RT-SDN: Adaptive Routing and Priority Ordering for Software-Defined Real-Time Networking

Sangeun Oh, Jiyeon Lee, Kilho Lee, Insik Shin
Dept. of Computer Science
KAIST, South Korea
insik.shin@cs.kaist.ac.kr

Abstract—**This work constitutes the first attempt towards an adaptive real-time communication system that leverages the Software-Defined Networking (SDN) paradigm to provide end-to-end deadlines guarantees. SDN is an emerging networking paradigm that allows to control the network through directly programmable software controllers, offering the flexibility to manage and optimize network resources dynamically. In this paper, we address the problem of determining routes and flow priorities to support real-time communication on SDN-enabled networks. We present a scalable routing algorithm that adaptively reconfigures the routes of existing flows in order to find bandwidth-guaranteed routes for all the flows. We also introduce an optimal priority assignment scheme, based on the well-known OPA algorithm, with respect to end-to-end delay analysis in multi-hop communication environments. In addition, we employ a feedback loop between routing and priority ordering schemes. We implement a prototype of the proposed system, called RT-SDN, as an SDN controller and deploy it on a testbed network of 30 BeagleBone devices. Our experiments show that RT-SDN can be deployed in real-world commodity networks to provide end-to-end deadline guarantees, while the proposed schemes collectively improve schedulability significantly.**

I. Introduction

The demand for real-time communication is spreading fast not only into many industrial networks (e.g., factory automation and automotive networks), but also into general-purpose commodity networks (e.g., enterprise networks). Supporting real-time communication with end-to-end deadline guarantees requires to address many important problems, such as *route selection* and *traffic scheduling*. For each real-time traffic flow, one must first select a route between its source and destination along which sufficient resources are reserved to meet its time-sensitive requirements. Yet, since flows are subject to contention for the shared network resources along their selected routes (i.e., router/switch queues), it entails ”good” scheduling disciplines (i.e., priority ordering) that guarantee less end-to-end delay bounds than the user-specified delay requirements.

In conventional networks, it is generally difficult to achieve ”good” real-time routing and traffic scheduling that collectively guarantee smaller end-to-end delay bounds, due to many challenges. For examples, 1) it is necessary to obtain an accurate and complete information on the global network state to make better or optimal routing and scheduling decisions. However, it is very expensive for individual routers/switches to maintain up-to-date global network view to make routing decisions in a distributed manner. 2) In addition, an ability to reconfigure resource allocations (i.e., routes, flow priorities) is also necessary to respond better to dynamically changing demands. However, most conventional routers/switches are usually ”closed” systems that provide static, manual configuration methods through limited- and vendor-specific control interfaces, making it quite difficult to reconfigure flexibly. 3) Furthermore, a close interaction between routing and traffic scheduling will provide opportunities for significant performance improvements, since they are interdependent on each other for real-time guarantees. However, a clear separation between IP and MAC layers in the network layered architecture makes it non-trivial to coordinate them tightly.

Software-Defined Networking (SDN) is an emerging network architecture towards a new control paradigm by separating the roles of network control and packet forwarding functions. The control function, formerly tightly bounded in individual network devices, is migrated into external software, becoming directly programmable. This new programmatic way of controlling the forwarding function allows network managers to add new functionalities into their network, while reproducing the behavior of existing protocols. To this end, software-based SDN controller exchange control messages with SDN-enabled switches using a standard protocol, such as OpenFlow [1], to collect global network state information and manage switches how to forward packets. This way, SDN provides the capability to (re)configure and optimize network resources flexibly across protocol layers via the logically centralized SDN controller. The goal of this work is to explore the potential for supporting hard real-time communication while leveraging the benefits of such SDN platforms.

Related work. During the past several decades, substantial research has been conducted to support hard real-time communication in various networks, such as CAN (Control Area Network) [2], on-chip network [3], and Ethernet [4], [5], [6], [7], [8]. In particular, recent studies on Ethernet switches are more closely related to our work, since we consider SDN-enabled Ethernet switches. A group of studies consider TDMA(Time Division Multiple Access)-based master/slave models, called FTT-Ethernet, to schedule the transmissions of Ethernet frames in specific network topologies, such as bus [4], a single switch [5], and a tree topology [6]. RT-NET [7] supports end-to-end delay bounds by maintaining bounded queue lengths within Ethernet switches through traffic admission control. RT-layer [8] is introduced as an intermediate layer between routing and Ethernet MAC (Medium Access Control) layers to carry out admission control and packet scheduling. The above approaches for hard real-time support share in common the assumption that routes are given and fixed and do not consider re-routing issues to improve schedulability. A recent study [9] proposes an SDN-based approach to reconfigure routes for multimedia traffic, but no end-to-end delay guarantee is supported.

This work. We aim to develop an adaptive real-time communication system, RT-SDN, that addresses routing and traffic scheduling to provide end-to-end deadline guarantees on software-define networking (SDN) platforms. Towards this goal, in this paper, we first develop a scalable routing scheme that adaptively reconfigures existing routes to find new routes with bandwidth guarantees from a global point of view. In

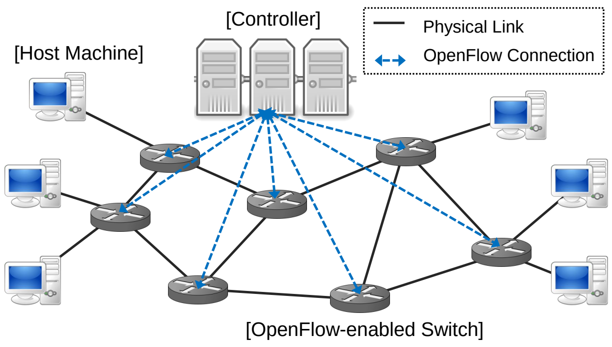


Fig. 1: Software-Defined Networking (SDN) environment

SDN, its underlying Ethernet switching largely eliminates medium contention concerns. However, the flows sharing the same switch are yet subject to contention if they are heading for the same output port within the switch. Hence, we present a priority assignment scheme for fixed-priority switch scheduling to arbitrate such a contention. A key feature of our priority ordering scheme is that it enables to use the well-known OPA (Optimal Priority Assignment) algorithm [10] with a provable performance bound in multi-hop communication environments, where traditional end-to-end response time analysis (i.e., holistic analysis [11]) is not compatible with OPA while capturing precedence relationship with release jitters. Furthermore, we explore the benefit of coordinating route selecting and priority ordering based on a feedback loop. When the latter fails to find a proper priority to make a flow deemed schedulable, it gives to the former some feedback information to help to find a better route for schedulability improvement. Our simulation results indicate that adaptive route reconfiguration and priority ordering schemes improve performance individually and that the feedback-based coordination can improve schedulability even further.

We implement a prototype of RT-SDN as an SDN controller and deploy it on an SDN network testbed, where 30 small commodity computing devices (BeagleBone) play the roles of OpenFlow-enabled switches (15 devices) and end-hosts (15 devices), respectively. Our experiments with the prototype show that our proposed adaptive routing and priority ordering are able to provide end-to-end deadline guarantees while improving schedulability significantly.

II. System Environments

A. Software-Defined Networking Environments

In this paper, we consider a software-defined networking (SDN) platform, which consists of a (logically) centralized controller and a set of SDN-enabled Ethernet switches (See Figure 1). The controller communicates with switches using a well-defined standard protocol (i.e., OpenFlow) to instruct the switches to take specified actions for specified flows in the network. The controller is aware of the global network topology and thereby able to make intelligent decisions on which route and in which priority each packet should be forwarded and scheduled. The controller installs such forwarding and scheduling rules in each individual switch using a *flow table* with *match* entry and *action* field.

The switches become simple devices that forward and drop packets according to the rules defined by the controller. All incoming packets are matched against the match entry in the

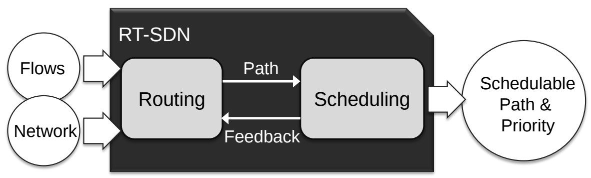


Fig. 2: Overview of the proposed system, RT-SDN

flow table, and corresponding actions in the action field are performed on the matched packets. For example, the action might be to forward a packet to a specified output port or to modify the header of the packet for some purposes. In addition, the action can be specified further to indicate into which priority queue of the output port the packet should be placed. If no match is found, the packet is forwarded to the controller, and the controller is responsible for determining how to handle packets.

B. Notations and Terms

A network topoloy is characterized by a directed graph $G\langle V, E\rangle$, where V is a set of nodes and E is a set of directed links. Each link $l(a,b) \in E$ points from one node $a \in V$ to another $b \in V$, and it is associated with a capacity constraint $c(a,b)$. For simplicity of notations, let us denote a link by l_i when it is not important to indicate which two nodes the link l_i connects. Likewise, c_i then represents the capacity of l_i.

We consider a set F of sporadic real-time flows. Each flow $f_k \in F$ is specified as $f_k(s_k, t_k, T_k, M_k, D_k)$ such that a sequence of messages are delivered from a source node s_k to a destination node t_k, T_k is a minimum separation time between two consecutive messages, M_k is the maximum message size, and D_k is a relative end-to-end deadline ($D_k \leq T_k$). In addition, we introduce a few more notations. The route R_k is a sequence of links that connects s_k to t_k. The maximum bandwidth requirement b_k is defined as $b_k = M_k/T_k$. In this paper, we consider fixed-priority scheduling for each output port within a switch, and P_k denotes the priority of flow f_k.

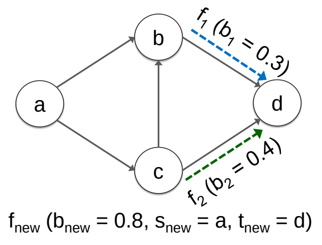
III. System Overview

The goal of this paper is to support real-time communication on software-defined networking platforms, focusing on addressing routing and priority ordering.

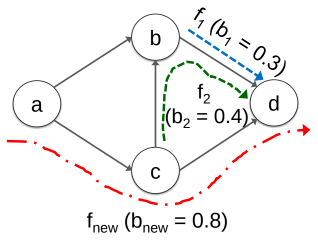
Problem statement. Given a network $G\langle V, E\rangle$ and a set F of real-time flows, we consider the problem of determining the route R_k and the priority P_k of each flow f_k such that all the messages of each flow f_k are delivered from its source s_k to destination t_k with an end-to-end delay guarantee of less than or equal to a deadline D_k.

The above problem involves two sub-problems of routing (to determine R_k) and priority ordering (to assign P_k), while those two are typically dependent on each other when end-to-end deadline guarantees are required. The whole problem is known to be NP-hard [12], and a vast majority of literature shares the principle of addressing sub-problems one by one to simplify complex relationships, focusing on one sub-problem while assuming the other.

One possible direction is to select routes with delay guarantees under the assumption that underlying traffic scheduling disciplines are determined [13]. This approach basically considers the problem of admitting a new flow f_{new} by finding



(a) Failure with fragmentation



(b) Success

Fig. 3: Routing example

its route with a deadline guarantee, while not violating the deadline guarantees provided to any existing flows. To this end, this approach entails to compute the maximum delay bound that any link incurs to f_{new}. Given that f_{new} can impose interference on other flows traveling on the same links, this requires to assign to f_{new} a proper priority that does not invalidate the other flows' delay guarantees. However, this is quite complicated since the amount of interference that f_{new} can impose on other flows depends not only on their priority relationship, but also on the route of f_{new}. One naive approach is to assign to f_{new} the lowest possible priority, but this is significantly disadvantageous to admitting new flows.

On the other hand, a number of studies [2], [4], [5], [6], [7], [8] consider another approach, addressing real-time traffic scheduling to provide end-to-end delay guarantees while assuming routes are given. We basically follow this direction to address routing and traffic scheduling one by one with an additional cross-layer coordination between them. As shown in Figure 2, our proposed system, RT-SDN, is designed to first select (delay-oblivious) routes with bandwidth guarantees[1] and then assign flow priorities based on the selected routes aiming to meet end-to-end deadlines. If the system becomes schedulable, the RT-SDN controller configures switches according to the routes and priorities determined. If not, it employs a feedback loop between routing and priority ordering to improve schedulability. Delay information is given as a feedback such that delay-aware decisions can be made to select better routes.

IV. QoS Routing with Bandwidth Guarantees

The goal of QoS routing is to find feasible, possibly optimal routes that satisfy QoS constraints. In this section, we consider QoS routing with bandwidth constraint. Specifically, we assume that a network $G\langle V, E\rangle$ and a set F of flows are given in a way that all flows $f_k \in F$ are assigned their own routes and their bandwidth requirements are met along the routes. We then consider the issue of admitting a new flow f_{new} with bandwidth guarantees.

A. *Existing Approaches*

We first discuss two representative existing approaches for static and dynamic (reconfigurable) routing to the above issue: Constraint-Based Routing (CBR) and Multi-Commodity Flow (MCF).

Constraint-Based Routing (CBR) refers to a set of routing algorithms that find an optimal route subject to some constraints. For example, one can formulate the problem of admitting a new flow f_{new} subject to bandwidth constraints as

$$\begin{array}{ll} \text{minimize} & |R_{new}| \\ \text{subject to} & \forall l(a,b) \in R_{new}, \sum_{\forall k: l(a,b)\in R_k} b_k \le c(a,b) \end{array}$$

One can then find an optimal route R_{new} with shortest path by running Dijkstra shortest path algorithm, after pruning the links with insufficient residual bandwidth [14], which runs in complexity $O(|E| + |V| \ln |V|)$.

A key characteristics of CBR is a static nature. It does not consider changing the routes of existing flows. While this static routing is fast, it can lead to poor performance due to fragmentation, as illustrated in Figure 3(a). In the example, each link has a capacity of 1.0, and two flows f_1 and f_2 use the links $l(b, d)$ and $l(c, d)$ with the bandwidth requirements of $b_1 = 0.3$ and $b_2 = 0.4$, respectively. Suppose a new flow f_{new} requests a route from a node a to a node d that satisfies the bandwidth requirement of $b_{new} = 0.8$. In this example, CBR fails to find such a route for f_{new} since both $l(b, d)$ and $l(c, d)$ have less available bandwidth than b_{new} each, even though their total aggregate bandwidth available is greater than b_{new}.

In this paper, such links, $l(b, d)$ and $l(c, d)$, are said to be *critical links* to finding a route for f_{new}. More precisely, a link $l_{i,j}$ is critical to finding a route of f_k if (i) the link has a less available bandwidth than b_k and (ii) pruning all the critical links $l_{i,j}$ partitions the entire network into two or more subnetworks such that s_k and t_k are placed in different partitioned subnetworks. Thus, by definition, no route for f_k can be constructed if a critical link exists.

Multi-Commodity Flow (MCF) refers to a set of network flow optimization problems. For example, Algorithm 1 shows a Mixed-Integer Linear-Programming (MILP) optimization problem that finds the routes of all flows in a way that a total bandwidth consumption is minimized subject to the bandwidth requirements of all the flows. A solution to this MILP problem can be naturally used for addressing the above issue of admitting a flow f_{new} with a bandwidth guarantee. The solution to the MILP problem basically gives a potentially new route to every single existing flow while removing all critical links, if any, in an optimal way. This way, the solution serves as the point of maximal performance benefit of dynamic routing reconfiguration. However, the search space of the MILP problem is huge, $O(2^{|E|+|F|})$, and it thereby quickly becomes intractable when the network size and/or the number of flows grows larger.

B. *Our Approach - Cluster-based Adaptive Routing*

The previous subsection describes the benefits and costs of static vs. dynamic QoS routing, which can be represented as a tradeoff between performance vs. complexity. Here, we present an approach, called *Cluster-based Adaptive Routing (CAR)*, that balances the tradeoff efficiently in a scalable manner.

The proposed CAR scheme can be outlined as follows. We consider a network $G\langle V, E\rangle$ and a set F of flows such that each flow $f_k \in F$ is assigned a bandwidth-guaranteed route R_k. When accepting a new flow f_{new} with a bandwidth guarantee, we potentially face a critical link problem for f_{new}. Our approach is to transform $G\langle V, E\rangle$ into a virtual network $G^*\langle V^*, E^*\rangle$ such that if there exists a feasible route R^*_{new} for f_{new} in G^*, then there must exist a set of routes R_k for each $f_k \in F$ as well as R_{new} with all bandwidth guarantees in the given network G, while each R_k can change.

[1]In this paper, we assume that switch queues are long enough to avoid any packet drop if the total bandwidth requirements on a link is no greater than its capacity, and it is out of the scope of the paper to address packet loss issues.

Algorithm 1 Multi-Commodity Flow (MCF)

```
input : G(V,E), F={$f_k \mid f_k = (s_k,\, t_k,\, b_k)$ }, $c_l$: capacity of link l,
        E={$E_p | p \in E_p, E_p = I_p \cup O_p$}, $I_p$: ingress link, $O_p$: egress link
output: $\forall f_k \in F, \forall l \in E$,
```

$$X_k^l = \begin{cases} 1, & \text{if flow } f_k \text{ flows into link } l \\ 0, & \text{otherwise} \end{cases}$$

$$\begin{aligned}
\underset{X}{\text{minimize}} \quad & \sum_{l \in E} \sum_{f_k \in F} X_k^l b_k \\
\text{subject to} \quad & \forall l \in E, \sum_{f_k \in F} X_k^l b_k \le c_l \\
& \forall f_k \in F, \sum_{l \in O_{s_k}} X_k^l b_k = \sum_{l \in I_{t_k}} X_k^l b_k = b_k \\
& \forall f_k \in F, \forall p \in V, p \neq s_k, t_k, \sum_{l \in I_p} X_k^l = \sum_{l \in O_p} X_k^l \\
& \forall f_k \in F, \forall p \in V, \sum_{l \in E_p} X_k^l \le 2 \\
& \forall f_k \in F, \sum_{l \in E_{s_k}} X_k^l = 1 \text{ and } \sum_{l \in E_{t_k}} X_k^l = 1
\end{aligned}$$

For ease of presentation, let us introduce some terms and notations. In order to transform $G\langle V, E\rangle$ to $G^*\langle V^*, E^*\rangle$, we first partition $G\langle V, E\rangle$ into a set of disjoint clusters $G_i\langle V_i, E_i\rangle$. For each node $v \in V$, v belongs to exactly one cluster G_i ($v \in V_i$). For each link $l(a,b) \in E$, $l(a,b)$ is said to be either an *internal* link of a cluster G_i if both two nodes a and b belong to the same cluster G_i ($l(a,b) \in E_i$), or an *external* link connecting two clusters otherwise. For each node $v \in V_i$, v is either a *border* node if it is associated with one or more external link, or an internal node otherwise. For each node $v \in V$, v is said to be a *virtual* node if it is a source s_{new} of a destination t_{new} of flow f_{new}, or a border node. For each pair of two virtual nodes (a,b) that belong to the same clusters ($a, b \in V_i$), we construct two directional virtual link $l^*(a,b)$ and $l^*(b,a)$ between a and b. Then, we construct V^* as a set of all virtual nodes and E^* as a set of all virtual and external links.

We then explain how to partition G into a set of clusters G_i and define virtual links.

Cluster construction. One can view that the MILP formulation in Algorithm 1 yields very high complexity while adding all the links $l(a,b) \in E$ into search space to address potential critical link problems. In order to reduce search space efficiently, we consider critical link candidates: a link $l(a,b) \in E$ is a *critical link candidate* for a flow f_k if the available bandwidth of the link is smaller than b_k. Our approach is designed to investigate a much smaller search space with some pre-specified bound (N_x) to resolve critical links. Thereby, it constructs clusters with the notion of the critical link candidate of a new flow f_{new}, as follows:

Step 1: We first transform $G\langle V, E\rangle$ into $G'\langle V', E'\rangle$ such that $V = V'$ and E' contains only critical link candidates of f_{new}. This transformation is done by pruning all the non-critical link candidates, which can be simply done in $O(|E|)$. We then construct individual clusters G_i iteratively through Steps 2 and 3.

Step 2: We go to Step 3 (initializing i if not initialized) if V' is empty, or finish otherwise.

Step 3: For any arbitrary-chosen node $v' \in V'$, we find a set V_i of up to N_x nodes connected in G' such that $v' \in V_i$. This can be easily done by breath-first-search. Then, for each node $v' \in V_i$, we remove $v' \in V_i$ from V'. We increase i by one and go to Step 2.

A key characteristic of the above cluster construction is that since we build clusters based on critical link candidates, it is more likely to place critical link candidates within clusters. This way, it is more likely to address critical link problems when we solve the MILP problem in Algorithm 1 for each cluster.

Dynamic routing within clusters. Upon the construction of individual clusters, we aim to resolve any potential critical links through dynamic routing reconfiguration based on the MILP optimization. To this end, for each cluster G_i, we define a set F_i of flows that belong to the cluster as follows. For each flow $f_k \in F$ that uses any link $l(a,b) \in E_i$ and thereby belongs to G_i, we synthesize a new flow f'_k such that it has a source s'_k and a destination t'_k within the cluster G_i ($s'_k, t'_k \in V_i$) as follows: s'_k is defined as s_k if $s_k \in V_i$ or an incoming border node $v' \in V_i$, and similarly, t'_k is defined as t_k if $t_k \in V_i$ or an outgoing border node $v" \in V_i$. Here, it is not yet determined whether f_{new} will belong to this cluster G_i, unless its source and destination are located within G_i. Hence, we consider all the scenarios in which f_{new} belongs to G_i, exploring all the combinations of which incoming and outgoing border nodes will be used by f_{new}. Thus, let V_i^* denote a set of virtual nodes that belong to G_i, that is, $V_i^* \equiv V^* \cap V_i$. Then, we run the MILP optimization for each possible scenario of pairing a source $s'_{new} \in V_i^*$ and a destination $t'_{new} \in V_i^*$ for f'_{new}. When we find a solution to the MILP optimization, it means that it is guaranteed to construct all the routes with bandwidth guarantees when f_{new} travels from s'_{new} to t'_{new} within the cluster. We then add a virtual link $l^*(s'_{new}, t'_{new})$ into the virtual network $G^*\langle V^*, E^*\rangle$ with its capacity set to ∞.

A key feature of the above cluster-based optimization is that it not only reduces the problem space for the MILP optimization, but also allows to parallelize the optimization for each cluster and even for each scenario of pairing s'_{new} and t'_{new}.

Route construction. For each link $l^*(a,b)$ in the virtual network $G^*\langle V^*, E^*\rangle$, it is either an external link with a larger bandwidth available than b_{new} or a virtual link that is able to resolve any underlying critical link problems to support f_{new}. Then, we first run CBR to find a route R_{new} for f_{new} in G^* and then do some processing for each virtual link $l^*(a,b) \in R_{new}$ as follows. For each virtual link $l^*(a,b) \in R_{new}$, there must exist a cluster G_i such that, by definition of virtual link, $a, b \in V_i$ and must also exist a solution to the MILP optimization for G_i in the scenario of $a = s'_{new}$ and $b = t'_{new}$. According to the solution, we reconfigure the routes of existing flows and replace the virtual link $l^*(a,b)$ with other internal links properly.

It is worthy noting that this approach cannot find an optimal solution all the time. It just finds sub-optimal solutions when critical links are placed across clusters.

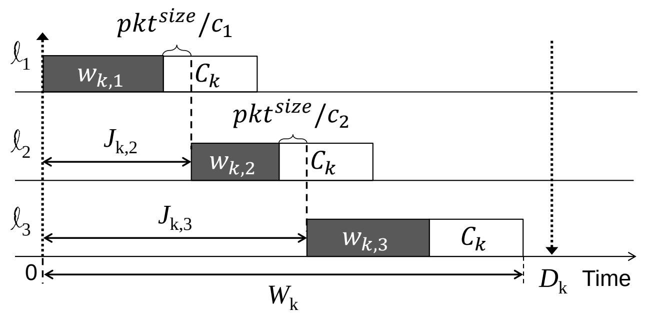


Fig. 4: Holistic communication analysis

V. Real-Time Traffic Scheduling

In this section, we consider real-time traffic scheduling with a focus on priority ordering. Here, we assume that a network topology $G\langle V, E\rangle$ and a set F of real-time flows are given such that each flow $f_k \in F$ is assigned its own route R_k. We then address the problem of determining the priority P_k of each flow f_k such that the end-to-end deadline D_k of each flow f_k is met. We first discuss an end-to-end delay analysis and then propose a new priority assignment scheme.

A. *End-to-end Delay Analysis*

Holistic analysis, first proposed by Tindell and Clark [11] and later improved in [15], [16], [17], [18], presents the worst-case end-to-end response time analysis for distributed task models, where a task $\tau_i(T_i, D_i, \{C_{i,j}\})$ is sequence of subtasks, $\tau_{i,j}, \cdots, \tau_{i,N_i}$, with precedence constraints. A subtask $\tau_{i,j+1}$ can start executing with the worst-case execution time $C_{i,j}$, possibly on a different node, only after the preceding subtask $\tau_{i,j}$ has completed execution. Since it is hard to consider precedence constraints accurately in schedulability analysis (it is generally known as NP-hard [3]), release jitters are introduced to enforce the precedence constraints. The jitter $J_{i,j}$ of a subtask $\tau_{i,j}$ represents the maximum release time deviation from its activation time to the completion time of the preceding subtask $\tau_{i,j-1}$ (see Figure 4), i.e.,

$$J_{i,j} = \sum_{k=1}^{j-1} w_{i,k}. \tag{1}$$

Holistic analysis shows that a bound W_i on the end-to-end response time of τ_i can be derived by computing the maximum delay $w_{i,j}$ that each subtask $\tau_{i,j}$ experiences. Here, $w_{i,a}$ is computed through the following fixed-point iteration:

$$w_{i,j}^{n+1} = C_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{J_{k,j} + w_{i,j}^n}{T_k} \right\rceil C_k, \tag{2}$$

where $hp(i)$ denotes the set of higher-priority tasks of τ_i. The iterative computation of $w_{i,j}$ starts with $w_{i,j}^0 = 0$ and continues until it converges, i.e., $w_{i,j}^{n+1} = w_{i,j}^n$, or reaches some predefined value greater than D_i. Then, the worst-case end-to-end response time W_i of task τ_i is $W_i = \sum_j w_{i,j}$, and it is schedulable if $W_i \le D_i$.

Holistic communication analysis. In addition to distributed computing environments, the holistic analysis has been widely used and extended in many other network environments, such as Control Area Network (CAN) [2] and on-chip networks [3], for the end-to-end delay analysis of real-time flows. While sharing many key attributes such as precedence constraints and interference, one significant difference between real-time distributed computing model and packet-switched networking model is the unit of parallel execution/communication. In traditional distribution computing models, subtasks are under strict precedence constraints such that a subtask must start execution only after the completion of its preceding subtask. In packet-switched networking model, a message is unit of communication from the flow's point of view, but a smaller unit, a packet, is the actual unit of communication from the network's point of view. This allows a packet of a message to be transmitted on a link, while another packet of the same message is being transmitted on a different link, leading to hiding the transmission times of packets over multiple links for a single message. Thereby, as discussed in the literature [19], [3], the maximum transmission time of a message M_k can be given by

$$C_k = \max_{l_a \in R_k} C_{k,a} + T_P \cdot |R_k|, \text{ where } C_{k,a} = (M_k + \mathsf{pkt}_k^{\mathsf{hd}})/c_a, \tag{3}$$

while $\mathsf{pkt}_k^{\mathsf{hd}}$ is the total packet header size of M_k (such as UDP/IP and ethernet header) and T_P indicates the maximum constant processing delay in a node.

We can then extend the holistic analysis, originally developed for distributed computing environments, to packet-switched networking environments, as follows. Now, $J_{k,a}$ is defined as $w_{k,a-1}$, where l_{a-1} indicates the previous link of l_a in R_k. Then, the maximum queuing delay $w_{k,a}$ that f_k experiences on a link l_a is calculated with the following fixed-point iteration:

$$w_{k,a}^{n+1} = B_{k,a} + \sum_{\forall i \in hp(k,a)} \left\lceil \frac{J_{i,a} + w_{k,a}^n}{T_i} \right\rceil C_{i,a}, \tag{4}$$

where $B_{k,a}$ is the maximum blocking time from lower-priority flows due to the non-preemptable nature of transmission. That is, $B_{k,a} = \mathsf{pkt}^{\mathsf{size}}/c_a$, where $\mathsf{pkt}^{\mathsf{size}}$ is a single packet size. The iteration starts with $w_{k,a}^0 = C_{k,a}$ and terminates in the same way as in Eq. (2). The jitter $J_{i,a}$ is defined as

$$J_{i,a} = \sum_{\forall l_b \in R_i^a} (w_{i,b} + \frac{\mathsf{pkt}^{\mathsf{size}}}{c_b}), \tag{5}$$

where R_i^a represents a subset of R_i that includes only the previous links of l_a in the route R_i. Then, the worst-case end-to-end response time W_k of flow f_k is derived as

$$W_k = \sum_{\forall l_a \in R_k} (w_{k,a} + \frac{\mathsf{pkt}^{\mathsf{size}}}{c_a}) + C_k. \tag{6}$$

B. *Priority Assignment*

We now consider priority ordering schemes for real-time flows. Davis and Burns [20] showed that Audsley's Optimal Priority Assignment (OPA) algorithm [10] provides an optimal fixed-priority assignment with respect to any so-called *OPA-compatible* schedulability test that satisfies the following three conditions:

1) The schedulability of a flow may, according to the test, depend on the set of higher- and equal-priority flows, but not on their relative order.

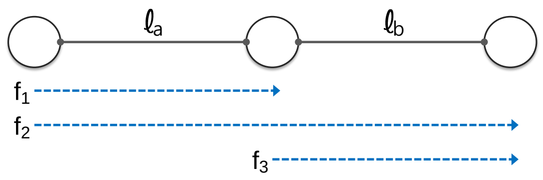


Fig. 5: OPA example

2) The schedulability of a flow may, according to the test, depend on the set of lower-priority flows, but not on their relative order.
3) A flow being assigned a higher priority cannot become unschedulable according to the test, if it was previously schedulable at a lower priority.

Unfortunately, the holistic (communication) analysis with the release jitter $J_{k,a}$ defined in Eq. (5) is not OPA-compatible (and neither is the one with Eq. (1)). This is because the first condition above may not hold, as illustrated with the example shown in Figure 5. In the example, three flows f_1, f_2, and f_3, are given with their routes such that $W_1 = \langle l_a \rangle, W_2 = \langle l_a, l_b \rangle$, and $W_3 = \langle l_b \rangle$, respectively. Suppose f_1 and f_2 are higher-priority flows of f_3. Then, we wish to show that the response time W_3 of f_3 can vary depending on the priority order between f_1 and f_2. The maximum queuing delay $w_{3,b}$ of a flow f_3 depends on the jitter $J_{2,b}$ of a higher-priority flow f_2, and $J_{2,b}$ is determined largely by the maximum queuing delay $w_{2,a}$ that f_2 experiences on its previous link l_a. Then, the relative priority order between f_1 and f_2 directly affects $w_{2,a}$ and, subsequently, $J_{2,b}$ and $w_{3,b}$, as well. This way, W_3 can increase when P_1 is higher priority than P_2, compared to the other case where P_1 is smaller priority than P_2.

Local OPA with intermediate deadlines. Given that it is infeasible to find an optimal priority assignment by applying OPA with respect to the holistic (communication) analysis, one possible approach is to decompose an end-to-end deadline into intermediate deadlines of individual links and apply OPA to find per-link priority of each flow according to the intermediate deadlines, as proposed by a recent study [21]. However, this is only applicable to distributed computing environments, but not to communication environments, particularly, with the transmission time model given in Eq. (3). This is because intermediate deadlines enforce a clear separation such that one can start transmitting a message on one link only after the message was sent completely on a previous link prior to an intermediate deadline.

Thus, we seek to apply OPA for priority assignment without decomposing an end-to-end deadline into intermediate deadlines. The source of OPA incompatibility of the holistic analysis is the release jitter $J_{k,a}$ that depends on the priority order of higher-priority flows. Hence, we introduce an upper-bound on the jitter (denoted as $\bar{J}_{k,a}$) such that $\bar{J}_{k,a}$ is insensitive to the priority order. For instance, in the previous example above, we can use an upper bound $\bar{J}_{2,b}$, rather than an exact value of jitter $J_{2,b}$, in computing $w_{3,b}$ and thereby W_3. This way, we can get the worst-case end-to-end response time that is invariant to the priority order of f_1 and f_2. Motivated by this, we define a jitter bound $\bar{J}_{k,a}$ as $\bar{J}_{k,a} \triangleq D_k - C_k$. If there exists a flow f_k such that $w_{k,a-1} > D_k - C_k$, then the actual jitter value becomes larger than the bound $\bar{J}_{k,a}$, making it invalid to analyze the schedulability other flows f_i based on the bound. However, in this case, W_k becomes greater than D_k, and the flow f_k is thereby determined as unschedulable. This way, without regard to the schedulability of f_i, the entire system is determined unschedulable as well.

Algorithm 2 Optimal Priority Assignment (OPA) Algorithm

```
1: for each priority level $i$, lowest first do
2:     for each unassigned flow $f_k$ do
3:         if $f_k$ is schedulable in priority level $i$ according to the
           schedulability test $S$ assuming all unassigned flows to
           have a higher priority then
4:             assign $f_k$ to priority $i$
5:             break the loop
6:         end if
7:     end for
8:     return unschedulable
9: end for
10: return schedulable
```

We can extend the holistic communication analysis with $\bar{J}_{i,a}$ plugged in the computation of $w_{k,a}$ in Eq. (4), rather than $J_{i,a}$. Such an extension to the holistic communication analysis is more pessimistic, but is now OPA-compatible.

Lemma 1: The OPA algorithm (described in Algorithm 2) provides an optimal priority assignment with respect to the holistic (communication) analysis with a jitter bound $\bar{J}_{k,a}$.

Proof: Examination of Eqs. (4) and (6) with the definition of $\bar{J}_{k,a}$ shows that the three OPA-compatible conditions hold. Neither the length of the queuing delay, nor the length of the jitter bound depends on the specific priority order of higher- and equal-priority flows. In addition, when the priority of a flow increases, it can get a larger blocking term but will have a decrease in interference that is at least as large, leading to a non-increasing response time only. ■

Capacity augmentation bound. With the pessimism introduced to the jitter bound $\bar{J}_{k,a}$, the holistic communication analysis with $\bar{J}_{k,a}$ becomes more pessimistic and so does OPA with respect to the analysis. Here, we derive a *capacity augmentation bound* (α) of the holistic communication analysis with $\bar{J}_{k,a}$ (denoted as HCA*) with respect to the same analysis but with a different jitter $J_{k,a}$ (denoted as HCA) in a sense that if an optimal priority assignment (for instance, through exhaustive search) is found with respect to HCA on a network with capacity c, then OPA always finds a schedulable priority assignment with respect to HCA* on a network with capacity $c \cdot \alpha$.

Theorem 1: The resource augmentation bound of HCA* with respect to HCA is 2.

Proof: Due to space limit, we present a proof sketch. For any schedulable priority ordering, we have $0 \le J_{k,a} \le \bar{J}_{k,a} \le W_k \le T_k$. We wish to show that any priority ordering schedulable with HCA is also schedulable with HCA* on a network of a capacity α times larger, where a message transmission time becomes α times shorter. Then, we wish to show $\bar{w}_{k,a} \le w_{k,a}$, where $w_{k,a}$ and $\bar{w}_{k,a}$ are calculated with $J_{k,a}$ and $\bar{J}_{k,a}$ in Eq. (2), respectively, as follows.

$$
\begin{aligned}
w_{k,a}^{(n+1)} &= B_{k,l} + \sum_{\forall i \in hp(k,a)} \left\lceil \frac{J_{i,a} + w_{k,a}^{(n+1)}}{T_i} \right\rceil C_i \\
&\ge B_{k,l} + \sum_{\forall i \in hp(k,a)} \left\lceil \frac{w_{k,a}^{(n+1)}}{T_i} \right\rceil C_i \ (\because 0 \le J_{i,a}) \quad (7)
\end{aligned}
$$

$$\bar{w}_{k,a}^{(n+1)} = \frac{B_{k,l}}{\alpha} + \sum_{\forall i \in hp(k,a)} \left\lceil \frac{\bar{J}_{i,a} + \bar{w}_{k,a}^{(n+1)}}{T_i} \right\rceil \frac{C_i}{\alpha}$$
$$\leq \frac{B_{k,l}}{\alpha} + \sum_{\forall i \in hp(k,a)} \left(1 + \left\lceil \frac{\bar{w}_{k,a}^{(n+1)}}{T_i} \right\rceil\right) \frac{C_i}{\alpha} \quad (\because \bar{J}_{i,a} \leq T_i) \tag{8}$$

Iteration starts with $w_{k,a}^{(0)} = C_k$ and $\bar{w}_{k,a}^{(0)} = C_k/\alpha$. A full proof can be then constructed via induction that the RHS of Eq. (8) is always no greater than the RHS of Eq. (7). ■

C. Feedback-based Cross-Layer Coordination

So far, we considered the problem of assigning flow priorities assuming that routes are given and fixed. However, when the OPA algorithm fails to assign a priority to a flow that guarantees all end-to-end deadline satisfaction, one reasonable approach is to reconfigure the routes of some flows to increase schedulability. Thus, we consider cross-layer coordination between routing and scheduling layers to proceed another round of route and priority assignment. To this end, we present an approach based on feedback loop such that the routing scheme finds a route according to some feedback information obtained during the process of priority assignment.

A link $BL_k \in R_k$ is said to be a *bottleneck* link of a flow f_k if it incurs the largest queuing delay among all the links in the route R_k, i.e.,

$$BL_k = \arg\max_{l \in R_k} w_{k,l}. \tag{9}$$

When the OPA algorithm fails to assign a priority for a flow f_k, we pass the bottleneck link of f_k to the routing scheme. Then, the routing scheme runs CBR to select a new bandwidth-guaranteed route, after pruning the bottleneck link. Upon finding a new route successfully, the route is passed to the scheduling component to find a priority that provides end-to-end delay guarantees. Such a feedback loop iterates until all the flows are schedulable or no new route is found. It is easy to see that this performs a finite number of iteration steps, since it prunes some link at each iteration step.

VI. Simulation Results

Simulation Environment. For simulation, we constructed a network topology $G\langle V, E\rangle$ with two parameters: the number of nodes (N_V) and the link probability p. For each pair of two nodes in V, a link is randomly constructed with the link probability p. We set either 25, 50, or 75 to N_V and either 0.2, 0.4, or 0.6 to p. We assume that the capacity of all links is $100Mbps$. For each flow f_k, its source s_k and destination t_k are randomly chosen in V as long as $s_k \neq t_k$. Its message size M_i is uniformly distributed from $10Kbits$ to $25Mbits$ and a period T_i from $10ms$ to $1s$. We assume that each deadline D_i is equal to T_i.

In order to evaluate the performance of routing schemes, we define a parameter, called *load*, as the ratio of a total link consumption to a total link capacity in the network, that is,

$$load = \frac{\sum_{\forall f_k \in F} |R_k| \cdot M_k/T_k}{\sum_{\forall l(a,b) \in E} c(a,b)}. \tag{10}$$

Simulation Results. Figure 6 compares three routing schemes: MILP, CAR, and CBR, in terms of the ratio of accepting new flows, where CAR is our proposed routing algorithm. For some given value L', where $0.1 \leq L' \leq 0.9$, we generate a set of flows until its load reaches L' under the CBR routing scheme. At this point, we generate 100 different flows as a new flow f_{new} and run different routing schemes to see whether f_{new} can be accepted with a bandwidth-guarantee route. The figure shows that MILP always finds a route for f_{new} during the simulation. On the other hand, CBR and CAR become incapable of accepting every new flow when the load increases. In particular, the figure shows a performance gap between CBR and CAR when the load is medium. This indicates that when the load is between 0.3 and 0.7, critical links exist and make CBR fail to accept a large fraction of new flows, while CAR is able to resolve those critical links via adaptive route reconfiguration. On the other hand, when the load is high (i.e., greater than 0.7), there are more critical links that make both CBR and CAR fail to accept many new flows. In this case, some critical links exist across clusters, and CAR is not able to resolve them.

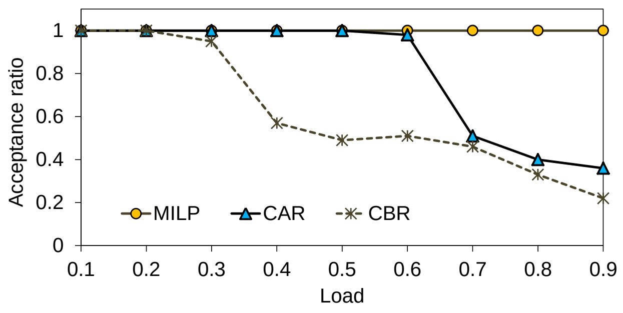


Fig. 6: Routing

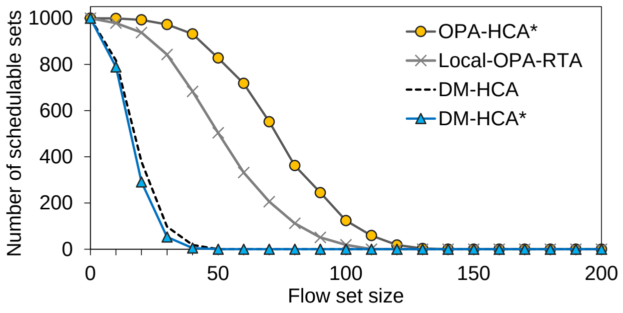


Fig. 7: Priority assignment

Figure 7 compares the number of flow sets deemed schedulable by different priority assignment methods: DM-HCA, DM-HCA*, OPA-HCA*, and Local-OPA-RTA [21]. We first discuss the first three methods. For priority assignment, DM is the well-known Deadline-Monotoic algorithm [22] as a representative static method, and OPA-HCA* is our proposed scheme, where HCA and HCA* indicate the holistic communication analysis with the conventional jitter ($J_{k,a}$) and our proposed jitter bound ($\bar{J}_{i,a}$), respectively. Due to the pessimism involved in $\bar{J}_{i,a}$, HCA* is expected to yield a more pessimistic schedulability analysis than HCA does. The figure shows that some noticeable difference is made between the cases in which schedulability analysis is performed for DM scheduling with respect to HCA and $\bar{J}_{i,a}$. However, such a difference is relatively very small compared to the gap between DM-HCA* and OPA-HCA*. This shows the benefit of using OPA-compatible HCA* in priority assignment for real-time flows.

For an additional comparison, we add another method, Local-OPA-RTA [21], that assigns an intermediate (local) deadline to each link $l_a \in R_k$ for flow f_k and applies OPA

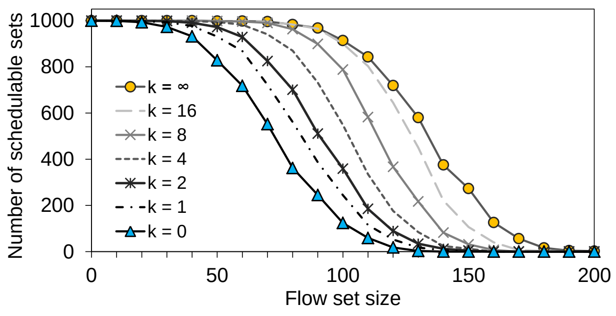


Fig. 8: Feedback

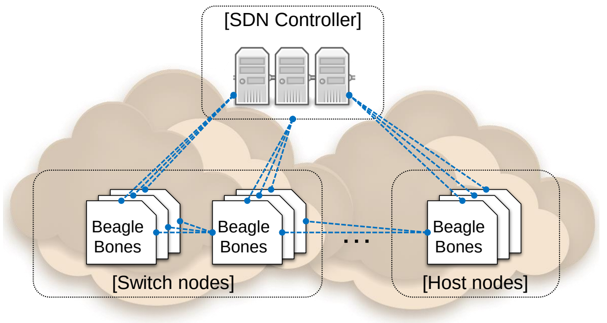


Fig. 9: Testbed overview

Fig. 10: Testbed with 30 BeagleBone devices

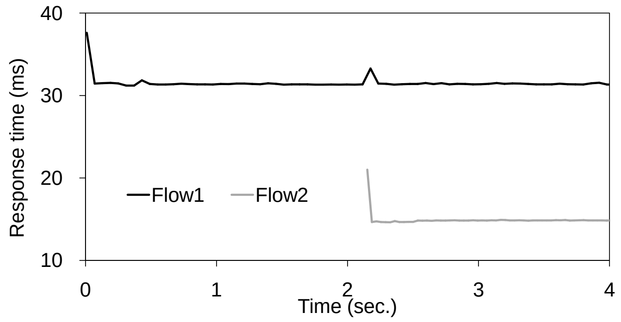


Fig. 11: Route reconfiguration scenario

to determine a per-link priority $P_{k,a}$ for each link l_a. As discussed in Section V, such intermediate deadlines enforce precedence constraints clearly between messages in a way that a message becomes the unit of transmission, which makes it infeasible to use holistic communication analysis. Thereby, it uses the uniprocessor response-time analysis (RTA) [23] without jitters. The figure shows that Local-OPA-RTA performs better than DM-HCA and DM-HCA*, since it assigns priorities responding to the characteristics of the current flow set and network state information. On the other hand, its performance is shown to fall behind OPA-HCA*'s. This is because the analysis based on intermediate deadlines is more pessimistic than the holistic communication analysis. That is, with the intermediate deadlines, the end-to-end response time is derived to include the transmission time of a message as many times as the number of hops (i.e., $|R_k|$ for flow f_k). On the other hand, the holistic communication analysis adds the transmission time of a message only once to the end-to-end response time.

Figure 8 evaluates the performance of coordinating routing and priority ordering through a feedback loop. The figure plots the number of flows deemed schedulable by OPA-HCA* while going through the feedback loop up to k times. This way, when $k = 0$, it is exactly OPA-HCA* with no feedback loop involved. It shows that schedulability can be significantly improved through even a small number of feedback loop iterations. For example, when the flow set size is around 100, a huge gain was obtained by iterating the feedback loop only a few times. This implies the importance and effectiveness of tight coordination between routing and flow scheduling when supporting real-time communication.

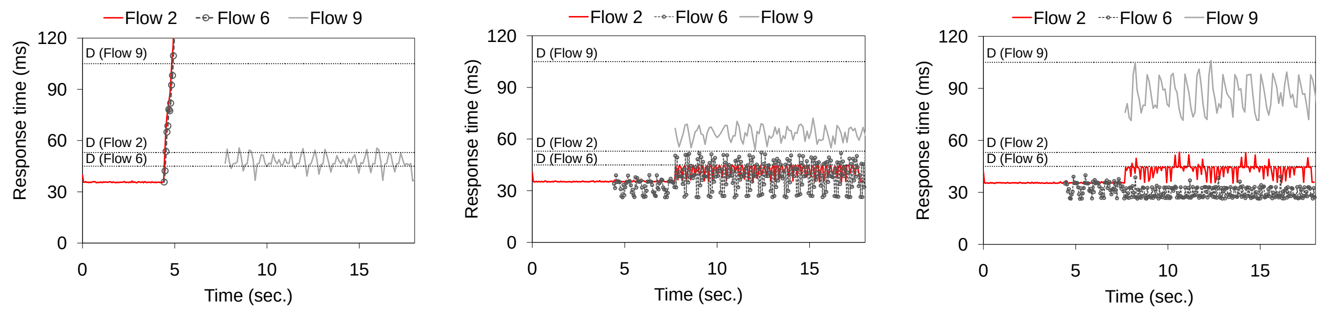
VII. Experiment Results

In this section, we evaluate our RT-SDN controller prototype using an OpenFlow-enabled testbed network. Our experience with the prototype shows that adaptive routing and priority ordering are necessary to improve schedulability.

A. Testbed Platform

Testbed setup. Figure 9 depicts the overview of our testbed network. It consists of 30 Linux-based BeagleBone Black development boards (ARM Cortex-A8 1GHz, 512MB RAM) [24] with 15 working as switch nodes and the other 15 as host nodes. A switch node is operated as a OpenFlow software switch by the use of OpenVSwitch 1.9.3 [25]. Due to the lack of network interfaces, we additionally mounted USB hubs (Belkin F4U040kr [26]) and USB Ethernet cards (realtek r8152 [27]) on the BeagleBone Black boards. As an SDN controller, Beacon [28], one of the most popular SDN controllers, is chosen to run on a desktop machine (core i5-4670, 32GB RAM). Our system, RT-SDN, is implemented as a module in the Beacon controller. The photograph in Figure 10 shows our testbed network.

Implementation issues. We resolved several implementation issues to build the SDN-enabled testbed. In order to avoid network traffic congestion due to packet flooding, we disabled MDNS (Multicast Domain Name System) [29] and used static ARP (Address Resolution Protocol) [30] entries. Time is synchronized among nodes by the use of NTP (Network Time Protocol) [31], which enables an accurate one-way trip time measurement for each flow. Most of the time during the experiment, the synchronization error was less than 1ms. In addition, we implemented a priority queue for each link by using linux TC (Traffic Control) tool with HTB (Hierarchical Token Bucket) Queue [32]. Since most OpenFlow-enabled switches offer 8 priority queue for each output port, we consider the number of priority levels is limited to only 8. Thus, we slightly modify the OPA algorithm to reflect this limited number of priority levels, by removing Line 5 in Algorithm 2. Its optimality still holds when its underlying schedulability analysis considers equal-priority flows as higher-priority ones.



(a) OSPF (b) CAR (c) RT-SDN

Fig. 13: Case study: The response times of three representative flows

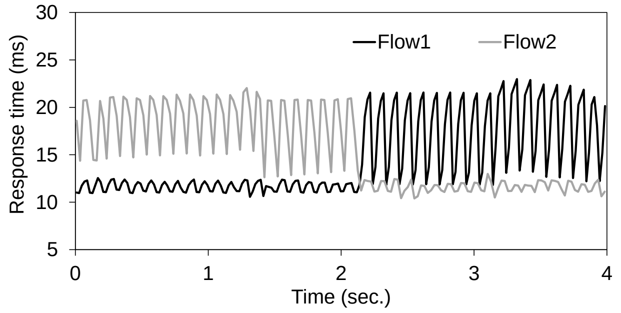


Fig. 12: Priority reconfiguration scenario

Flow #	Src.	Dst.	Bandwidth Req.(Mbps).	T(ms)	C(ms)	D(ms)
0	H7	H3	1.9	46	9	27
1	H8	H4	2.0	50	10	36
2	H6	H1	5.9	57	34	53
3	H9	H10	6.0	60	36	47
4	H0	H7	2.0	31	6	26
5	H11	H8	2.9	59	17	37
6	H0	H2	5.0	50	25	45
7	H11	H5	4.8	39	19	38
8	H7	H4	3.1	39	12	36
9	H8	H3	3.2	108	35	105

TABLE I: Case study: Flow configurations.

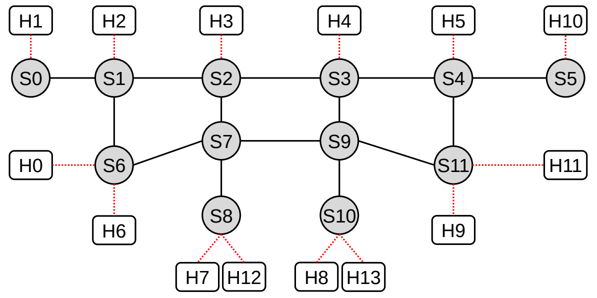


Fig. 14: Case study: network topology configuration.

B. OpenFlow Primitives for Routing and Priority Ordering

Route selection. One of the key features of the proposed system is adaptive route reconfiguration that leverages the up-to-date global network state maintained in the SDN controller. The controller exchanges OpenFlow control messages with switches to install a new packet forwarding rule when a dynamic route configuration is needed (e.g., when the route of a new flow cannot be found). More specifically, the controller sends a `flow mod` control message with an output port specified in the `action` field. Figure 11 shows a scenario where the route of a flow f_1 is reconfigured when another flow f_2 joins. In this case, when f_1 joins at time 0, our controller accepts this flow with a new route. With some delay, as shown in the figure, f_1 is able to reach a destination. When f_2 joins at time 2, the controller decides to reconfigure the route of f_1. The figure also shows the overhead of this adaptive route reconfiguration is smaller than that of a new route establishment. During our experiments, no packet loss was observed from route reconfiguration. One reason is that the controller has a dedicated link connection to each individual switch, leading to a very low probability of control message loss. Another reason is that the controller sends the control messages for route reconfiguration to the switches in the increasing order of their distance to the destination, leading to a smaller probability of packet loss due to inconsistent forwarding rules across switches.

Priority ordering. Another key feature of the proposed system is adaptive priority ordering for switch scheduling that exploits the global network state information. The SDN controller configures the flow tables of switches for flow priorities by sending a `flow mod` control message with a priority queue information specified in the `action` field. Figure 12 shows a scenario where two flows reconfigure priorities dynamically, while sharing the same route. In this scenario, two flows f_1 and f_2 join at time 0 while f_1 is a higher-priority flow of f_2. The figure shows that f_1 has a smaller response time than f_2 does. At time 2, the SDN controller instructs switches to change the priorities of f_1 and f_2 such that f_2 becomes higher-priority. Then, the figure shows that this priority reconfiguration takes place smoothly. After the reconfiguration, the response time of f_2 becomes smaller than that of f_1 without much fluctuation.

C. Case Study

We performed a case study on our testbed network to investigate the behavior of real-time flows depending on different routing and priority ordering schemes. To this end, we examined the response times of 10 synthesized real-time flows on a network topology of 12 switches and 14 host nodes (see Figure 14). Table I shows the timing and source/destination requirements of the 10 flows. In order to observe the effect of adding individual flows to the system, we added 10 flows one by one with the time delay of 1 second. Our experiments were performed under three different schemes, OSPF, CAR, and RT-SDN, where OSPF [33] is one of the most widely used routing algorithms without real-time support.

Figure 13 shows the response times of three representative flows, f_2, f_6, and f_9, that experienced most contention in the experiments. Those flows had the bandwidth requirements of 5.9, 5.0, and 3.2 Mbps, respectively, while each link has the capacity of 10Mbps. Figure 13(a) shows the baseline case where OSPF routing was employed. Since OPSF makes a routing decision from an individual flow perspective (i.e., the shortest-path-first principle), it resulted in a case where f_2 and f_6 shared the same links exceeding their link capacity. This led to substantial increases in response times for f_2 and f_6 beyond their deadlines, while f_9 was not interfered at all. In total, five out of ten flows missed deadlines.

Figure 13(b) indicates how our routing scheme, CAR, is beneficial to improving response times. Since CAR aimed to find bandwidth-guaranteed routes, it allowed f_2 and f_6 to use different links in order not to collide on bandwidth-limited links, reducing their response times substantially. However, f_6 and f_9 happened to share some links while interfering with each other and thereby increasing the response times of both flows, leading to some deadline misses of f_6. As such, 4 out of 10 flows missed deadlines.

Figure 13(c) shows how RT-SDN can eventually improve schedulability via its priority ordering scheme (i.e, OPA with the feedback loop). Here, f_6 and f_9 shared the same link too, but f_9 was assigned a lower priority such that f_9 no longer interfered f_6. This way, f_6 was able to meet deadlines. Likewise, all flows met deadlines in this case.

VIII. Conclusion

In this paper, to the best of our knowledge, we made the first attempt to design and implement an SDN-based communication system, called RT-SDN, that supports end-to-end hard real-time guarantees by addressing routing and priority ordering together. Towards this, we developed an adaptive, scalable scheme for QoS routing with bandwidth guarantees, an OPA-based priority ordering scheme with a more pessimistic but OPA-compatible holistic communication analysis, and a feedback-based close coordination between the two schemes. Our prototype implementation deployed in an SDN-enabled 30-node testbed network showed that the proposed schemes support end-to-end deadline guarantees with significant schedulability improvement, as is indicated by our simulation results as well.

However, this work here is a starting point with many open issues remaining to be explored. One issue is that the entire SDN network is susceptible to a single point of failure if the centralized controller fails or misbehaves, as well as scalability issues in large-scale networks. A possible approach is to extend the system architecture with distributed controllers to eliminate/alleviate performance and reliability bottlenecks. In this direction, our future work includes developing new methods that can conduct schedulability analysis and priority ordering efficiently in a distributed and/or hierarchical manner.

References

[1] 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.

[2] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems Journal*, pp. 239–272, April 2007.

[3] Z. Shi and A. Burns, "Schedulability analysis and task mapping for real-time on-chip communication," *Real-Time Systems*, vol. 46, no. 3, pp. 360–385, 2010.

[4] P. Pedreiras, P. Gai, L. Almeida, and G. Buttazzo, "Ftt-ethernet: a flexible real-time communication protocol that supports dynamic qos management on ethernet-based systems," *Industrial Informatics, IEEE Transactions on*, vol. 1, pp. 162–172, 2005.

[5] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over cots ethernet switches," in *In WFCS06: IEEE International Workshop on Factory Communication Systems*, 2006, pp. 295–302.

[6] M. Behnam, Z. Iqbal, P. Silva, R. Marau, L. Almeida, and P. Portugal, "Engineering and analyzing multi-switch networks with single point of control," in *Proceedings of the 1st International Workshop on Worst-Case Traversal Time*, 2011, pp. 11–18.

[7] J. Loeser and H. Haertig, "Low-latency hard real-time communication over switched ethernet," in *ECRTS*, 2004, pp. 13–22.

[8] H. Hoang, M. Jonsson, U. Hagström, and A. Kallerdahl, "Switched real-time ethernet and earliest deadline first scheduling - protocols and traffic handling," in *IPDPS*, 2002.

[9] H. Egilmez, S. Dane, K. Bagci, and A. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *APSIPA ASC*, 2012.

[10] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Department of Computer Science, University of York, Tech. Rep. YCS164, 1991.

[11] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 40, no. 2-3, pp. 117–134, April 1994.

[12] A. Burns, "Scheduling hard real-time systems: a review," *Software Engineering Journal*, vol. 6, no. 3, pp. 116–128, May 1991.

[13] A. Orda, "Routing with end to end qos guarantees in broadband networks," in *INFOCOM*, 1998.

[14] R. A. Guerin, A. Orda, and D. Williams, "Qos routing mechanisms and OSPF extensions," in *GlobeCom*, 1997.

[15] M. Spuri, "Holistic analysis for deadline scheduled real-time distributed systems," INRIA, Tech. Rep. RR-2873, 1996.

[16] J. Palencia and M. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *RTSS*, 1998.

[17] J. Palencia and H. Harbour, "Exploiting precedence relations in the schedulability analysis of distributed real-time systems," in *RTSS*, 1999.

[18] R. Pellizzoni and G. Lipari, "Improved schedulability analysis of real-time transactions with earliest deadline scheduling," in *RTAS*, 2005.

[19] A. Kumar, D. Manjunath, and J. Kuri, *Communication Networking: An Analytical Approach*. Morgan Kaufmann Publishers Inc., 2004.

[20] R. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *RTSS*, 2009, pp. 398–409.

[21] R. Garibay-Martnez, G. Nelissen, L. L. Ferreira, and L. M. Pinho, "Task partitioning and priority assignment for hard real-time distributed systems," in *REACTION*, 2013.

[22] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: the deadline-monotonic approach," in *Proceedings of the IEEE Workshop on Real-Time Operating Systems and Software*, 1991.

[23] ——, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[24] "Beaglebone black." [Online]. Available: http://beagleboard.org

[25] "Open vswitch." [Online]. Available: http://openvswitch.org/

[26] "Belkin f4u040v." [Online]. Available: http://www.belkin.com/us/p/P-F4U040/

[27] "Realtek r8152b(n) chipset." [Online]. Available: http://www.realtek.com.tw

[28] E. David, "The Beacon OpenFlow Controller," in *ACM SIGCOMM HotSDN*, 2013.

[29] S. Sheshire and M.Krochmal, "Multicast dns," in *RFC 6762*, 2013.

[30] D. C. Plummer, "An ethernet address resolution protocol," in *RFC 826*, 1982.

[31] J. B. W. D. Mills, J. Martin, "Network time protocol version 4: Protocol and algorithms specification," in *RFC 5905*, 2010.

[32] "Hierarchical token bucket." [Online]. Available: http://www.tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/

[33] J. Moy, *OSPF, Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1998.