Proof-of-concept for a Software-defined Real-time Ethernet

Matthias Herlich, Jia Lei Du, Fabian Schörghofer, Peter Dorfinger
Salzburg Research, Austria
Email: {matthias.herlich, jia.du, fabian.schoerghofer, peter.dorfinger}@salzburgresearch.at

***Abstract*—In this paper we present a proof-of-concept implementation to demonstrate advantages of applying software-defined networking (SDN) to real-time Ethernet (RTE) networks. Potential advantages include the support of arbitrary network topologies, central and dynamic network (re-)configurations, the efficient use of bandwidth, and fast failover mechanisms at network level. To illustrate the advantages we implemented small proof-of-concept setups that demonstrate the features based on OpenFlow and openPowerlink. Finally, we built a practical showcase. Based on our results we believe the concept of a software-defined real-time Ethernet is worth pursuing as it may lay the groundwork for more advanced real-time networks.**

I. Introduction

Real-time Ethernet (RTE) [1] allows the use of cost effective, widespread, and high-bandwidth Ethernet technology in industrial environments like automation, process control, and transportation where one key challenge is real-time communication. Typical RTE deployments in the past have been configured once to run without re-configuration for years or even decades. However, in the future these network configurations will likely be required to change more dynamically, for example for highly flexible production lines or software upgrades in modern cars that add new features which require changes to the in-vehicle network. Software-defined networking (SDN) is already increasingly used to centrally and dynamically configure and control non-real-time networks. The basic idea of SDN is to control network flows through a centralized intelligent controller with "dumb" forwarding devices in the data plane of the network. This approach allows a fine-grained control of network flows that can be used to implement, for example, traffic engineering and security applications. The idea of a software-defined real-time Ethernet has already been theoretically discussed [2]–[5]. In this paper we instead provide proof-of-concept implementations of previously identified key advantages [5] of software-defined real-time Ethernet solutions.

We discuss related work and shortly describe SDN in the next sections. Then we present our network setups and describe the advantages of using SDN which we demonstrate and show how we implemented these advantages in our testbed.

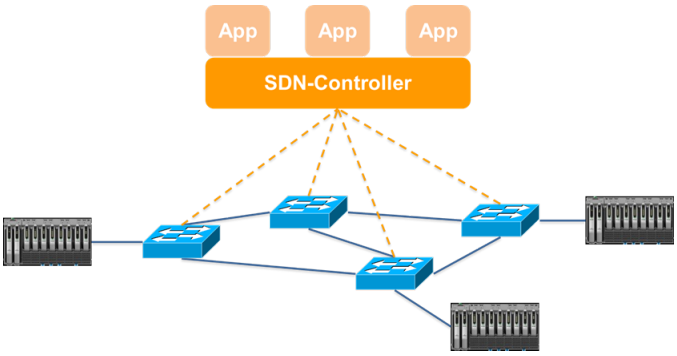

Fig. 1. The switches (middle) forward traffic flows programmed by a centralized SDN controller (top) between end devices (left/right). Note that the SDN controller could also be integrated into one of the end devices.

In the section after that we present the implementation of a real use case that could be easily implemented transparent for the RTE protocol using SDN.

II. Related Work

Gopalakrishnan [2] provides a general list of SDN features and gives some examples how the advantages could be applied to an IEC 61850-based network, but only mentions real-time capabilities in passing. Kalman [3] focuses on hardware abstractions and the ability to automatically configure networks using SDN. While both consider the advantages of SDN, they focus on industrial communication networks in general and not the advantages SDN can bring to real-time networks in particular. Dürkop et al. [4] provide a high-level concept for the automatic configuration of real-time Ethernet solutions. Our paper focuses on the communication aspect in more detail and proposes using SDN as an approach for network management. Furthermore, automatic (re-)configuration is only one of the advantages we describe in this paper that software-defined networking can bring to RTE networks. In our previous paper [5] we presented a theoretical overview of the advantages. All cited papers are of theoretical nature, whereas the key contributions of this paper are the proof-of-concept implementations.

III. Software-defined Networking

In most conventional communication networks, traffic is forwarded based on rules that are determined using distributed algorithms. In contrast to this approach, it has been suggested

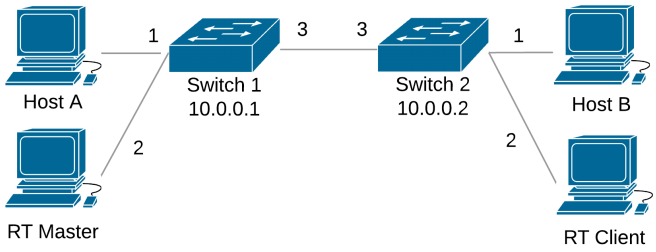

Fig. 2. We use this setup to demonstrate the basic use of OpenFlow for RTEs.

to use software applications in centralized controllers to direct traffic flows in software-defined networks (SDNs) [6]. This approach effectively decouples the control plane, which determines where traffic is sent, from the data plane, which forwards packets to their destinations (Fig 1). The SDN concept provides an approach for the implementation of a wide range of applications: flows can, for example, be dynamically rerouted based on load, failure, or security scenarios to provide bandwidth or latency properties, fast failover mechanisms or security. From an economic point of view, through standardization and centralization, SDN has the potential to simplify and reduce costs for network setup and operation. One standard for the implementation of software-defined networks is OpenFlow [7], which defines a communication protocol between network switches and one or more SDN controllers. The ideas in this paper are applicable to SDNs in general, but we will use OpenFlow as example.

IV. Implementation

In this section we present proof-of-concept implementations for the identified advantages based on OpenFlow and openPowerlink. openPowerlink[1] is an open source implementation of the Powerlink real-time Ethernet protocol. It supports all key protocol features and is compatible with Powerlink real-time Ethernet devices. We built multiple real-time Ethernet networks with a cycle time of 1 ms using OpenFlow-switches in our test lab. In earlier work [5] we described features that SDN enables in RTE networks: (1) centrally controlled network (re-)configuration, (2) the use of standardized, interchangeable network devices, (3) integrated standard interfaces for the simple retrieval of global network information and statistics, (4) simple addition and removal of network nodes, (5) the support of arbitrary topologies (including loops, parallel links, etc.), (6) built-in fast failover mechanisms, (7) support for multiple simultaneous communication paths, (8) support for multiple separated networks over one hardware infrastructure, (9) the isolation of faulty or adversary nodes, (10) dynamic load balancing and (11) efficient multicasting. The following 4 network setups demonstrate these features.

A. Setup 1: Basic Concept

In setup 1 (Fig. 2), hosts A and B exchange best-effort non-real-time traffic, and RT Master and Client run the Powerlink RTE protocol. This setup demonstrates features 1, 2, 8, and 11.

To create the network setup, the following OpenFlow rules are applied to switch 1. For this purpose we use the `ovs-ofctl`[2] command line tool, which is part of the Open vSwitch software package, to configure OpenFlow-switches. Note that any other combination of OpenFlow-compatible configuration/control tools and network devices could be also used to configure OpenFlow-Switches of any manufacturer.

```
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=1,actions=output:3
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=2,actions=output:3
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=3,dl_type=0x88ab,actions=output:2
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=3,dl_type=0x0800,actions=output:1
```

These rules forward traffic from host A and the RT Master to port 3 on switch 1. In the other direction filtering is done by the Ethertype values. IP traffic (`dl_type=0x0800`) is forwarded to host A and real-time traffic (Powerlink traffic: `dl_type=0x88ab`) is forwarded to the RT Master connected on port 2. Alternatively, the classification of traffic could be based on other parameters (e.g., VLAN tags). The mirrored commands are necessary to configure switch 2.

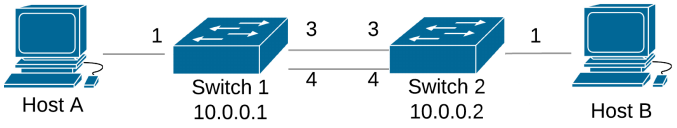

Fig. 3. The failure of one link between switches 1 and 2 will not be detected by the RTE.

B. Setup 2: Transparent Failure Recovery

SDN allows for arbitrary topologies, so adding a redundant link between two switches will not introduce routing loops (when configured correctly). Setup 2 (Fig. 3) demonstrates features 1, 2, 5 and 6. The additional link can, for example, be used for failure recovery. The necessary commands to create a group for fast failover are:

```
ovs-ofctl add-group tcp:10.0.0.1
  group_id=1,type=fast_failover,
  bucket=watch_port:3,output:3,
  bucket=watch_port:4,output:4
```

Here a 'group' with ID 1 is created. The group ID can be used as an output for a traffic flow:

```
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=1,in_port=2,actions=group:1
```

This fast failover rule will use port 4 in case of link outage on port 3 and will accept data from port 1 and 2. These setup commands have to be mirrored for switch 2.

[1] http://openpowerlink.sourceforge.net

[2] http://openvswitch.org/

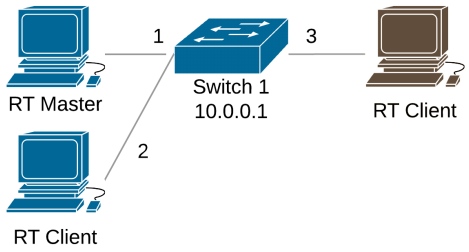

Fig. 4. An RT Client is connected to port 3 dynamically.

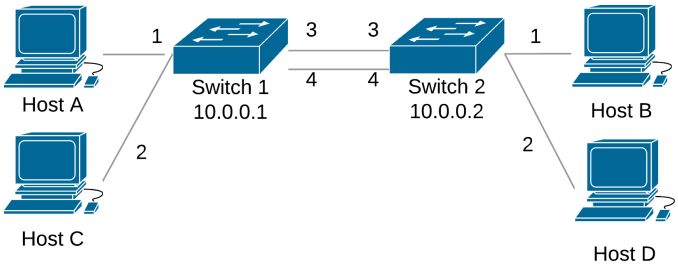

Fig. 5. The traffic between the hosts can be dynamically assigned to one of the two links between the switches.

C. Setup 3: Dynamic Nodes

Using SDN and OpenFlow, nodes can be added and removed centrally. This can for example be used to prevent wrongful connection of non-RT hardware to RT enabled ports and vice versa if Ethertype fields are matched. Another scenario would be the removal of faulty or adversary nodes. This setup (Fig. 4) demonstrates features 1, 2, 4, and 9. The network is initialized with one client:

```
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=1,dl_type=0x88ab,actions=output:2
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=2,dl_type=0x88ab,actions=output:1
```

To add another Client on port 3, traffic from the RT Master needs to be sent to ports 2 and 3 now:

```
ovs-ofctl mod-flow tcp:10.0.0.1
  in_port=1,dl_type=0x88ab,
  actions=output:2,3
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=3,dl_type=0x88ab,actions=output:1
```

A faulty node (e.g. the client on port 3 transmits incorrect data) can be also be easily disconnected from the network:

```
ovs-ofctl mod-flow tcp:10.0.0.1
  in_port=1,dl_type=0x88ab,actions=output:2
ovs-ofctl del-flows tcp:10.0.0.1
  in_port=3,dl_type=0x88ab
```

This removes the client on port 3 from the network: it will not receive frames and frames from port 3 will be dropped.

D. Setup 4: Dynamic Rerouting

Finally, in setup 4 (Fig. 5) features 1, 2, 3, 5, 7, and 10 are demonstrated. Flow statistics such as the number of bytes transmitted can be centrally collected using:

```
ovs-ofctl dump-flows tcp:10.0.0.1
```

Via a monitoring process, an increase in transmitted bytes per second could be detected and dealt with by adding another link to distribute traffic more evenly. In the topology in Fig. 5 port 4 is initially not used on any switch. If traffic from host C to host D increases over a threshold the additional link can be activated and used for traffic between hosts C and D using:

```
ovs-ofctl mod-flow tcp:10.0.0.1
  in_port=2,actions=output:4
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=4,actions=output:2
```

E. Implementation Remarks

In these setups we have shown the simplicity of implementing network-level features in a transparent way for the Powerlink RTE protocol; some of which would be difficult to realize with standard Ethernet. The examples used port numbers for simplicity and clarity. However, OpenFlow generally allows matching against and modification of fields in a frame to create complexer matching rules and actions. The difficulty of the implementation and the advantages depend on the feature and the RTE protocol. The Powerlink protocol only serves as an example. Some features can be implemented without any changes to the RTE protocol. For other features and combinations with other RTE protocols a tighter integration between software-defined network configuration and RTE protocol is necessary. Finally, some advantages, which SDN enables, may already be supported or could somehow be implemented with varying effort using combinations of existing RTE protocols and managed switches. However, even in such cases the use of SDN provides the advantage of an open and consistent framework to implement all advantages in a simple and manufacturer-independent way.

V. Final Example

For the proof-of-concept of a real usage example we created a new setup (Fig. 6). Two OpenFlow capable SDN switches were used, as well as one ordinary switch in between. This could also be an OpenFlow switch, however, we use an ordinary switch for simplicity.

In this scenario switch 2 will be replaced with a new switch (switch 3) during live operation of the RTE network (e.g. for maintenance purposes). During the exchange of the devices, packet losses must be avoided.

Setup: The basic setup, allowing data communication between hosts A and B can be achieved with the following OpenFlow rules:

```
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=1,actions=output:3
ovs-ofctl add-flow tcp:10.0.0.1
```

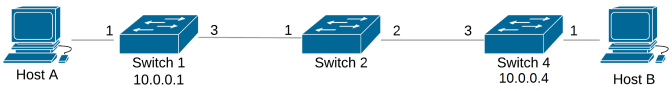

Fig. 6. The final example will exchange switch 2 without the RTE noticing.

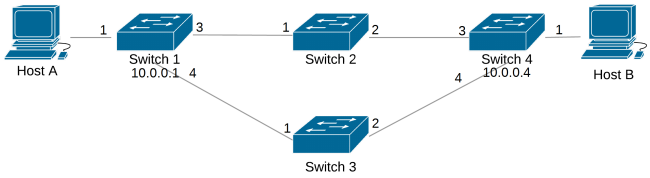

Fig. 7. Switch 3 is added to the topology without routing problems occurring.

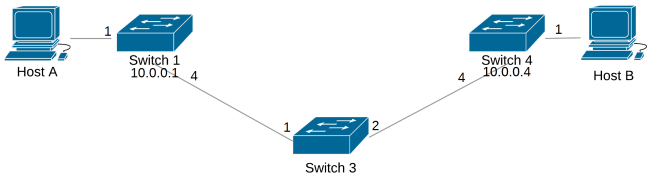

Fig. 8. After the reconfiguration, switch 2 can be removed.

```
  in_port=3,actions=output:1
ovs-ofctl add-flow tcp:10.0.0.4
  in_port=1,actions=output:3
ovs-ofctl add-flow tcp:10.0.0.4
  in_port=3,actions=output:1
```

Feature: Then, we add another switch to the topology (Fig. 7) by connecting it to two unused ports on switches 1 and 4. The new switch will not participate in data transmission, unless rules are changed on switches 1 and 4. To exchange switch 2 with switch 3, the following OpenFlow rules are changed:

```
ovs-ofctl add-flow tcp:10.0.0.4
  in_port=4,actions=output:1
ovs-ofctl add-flow tcp:10.0.0.1
  in_port=4,actions=output:1
```

This will transmit incoming traffic from port 4 to port 1 on switches 1 and 4, but as no traffic is yet originating from these ports, this will not change any flows yet.

```
ovs-ofctl mod-flows tcp:10.0.0.4
  in_port=1,actions=output:4
ovs-ofctl mod-flows tcp:10.0.0.1
  in_port=1,actions=output:4
```

The 'mod-flows' sub-command changes an existing flow to output on port 4. As each line is executed individually, data traffic uses switch 2 in one direction and switch 3 in the other for a short time. Traffic is symetric again after the last command is executed and switch 2 can be removed (Fig. 8).

The complexity of the final example without SDN depends on the real-time network. We compared the SDN solution

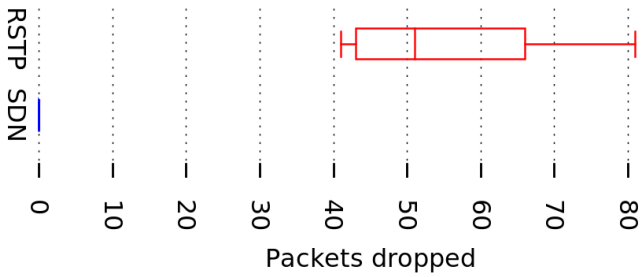

Fig. 9. Our SDN-based solution loses no packets (9 repeats).

to Cisco 2960 switches and the RSTP protocol, but other solutions based on methods in [8] would also be possible. The exchange in RSTP is made by changing the root bridge ID from switch 3 to a lower value than that of switch 2. Switch 3 will become the root bridge and traffic will be forwarded with the newly configured root bridge. Our results showed that the RSTP approach will result in packet loss during convergence of the spanning tree. We detected a median packet loss of 51 packets per test (with a mean of 55). No packet loss did occur with the SDN approach (Fig. 9). This is completely transparent for the RTE (or any other) protocol and can be done during runtime. In particular, this means that the delays of all packets stay the same (as long the substituted switches are equally fast and the cables are equally long).

VI. Conclusion

We provided an introduction to software-defined networking and presented our idea of applying SDN concepts to RTE networks. We implemented proof-of-concept setups and a real use case in our testbed with only few lines of code. This use case could be easily implemented transparent to the RTE protocol using SDN. We think that a fully functional software-defined real-time Ethernet, is a promising endeavor which will make future RTE networks more flexible, efficient and robust.

Acknowledgment

This work was partially funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) in the project OpenheaRTEd (FFG Project No. 849972).

References

[1] M. Felser, "Real-time ethernet–industry prospective," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1118–1129, 2005.

[2] A. Gopalakrishnan, "Applications of software defined networks in industrial automation," 2014.

[3] G. Kalman, "Applicability of software defined networking in industrial ethernet," in *Telecommunications Forum Telfor*, Nov 2014, pp. 340–343.

[4] L. Dürkop, J. Jasperneite, and A. Fay, "An analysis of real-time ethernets with regard to their automatic configuration," in *Factory Communication Systems (WFCS), IEEE World Conference on*, May 2015, pp. 1–8.

[5] J. Du and M. Herlich, "Software-defined networking for real-time ethernet," in *Proceedings of the International Conference on Informatics in Control, Automation and Robotics*, 2016.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[7] *OpenFlow Switch Specification Version 1.5.1*. Open Networking Foundation, 2015.
[8] L. Wisniewski, M. Hameed, S. Schriegel, and J. Jasperneite, "A survey of ethernet redundancy methods for real-time ethernet networks and its possible improvements," *Proc. IFAC FeT*, pp. 163–170, 2009.