TSN-Builder: Enabling Rapid Customization of Resource-Efficient Switches for Time-Sensitive Networking

Jinli Yan, Wei Quan, Xiangrui Yang, Wenwen Fu, Yue Jiang, Hui Yang and Zhigang Sun*
College of Computer, National University of Defense Technology
Email: {yanjinli10, w.quan, yangxiangrui11, fuwenwen16, jiangyue17, yanghui, sunzhigang}@nudt.edu.cn

***Abstract*—Time-Sensitive Networking (TSN) emerges as a promising technique empowering deterministic forwarding on standard Ethernet without sacrificing compatibility. There are some commercial off-the-shelf (COTS) switches that support TSN recently. However, the resource partitioning on these switches is normally inefficient for the on-chip memory resource in many specific application scenarios. We observe that the critical requirements (*e.g.*, topology, flow features) of these scenarios are pre-determined. Thus, developing a TSN switch in a Top-down approach is feasible and urgently needed.**

In this paper, we propose TSN-Builder, a template-based developing model for customizing resource-efficient TSN switches rapidly with targeted application-dependent requirements. TSN-Builder decomposes the integrated TSN switching function into multiple function templates. With a fine-grained resource abstraction, TSN-Builder provides platform-independent customization interfaces for developers to customize the resource parameters. We prototype TSN switches on FPGA to evaluate the resource consumption and performance under different application scenarios. Experimental results show that TSN-Builder reduces the on-chip memory by up to 80.53% under the same Quality-of-Service, compared to the resource configuration in the COTS switch.

I. Introduction

Ethernet has a great potential to replace field buses for the ever-increasing bandwidth and high compatibility in distributed hard real-time domains, *e.g.*, industrial control, automotive and aerospace [22]. However, the standard Ethernet switches provide best-effort forwarding services where the packet loss and queuing are unpredictable [19]. Most existing congestion control and flow scheduling works [17] [13] focus on alleviating these problems from the spatial perspective, but they cannot guarantee the determinism of the service.

To empower standard Ethernet with deterministic capability, Time-Sensitive Networking (TSN) is proposed as a new paradigm that introduces new time-based features on Ethernet devices. Currently, TSN Task Group [8] has published comprehensive standards and drafts on time synchronization, flow control, flow management, and flow integrity [2] [7] [6] [5] [4]. Some network equipment providers, *e.g.*, Broadcom [1], Marvell [9], have released a series of TSN switching chips supporting typical TSN standards.

The commercial off-the-shelf (COTS) TSN switches are developed in a Bottom-up method without considering specific application requirements. Since the resource partitioning for tables, queues, and buffers in such switches is fixed, the traffic planning algorithms are constrained by the underlying resources. In many cases, the resource partitioning does not adapt to the specific application features very well. The on-chip memory resource is often under low utilization. Besides, if the end-to-end deterministic communication of critical flows cannot be guaranteed, an upgrade of switches would be mandatory, which brings about extra costs. Thus, customizing a resource-efficient TSN switch in an application-driven mode is urgently needed.

Along this road, TTTech is concentrating on designing an integrated autonomous toolchain for mapping the upper requirements into underlying resources [10]. Recently, they collaborated with Intel and published a white paper about the system-level design of TSN switch based on FPGA [11]. However, until now, these studies only provide preliminary ideas without concrete schemes and products. According to their work, the main challenges for customizing TSN switches include two aspects. (1) Composition of TSN switch. TSN switches introduce time-based features on Ethernet switches, including gPTP (generalized Precision Time Protocol) [2], Time-Aware Shaper (TAS) [5], etc. It is crucial to figure out the core components and their connections. (2) Resource abstraction. The various requirements of application scenarios determine the difference of resource specification in each component. Thus, providing a fine-grained resource abstraction is critical to customize application-specific switches.

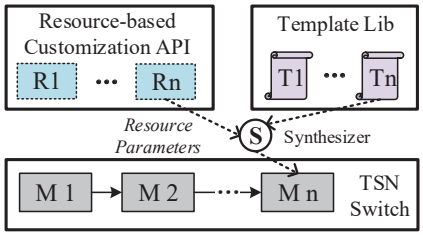


Fig. 1. Overview of TSN-Builder

In this paper, we propose TSN-Builder, a template-based developing model for the rapid customization of resource-efficient TSN switches. The core idea of TSN-Builder is demonstrated in Fig. 1. By referring to typical TSN standards, TSN-Builder decomposes a TSN switch into five core components, which provides a general reference to develop a TSN switch. Further, TSN-Builder decouples the various resource specifications of each module from the fixed processing logic with a comprehensive abstraction of memory-related resources. The fixed processing logic is encapsulated into multiple platform-specific templates. These resource specifications are parameterized into unified resource-based customization APIs for users to inject the optimal configurations without considering platform differences. Therefore, TSN-Builder supports the rapid customization of TSN switches for diverse application scenarios without programming the same functions repeatedly.

We prototype a FPGA-based TSN switch that supports user-defined resource configurations. FPGAs offer flexible re-programmability, ultra-low processing latency, and power consumption. We set up three topologies that are typical in the industrial control network, including ring, linear, star, etc. We select the resource configuration parameters of Broadcom TSN chip BCM53154 as a reference and then compute the customized parameters according to the application features. Compared to the resource parameters of COTS TSN switches, the experimental results show that the on-chip memory resources of customized TSN switches are reduced by 80.53% at most while guaranteeing the same performance of time-sensitive flows in latency, jitter and packet loss.

*Zhigang Sun is the corresponding author.

II. Motivation and Design Goals

A. Background and Motivation

Features of TSN-related application scenarios. The network features of TSN-related scenarios are quite different from the open data center network. The features of data centers are **dynamically changing and complicated**, which makes it difficult to predict the arrival of nondeterministic bursts and congestions.

Differently, the features in TSN-related domains are **predetermined and simple**, which is the basis of customization that we put forward. The topology and characteristics of the critical flows in a specific scenario are known in advance. The typical topologies include star, ring, linear, etc. [21] The flows in a TSN network are divided into three typical types [19]. (1) Time-Sensitive (TS) flows (Highest priority). The TS packets are generated periodically. These critical packets must arrive at the destination within the deadline with ultra-low jitter and packet loss. (2) Rate-Constrained (RC) flows (Medium priority). They require the network to reserve the specified bandwidth. (3) Best-Effort (BE) flows (Lowest priority). The left bandwidth resource is allocated to them.

Relationships between upper requirements and underlying switches. Application scenarios diversify in topology, flow features, the precision of time synchronization, etc. These differences significantly affect the internal design and implementation of the underlying TSN switches. (1) Topology. It determines the number of switches and their connections. Since the number of enabled ports is different, the number of queues, the volume of the packet buffer and the number of per-port tables would be affected. (2) Flow Features. It includes the number of flows and their performance requirements, such as latency, jitter, bandwidth, packet loss, etc. These influence the size of tables, per-queue depth, and packet buffer size [1]. (3) Synchronization precision. It determines the selection of synchronization algorithms and the implementation platforms, which could affect memory and compute resources. To sum up, application requirements affect the selection of function modules and the resource specification of each module. Since the resource configuration varies even there are minor changes in upper applications, we mainly concentrate on the customization of on-chip memory resources in this paper.

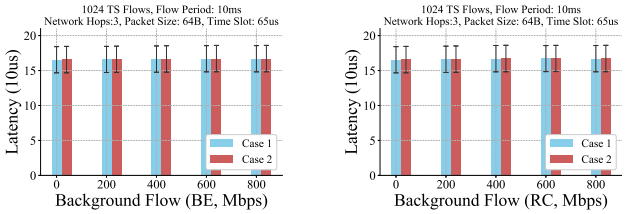
Significance of Customization. The resource partitioning of commercial TSN switches with ASIC chips is fixed according to the coarse-grained evaluation of the applied fields. Generally, in order to guarantee the quality of service (QoS), the resource configurations are much higher than the actual requirements of applications. Thus, precious on-chip memory is under low utilization. In order to reach the performance targets while saving memory resources, an application-driven fine-grained customization model is urgently needed.

To stress it clearly, we take the customization of per-queue depth and packet buffer of a simple TSN network for example. There are three TSN switches based on Xilinx Zynq 7020 FPGA SoC with one enabled port connected with each other. These TSN switches are rapidly built based on the function templates proposed in Section III.C. A network tester is configured to inject 1024 TS flows into this network. The packet size of each flow is 64B, and the period is 10ms. There are two different resource configurations shown in Table I, where the depth of each queue (the width of metadata is 32b here) and the number of packet buffers (the size of each packet buffer is 2048B) are different. Then we evaluate the latency of each TS packet with different bandwidth of RC flows and BE flows as background traffic. The experimental results show that the latency and jitter of TS flows with the highest priority are very stable despite the interference of other flows. Case 2 reduces the resource consumption of queues and buffers by 540Kb BRAM in total. This result proved that the resource parameters in Case 1 are larger than the traffic-dependent threshold and the extra memory resources are free. Therefore, the application-driven customization is feasible when the topology and flow features are pre-determined.

[1]In most switches, queue stores packet descriptor (also named metadata) while buffer stores packet payload.

TABLE I
Configuration of Queue and Packet Buffer

	Queue Num Per-Port	Pkt Num Per-Queue	Packet Buffer Num	Total BRAMs
Case 1	8	16	128	2304Kb
Case 2	8	12	96	1764Kb



(a) (b)

Fig. 2. The latency of TS flows. (a) Different bandwidth of BE flows as background traffic. (2) Different bandwidth of RC flows as background traffic.

B. Design Goals

Providing a general composition of TSN switch. Although the TSN Task group [8] has released detailed standards for empowering standard Ethernet switches with time-based features, integrating these TSN features with Non-TSN features into a TSN switch is still difficult for developers. Thus, it is critical to reasonably decompose TSN switches into multiple modules and analyze their connections.

Providing a fine-grained resource abstraction. The key point for customizing a TSN switch is to determine the memory-related resource specification according to the specific application features. Therefore, a fine-grained resource abstraction is essential for allocating the on-chip memory properly.

Providing platform-independent customization interfaces. TSN-Builder should provide customization interfaces for decoupling the resource configuration from the implementation. The resource specification is determined by mapping the application requirements onto the parameters of each interface, which is independent of the underlying implementation platform.

Supporting rapid customization with high reusability. We observe that the types of function modules are the same while the resource specifications change in most cases. TSN-Builder should extract the processing logic of each component into a template that supports the resource configurations. Thus, the development effort is reduced dramatically by reusing the templates.

III. TSN-Builder Design and Implementation

The composition of a general TSN switch is described firstly, followed by a comprehensive resource abstraction. Finally, we introduce how to customize a TSN switch with TSN-Builder.

A. Composition of TSN Switch

By analyzing all the standards related to the design of the TSN switch, we observe that these standards mainly focus on the description of the partial TSN-related mechanisms. However, it lacks a

general global design of the TSN switch that integrates TSN functions with Non-TSN functions together.

We decompose the whole processing of the TSN switch into five key elements with a loosely-coupled modular design. Our core principle is to decouple the TSN-related logic from the shared logic of standard Ethernet switches and encapsulate the TSN features into independent compositions. Therefore, the general switching functions can be reused without further modifications. According to [20] [5] [7], the core mechanisms of TSN consist of time synchronization and gate control. Time synchronization provides a global uniform time among all TSN switches. The gate control is used to control the enqueue and dequeue time of each packet with two Gate Control lists (GCL) attached to the ingress and egress of each queue, respectively. Each gate control entry defines open/close state for controlling whether the packets in this queue are eligible to be enqueued/dequeued. Therefore, the transmission latency of each packet on each hop is predictable and the end-to-end deterministic communication is guaranteed. The function of each component is described as following.

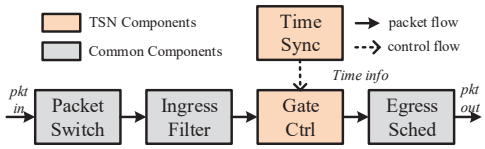


Fig. 3. Composition of TSN switch

Packet Switch. It is used to lookup the outport for each packet with the specified packet fields. **Ingress Filter.** It is used to classify the input packets with the user-defined packet features and map them onto different flows for the following traffic policing. **Gate Ctrl.** It augments traditional queue management with the gate control mechanism. The ingress and egress gates attached to each queue are set as open/close state periodically according to the configured GCLs from a global scheduler. **Egress Sched.** It is used to select the packets in which queue should be sent out. There are shapers limiting the bandwidth of RC queues for alleviating the traffic burst. **Time Sync.** It is used to synchronize the global time that is used to set the current state of each gate by *Gate Ctrl*.

B. Resource Abstraction

With a clear function composition of the TSN switch, the most crucial challenge is to identify the objects that consume the on-chip memory. By referring to 802.1 Q standard [3] [2] and TSN standards [2] [7] [5] [4] [6], we build a resource view of each module, as depicted in Fig. 4, including the key tables, queues and packet buffers.

Table resources. Except for the *Time Sync* component, the other four components have multiple tables, with which the packet flows are processed. In *Packet Switch*, the unicast table is firstly matched with the *Dst Mac* (destination MAC address) and *VID* (VLAN ID) in the packet header for finding the outport. If *Dst MAC* is a multicast address, the multicast index (*MC ID*) is used to find a set of outports from the multicast table.

The classification table in *Ingress Filter* is used to get Meter and Queue ID based on the combination of *Src MAC* (source MAC address), *Dst MAC*, *VID* and *PRI* (priority) carried in the packet header. Then, the *Meter ID* is used to find the corresponding meter that regulates a flow with its current rate. The *Queue ID* indicates which queue the packet would be enqueued.

The input and output gate tables in *Gate Ctrl* are used to set the states of ingress and egress gates and the eligible gate entries are selected based on the current time. The time is divided into multiple time units with the same interval (time slot). In each time slot, the queue stays in an open or a close state. The open state means the packet can be enqueued/dequeued while the close state forbids the enqueued/dequeued operation.

In the *Egress Sched* component, the CBS MAP Table is used to set up the relation between the queues and credit-based shapers (CBS). The *idleslop* and *sendslop* in the CBS Table of each port represent the increase rate and decrease rate of the credits, respectively.

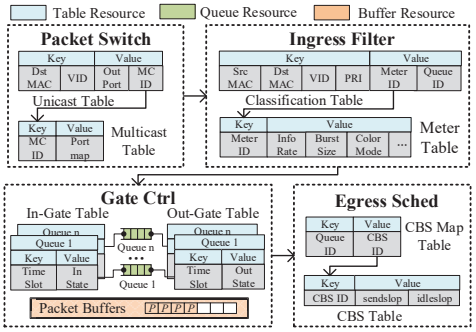


Fig. 4. Resource view of TSN switch

Queue and Buffer Resource. The number of metadata that a queue (queue depth) holds at most determines the volume of packet buffer pool in each port. If the queue depth is not adapted to the number of packet buffers, some packets may be dropped directly or some packet buffers are wasted.

Based on the above resource view, TSN-Builder provides resource-based customization interfaces to parameterize the variables that determine the resource consumption in Table II. With the high-level customization interfaces, TSN-Builder decouples the resource configuration from the fixed switching processing logic. Therefore, developers can design some function templates to encapsulate these processing logics. The implementation of each template depends on the underlying hardware platform and design. Whereas, the customization APIs are platform-independent so that the implementation of templates are transparent to the upper control algorithms.

C. Customizing a TSN switch with TSN-Builder

Since TSN-Builder is proposed to rapidly customize a TSN switch, here we show how to achieve this goal with TSN-Builder.

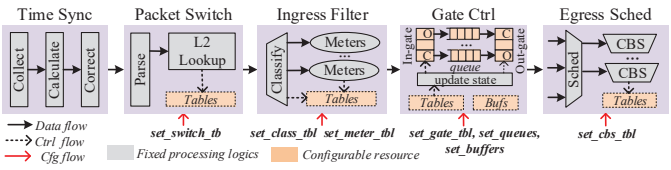


Fig. 5. Design of five templates in TSN-Builder

Firstly, we design and implement five templates as shown in Fig. 5. The gPTP protocol is selected to implement the *Time Sync* template. It includes three submodules: collection of clock time, calculation of correction time and clock correction submodules. The *Packet Sync* template composes of a parser submodule for extracting the packet fields and a lookup submodule to execute the lookup algorithm. The *Ingress Filter* template firstly uses a classifier to differentiate the flows and then puts packets into the specified meters. The *Gate Ctrl* template provides an update module to set the state of In/Out GCL periodically. The *Egress Sched* template consists of a scheduler and CBS submodules. The scheduler selects a packet with a strict priority algorithm. The CBS is implemented based on a token bucket.

[2]It defines the design of standard Ethernet switches.

TABLE II
CUSTOMIZATION APIS OF TSN-BUILDER

Function	Parameters	Description
set_switch_tbl	unicast_size, multicast_size	set the size of unicast table and multicast table
set_class_tbl	class_size	set the size of classification table
set_meter_tbl	meter_size	set the size of meter table
set_gate_tbl	gate_size, queue_num, port_num	set the size of each gate table, the number of queues in each port and the number of ports
set_cbs_tbl	cbs_map_size, cbs_size, port_num	set the size of cbs map table and cbs table, the number of ports
set_queues	queue_depth, queue_num, port_num	set the depth of each queue, the number of queues in each port and the number of ports
set_buffers	buffer_num, port_num	set the number of packet buffers for each port, the number of ports

Secondly, the resource parameters are computed based on the network topology and flow features. Here we make some brief discussions on the guidelines of resource configurations because the configuration algorithm is closely related to the upper flow scheduling algorithm. (1) Switch/Classification/Meter table. These tables are shared by all ports. The number of entries for each table is equal to the number of application flows in the worst case. For optimal configurations, some table entries could be aggregated according to the transmission path, etc. (2) In/Out-Gate tables. These tables are exclusive for each port. The time is divided into multiple equally sized "time slots". The number of entries for each table equals to the number of time slots within a scheduling cycle. The scheduling cycle defines a complete iteration and equals to the least common multiple of all flow periods. (3) CBS Map/CBS table. These tables are exclusive for each port. The number of entries for each table is equal to the number of queues used for RC flows. (4) Queues/Buffers. Each port has a set of exclusive queues and the buffers are shared by all the ports. As for the depth of each TSN queue, the queue should hold all the packets that arrive at the queue in the same slot. Otherwise, some packets would be dropped. The upper flow scheduling algorithms have significant effects on the queue depth. The overall buffer size equals to the queue depth of all queues times per-buffer size. (5) Enabled ports. Some tables above are set for each enabled port. The number of enabled ports for deterministic transmission is closely related to the topologies and transmission direction.

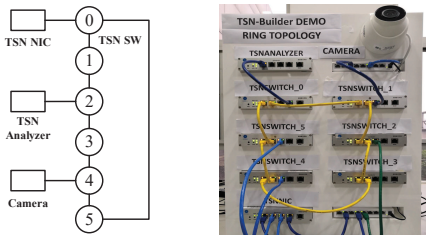
Finally, in the synthesis stage, developers select these templates and inject the application-specific resource configurations by the customization APIs. When the application scenario changes, users only need to regulate the related parameters and reuse these templates without reprogramming in many cases. Thus, the development effort is greatly reduced without reprogramming.

IV. EVALUATION

We evaluate customized TSN switches based on TSN-Builder under different application scenarios. The evaluations focus on two main aspects. (1) Can TSN-Builder reduce overall resource usage? (2) Can TSN-Builder achieve the same QoS as COTS TSN switches?

A. Experiment Setup

We customized a TSN switch prototype on Xilinx Zynq 7020 FPGA SoC with FAST [25] framework. FAST provides a modular programming model and hides the platform-related logic (such as DMA, PCIe, and Linux kernel) for users to rapidly develop their application-related logic on CPU/FPGA heterogeneous platform. All the function templates in Section III.C are programmed with Verilog from scratch based on the interfaces of FAST and run on FPGA. The CPU embedded in Zynq 7020 is used to configure the register and table entries at run-time. The synchronization precision on FPGA is less than 50ns. The FPGA clock frequency in our experiments is 125MHz. We build three network topologies with our TSN switches: star topology where the core node has three child nodes (4 switches), ring topology where each node with one port supporting unidirectional deterministic transmission (6 switches) and linear where each node with two ports supporting bidirectional forwarding (6 switches). The ring topology is described in Fig. 6(a). The maximum of enabled TSN ports in each topology is 3, 2, 1 respectively. To inject user-defined TS/RC/BE flows into a TSN network, a network tester named TSNNic is developed based on Xilinx Zynq 7020 FPGA SoC. The demo of the ring topology is described in Fig. 6(b), where all the devices are connected with Ethernet cables (1Gbps). The TSN analyzer is used to receive the TS/RC/BE flows and analyze the latency, jitter and packet loss.



(a) (b)

Fig. 6. Experiment setup with 6 TSN switch nodes and 3 end-devices in a ring topology. (a) Ring topology. (b) TSN-Builder Demo.

We put a static configuration on the In/Out Gate Control list to implement Cyclic Queuing and Forwarding model (CQF), where two TSN queues perform enqueue and dequeue operations in a cyclic manner. The detailed principles of CQF for guaranteeing the determinism are described in [6]. The reason why CQF provides the deterministic latency relies on two principles. Firstly, the sending time slot and the receiving time slot of a packet on two adjacent switches must be the same. Secondly, a packet received at a time slot must be sent at the next time slot in a switch. The maximum and minimum end-to-end latency bounds in CQF is expressed in Eq. (1).

$$\begin{aligned} L_{max} &= (hop + 1) \times slot_{size} \\ L_{min} &= (hop - 1) \times slot_{size} \end{aligned} \tag{1}$$

where the hop is the number of switches in the path of a TS flow and $slot_{size}$ is the size of time slot.

In order to demonstrate the resource consumptions of TSN-Builder, we select the resource configuration of the Broadcom BCM53154 TSN chip [1] as a baseline. The resource usage and performance are compared between the customized parameters and commercial parameters. The target application scenarios of this chip are the industrial control networks. The features of TS flows that we generate are guided with the IEC 60802 standard [12] that describes the typical flow features in the production cell and line. In our experiments, we generate 1024 periodic TS flows and the period of each TS flow is 10ms. The deadline of each TS flows is randomly selected from the

TABLE III
COMPARISON OF RESOURCE USAGE UNDER DIFFERENT SCENARIOS

Resource Type	Bit/Byte Width	Commercial Switch (4 ports)		Customized Switch (Star, 3 ports)		Customized Switch (Linear, 2 ports)		Customized Switch (Ring, 1 port)	
		Parameters	BRAMs	Parameters	BRAMs	Parameters	BRAMs	Parameters	BRAMs
Switch Tbl	72b	16K, 0	1152Kb	1024, 0	72Kb	1024, 0	72Kb	1024, 0	72Kb
Class. Tbl	117b	1024	126Kb	1024	126Kb	1024	126Kb	1024	126Kb
Meter Tbl	68b	512	36Kb	1024	72Kb	1024	72Kb	1024	72Kb
Gate Tbl	17b	2, 8, 4	144Kb	2, 8, 3	108Kb	2, 8, 2	72Kb	2, 8, 1	36Kb
CBS Tbl	72b	8, 8, 4	144Kb	3, 3, 3	108Kb	3, 3, 2	72Kb	3, 3, 1	36Kb
Queues	32b	16, 8, 4	576Kb	12, 8, 3	432Kb	12, 8, 2	288Kb	12, 8, 1	144Kb
Buffers	2048B	128, 4	8640Kb	96, 3	4860Kb	96, 2	3240Kb	96, 1	1620Kb
Total		/	10818Kb	/	5778Kb(**-46.59%**)	/	3942Kb(**-63.56%**)	/	2106Kb(**-80.53%**)

set {1ms, 2ms, 4ms, 8ms}. The packet size of these TS flows in each test is the same and selected from the set {64B, 128B, 256B, 512B, 1024B, 1500B}. The number of TSN switches that each TS flow traverses is the same in each test and selected from the set {1, 2, 3, 4}. The RC flows and BE flows are also generated with TSNNic. Since the RC/BE flows are background flows here, the packet size of each RC/BE flow is set as 1024B.

B. Resource Usage

The resource parameters of BCM53154 in datasheet includes 4 TSN ports, 16K MAC entries, 1K classification entries, 512 meters, 8 queues/shapers per port and 1MB buffers in total. Since there is only a rough description of these parameters, the other unknown parameters are set the same as the customized parameters. The resource usage of the commercial and customized switches in three scenarios with different topologies is illustrated in Table III. The parameters for each type of resource correspond to the input parameters of the APIs in Table II. The size of the allocated BRAM block is 18Kb or 36Kb and it is determined by the inputted width and depth.

We only create a unicast table in our TSN switch because the multicast flows can be split into multiple unicast flows. Since we generate 1024 TS flows in our testbed, the size of the unicast switch table and classification table is also 1024. The entry width of them is 72b and 117b, respectively. Here we set the size of the meter table as 1024 and the entry width of it is 68b. Since CQF is used as the gate control mechanism, the size of each gate table is only 2 and the width of each entry is 17b. Similarly, the entry width of CBS table and CBS MAP table is 72b in total. Each table size is set as 3 because there are three queues for RC flows in each port. As for the queues, the length of TSN metadata in each queue is 32b, and the queue depth is 8 here with our flow scheduling algorithm [24]. The size of the packet buffer is 2048B for holding the MTU packet.

Compared to the commercial switch in three different scenarios, the reductions of resource in the customized switch are up to 46.59%, 63.56%, and 80.53% respectively.

C. Transmission Performance

We test the transmission performance (latency, jitter and packet loss) under different topologies. The experiment results show that the transmission performance of different topologies is the same. Thus, here we only show the transmission performance in the ring topology in Fig. 7 due to space limitation. Since the priority of TS flows is the highest, and other background flows cannot preempt the bandwidth. The packet loss in all the experiments is 0.

In Fig. 7(a), we regulate the number of TSN switches (hops) that the TS flows traverse. The latency of each TS flow increases with the growth of hops. According to the latency bounds in Eq. (1), the average latency is increased manyfold and the jitter is related to the $slot_{size}$. Here we use the standard deviation of latency to describe the jitter. Since the $slot_{size}$ remains constant (65us), the jitter is nearly unchanged in different hops. The latency increases slightly as the packet size increases in Fig. 7(b). The reason is that the time for outputting the packet is positively correlated with the packet size. The third case shows the change of latency under different $slot_{size}$ in Fig. 7(c). The average latency and jitter are increased manyfold according to the upper and lower bound in Eq. (1). Finally, we inject RC flows and BE flows with different loads simultaneously as background flows. The bandwidth of RC flows and BE flows are the same here. As shown in Fig. 7(d), there is no affection on the latency and jitter of critical TS flows.

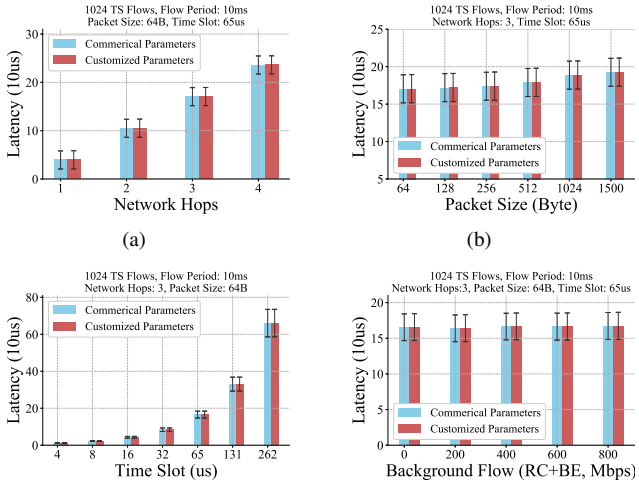


Fig. 7. End-to-End latency. (a) under different hops. (b) under different packet sizes.(c) under different time slots. (d) under different background flows.

In summary, the performance of the TSN switch with customized resource configuration is the same as the one with the commercial parameters. It demonstrates that TSN-Builder could reduce resource usage greatly while guaranteeing the same QoS of TS flows.

V. DISCUSSION

Relationships with existing works. To the best of our knowledge, TSN-Builder is the first research work for customizing the TSN switches based on a comprehensive resource abstraction in a Top-down approach. The relationship between TSN-Builder and the typical existing works, such as OMNeT++ [23], NetFPGA [18], FAST [25] is that TSN-Builder provides a fast TSN-specific development solution over these underlying platform-specific frameworks. To be specific, OMNeT++ is a general software framework for prototyping

network systems. NetFPGA and FAST provide a general programming framework for customizing L2-L7 network functions based on FPGA. However, building a TSN-enabled switch from scratch is tedious and time-consuming without high-level TSN-specific programming abstraction. TSN-Builder proposes a template-based developing model for customizing TSN switches rapidly via platform-independent interfaces. Thus, TSN-Builder could be implemented on OMNetT++/NetFPGA/FAST by developing TSN-related templates. In this paper, we select the FPGA platform for its high performance to meet deterministic requirements with FAST programming framework.

Selection of resource parameters. It is very critical to generate appropriate resource parameters for the practical application of TSN-Builder. We observe that the selection of parameters is an optimization problem and influenced by many factors, including flow features, topologies, lookup algorithms, flow scheduling algorithms, etc. We take the flow scheduling algorithm as an example, it influences the global queue resource occupation from temporal and spatial dimensions so that the queue and buffer size would be affected. In this paper, we focus on providing a novel customization model with comprehensive resource abstraction and propose some preliminary guidelines on how to select resource parameters. Moreover, The major queue/buffer resources are optimized by improving the global resource utilization with our proposed ITP mechanism in [24]. We hope developers design complex algorithms for better optimizations based on our abstractions with domain-specific optimization strategies, which goes beyond the scope of this paper.

VI. Related Work

SwitchBlade [14] provides a pipeline abstraction for the rapid deployment of network protocols. It decouples the common modules from the user-specific modules and developers only need to replace the user-specific modules. Thus, it is essential to provide a TSN-oriented abstraction for customizing a TSN switch. [16] proposes a time-triggered switch-memory-switch architecture (SMS) to build a memory-efficient TSN switch. SMS reduces the packet buffers via sharing them among all ports and provides a scheduler to improve the utilization of memory. Differently, TSN-Builder alleviates the waste of limited on-chip memory resources by customizing the resource-related parameters, including the tables, queues, and buffers. [15] puts forward a research plan of TSN switches based on FPGA. They come up with an idea about designing a parameterized and modular hardware IP core of the multi-stage TSN fabric. However, they did not give a general composition and parameter abstraction for TSN switches. Our work not only focuses on solving these problems, but also provides platform-independent customization APIs.

VII. Conclusion

We propose a template-based developing model named TSN-Builder for customizing a resource-efficient TSN switches. TSN-Builder provides a general decomposition of the integrated TSN switching function. It abstracts a high-level resource view that is closely related to the consumption of the on-chip memory. TSN-Builder decouples various resource configurations from the fixed processing logic and provides platform-independent customization APIs to configure the resource specifications of selected templates. Experimental results demonstrate that the application-oriented resource configurations save the on-chip memory greatly while achieving the same transmission performance as a typical commercial switch.

Acknowledgment

The authors gratefully acknowledge all the TPC members of DAC ESS5 for their constructive comments and suggestions. This work is supported by National Key Research and Development Program of China (Grant No.2018YFB1800505, 2018YFB1800402), National Natural Science Foundation of China (Grant No.61702538, 61802417, 61601483), Training Program for Excellent Young Innovators of Changsha (Grant No.kq1905006) and Research Project of National University of Defense Technology (Grant No.ZK17-03-53, ZK18-03-40).

References

[1] Broadcom 53154 TSN Chip. https://www.broadcom.com/products/ethernet-connectivity/switching/roboswitch/bcm53154.
[2] IEEE 802.1AS Standard. www.ieee802.org/1/pages/802.1as.html.
[3] IEEE 802.1Q Standard. www.ieee802.org/1/pages/802.1Q.html.
[4] IEEE 802.1Qav Standard. www.ieee802.org/1/pages/802.1av.html.
[5] IEEE 802.1Qbv Standard. www.ieee802.org/1/pages/802.1bv.html.
[6] IEEE 802.1Qch Standard. https://1.ieee802.org/tsn/802-1qch/.
[7] IEEE 802.1Qci Standard. https://1.ieee802.org/tsn/802-1qci.
[8] IEEE TSN Task Group. https://1.ieee802.org/tsn/.
[9] Marvell 88E6390X TSN Chip. https://www.marvell.com/switching/assets/LinkStreet_88E6390X_FINAL2.pdf.
[10] Overview of TTE Applications and Development at NASA/JSC. https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20160012363.pdf.
[11] Time-Sensitive Networking: From Theory to Implementation in Industrial Automation. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01279-time-sensitive-networking-from-theory-to-implementation-in-industrial-automation.pdf.
[12] Traffic Types Mapping to TSN Mechanism. http://grouper.ieee.org/groups/802/1/files/public/docs2019/60802-Hotta-Traffic-Types-Mapping-to-TSN-Mechanism-0119-v01.pdf.
[13] Ali Allahverdi. A Survey of Scheduling Problems with No-Wait in Process. *European Journal of Operational Research*, 255(3):665–686, 2016.
[14] Muhammad Bilal Anwer, Murtaza Motiwala, Mukarram bin Tariq, and Nick Feamster. SwitchBlade: a Platform for Rapid Deployment of Network Protocols on Programmable Hardware. *ACM SIGCOMM Computer Communication Review*, 41(4):183–194, 2011.
[15] Adnan Ghaderi, Masoud Daneshtalab, et al. Design Challenges in Hardware Development of Time-Sensitive Networking: A Research Plan. In *2019 Cyber-Physical Systems PhD Workshop*, volume 2457. CEUR-WS, 2019.
[16] Zonghui Li, Hai Wan, et al. Time-Triggered Switch-Memory-Switch Architecture for Time-Sensitive Networking Switches. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
[17] Hela Mliki, Lamia Chaari, and Lotfi Kamoun. A Comprehensive Survey on Carrier Ethernet Congestion Management Mechanism. *Journal of Network and Computer Applications*, 47:107–130, 2015.
[18] Jad Naous, Glen Gibb, Sara Bolouki, and Nick McKeown. NetFPGA: Reusable Router Architecture for Experimental Research. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, pages 1–7. ACM, 2008.
[19] Ahmed Nasrallah, Akhilesh S Thyagaturu, et al. Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research. *IEEE Communications Surveys & Tutorials*, 21(1):88–145, 2018.
[20] Ramon Serna Oliver, Silviu S Craciunas, and Wilfried Steiner. IEEE 802.1 Qbv Gate Control List Synthesis Using Array Theory Encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 13–24. IEEE, 2018.
[21] Csaba Simon, Miklós Máté, et al. Ethernet with Time Sensitive Networking Tools for Industrial Networks. *Infocommunications Journal*, 9(2):6–14, 2017.
[22] Johannes Specht and Soheil Samii. Urgency-based Scheduler for Time-Sensitive Switched Ethernet Networks. In *2016 28th Euromicro Conference on Real-Time Systems*, pages 75–85. IEEE, 2016.
[23] Andras Varga. OMNeT++. In *Modeling and tools for network simulation*, pages 35–59. Springer, 2010.
[24] Jinli Yan, Wei Quan, et al. Injection Time Planning: Making CQF Practical in Time-Sensitive Networking. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020.
[25] Xiangrui Yang, Zhigang Sun, et al. FAST: Enabling Fast Software/Hardware Prototype for Network Experimentation. In *Proceedings of the International Symposium on Quality of Service*, pages 1–10, 2019.