Application-Aware Industrial Ethernet Based on an SDN-Supported TDMA Approach

Eike Schweissguth, Peter Danielis, Christoph Niemann, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany
Email: eike.schweissguth@uni-rostock.de

***Abstract*—In industrial automation environments, networks providing a reliable and timely data delivery are required. Fulfilling this need, Industrial Ethernet (IE) systems have established as an important networking technology in many application areas. Although there are several IE solutions on the market, all of these systems have notable drawbacks, like limited scalability or the introduction of a Single Point of Failure (SPoF).**

Therefore, we propose a novel IE system that is based on Software Defined Networking (SDN). Originally meant for data center and IT networks, the SDN concept offers features like central network management functions and a fine-grained traffic control that allows to support many applications with diverse requirements even in the same network. Thereby, SDN is also perfectly suited for complex automation environments. To guarantee RT data transmission as well as scalability and an efficient resource usage, our IE system uses a Medium Access Control (MAC) scheme that is based on a Time Division Multiple Access (TDMA) mechanism that is extended by simultaneous data transmissions on physically separate links. The enhanced TDMA mechanism is configured by a joint routing and scheduling algorithm that takes application requirements into account. Our theoretical analysis as well as results achieved with a prototype implementation of the system confirm the applicability of our concept in demanding automation environments with applications that require a worst case communication latency below 1 ms.

I. Introduction

In industrial automation environments, many applications require a reliable real-time (RT) communication system that guarantees the timely delivery of messages (e.g., sensor or control data), because a violation of communication deadlines can have unacceptable consequences like damage imposed on parts of the production facility.

Traditionally, fieldbusses are therefore used in many application areas to realize an RT communication. In recent years, Industrial Ethernet (IE) systems, which promise several advantages [1], have established as a serious competitor to these fieldbusses. Similar to the fieldbusses, several IE solutions of different companies have emerged, which all solve the problem that standard Ethernet (IEEE 802.3) is not RT capable due to its medium access control (MAC) scheme. Even in full-duplex switched networks, unacceptable packet delays or packet loss can occur due to the over-utilization of links as, in general, all connected terminal devices (hosts) are allowed to send data at line rate at any time. Therefore, existing IE solutions introduce an additional MAC that assigns send permissions to the hosts in way that a certain worst case latency of a data transmission can be guaranteed.

Nevertheless, every IE system has at least one of the following drawbacks [1]:

- Limited scalability in terms of the number of devices
- Incorporation of a Single Point of Failure (SPoF)
- Insufficient self-configuration features
- Increased costs due to the use of proprietary hardware instead of standard IEEE 802.3 hardware
- Use of broadcast transmissions inhibits simultaneous transmissions of multiple data flows on separate links and thereby limits network efficiency

Especially the limited scalability is a major problem considering the future increase of the number of devices in automation environments, which will be necessary to support the more comprehensive approaches to horizontal and vertical integration and big data analysis (cf. Industrial Internet [2], Industrie 4.0 [3]).

Therefore, our objective was the design of a new RT communication system that avoids the mentioned drawbacks of the existing IE systems while providing comparable latency characteristics and independence of the used network topology. Additionally, it should be more flexible and based on open standards, so that the further development is simplified and an optimal cost-effectiveness is achievable. In our concept, we exploit the SDN controller's central view on the network to discover the topology and to collect the requirements of the applications using the network. Then, we use the collected information in a joint routing and scheduling algorithm that computes routes for all required network connections as well as a send schedule that ensures compliance with the application requirements (e.g., bandwidth, maximum latency). In contrast to classical TDMA systems, the provided network configuration allows simultaneous data transmissions on physically separated routes (Space Division Multiple Access or SDMA). Finally, we use the features provided by existing SDN technologies like OpenFlow to install the customized, application-specific routes in the network.

Specifically, our contributions are:

- Novel concept (SDN-based TDMA mechanism) for end-to-end Quality of Service (QoS) avoiding the drawbacks of the well-known IE systems

- Original heuristic algorithm for joint routing and scheduling in switched networks
- Results of a real hardware testbed running a fully functional prototype implementation with achievable latencies below 1 ms

The rest of the paper is structured as follows. Section II introduces the concept of SDN and explains the used network delay model. Section III presents our novel concept for RT communication systems. Section IV describes our prototype implementation of the concept and achieved results. Then, pros and cons of our concept are discussed, before Section V summarizes related approaches to RT networking. A conclusion is given in Section VI.

II. Basics

A. Network Delay Model

In this work, we consider the end-to-end network delay from the application layer of the sender to the application layer of the receiver. Therefore, network delay can be divided into five categories. The first delay is caused by the network stack (e.g., UDP, IP and Ethernet implementation) of the sender (denoted as $t_{Software_S}$). It is followed by the transmission delay $t_{Transmission}$ at the sender's Ethernet PHY and several propagation delays $t_{Propagation_i}$ and switch delays t_{Switch_i}, whose numbers depend on the route through the network to the receiver. Finally, the packet is delayed by the network stack of the receiver ($t_{Software_R}$). These five delays are summarized as $t_{Delivery}$.

A classical approach to enable a RT capable communication is the use of a TDMA scheme, in which the hosts get the send permission one at a time and each host is only allowed to send for a certain amount of time. If such a time slot based RT communication system is used, we must additionally consider that packets cannot be sent immediately, but they have to wait in a packet queue until the respective device obtains the send permission. Considering this additional delay t_{Queue}, the complete packet latency is given by Equation 1.

$$t_{Latency} = t_{Queue} + t_{Delivery} \tag{1}$$

Note that the switch delay normally depends on the link utilization because packet queuing delay increases for highly utilized link. In case of a completely time slot based system, queuing delay at switches is not a concern as we can assume that switch queues are always empty. Then, switch delay is only dependent on packet size and switch implementation. Moreover, $t_{Transmission}$ is of significant importance for time slot based systems, because the transmission time resembles the length of a required time slot. I.e., a time slot designated to an application must have at least the length that equals the transmission time of the data to be sent (including all headers). Therefore, the minimum slot length is given by the well-known formula for transmission delay computation (Equation 2):

$$t_{SlotLength} \geq t_{Transmission} = DataSize/Bandwidth \tag{2}$$

Within this work, the length of an assigned time slot is called $t_{SlotLength}$. If the amount of data that is generated between two time slots is small enough to be sent completely in the next time slot, the data's worst case (WC) delivery time $t_{DeliveryWC}$ is given by Equation 3.

$$t_{DeliveryWC} = t_{Software_S} + t_{SlotLength} + t_{Software_R} + \sum_{i=1}^{N+1} t_{Propagation_i} + \sum_{i=1}^{N} t_{Switch_i} \tag{3}$$

1) Requirement Parameters: When using a time slot based RT communication system, the requirements of an application can be defined by the following three parameters.

- $t_{DataInterval}$ describes the time interval with which an application generates data packets that must be sent. For example, in an automation environment this could be the time interval for the periodic sending of sensor data.
- $t_{SlotLength}$ determines how long an assigned time slot must be so that all data of a single data interval can be transmitted within a single time slot.
- $t_{MaxLatency}$ describes the maximum acceptable latency of the communication, including all latencies of Eq. 1.

The communication system can use this information about application requirements to configure all time slots appropriately, so that sufficient bandwidth is available for each application and all application deadlines are met.

B. Software Defined Networking

In traditional networks, each network infrastructure device (NID) that works on OSI layer 2 or higher has its own control logic that is responsible for routing decisions. For example, this control logic implements the MAC address learning mechanism in simple switches or routing protocols like RIP, OSPF or IS-IS in more advanced switches and routers. In contrast to this traditional approach, the SDN concept moves the control logic from the NIDs into a central SDN controller [4] [5]. The SDN controller then makes all routing decisions and establishes these in the network by installing appropriate packet processing rules in the NIDs. Additionally, the network becomes programmable and as a consequence, custom, application-specific routing strategies can be deployed easily. Compared to a distributed control logic, the controller's central view on the network also simplifies the implementation of important parts of the control logic (e.g., topology discovery or routing protocols). The interface for the communication between controller and NIDs, which is used for rule installation, is called Southbound API. Optionally, the controller can also offer a so called Northbound API for the communication with applications using the network. For example, applications can use this interface to request a specific QoS.

TABLE I
Exemplary OpenFlow Rule.

In Port	MAC SRC	MAC DST	Ether-type	IP SRC	IP DST	IP Proto.	TCP SRC	TCP DST	Action
*	*	*	IPv4	10.0.1.3	10.0.1.5	TCP	7777	6100	Out#3

TABLE II
STARTUP PROCESS OF THE PROPOSED RT COMMUNICATION SYSTEM.

0. Switches connect to the controller
1. Discovery of switch interconnections
2. Discovery of connected terminal devices
3. Request for communication requirements
4. Computation of routes and schedule
5. Route installation and send schedule distribution
6. Host synchronization
7. RT communication of applications

1) OpenFlow: One widespread and freely available Southbound API is the OpenFlow protocol [6] [5]. In the following, OpenFlow-capable NIDs are called OpenFlow switches. OpenFlow defines the communication interface between OpenFlow switches and the controller and it specifies the structure of packet processing rules. Table I shows an exemplary OpenFlow rule for a specific traffic flow. An OpenFlow rule installed in a switch is applied to every incoming packet whose header fields match those of the rule. Wildcards (*) in a rule match an arbitrary entry in the packet header. If none of the installed rules matches the header of an incoming packet, the OpenFlow switch requests the correct treatment of the packet at the SDN controller. The SDN controller typically answers such a request with the installation of a new, appropriate rule. If the expected traffic flows are known beforehand, an SDN controller might also install appropriate packet forwarding rules in the switches proactively, so that no further requests and responses between switches and controller are necessary during the actual communication. As OpenFlow has become a widespread protocol in many areas of networking and it is not very demanding to the hardware (i.e., implementations based on existing hardware are possible) [7] [6], it can be expected that future IE switches will also support OpenFlow.

III. CONCEPT

In our concept, we propose a MAC that is based on the traditional TDMA approach. We enhance the TDMA mechanism by exploiting two important features of SDN. At first, topology information is easily accessible by the central SDN controller. Secondly, standardized SDN interfaces like OpenFlow offer the possibility to install customized, application specific routes in the network. By taking topology information into account and controlling routes, it is possible to allow multiple devices to send data over physically separated links at the same time. This simultaneous transmission on separate links is also called Space Division Multiple Access[1] (SDMA). In this joint TDMA/SDMA approach managed by the central SDN controller, hard RT communication is ensured by the traditional TDMA principle, whereas the additional use of SDMA facilitates the efficient use of network resources and provides good scalability in terms of network size.

[1]The term SDMA is more common in the wireless context, but also describes the principle of our system very well.

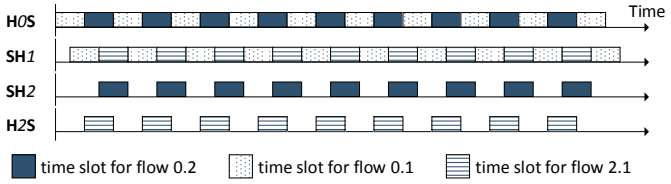

Fig. 1. Exemplary link schedule for a single switch topology with three hosts. Naming scheme: **H*0*S** denotes the link from host 0 to the switch. Flow 0.2 denotes a communication flow from host 0 to host 2.

A. Overview

During the startup phase of the network, several steps are required to configure the network appropriate for the application requirements. These steps are depicted in Table II.

Steps 0-2: At the beginning, all OpenFlow switches connect to the SDN controller, which then performs a network topology discovery mechanism. SDN Controller frameworks like POX [8] already ship with discovery modules that can be configured or slightly modified to provide such a topology discovery at system startup. The discovery of switch interconnections relies on specially crafted LLDP packets, whereas hosts can be discovered by passively monitoring network traffic or by an active search like a ping sweep.

Step 3: Afterwards, it is necessary to inform the controller about the communication requirements of each terminal device. This information is given by flow definitions (determining the endpoints of a communication) and corresponding per-flow requirement parameters according to Section II-A1.

Step 4: The controller then uses the collected information about the topology and the communication requirements to compute a route and a send schedule for each flow. During this computation, the controller creates a conflict-free link schedule for each single link in the network that specifies when the link is used by which flow. Thereby, a deterministic packet delivery is ensured, as neither unexpected delays through multiple packets in a switch queue nor packet loss due to link congestion can occur. An example of a conflict-free link schedule for a single switch topology with three hosts, which are communicating over three different traffic flows, is given in Figure 1. The send schedule that is relevant for the hosts is given by the assigned time slot on the first link of the route (**H*0*S** and **H2S** in the example). Note that only the hosts are expected to be able to send data following a given schedule, whereas switches simply forward data with a certain delay determined by packet size and switch implementation. The switch delay causes the time shift between time slots of a flow on successive links of a route.

Step 5: Following the computation of routes and link schedule, the SDN controller uses of the standardized OpenFlow interface to install the respective rules for the routes in the OpenFlow switches. Then, it transmits the corresponding send schedule to every host via a northbound interface.

Step 6: Before the RT communication in time slots can start, the hosts need to synchronize their clocks to a common

reference clock. Afterwards, they can autonomously detect if they currently have the permission to send data of a certain flow (clock time within respective send time slot). The host synchronization can be realized through the use of a well-known synchronization algorithm like the one used by the Network Time Protocol (NTP). Although this algorithm uses a single time server, it is possible to avoid a SPoF by equipping each device with the same synchronization functionality, so that each device can take the role of the time server.

Step 7: Finally, the RT communication can start. During this communication, the devices periodically repeat the communication pattern determined by the send schedule.

Resembling an important part of our work, details on the definition of communication requirements of the applications, as well as an original routing and scheduling algorithm for the computation of link schedules like depicted in Figure 1 are described in the following subsections.

B. Flow and Constraint Discovery (Step 3)

In this step, the controller collects information about the communication requirements of the hosts and their running applications. The endpoints of a communication are defined by an OpenFlow header (cf. Table I). On the one hand, it is possible to specify necessary connections on a device level by wildcarding transport protocol header fields. On the other hand, a definition of multiple connections between the same pair of hosts, each dedicated to a specific application, is possible by using the transport protocol header fields. For every given flow definition, the SDN controller also needs the corresponding constraints (cf. Section II-A1) to be able to reserve time slots of appropriate length and frequency. Additionally, it is possible to use a single device level flow and multiple application level flows between the same pair of hosts at the same time. In this case, the application level flows can be used to satisfy the requirements of specific applications, whereas the device level flow can be considered as a general communication budget for further applications, for example without RT requirements. As a flow only defines an unidirectional communication, bidirectional communication between two hosts requires the definition of dedicated flows for either direction.

Different strategies can be used for gathering flow definitions and constraints. With sufficient a priori knowledge about the topology and the applications using the network, it is possible that the controller either reads the flow definitions and constraints from a file or generates them following a predefined pattern. If topology and traffic pattern are completely unknown beforehand, it will be necessary that the controller requests the communication requirements from the hosts via a northbound interface.

C. Routing and Scheduling Algorithm (Step 4)

To create link schedules similar to the one shown in Figure 1, a routing and scheduling algorithm is necessary, which finds an appropriate (i.e., satisfying all flow constraints) route and a time slot configuration for every flow. Before our routing and scheduling algorithm is presented, we will briefly explain how the different parameters of latency and the flow-specific constraints (cf. Section II-A) are related and how they must be taken into account by the algorithm to satisfy the flow constraints.

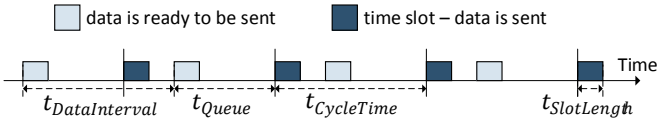

Fig. 2. Visualization of a time slot configuration.

At first, the time slot length for the considered flow is set according to the corresponding requirement parameter $t_{SlotLength}$. This configuration guarantees that the sender can transmit all data packets of a single data interval of the flow in a single time slot. If the period $t_{CycleTime}$ of the time slot is additionally chosen according to Equation 4, data of multiple data intervals cannot accumulate in the host's queue and all data packets are always sent in the next time slot. At the beginning, $t_{CycleTime}$ is set as high as possible (i.e. $t_{CycleTime} = t_{DataInterval}$), as this leads to a lower link utilization.

$$t_{CycleTime} \leq t_{DataInterval} \tag{4}$$

After setting a time slot length and cycle time, it is necessary to search for an appropriate route for the flow. The goal of the route search is to find a short route that does not cause time slot conflicts in the link schedule. Based on this route, it is possible to compute $t_{DeliveryWC}$ (cf. Eq. 3). Furthermore, as the generation of data packets to be sent is typically not synchronized with the time slots, in the worst case, packets are generated right after the beginning of a time slot and therefore have to wait a full cycle time until they can be sent (worst case for t_{Queue}, cf. Figure 2). This results in the worst case latency given by Equation 5.

$$t_{LatencyWC} = t_{CycleTime} + t_{DeliveryWC} \tag{5}$$

Then, Eq. 6 must be satisfied to meet the flow requirements. If Eq. 6 is not satisfied, there is still the option to reduce $t_{CycleTime}$. In contrast, lowering $t_{DeliveryWC}$ is not possible assuming that our route search already chose the route with the lowest delivery time.

$$t_{LatencyWC} = t_{CycleTime} + t_{DeliveryWC} \leq t_{MaxLatency} \tag{6}$$

After changing $t_{CycleTime}$, it is necessary to compute a new route because new time slot conflicts can arise in the link schedule on the previously computed route due to the new time slot configuration. Finally, Eq. 6 must be checked again with the new values for $t_{DeliveryWC}$ and $t_{CycleTime}$. Potentially, these steps of lowering the cycle time and computing a new route have to be repeated multiple times.

Additionally, some applications may benefit from jitter-free packet delivery times. This is generally not given if $t_{CycleTime}$ is chosen smaller than $t_{DataInterval}$ because t_{Queue} then varies for every cycle. If necessary for the application, it is

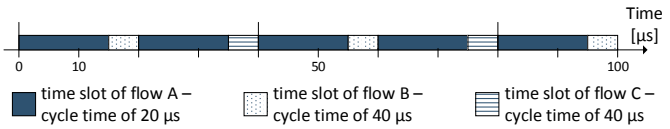

Fig. 3. Arbitrarily chosen example for the illustration of per-flow cycle times and slot lengths.

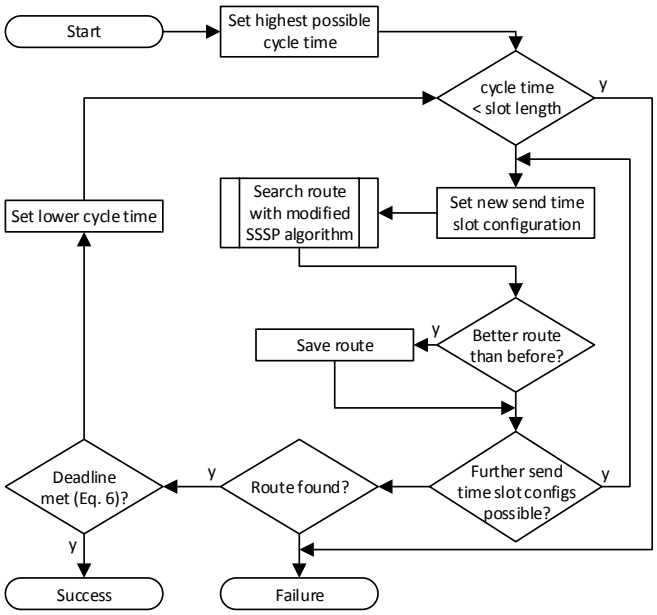

Fig. 4. Flowchart of the main part of the routing and scheduling algorithm.

possible to avoid that the time slot mechanism causes jitter by additionally choosing $t_{CycleTime}$ in accordance with Eq. 7.

$$t_{CycleTime} = \frac{t_{DataInterval}}{n}, n \in N \tag{7}$$

As mentioned above, the route search requires a check if all time slots assigned to a link never overlap. This check must also work if the time slots use different cycle times and have an arbitrary length. Such a check can be realized as follows. At first, the least common multiple (LCM) of the cycle times of all time slots is computed. Then, the schedule of the link is checked for time slot conflicts up to the LCM. If no time slot conflicts occur up to the LCM, there will not be conflicts in the future as the time slot pattern repeats after the LCM. For example, Fig. 3 shows a link with time slots of three flows. The LCM of the cycle times is 40 µs. As the first 40 µs of the link schedule are conflict-free, the schedule for this link is feasible.

Following these considerations, it is possible to formulate a routing and scheduling algorithm that enables the use of different time slot lengths and cycle times per flow. For every required RT flow, our routing and scheduling algorithm executes the steps depicted in Figure 4. At first, the cycle time for the time slots of the flow is set to the highest possible value considering Eq. 4 , as it is preferable to use a high cycle time due to a lower link utilization and a lower chance to conflict with other flows. Then, a send time slot is determined. The send time slot is the time slot of the first link of a route and thereby it defines when the host is allowed to transmit the data of the flow. With the given send time slot configuration, a modified version of Dijkstra's single source shortest path (SSSP) algorithm is run to search for a possible route. For this purpose, the SSSP algorithm has been extended to be aware of time slots: It knows the time slots that have been assigned to previously processed flows and when searching for a route for another flow, it only adds links to a route if this does not lead to a time slot overlapping. During this process, the required time slots for every link of a route can be computed based on the given send time slot and known switch delays. Finally, the modified SSSP algorithm behaves like the original SSSP algorithm but without adding a link to a route if this would lead to a time slot conflict. This way, the modified SSSP algorithm jointly solves the routing and scheduling problem. As the host can start the data transmission at an arbitrary point in time between 0 and $t_{CycleTime}$ of the respective flow (with time steps limited by the hosts time resolution), many different send time slot configurations are possible. Therefore, the modified SSSP algorithm is repeated with every possible send time slot configuration and only the best route found (by hop count metric) is saved. Afterwards, the worst case latency (Eq. 5) of this route and its corresponding time slot configuration is computed and the algorithm checks if it meets the deadline (Eq. 6). If the deadline is met, appropriate forwarding rules for the route are installed in the switches and the corresponding time slots are added to the link schedule. If the best route found does not satisfy Eq. 6, the search for routes is repeated with a lower cycle time (still considering Eq. 4 and Eq. 7). The algorithm stops with an error if no route is found at all for a certain cycle time because it is unlikely that a route can be found with an even lower cycle time as the chance of schedule conflicts increases. The algorithm also stops if the cycle time drops below the required slot length.

IV. Implementation and Results

A. Implementation

The SDN controller of our implementation of the presented system is based on the POX framework. The topology discovery was realized by means of modified POX modules. To manage the different phases of the system (cf. Table II) and for the routing and scheduling algorithm as well as the custom northbound interface, new modules were added to POX. The northbound interface for the communication between controller and hosts uses a custom JSON format. Python's standard library functions were used for parsing and creating JSON documents. As hosts, ZedBoards (667MHz ARM Cortex A9 Dual Core, 512MB DDR3 RAM) running FreeRTOS with the lwIP [9] network stack were used, with additional custom features (synchronization, packet queuing, time slots, JSON interface) implemented in C++. The library rapidJSON was used for JSON message parsing. These hosts

offer a time resolution of 10 µs. A higher time resolution can be configured but the platform then produces unreliable results.

In the test setup, the OpenFlow-capable HP 2920 datacenter switch was used to connect the hosts. This store-and-forward switch processes the relevant subset of OpenFlow rules in hardware. Therefore, it offers the possibility to conduct latency measurements that provide realistic results for a network with gigabit Ethernet.

Note that we use standard hardware components and NIC drivers, as they provide all features required by our system. For the use in a production network, it will be necessary to certify such components with respect to their RT capabilities, which will require a detailed analysis of the components to verify that unexpected packet delays can never occur.

B. Initial Parameter Measurement

Before taking the developed system into operation, the parameters t_{Switch}, $t_{Software_S}$ and $t_{Software_R}$ of the used components need to be measured and made known to the controller.

We implemented the queuing mechanism and the ability to send in flow-specific time slots as an additional wrapper around the used network stack. Applications have to use API functions of this wrapper to send data. The wrapper then implements the queuing and time slot functionality and calls API functions of the network stack when a time slot is reached and packets can be sent.

When the wrapper calls an API function of the network stack to send a packet, the time $t_{Software_S}$ passes before the data is transmitted. To comply with the given time slot, this software delay must be known and compensated, e.g., by making the respective API call exactly by $t_{Software_S}$ before the beginning of the respective time slot or by adding an appropriate safety margin between time slots. In practice, $t_{Software_S}$ is not perfectly constant (e.g., dependent on packet sizes) and thus cannot be fully compensated by making the respective API call earlier by a fixed amount of time. Therefore, a safety margin (B in Fig. 5) must be added between the time slots, with a length that equals the difference between the maximum and minimum possible value of $t_{Software_S}$.

Furthermore, t_{Switch} must be determined because it is used by the controller during computation of the link schedule. In practice, t_{Switch} is not constant either and therefore it is necessary to introduce an additional safety margin between time slots for every switch a packet traverses (E in Fig. 5). The length of this per-switch safety margin must equal the difference of the maximum and minimum of t_{Switch}.

Additionally, the maximum values of $t_{Software_S}$, $t_{Software_R}$ and t_{Switch} must be known to the controller beforehand to correctly compute the expected worst case delivery time of packets (cf. Eq. 3) and check if the deadlines are met (Eq. 6).

As we measured the switch and software delay values with our ZedBoard hosts, we pessimistically rounded the measured minimum delay values down by an additional 10 µs (i.e., 1 clock tick) and the measured maximum delay values up an additional 10 µs to accommodate the host's limited measurement accuracy.

TABLE III
FLOW CONSTRAINTS FOR THE PRESENTED TEST CASE.

Flow Name	Source Host	Dest. Host	$t_{MaxLatency}$	$t_{DataInterval}$	$t_{SlotLength}$
0.1	0	1	600	400	20
2.3	2	3	600	400	20
4.5	4	5	600	400	20
6.7	6	7	600	400	20
0.7	0	7	600	400	20
2.1	2	1	600	400	20
4.3	4	3	600	400	20
6.5	6	5	600	400	20

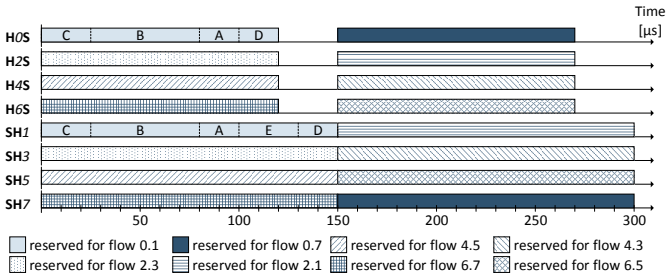

Fig. 5. Link Schedule for the presented test case. Reservation consists of: A: Actual transmission time slot B: Safety margin for variable software delay C: Safety margin for synchronization inaccuracy (early clock) D: Safety margin for synchronization inaccuracy (late clock) E: Safety margin for variable switch delay.

TABLE IV
MEASUREMENT RESULTS FOR THE PRESENTED TEST CASE.

Flow Name	$t_{Latency}$ Minimum [µs]	$t_{Latency}$ Maximum [µs]	$t_{Latency}$ Average [µs]	$t_{Latency}$ Std. Dev. [µs]
0.1	320	320	320,0	0,0
2.3	410	430	423,87	4,87
4.5	370	430	375,27	5,05
6.7	380	390	386,42	4,79
0.7	70	80	73,78	4,85
2.1	180	180	180,0	0,0
4.3	120	290	126,92	7,69
6.5	110	120	116,42	4,79

The accuracy of the synchronization mechanism is also limited by the host's time resolution and in practice, the clocks of the master and synchronized slaves differ. This must be taken into account by adding further safety margins (C and D in Fig. 5).

Figure 5 visualizes all mentioned safety margins and their exact values for our implementation.

C. Results

We verified that our routing and scheduling algorithm as well as our prototype implementation are correctly working with several successful tests using ring, tree and single switch topologies and multiple traffic patterns. For simplicity, we present our results of an example using a single switch topology with eight ZedBoard hosts (0-7) connected to an HP

2920. Traffic pattern and flow constraints are shown in Table III. The link schedule generated by our algorithm is shown in Figure 5. By the example of two of the Links (**H*0*S** and **SH*1***) the figure also shows how each reserved time is composed of the actual time slots and the safety margins. Note that it is correct that the time slots on links successively used by a flow start at the same time, because the minimum switch delay of our used switch was measured as 0 µs. The cycle time of all time slots was set to 400 µs. With the described setup, one-way latency measurements were conducted by sending UDP packets with a payload of 1000 bytes over the respective flows. A measurement series consists of 5000 measured delay values per flow. Measured latency values are depicted in Table IV. The measurement was conducted on application level, thereby including all delay factors of Eq. 1.

The results satisfy the expectations and can easily be explained. As the chosen cycle time of the time slots and the data interval of the test application are equal, the queuing delay of packets of the same flow is always the same during a complete measurement series, resulting in a low standard deviation. Nevertheless, the first packet generation and packet queuing of a flow takes places at an arbitrary point in time between two time slots, leading to a random queuing delay between zero and $t_{CycleTime}$ for every flow and thereby to different average latencies per flow. $t_{Delivery}$ can be assumed as equal for all flows because of the equal single hop routes. The typical difference of up to 20 µs between minimum and maximum latency value of a single flow can be explained by random errors that can occur due to the limited clock accuracy and the used synchronization mechanism.

The results are generally very convincing because they confirm that the routing and scheduling algorithm works correctly and creates a feasible schedule that satisfies all flow constraints. Additionally, the results show that our prototype implementation can content the flow requirements without any exceptions (maximum latency value below $t_{MaxLatency}$). This is especially remarkable because with data intervals far below 1 ms the flow requirements are quite demanding and fulfill the needs of many hard RT applications.

D. Discussion

The introduction of safety margins is an issue that limits the efficient use of available network resources. Nevertheless, there are several possibilities to mitigate this problem. E.g., hosts with a higher time resolution will allow a more precise synchronization and the corresponding safety margins (C/D in Figure 5) can be reduced accordingly. Additionally, with a more precise measurement of switch and software delays (cf. Section IV-B) safety margins B and E can be reduced. By the use of OpenFlow-capable cut-through switches, safety margin E may be even negligible. For the use in data centers, switch fabrics offering such features already exist (e.g., Mellanox SwitchX-2 EN). Thus, it becomes apparent that our novel IE concept facilitates even a much higher efficiency than we were able to demonstrate with our prototype implementation.

In our concept, the additional use of SDMA besides TDMA allows the simultaneous communication of multiple traffic flows over physically separate parts of a network. Such an efficient use of the network topology is not possible in several existent IE solutions because they make use of broadcast transmission (or hubs). Thereby, our concept offers a significantly better scalability. As the configuration of the combined TDMA/SDMA system is autonomously performed by the routing and scheduling algorithm of the SDN controller, the manual configuration effort is decreased to a minimum. Additionally, every computed route and every time slot of the link schedule is flow-specific (cf. Table I) and not device-specific, which means that it is possible to (independently) take account of the different requirements of multiple communicating applications running on a single host. As the SDN controller is only involved in the startup phase of system but not in the actual real-time communication, it is not necessary to analyze the SDN controller with respect to RT capabilities. Therefore, even standard SDN controller frameworks and existing libraries of the respective programming language can be used for the controller application. Furthermore, the SDN controller is no Single Point of Failure (SPoF) because once the system is configured, RT communication can continue even in case of a controller failure. The controller is only required for a restart or reconfiguration of the communication system. As the controller does not require specialized hardware, but can be executed on any PC or embedded device, it is also easy to replace in case of a failure. By the implementation of the control logic in software, the system becomes also very flexible and adaptable to different application areas. Especially the opportunity to select different routing and scheduling modules depending on the expected topology, traffic pattern and application requirements offers great optimization potential. For the terminal devices and OpenFlow switches, no special hardware features are required and thereby, standard Ethernet hardware can be used.

V. Related Work

Although the use of SDN in industrial automation networks has been considered in the past [10] [11], there are few publications that propose an SDN-based networking concept which is able to provide application-specific, guaranteed worst case latencies. In [12], Guck and Kellerer present an SDN-based end-to-end QoS mechanism that works with flow-specific bandwidth allocations and additionally uses static priority queuing in the switches. Similar to our proposed concept, the controller collects information about the network topology and flow requirements and then uses these in a joint admission control and routing algorithm to compute an appropriate network configuration. The given analysis proves that the system is able to provide guaranteed worst case latencies. The advantage of the approach of Guck and Kellerer is that host implementations become simpler compared to our proposed concept, as they do not need to implement synchronization features and a TDMA mechanism. Nevertheless, with their approach it is difficult to achieve low latencies over multi-hop

routes, as queuing delays of every traversed switch must be considered. Additionally, low-jitter communication becomes difficult as packet delay depends on the behavior of multiple hosts communicating over the same links and packet queues.

Further work that addresses SDN-supported QoS-provisioning is typically focused on network optimization for soft RT multimedia applications (e.g. video streaming, VoIP, online gaming). For example, the authors of [13] formulated an Integer Linear Programming (ILP) solution that uses topology information and application requirements (minimum bandwidth, maximum delay and packet loss) to compute appropriate routes. Additionally, they propose the use of network monitoring features to detect requirement violations due to dynamically changing network load and they use this information to trigger path recomputations. In [14], the authors also propose a QoS mechanism in which the SDN controller monitors network utilization and reactively reconfigures routes if it detects that application requirements with respect to latency or bandwidth are violated. The authors of [15] propose an SDN-based network resource management that categorizes traffic by the use of an additional IP header field that contains an application ID. Based on the categorization in three classes, different routing algorithms are applied for the respective traffic flows. These algorithms use different cost functions to weight latency, jitter and packet loss and thereby enable a routing optimization for different applications. Similar to the previously mentioned works, the system uses monitoring features that report the current network status, which is considered by the routing algorithms. Although the basic principles in the related work with soft RT focus are partly similar to those of our proposed system and the concept of [12], the given analysis is typically not sufficient to provide any guarantees with respect to worst-case latency or jitter. Instead, these approaches consider the adaption of the network configuration to dynamically changing traffic patterns, which typically occur in IT and data center networks.

VI. Conclusion

We designed a new, SDN-based concept for RT communication systems. The concept exploits the capabilities of SDN to collect topology information and application requirements, which are then used by a newly developed routing and scheduling algorithm to autonomously determine an appropriate network configuration. This configuration is installed in the network by means of the northbound and southbound interfaces of the SDN concept. To ensure RT communication, the network uses the well-known TDMA approach, which has been extended by the exploitation of the network topology to allow simultaneous data transmissions (SDMA). By this SDMA-extension, the concept offers an increased potential to scale up the network size without decreasing the RT capabilities. Moreover, our concept takes application-specific requirements into account. We successfully implemented our proposed concept as a prototype based on standard hardware components. The latencies achieved with this prototype are below 1 ms for small networks. Even in very large networks with long multi-hop routes latencies far below 10 ms can be expected. These promising results confirm that systems based on the proposed concept are at least equal to existing IE solutions with respect to satisfiable cycle time and latency requirements. However, our concept outpaces them due to its superior scalability, the possible use of standard hardware, and the absence of an SPoF.

Additionally, we identified precise and scalable device synchronization strategies as well as the development of further routing and scheduling algorithms as interesting topics for our future research. In the future, we will also work on the important aspect of dynamically changing networks, i.e., how new devices and flows or changing flow requirements can be integrated into the network at runtime without disturbing RT communication of other flows.

References

[1] 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 *IEEE 19th International Conference on Emerging Technology and Factory Automation (ETFA)*, Sep. 2014, pp. 1–8.

[2] P. C. Evans and M. Annunziata, "Industrial internet: Pushing the boundaries of minds and machines," General Electric, Tech. Rep., Nov. 2012. [Online]. Available: http://www.ge.com/docs/chapters/Industrial_Internet.pdf

[3] Industrie 4.0 Working Group, "Recommendations for implementing the strategic initiative industrie 4.0," Communication Promoters Group of the Industry-Science Research Alliance and acatech - National Academy of Science and Engineering, Tech. Rep., Apr. 2013. [Online]. Available: http://forschungsunion.de/pdf/industrie_4_0_final_report.pdf

[4] "Software-defined networking (sdn) definition," Open Networking Foundation, Dec. 2015. [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition

[5] "Software-defined networking: The new norm for networks," White Paper, Open Networking Foundation, Apr. 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[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, Apr. 2008.

[7] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[8] Pox on github. [Online]. Available: https://github.com/noxrepo/pox

[9] lwip - a lightweight tcp/ip stack. [Online]. Available: http://savannah.nongnu.org/projects/lwip/

[10] D. Cronberger, "The software defined industrial network," *industrial ethernet book*, vol. 84, pp. 8–13, Oct. 2014.

[11] G. Kálmán, D. Orfanus, and R. Hussain, "Overview and future of switching solutions for industrial ethernet," *International Journal on Advances in Networks and Services*, vol. 7, no. 3&4, pp. 206–215, 2014.

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

[13] F. Ongaro, E. Cerqueira, L. Foschini, A. Corradi, and M. Gerla, "Enhancing the quality level support for real-time multimedia applications in software-defined networks," in *International Conference on Computing, Networking and Communications (ICNC)*, Feb. 2015, pp. 505–509.

[14] S. Gorlatch and T. Humernbrum, "Enabling high-level qos metrics for interactive online applications using sdn," in *International Conference on Computing, Networking and Communications (ICNC)*, Feb. 2015, pp. 707–711.

[15] L.-C. Cheng, K. Wang, and Y.-H. Hsu, "Application-aware routing scheme for sdn-based cloud datacenters," in *Seventh International Conference on Ubiquitous and Future Networks (ICUFN)*, Jul. 2015, pp. 820–825.