Fenglin-I: An Open-Source Time-Sensitive Networking Chip Enabling Agile Customization

Wenwen Fu, Wei Quan, Jinli Yan, and Zhigang Sun

Abstract—Time-Sensitive Networking (TSN) technology is experiencing diverse application requirements and forming a complicated standard system. It is extremely difficult to design a one-fits-all chip for all TSN applications. Therefore, application-driven TSN chip customization is inevitable. Generally, chip customization starts from a "clean-slate". For complicated ASIC chips, that results in significant development overhead. Inspired by RISC-V chips, an open-source template will significantly reduce the customization complexity. Along this road, we propose an open-source TSN chip named Fenglin-I. Fenglin-I includes a high-level abstraction to build a relationship between application requirements and chip implementation, source code of a real chip named FastTSN to provide reference code for chip implementation, and software tools to facilitate chip verification. Based on Fenglin-I, we further propose a TSN chip customization method that provides step-by-step guidance about customizing TSN chips agilely. To verify the effectiveness of Fenglin-I and the proposed customization method, we use FPGA arrays to prototype and verify FastTSN. The results show that FastTSN achieves microsecond-level transmission jitter for unicast and multicast time-critical traffic. Additionally, we demonstrate two domain-specific TSN chip customization cases in which the customized chips reuse at least 84$\%$ of FastTSN code while meeting their requirements.

Index Terms—Time-sensitive networking, chip customization, open-source chip, fenglin-I, FastTSN

1 Introduction

Time-Sensitive Networking (TSN) augments the native Ethernet with time-related functionalities, such as time synchronization and time-aware shaper (TAS), further to achieve bounded transmission delay and jitter, and inherit the characteristics of native Ethernet (good compatibility and high bandwidth). This expands TSN application scenarios to 5G fronthaul networks, automotive industries and avionics, etc [1], [2]. Nowadays, TSN applications exhibit a large diversity in traffic size, quality of service (QoS) requirements, etc. In order to be widely applied in diversified applications, the TSN task group has proposed 23 standards and is working on 19 ongoing projects, forming a complicated standard system with thousands of pages of content [3].

As Ethernet chips are already very complex, adding more TSN functionalities increases development cost and implementation complexity, and drains more power. It is tough to integrate all TSN function options onto a single Ethernet chip. Currently, commercial off-the-shelf (COTS) TSN chips are designed based on existing Ethernet chips and only support partial TSN standards [4], [5]. Moreover, the COTS TSN chips are developed in a bottom-up mode (just realizing the mainstream functionalities in TSN standards rather than tackling specific application requirements) [6]. However, sometimes it is impossible for users to select a COTS chip that satisfies all their requirements. For instance, limited by system power consumption, some industrial applications require chip power to be less than 1W to substitute the original Ethernet low-power chip [7]. As COTS TSN chips installed many unnecessary functionalities for these applications, there is no COTS chip that satisfies this low-power requirement, as far as we know. Therefore, customizing a TSN chip in an application-driven mode is urgently needed.

For application-driven TSN chip customization, a typical method starts from a "clean-slate" like other chips. To design a TSN chip from a "clean-slate", designers need to first choose appropriate TSN standards (thousands of pages) according to the application requirements and then design a chip architecture considering the TSN functionalities and the general Ethernet functionalities. For the selected functionalities, implementation details, including logic workflow and table configuration of each function module, are determined according to the application requirements. After that, the most time-consuming parts, coding and verification of the chip, need to be done. In the verification process, if the requirements are not satisfied, feedbacks are provided to the previous two steps for a new round. Normally, a TSN chip involves tens to hundreds of thousand lines of hardware and software code [8], [9]. All the steps mentioned above are not trivial. Therefore, customizing a TSN chip from a "clean-slate" is complicated and time-consuming.

According to the consensus of academia and industry, open-source chips share design expertise and source code, facilitating agile chip customization. Along this road, we propose the first open-source TSN chip named Fenglin-I. To serve as a foundation for TSN chip customization, Fenglin-I needs to satisfy common requirements of TSN applications. TSN application requirements comprise functional and performance requirements. The common functional requirements are to support mixed-critical traffic and guarantee high-precision

• *The authors are with the School of Computer, National Universith of Defense Technology, Changsha, Hunan 410073, China. E-mail: {fuwenwen94, w.quan, yanjinli10}@nudt.edu.cn, sunzhigang@263.net.*

Manuscript received 1 February 2022; revised 26 May 2022; accepted 26 June 2022. Date of publication 4 July 2022; date of current version 13 December 2022.
(Corresponding author: Wei Quan.)
Recommended for acceptance by D. Mosse, T. Chantem, E. Bini.
This article has supplementary downloadable material available at https://doi.org/10.1109/TC.2022.3188179, provided by the authors.
Digital Object Identifier no. 10.1109/TC.2022.3188179

deterministic transmission delay and jitter for time-critical traffic (time-critical traffic is the scheduled object in TSN networks, so it is called Scheduled Traffic, ST). The common performance requirements are to reduce chip area and power.

Similar to RISC-V chips [10], Fenglin-I is composed of a high-level abstraction, source code of a real chip following the Fenglin-I high-level abstraction specification, and related software tools. Since the high-level abstraction is described with tables, same as the mainstream description of network device abstraction [11], [12], we name the high-level abstraction as Fenglin-I TTP (table type pattern). It is used to define a TSN chip that satisfies the common application requirements. In detail, Fenglin-I TTP abstracts a TSN chip into multiple function chains and a table map. Each function chain defines the processing path in the TSN chip for a specific traffic class (such as ST traffic). These function chains build a relationship between the common functional requirements and chip function options. Different from a typical Ethernet function chain, Fenglin-I TTP includes a function chain for ST traffic to achieve bounded transmission delay and jitter. This function chain is designed based on a novel deterministic forwarding model that delivers a high-precision determinism service and greatly reduces the traffic planning complexity. Moreover, the table map gathers the table format in all functionalities, building a relationship between the common performance requirements and table implementation details. Based on Fenglin-I, users can re-utilize and extend Fenglin-I TTP for a new application-specific TSN chip, which significantly reduces the complexity of defining a TSN chip.

Currently, we have implemented the first real chip (FastTSN) following the Fenglin-I TTP specification to provide "wheels" for the Fenglin-I based chips. FastTSN code is shown at.[1] Software tools are necessary components for both verifying a customized TSN chip and building a real TSN system. Thus, we provide Fenglin-I software tools at [2] to facilitate chip verification.

Based on Fenglin-I, we propose an agile TSN chip customization method, namely ATC (Agile TSN chip Customization). With this method, for an application-driven TSN chip customiztion, designers only need to customize new function chains by extending or pruning the FastTSN function chains, configure table size for the FastTSN table map, design table format and configure table size for the extended functionalities, and verify the custom chips by using the Fenglin-I software tools. Compared with customizing a TSN chip from a "clean-slate", ATC method significantly decreases the workload of chip architecture design, function and table customization, chip implementation, and the implementation of software tools.

In order to demonstrate the effectiveness of Fenglin-I, we extensively verify FastTSN on FPGA arrays. In a 6-nodes ring network whose transmission rate is 1Gbps (aircraft networks adopt this topology [13]), FastTSN achieves better than 120ns transmission jitter for unicast ST traffic. In a 3-nodes linear network whose transmission rate is 1Gbps, which is adopted in industrial control scenarios [14], FastTSN achieves transmission jitter better than $1\mu s$ for multicast ST traffic. Based on Fenglin-I, we verify the ATC customization method with the following two use cases.

(i) *Use case in an industrial control network.* We customize an ASIC chip named HX-DS09 for an industrial control network that requires less than 1W chip power and $2\mu s$ end-to-end transmission jitter for multicast ST traffic. The HX-DS09 achieves ultra-low power (0.5W) under the 130nm process and $1.2\mu s$ jitter. In this use case, only 15.9% additional hardware code is required to achieve the design goal. (ii) *Use case in an aircraft network.* A third party customizes an FPGA-based chip named TZ-TS01 for an aircraft network that requires the single-hop delay should keep less than $200\mu s$ under 100Mbps physical interface, and end-to-end transmission jitter should be less than $100\mu s$. As FastTSN satisfies these requirements, the designers of TZ-TS01 only configure table size in FastTSN without developing new function modules.

The main contributions of this paper are:

- We propose, as far as we know, the first open-source TSN chip (Fenglin-I) and provide its high-level abstraction, software tools, and source code of a Fenglin-I instance (FastTSN) to facilitate chip customization.
- In Fenglin-I, we implement a novel deterministic forwarding model that delivers high-precision determinism service and reduces traffic planning complexity.
- Based on Fenglin-I, we propose the ATC method to provide step-by-step guidance about customizing a TSN chip.
- We prototype and verify FastTSN on FPGA arrays, and the results show that FastTSN satisfies the common requirements of TSN applications.
- Two real TSN chips, HX-DS09 for an industrial control network and TZ-TS01 for an aircraft network, are demonstrated using the ATC method.

2 Motivation and Challenge

2.1 Background and Motivation

As mentioned in the introduction, TSN can be applied in many application domains. However, the requirements of different TSN applications vary widely. For instance, the applications like power grid systems in the automotive industry have stringent determinism requirements, e.g., only a few microseconds [17], while others (such as telesurgery and haptic feedback) have more relaxed determinism requirements up to a few milliseconds [16]. The reliability of a high voltage distribution network is three orders of magnitude higher than that of a medium-voltage distribution network [8]. Moreover, the traffic size of augmented and virtual reality (AR/VR) applications is thousands of times larger than that of industrial control applications [15], [17].

Each application requirement has a direct impact on defining a TSN chip. Here we take determinism, reliability, and traffic size requirements as examples. In order to deliver the high-precision determinism required by power grid systems, TSN chips generally integrate time synchronization functionality [23] and time-aware shaper [21]. As for

1. https://github.com/fast-codesign/OpenTSN2.0/tree/distributed/Hardware/code
2. https://github.com/fast-codesign/OpenTSN2.0/tree/distributed/Software

the loose determinism required by telesurgery and haptic feedback, TSN chips tend to integrate asynchronous traffic shaper (ATS) [20]. Moreover, the per-stream filtering and policing functionalities [19], frame replication and elimination functionalities [18] are urgently needed to improve reliability for high voltage distribution networks. In addition, the traffic size correlates with the values of memory resource parameters, such as table size. In order to save precious memory resources, the traffic size of the target application is an important constraint when allocating memory resource parameters.

In order to satisfy the diverse application requirements, the TSN working group has proposed 23 standards and is working on 19 projects, forming a complex TSN standard system with thousands of pages of content. Designers need to integrate a large number of TSN functionalities and allocate huge memory resources to design a TSN chip that satisfies all application requirements. This is practically not feasible due to implementation complexity, power consumption, and cost limits. *Therefore, the application-driven TSN chip customization will be a better choice for most TSN application scenarios.*

Typical chip customization starts from a "clean-slate". Since a TSN chip generally involves tens to hundreds of thousand lines of hardware code, it will take a massive amount of time and labor to customize a TSN chip using this typical model. Fortunately, the customization method based on an open-source template, which RISC-V chips adopt, has proven to be excellent for reducing the development workload. Consequently, our work is dedicated to contributing an open-source TSN chip that other designers can reuse.

2.2 Challenges

There are lots of challenges for designing and implementing an open-source TSN chip. In this paper, we mainly focus on the most important challenges.

Challenge I: providing a description to build a the relationship between the target application requirements and chip implementation. During the customization process, mapping target application requirements onto an underlying chip becomes unwieldy. Specifically, the requirements of TSN applications can be divided into functional and performance requirements. The functional requirements mainly indicate traffic requirements, such as determinism and reliability requirements. The performance requirements of a TSN application are proposed to adapt to network, device and traffic features, including traffic size, network topology, chip power and area, etc.

To satisfy the functional requirements of the target application, designers need to have a deep understanding of function descriptions in the TSN standards, which is hard and time-consuming since the TSN standards cover thousands of pages. Moreover, satisfying the target performance requirements is also extremely difficult. This requires designers to clearly understand the reasons for every design detail, such as table format, resource settings, implementation trade-offs, etc. In order to simplify the above work, an open-source TSN chip is recommended to provide a description template that builds a relationship between basic TSN application requirements and chip implementation. By doing this, designers can re-utilize and extend it with other application requirements. Specifically, an open-source TSN chip is recommended to

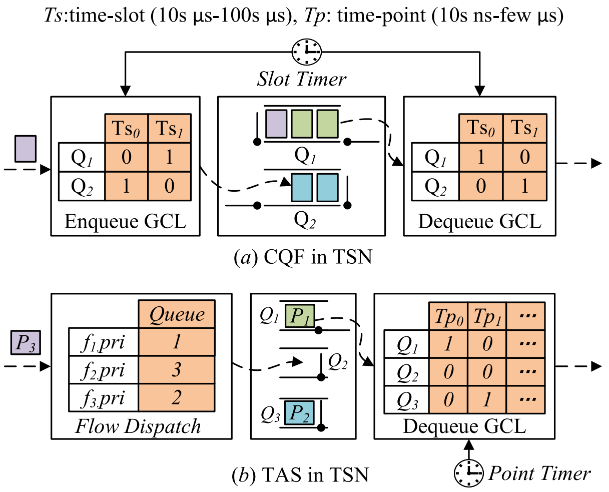


Fig. 1. TAS and CQF control the sending time of packets through GCL (gate control list). GCL records the gate states and duration of each state. When the gate state is open, the packets in the corresponding queue are allowed to be scheduled. Otherwise, the packets are forbidden from being scheduled. The time unit of gate state duration in CQF is *time-slot*, which is much larger than the *time-point* in TAS.

provide an appropriate function description to show the relationship between functional requirements and function options, and a detailed description (such as a table format description) to bridge performance requirements and corresponding design details.

Challenge II: designing a high-precision deterministic forwarding model with low traffic planning complexity. The deterministic forwarding models for ST traffic provided in the current TSN standards mainly include Cyclic Queuing and Forwarding (CQF) and TAS. As illustrated in Fig. 1a, CQF deploys the time-controlled gate for two queues to perform en-queue and de-queue operations cyclically. The time interval between adjacent en-queue and de-queue operations called time-slot is fixed. In the CQF mechanism, an ST packet received at a time-slot must be sent at the next time-slot in a switch. Differently, TAS (as shown in Fig. 1b) uses the flow-dispatch table before the queues to allocate queue-ID. Moreover, TAS adopts a time-controlled gate after each queue, which enables controlling sending time-point for each ST packet by fine-grained gate state configuration.

Since an ST packet may be sent at any time point in the pre-planned time-slot, the deterministic accuracy achieved by CQF is consistent with a time-slot, which is generally configured to hundreds of microseconds or tens of microseconds [22], [25]. This determinism precision is unacceptable for many real-time applications that require strict transmission determinism [1]. In order to improve determinism precision, TAS supports planning gate states at time-point granularity (sub-microsecond or few microseconds). This dramatically increases the calculation complexity and delay, making it hard to meet the requirement of online planning for calculation delay in many TSN scenarios [26]. Thus, it is urgently needed to design a high-precision deterministic forwarding model with low traffic planning complexity.

3 Fenglin-I Overview

In order to easily understand Fenglin-I, we compare Fenglin-I with another famous open-source chip (RISC-V). As

TABLE 1
Comparison Between RISC-V and Fenglin-I Chips

	RISC-V	Fenglin-I
High-level abstraction	Base ISA, Standard extended ISA, Non-standard extended ISA, (Instruction Set Architecture)	Best-Effort TTP, Rate-Constrained TTP, Time-Sensitive TTP, (Table Type Pattern)
Software	Compiler, Debugger,	Planner, Builder,
tool	Simulator	Controller, SDK
Real chip	Hummingbrid, Rocket, etc.	FastTSN

shown in Table 1, both RISC-V and Fenglin-I chips include high-level abstraction, software tools, and source code of real chips following the high-level abstraction (the correlations among Fenglin-I components are shown in Fig. 2). Since RISC-V chips are used as CPUs, the RISC-V instruction set architecture (ISA) accurately describes chip capabilities and is used as the high-level abstraction for hardware and software development. Differently, Fenglin-I is used in switches. Similar to CPU ISA, Table Type Pattern (TTP) can serve as the high-level abstraction of switch chips. The concept of TTP originates from OpenFlow switches [11]. It is used to describe specific switch forwarding behaviors for switches. The OpenFlow TTP is beneficial for improving and clarifying product interoperability, simplifying implementation, and improving resource utilization [12]. Inspired by the OpenFlow switch, we use Fenglin-I TTP as the high-level abstraction to describe a Fenglin-I based chip.

To serve as a foundation for TSN chip customization, Fenglin-I needs to satisfy the common requirements of TSN applications. Specifically, to satisfy the common functional requirement of supporting mixed-critical traffic, Fenglin-I TTP is divided into Best-Effort TTP, Rate-Constrained TTP, and Time-Sensitive TTP, as shown in Table 1 and Fig. 2. Each TTP describes a function chain. A function chain defines the processing path in the TSN chip for a specific traffic class. Specifically, the Best-Effort, Rate-Constrained, and Time-Sensitive TTP respectively describe the function chain for best-effort traffic (BE, which represents the traffic in traditional Ethernet), rate-constrained traffic (RC, which mainly represents audio and video traffic), and ST traffic. In order to satisfy the common functional requirement of guaranteeing high-precision deterministic transmission delay and jitter for ST traffic, the function chain for ST traffic is designed based on a novel deterministic forwarding model that delivers a high-precision deterministic transmission service and reduces traffic planning complexity, solving the *challenge II*. Moreover, in order to satisfy the common performance requirement of reducing chip area and power, we meticulously design each table format of Fenglin-I. Fenglin-I TTP gathers all table format of Fenglin-I to form an overall table map. Therefore, Fenglin-I TTP abstracts a TSN chip into multiple function chains and a table map to show the relationship between the target application requirements and chip implementation, solving the *challenge I*.

Moreover, verifying whether the custom chip satisfies the target application requirements is a necessary process for customizing a TSN chip, which requires building a TSN system. Normally, a TSN system contains multiple TSN

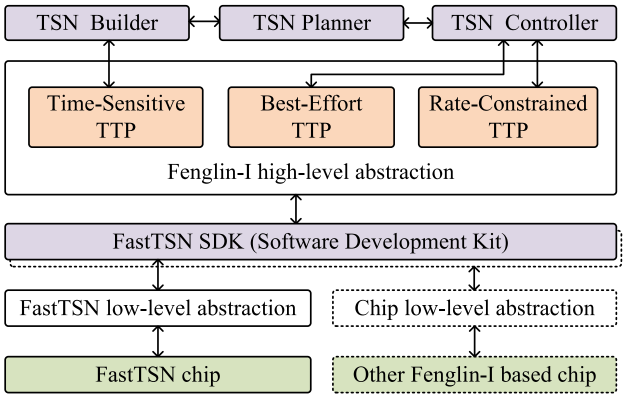

TSN Builder
TSN Planner
TSN Controller
Time-Sensitive TTP
Best-Effort TTP
Rate-Constrained TTP
Fenglin-I high-level abstraction
FastTSN SDK (Software Development Kit)
FastTSN low-level abstraction
Chip low-level abstraction
FastTSN chip
Other Fenglin-I based chip

Fig. 2. Shows the correlations among Fenglin-I components and low-level abstraction of real chips. The low-level abstraction describes chip implementation details.

chips forming a target network and software tools. Designing software tools requires designers to be clear on the implementation details of the TSN chip. For example, in order to configure a table entry, the designers of software tools are required to clearly understand the table format and the memory address of each table entry. Therefore, designing software tools is complicated and time-consuming for designers. In order to enable agile verification, we provide Fenglin-I software tools containing a planner, a controller, a builder, and software development kits (SDK).

The TSN planner mainly allocates spatial-temporal resources by a planning algorithm and dispatches the planning results to the builder and controller. The planning algorithm only focus on the planning object (such as gate states and the transmission paths of ST flows), without considering chip implementation details. Thus, the planning algorithm can be flexibly selected. The controller mainly collects application requirements for the TSN planner, and generates table information to configure the Best-Effort and Rate-Constrained tables based on the planned spatial results. The builder mainly submits the table sizes in Time-Sensitive TTP to the TSN planner and encapsulates the planned temporal results according to the format of Time-Sensitive TTP tables. The SDK receives the table entries from the builder and controller, then encapsulates them into hardware-identifiable packets according to the low-level abstraction.

Currently, we have implemented the first real chip (FastTSN) following the Fenglin-I TTP specification to provide "wheels" for the Fenglin-I based chips. The low-level abstraction of FastTSN is closely related to the chip implementation. It records interface signals of each function module, state machines, interface connection relationship between modules, etc. Sometimes, in order to satisfy specific application requirements, designers may choose a different operating system or coding language to redevelop the SDK. Thus, we provide the low-level abstraction at [3] to simplify SDK redevelopment.

Due to paper size limitations, we focus on the design and implementation of Fenglin-I TTP in this paper, which clearly reflects the design idea of Fenglin-I hardware.

3. https://github.com/fast-codesign/OpenTSN2.0/tree/distributed/Hardware/doc

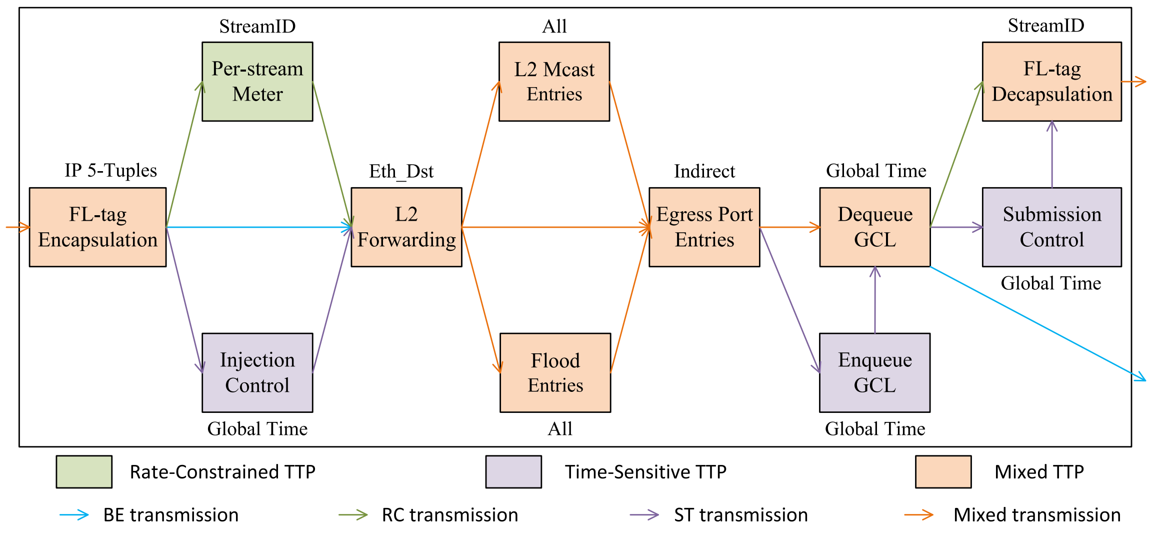


Fig. 3. Fenglin-I TTP describes a function chain for each traffic class and an overall table map. It is worth noting that Fenglin-I TTP follows the group table in OpenFlow abstraction to describe the forwarding behavior. Specifically, the Flood, L2 Mcast and Egress Port Entries are entries of the Fenglin-I group table, respectively recording the output ports of broadcast, multicast and unicast traffic.

4 Fegnlin-I TTP

In this section, we introduce Fenglin-I TTP in details including function chains and a table map.

4.1 Fucntion Chains in Fegnlin-I TTP

The function chains describe the processing paths for all traffic class of data packets.[4] In order to satisfy the common functional requirements, supporting mixed-critical traffic and guaranteeing high-precision deterministic transmission delay and jitter for ST traffic, we design the following function chains.

Function Chain for BE Traffic. As BE traffic only requires a best effort service, a basic Layer 2 forwarding function chain can satisfy its requirements. Therefore, the function chain for BE traffic integrates the functionalities connected by blue and orange lines in Fig. 3. The first functionality processing BE traffic is the FL-tag Encapsulation functionality. It directly forward BE packets to the L2 Forwarding functionality. The L2 Forwarding functionality figures out the forwarding mode (namely broadcast, multicast and unicast) of each packet and forward the packet to the corresponding forwarding functionality (namely broadcast, multicast, and unicast functionality). These forwarding functionalities acquire output ports and then forward packets to the Dequeue GCL functionality at the corresponding output port logic. The Dequeue GCL functionality controls whether the BE traffic is allowed to be dispatched, which eliminates interference with ST transmission time by forbidding BE packet dispatching when the planned transmission time of an ST packet arrives.

Function Chain for RC Traffic. RC traffic requires loose transmission delay and jitter. In order to achieve this goal, the function chain for RC traffic integrates the functionalities connected by green and orange lines in Fig. 3. Specifically, except for the basic layer 2 forwarding functionalities, this function chain integrates the Per-stream Meter and FL-tag Decapsulation functionalities. The Per-stream Meter functionality ensures that there will be a minimum inter-frame gap between any adjacent packets of a RC flow. Moreover, the FL-tag Encapsulation and Decapsulation functionalities in this function chain are used to implement a multi-semantic flow label mechanism. This mechanism proposes a custom tag named FL-tag (the FL-tag contents are shown in the FL-tag Encapsulation table, Section 4.2) to carry essential control information in the packet header. It can reduce the overhead of table look-ups along the processing path of a packet. Specifically, the FL-tag Encapsulation functionality overwrites the DMAC field with FL-tag at the first hop, which is reverted by the FL-tag Decapsulation functionality at the last hop to ensure end-to-end reachability.

Function Chain for ST Traffic. ST traffic requires high-precision deterministic transmission delay and jitter. In order to achieve this goal, this function chain integrates the functionalities connected by purple and orange lines in Fig. 3. Specifically, excepting the basic layer 2 forwarding functionalities and the multi-semantic flow label mechanism, this function chain integrates the Injection Control, Enqueue GCL, and Submission Control functionalities. The Enqueue GCL functionality detects whether ST packets arrive within the expected time windows. If an ST packet does not arrive within the planned time window, it will be discarded. That prevents erroneous ST packets from preempting the spatial-temporal resources of other ST packets.

The Injection Control, Dequeue GCL, and Submission Control functionalities are core contents of the Fenglin-I forwarding model (as shown in Fig. 4), which aims to deliver high-precision transmission determinism with low-complexity traffic planning. The Injection Control functionality controls the time-slot when ST packets are injected from the first hop, and the Submission Control functionality controls the time-slot when ST packets are submitted from the last hop. When multiple ST packets are planned to be injected (submitted) in the same time-slot, the Injection (Submission)

4. AS for the processing path for control packets, which is designed to achieve basic configuration and clock synchronization functionalities, we recommend not to modify them when customizing a TSN chip based on Fenglin-I.

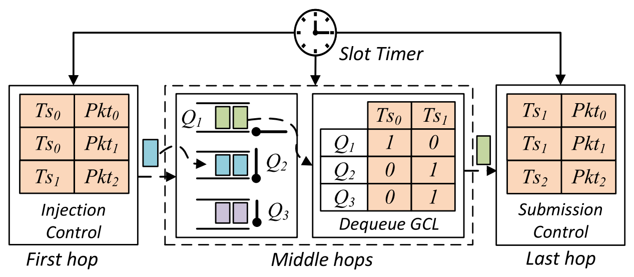


Fig. 4. Fenglin-I forwarding model. Specifically, the Injection (Submission) Control functionality controls injection (submission) *time-point* of each ST packet at the first (last) hop, which is achieved by planning the injection (submission) time-slot and the packet injection (submission) order in each injection (submission) time-slot.The Dequeue GCL functionality controls the sending *time-slot* at each middle hop.

Control functionality controls the injection (submission) order. Moreover, The Dequeue GCL functionality roughly controls the sending time-slot of each ST packet in all middle hops to ensure that the ST packets can reach the last hop node before the submission time-slot.

Different from the CQF model, the Fenglin-I forwarding model supports assigning the injection (submission) order, further controlling the fine-grained injection (submission) time-point at the first (last) hop. By doing this, Fenglin-I significantly improves deterministic accuracy. Moreover, compared with planning a time-point for each ST packet in recommended TAS-adapted planning algorithm [27], the Fenglin-I forwarding model supports planning a time-slot (much larger than a time-point) for each ST packet. As the calculation complexity and delay of a planning algorithm are positively related to the time granularity, the Fenglin-I forwarding model is able to reduce calculation complexity and delay significantly. The fine-grained theoretical analysis and practical proof are shown in Appendix A, available online.

4.2 Table Map in Fenglin-I TTP

In order to satisfy the common performance requirements, reducing chip area and power, we meticulously design queue structures, buffer structures, and table format. Since modifying the structure of queues and buffers requires designers to deeply understand the chip implementation details (such as the access logic of queues and buffers), which is hard work. We recommend designers not to modify the queue and buffer structures. Moreover, since tables are objects of software planning and configuration, we introduce the following table format in this subsection. *Each table entry is a key-value pair.* The key is the matching field, and the value defines a action. For instance, a port ID implies forwarding the matched packet to that port.

FL-tag Encapsulation Table. The FL-tag Encapsulation functionality mainly converts the 5-tuples in an IP-packet header to an FL-tag. Thus, the key of this table is 5-tuples, and the value is an FL-tag. *the FL-tag contains streamID, flow-priority, injection-address, submission-address*, stored in the DMAC field. The streamID is used to differentiate flows. The flow-priority is used to map a flow to the right queue. The injection-address/submission-address represents the memory addresses at the first/last hop for ST flows. Based on FL-tag, it is unnecessary to perform additional look-ups and keep the memory address for this control information, which significantly reduces chip area and power.

Per-stream Meter Table. This table is used to perform rate-based metering for each RC flow. The key of this table is streamID living in FL-tag, and its value is the maximum flow rate allowed for the corresponding RC flow. Directly mapping from streamID to rate does not require recording status information, such as the status of ST traffic in another meter for RC traffic (credit-based shaper [24]), which helps to save logic resources. Since the packet length is recorded after receiving the first packet, the minimum interval of sending time between any two adjacent packets can be calculated by the

$$interval_{min} = \frac{length_{pkt}}{rate_{max}} - \frac{length_{pkt}}{bandwidth}. \quad (1)$$

The $interval_{min}$ is the minimal interval allowed for the RC flow, the $length_{pkt}$ is the length of the RC packet, the $rate_{max}$ is the maximal transmission rate acquired by looking up this table, and the $bandwidth$ represents the bandwidth of the link. The Per-stream Meter functionality guarantees that the actual time interval of any adjacent packets of the RC flow is not less than the $interval_{min}$.

Injection Control Table. The Injection Control table is used to control the injection time-slot of each ST packet at the first hop. The key of this table is a sequence number of time-slot, and the value is the injection address. To be specific, at the beginning of each time-slot, the Injection Control functionality uses the current time-slot as the key to match this table for acquiring the injection address, and then dispatches the packet according to the injection address. The streamID of the packets living in the injection address is recorded in the streamID field of the FL-tag.

L2 Forwarding and Group Tables. These tables are used to acquire the output ports. The key of the L2 Forwarding table is the DMAC, and the value is the entry sequence number of the group table. Each entry of the group table records operation type and instructions. For example, a group table entry for a BE multicast packet records *operation type: All, Instructions: (Output port: 1, Go to Dequeue GCL; Output port: 2, Go to Dequeue GCL; Output port: 3, Go to Dequeue GCL)*, which means the packet should be copied three times, and each copy will be output from port 1, port 2 and port 3, respectively. See more details about the group table at [11]. In order to unify the operations for broadcast, multicast and unicast traffic, we combine the two tables into one table to save resources. The key is the DMAC, and the value is the output ports that is encoded in the form of a bitmap. For example, if the output ports obtained by matching this table is 4’b1101, it means that the output ports of the flow at this node are port0, port2, and port3.

Enqueue and Dequeue GCLs. The key of Enqueue (Dequeue) GCL is a sequence number of time-slot, and the value is gate states. The processing logic obtains the enqueue (dequeue) gate state of each time-slot by matching the Enqueue (Dequeue) GCL. Compared with the GCL format in TSN standard [21], these GCLs replace the duration field of each gate state with the time-slot field. The duration is encoded in 4 octets as a 32-bit unsigned integer, representing a number of nanoseconds. The time-slot field takes

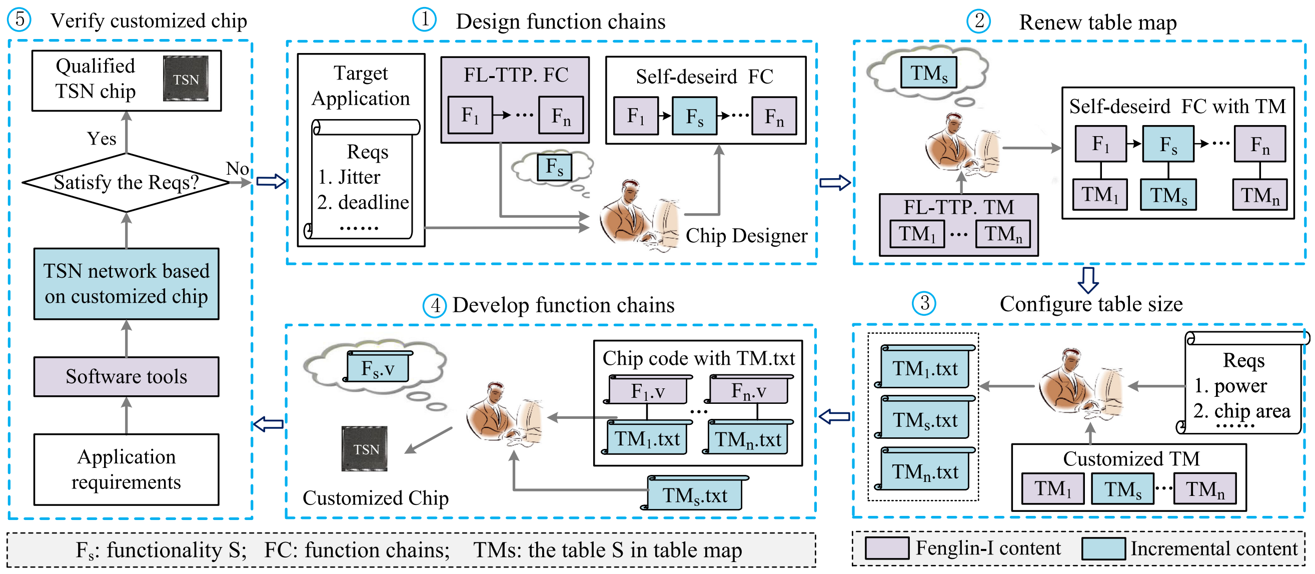


Fig. 5. ATC method. The TP$_s$.txt file is the configuration file of table parameters in functionality s, and the F$_n$.v file is the code file of functionality n.

10 bits, generally representing a number of hundreds or tens of microseconds. Therefore, replacing the duration field with the time-slot field saves table resources.

Submission Control Table. The submission Control functionality is used to control the submission time-slot of each ST packet at the last hop. The key of this table is a time-slot, and its value is the submission address. Similar to the injection Control table, this table is used to control submitting an ST packet of the specified flow at a predetermined time-point.

FL-tag Decapsulation Table. The key of this table is the streamID, and its value is the DMAC. The FL-tag Decapsulation functionality acquires the DMAC by looking up this table with the streamID, and replaces the FL-tag occupying the DMAC field with the acquired DMAC.

5 ATC Method

Based on Fenglin-I, we propose an agile TSN chip customization method named ATC. Using this method to customize a TSN chip requires the following five steps, as illustrated in Fig. 5.

Step 1: Design Function Chains. At this step, designers first collect the functional requirements of the target application. By comparing the functional requirements that Fenglin-I satisfies and the ones of the target application, designers will extract the same functional requirements between them. Then, designers can re-utilize the Fenglin-I function chains targeting these same functional requirements. As for the functional requirements that Fenglin-I does not satisfy, designers extract adaptive functionalities from the TSN standards. Based on the Fenglin-I function chains and the extracted functionalities, designers can customize new function chains by extending or pruning the Fenglin-I function chains.

For instance, as Fenglin-I does not provide high reliability, it is advisable to extend frame replication and elimination functionalities according to 802.1 CB [18] for those applications which require high reliability, such as a high-voltage distribution network. When designing new function chains based on Fenglin-I function chains, it is crucial to find appropriate locations in the Fenglin-I function chains to insert the extended functionalities. Since the frame replication and elimination functionalities are recommended to deploy after traffic convergence [18], we insert it after FL-tag Encapsulation functionality. As for inserting a functionality outside of the TSN standards, designers need to figure out the traffic class handled by the functionality and then insert this functionality into the corresponding function chain.

Step 2: Renew Table Map. After designing the function chains, designers need to determine the table format for each functionality. Fenglin-I already provides its table format. Since the Fenglin-I table format is designed to satisfy the common performance requirements (reducing chip power and area), and modifying the table format requires a deep understanding of the implementation details, which is a hard challenge for designers, the Fenglin-I table format is recommended not to modify. Designers only focus on the table format in incremental functionalities. For instance, the table format in the frame replication and elimination functionalities is a key-value pair. The key is the streamID, and the value is the sequence number of the latest received packet. To reduce the lookup time, designers can directly use the streamID as the corresponding table address.

Step 3: Configure Table Size. At this step, designers configure the table size of both incremental modules and FastTSN modules. The primary goal of configuring these parameters is to save as many resources as possible while meeting the performance requirements of the target application. For instance, as for the table for recording the sequence number of the latest received packet in the frame replication and elimination functionalities, each ST flow consumes a table entry. The table size should be larger than the maximum number of ST flows that each switch or host may load.

Step 4: Develop Function Chains. At this step, designers focus on developing incremental functionalities. In the developing process, designers only cares about the interfaces of adjacent function modules without figuring out the implementation details of adjacent modules. For instance, when inserting the frame replication and elimination

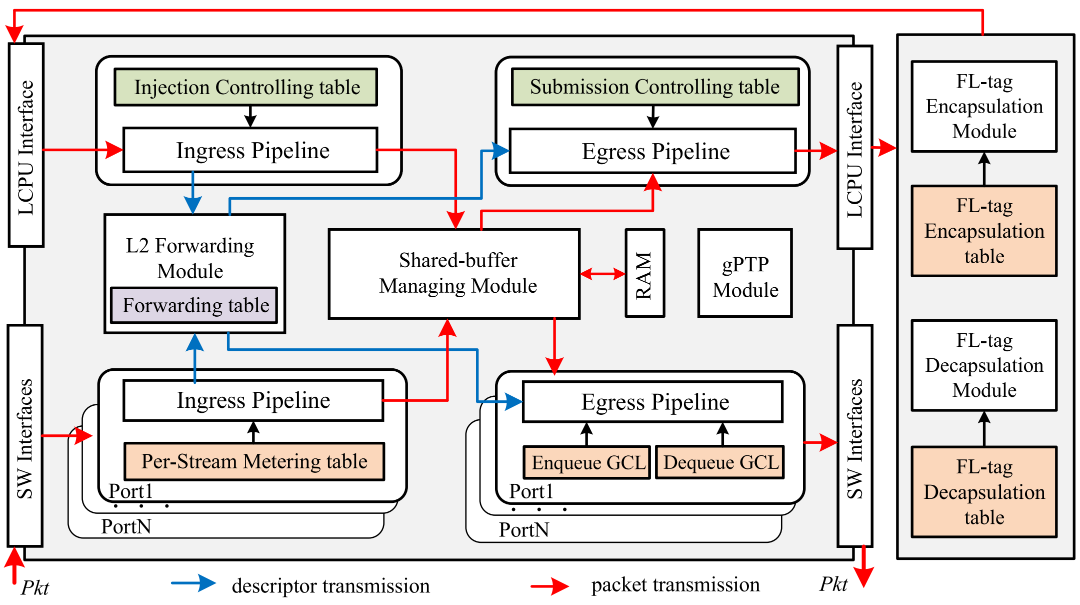


Fig. 6. FastTSN implementation framework. The red line indicates the direction of packet transmission, and the blue line indicates the direction of descriptor transmission.

module after the FL-tag Encapsulation module, the input interfaces of this module should be consistent with the output interfaces of the FL-tag Encapsulation module. In order to simplify the Interface adaptation, FastTSN uses FIFO as the interface between modules. The content transferred between modules is the packet descriptor which contains the required data information (such as FL-tag and partial header information) for packet processing in function chains or the packet carrying with FL-tag.

Step 5: Verify Customized Chip. At this step, designers verify whether the customized chip satisfies the functional and performance requirements of the target TSN application. To achieve this goal, designers first use the customized chips to build a network required in the target TSN application, and adopt the software tools to configure these underlying chips. The work of each software tool is shown in section 3. At run time, designers collect critical status information such as timestamps. After that, comparing the actual collected parameter values with the theoretical parameter values analyzed based on the forwarding model and planning results, if they are not consistent, feedback is provided to step 1 for a new round.

Compared with customizing from a "clean-slate", customizing a TSN chip based on Fenglin-I eliminates the workload of designing function chains and Fenglin-I table format, implementing FastTSN modules and software tools.

6 FastTSN Implementation and Evaluation

Based on Fenglin-I TTP, we implement a real chip named FastTSN. In this section, we introduce FastTSN from the implementation architecture and processing workflow. Moreover, we extensively evaluate it on FPGA arrays.

6.1 Implementation Architecture

FastTSN implementation architecture contains many design details, as shown in Fig. 6. We mainly introduce how to map the function chains in Fenglin-I TTP into this implementation architecture. Before introducing the mapping relationship, it is necessary to introduce the interfaces that FastTSN integrates, namely the Local CPU (LCPU) interface and switching (SW) interface. Both the LCPU and SW interfaces follow the standard Ethernet interface format. Between them, the LCPU interface connects the Fl-tag Encapsulation and Decapsulation modules with other modules. By doing this, it is beneficial for users to flexibly choose a platform (such as ASIC, FPGA, and CPU) to develop the FL-tag Encapsulation and Decapsulation modules. Moreover, FastTSN can be used as an L2-Switching Ethernet chip without deploying the FL-tag Encapsulation and Decapsulation modules. Differently, the SW interface is used to connect a host or a switch.

When a BE packet enters from an SW interface, it goes through the L2 Forwarding module and the Dequeue GCL module in the egress pipeline connecting with the SW interface. The Forwarding module is used to acquire output ports. The Dequeue GCL module is used to eliminate interference with ST transmission time by controlling whether the BE traffic is allowed to be dispatched. The above processing path for BE traffic is accorded with the BE function chain described in Section 4.1.

As for an RC packet or an ST packet, the actions in the L2 Forwarding module and the Dequeue GCL module are the same as those for a BE packet. However, other actions for an RC packet or an ST packet are different when the FastTSN chip is deployed in a different location, namely the first hop, a middle hop, and the last hop. Specifically, in the FastTSN deployed as the first hop, the FL-tag Encapsulation module replaces the DMAC with an FL-tag for each RC packet. In the FastTSN deployed as a middle hop, the Per-stream Meter module in the ingress pipeline connecting with an SW interface limits the transmission rate of each RC flow. In the FastTSN deployed as the last hop, the FL-tag Decapsulation module replaces the FL-tag with the DMAC. The above processing path for RC traffic is accorded with the ones of the RC function chain described in Section 4.1.

Moreover, the actions in the L2 Forwarding module, the Dequeue GCL module, the FL-tag Encapsulation, and the Decapsulation modules for an ST packet are the same as those for an RC packet. Additionally, in the FastTSN deployed as the first hop, the Injection Control Module in the ingress pipeline connecting with the LCPU interface controls the injection time-point of each ST packet. In the FastTSN deployed as a middle hop, the Enqueue GCL module in the egress pipeline connecting with the SW interface discards the ST packets that do not arrive within the planned time window. In the FastTSN deployed as the last hop, the Submission Control module in the egress pipeline connecting with the LCPU interface controls the submission time-point of each ST packet. The above processing path for an ST packet is accorded with the ST function chain described in Section 4.1.

6.2 Processing Workflow

In this section, we introduce the overall processing workflows of FastTSN from the processing workflows at the first hop, at a middle hop, and at the last hop.

Algorithm 1. Processing Workflows at the First Hop

```
Input: pkt_x, FL-tag Encapsulation table (FE) and Injection
       Control table (IC)
1:  for $a \leftarrow 0$ to $FE_{valid\_depth}$ do
2:    if $pkt_x.5\_tuples == SG[a].5\_tuples$ then
3:      pkt_x.FL_tag = FE[a].FL_tag;
4:  if $pkt_x.FL\_tag.flow\_priority \in ST$ then
5:    pkt_x.inject_addr = pkt_x.FL_tag.inject_addr;
6:    store pkt_x according to the injection_addr;
7:    for $b \leftarrow 0$ to $IC_{valid\_depth}$ do
8:      if $pkt_x.inject\_addr == IC[b].inject\_addr$ then
9:        pkt_x.inject_time = IC[b].inject_time;
10: while $pkt_x.inject\_time == current\_time$ do
11:   read pkt_x from pkt_x.inject_addr and injects it;
```

Processing workflows at the first hop. This process is shown in Algorithm 1. First, when a packet (pkt$_x$) reaches the first hop, and pkt$_x$ belongs to ST or RC traffic, the FL-tag Encapsulation module replaces the pkt$_x$ DMAC with a corresponding FL-tag, which is acquired by looking up the FL-tag Encapsulation table (lines 1-3), then forwards pkt$_x$ to the LCPU interface. If pkt$_x$ is a BE packet, the FL-tag Encapsulation module directly forwards it to the LCPU interface. After the actions of the FL-tag Encapsulation module, if pkt$_x$ belongs to an ST packet, the Injection Control module looks up the FL-tag to acquire the injection address (lines 4-5) and then pushes pkt$_x$ into the buffer according to the injection address (line 6). When the current time reaches the injection time-slot of pkt$_x$, acquired by looking up the Injection Control table, the Injection Control module fetches pkt$_x$ according to its injection address, then injects it to the next hop as soon as possible (lines 7-11).

Processing Workflows at a Middle Hop. The process is shown in Algorithm 2. When a packet (pkt$_x$) reaches a middle hop, the Per-stream Meter module checks whether the packet belongs to an RC flow. The Per-stream Meter module acquires the maximum transmission rate by looking up the Per-stream Meter table for RC packets (lines 1-4) and then keeps the actual flow rate less than the maximum transmission rate. If pkt$_x$ is not an RC packet, it will not go through the Per-stream Meter module. After that, the L2 Forwarding module looks up the Forwarding table to acquire the output ports regardless of the traffic type of pkt$_x$, then forwards the packet to the corresponding egress pipeline according to the acquired output ports (lines 5-8). If pkt$_x$ is an ST packet, the Enqueue GCL module in the egress pipeline acquires the queue-id of pkt$_x$ according to the flow-priority attached in the FL-tag (line 9), and inquires the en-queue gate state of the corresponding queue at the current time-slot by looking up the Enqueue GCL. If the en-queue gate state is open, put pkt$_x$ into the corresponding queue. Otherwise, pkt$_x$ will be discarded (lines 9-15). If pkt$_x$ is not an ST packet, it will not go through the Enqueue GCL module. Next, the scheduler in the egress pipeline dispatches the packets in the queues whose de-queue gate states are open according to the strict priority policy (ST traffic has the highest priority, RC traffic has the medium priority, and the priority of BE traffic is the lowest). The packets in the same queue are scheduled in first-in-first-out order (lines 16-19).

Algorithm 2. Processing Workflows at a Middle Hop

```
Input: pkt_x, Per-stream Meter table (PM), Forwarding table
       (Fwd), Enqueue GCL (EG), Dequeue GCL (DG)
1:  if $pkt_x.FL\_tag.flow\_priority \in RC$ then
2:    for $c \leftarrow 0$ to $PM_{valid\_depth}$ do
3:      if $pkt_x.stream\_id = PM\_[c].stream\_id$ then
4:        RC[pkt_x.stream_id].rate=PM[b].rate;
5:  pkt_x.outports = Fwd[pkt_x.stream_id].outports;
6:  for $d \leftarrow 0$ to $port_{num}$ do
7:    if $pkt_x.outports[d] = 1$ then
8:      forward pkt_x to port d;
9:  pkt_x.queue_id = pkt_x.FL_tag.flow_priority;
10: if $pkt_x.FL\_tag.flow\_priority \in ST$ then
11:   gate_states = EG[cur_timeslot].gate_states;
12:   if $queue\_id.gate\_state = 1$ &&
      $queue\_id.used\_depth \neq Full$ then
13:     put pkt_x into queue_id;
14:   else
15:     discard pkt_x;
16: for $queue\_id \leftarrow 0$ to $(queue_{num} - 1)$ do
17:   gate_states = GCL[current_timeslot].gate_states;
18:   if $queue\_id.gate\_state = 1$ && $queue\_id.empty = 0$ then
19:     allow scheduling packets from queue_id;
```

Processing Workflows at the Last Hop. The process is shown in Algorithm 3. When pkt$_x$ arrives at the last hop, the Submission Control module inquires about the packet priority of pkt$_x$. If pkt$_x$ is an ST packet, the Submission Control module looks up the FL-tag to acquire the submission address (line 1-2) and then pushes pkt$_x$ into the memory according to the submission address (line 3). When the current time reaches the submission time of pkt$_x$ acquired by looking up the Submission Control table, the Submission Control module fetches pkt$_x$ according to its submission address, then submits it to the FL-tag Decapsulation module (lines 4-8). If pkt$_x$ is an RC packet, it will directly enter the FL-tag Decapsulation module without going through the Submission Control module. After that, the FL-tag Decapsulation module looks up the FL-tag Decapsulation table to acquire the pkt$_x$ DMAC and fills the pkt$_x$ DMAC back to the DMAC

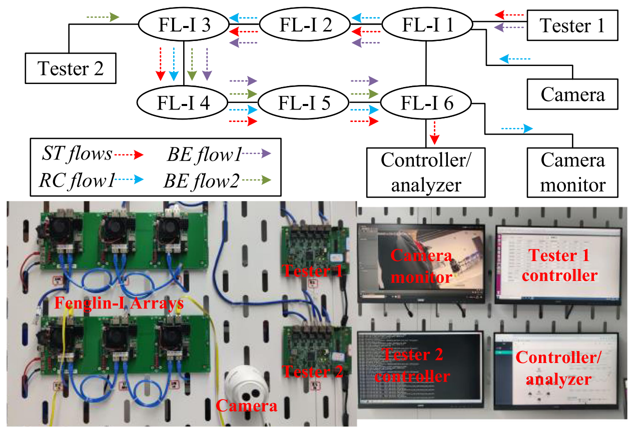


Fig. 7. Topology of experiment I.

field(lines 9-12). However, if pkt_x is not an ST or RC packet, pkt_x will not go through the Submission Control and FL-tag Decapsulation modules.

Algorithm 3. Processing Workflows at the Last Hop

```
Input: $pkt_x$, FL-tag Decapsulation table (FD) and Submission
       Control table (SC)
1:  if $pkt_x.FL\_tag.flow\_priority \in ST$ then
2:      $pkt_x$.submit_addr = $pkt_x$.FL_tag.submit_addr;
3:      store $pkt_x$ according to the submission_addr ;
4:      for $e \leftarrow 0$ to $SC_{valid\_depth}$ do
5:          if $pkt_x.submit\_addr = SC[e].submit\_addr$ then
6:          $pkt_x$.submit_time = SC[e].submit_time;
7:  while $pkt_x.submit\_time == current\_time$ do
8:      read $pkt_x$ from $pkt_x$.submit_addr, submit it;
9:  for $e \leftarrow 0$ to $FD\_entry_{num}$ do
10:     if $pkt_x.stream\_id = FD[e].stream\_id$ then
11:         $pkt_x$.dmac = FD[e].dmac;
12: fill the dmac back to the corresponding field;
```

6.3 Evaluation

To verify the feasibility of FastTSN chip, we prototype FastTSN on FPGA and deploy two typical TSN scenarios.

Experiment I: ring topology, unicast flows.

Devices settings.

In this experiment, as shown in Fig. 7, all the FL-Is (FastTSN chips) are connected, forming a ring topology. The controller/analyzer is used to configure and monitor all FL-Is. Moreover, both Tester1 and Tester2 are implemented by FPGA, which construct flows according to the configuration parameters (such as packet payload and sending interval) generated by the tester controllers. The camera and camera monitor are used to generate an RC flow. In addition, the bandwidth of each link is 1Gbp/s.

Traffic Settings. As illustrated in Fig. 7, there are BE flow1, BE flow2, RC flow1, and 32 ST flows. The bandwidth of BE flow1 is 400Mbps/s. The RC flow1 bandwidth is 5Mbps/s. The bandwidths of BE flow2 and each ST flow are variables. Among them, the BE flow1 and 32 ST flows are generated from Tester 1, which aims to simulate the host generating ST flows and non-ST flows simultaneously. BE flow2 is used to construct a congested link between FL-I 3 and FL-I 4.

Synchronization Offset Evaluation. We set FL-I 1 as the master clock (actively triggering synchronization) and the other FL-Is as slave clocks (passively executing synchronization). The synchronization interval is 100ms. We collect offset values in 400 minutes at runtime. As shown in Fig. 8, *the offset between slave clocks and the master clock always keeps less than 64ns.* Moreover, the offset value is positively correlated with the distance between the slave clock and master clock, which is consistent with theoretical analysis [23]. Specifically, as the distance between FL-I 4 and the master clock is the farthest, and the one between FL-I 2 or FL-I 6 is the nearest, the average offset value of FL-I 4 is the maximum, and the average offset value of FL-I 3 is over than the one of FL-I 2 (as shown in Fig. 8a). Moreover, the average offset value of FL-I 2 is roughly consistent with the one of FL-I 6 (as shown in Fig. 8b), and the average offset value of FL-I 3 is roughly consistent with the one of FL-I 5 (as shown in Fig. 8c).

Delay and Jitter Evaluation. We evaluate the influence of background traffic (such as BE traffic) workload on ST determinism by setting different bandwidths of BE flow2, 400Mbp/s and 700Mbp/s. When the bandwidth of BE-flow2 is 700Mbp/s, the accumulated traffic bandwidth from FL-I3 to FL-I6 is over than the link bandwidth. As illustrated in Fig. 9a, the difference between the average end-to-end delays of 8 ST flows under 400Mbp/s and 700Mbp/s BE flow2 keeps less than 40ns. The difference mainly results from the clock offset. Thus, the workload size of BE traffic has no influence on the determinism of ST traffic.

Next, we evaluate the influence of interval (the submission time-slot minus the injection time-slot), time-slot value, and ST bandwidth on ST determinism by setting different intervals (6time-slots and 1000 time-slots), different time-slot values (namely 64us and 128us) and different ST bandwidths (16Kbp/s and 16Mbp/s). The difference between the actual average end-to-end delays of 8 ST flows and the theoretical interval (shown as the red dotted line in Fig. 9) is less than 30ns, and the jitters of the end-to-end delays always keep less than 120 ns, as illustrated in Fig. 9b, Fig. 9c. Thus, the interval, time-slot value, and ST bandwidth do not influence the determinism of ST traffic. The above experimental results demonstrate that FastTSN is able to provide microsecond-level determinism.

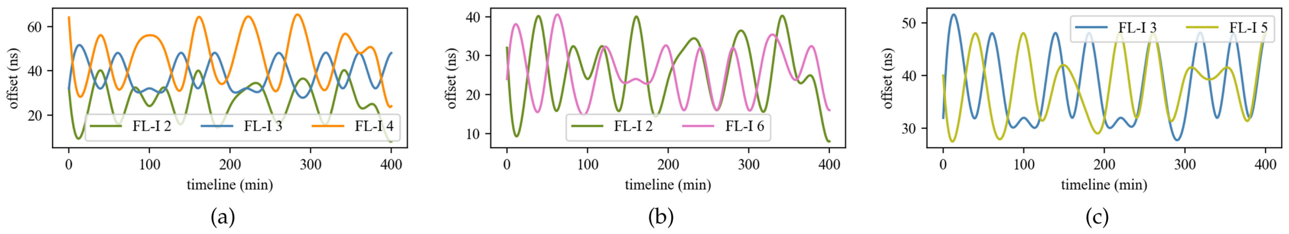


Fig. 8. Offset between slave and master clocks. (a) FL-I 1, FL-I 2 and FL-I 4 offset. (b) FL-I 2 and FL-I 6 offset. (c) FL-I 3 and FL-I 5 offset.

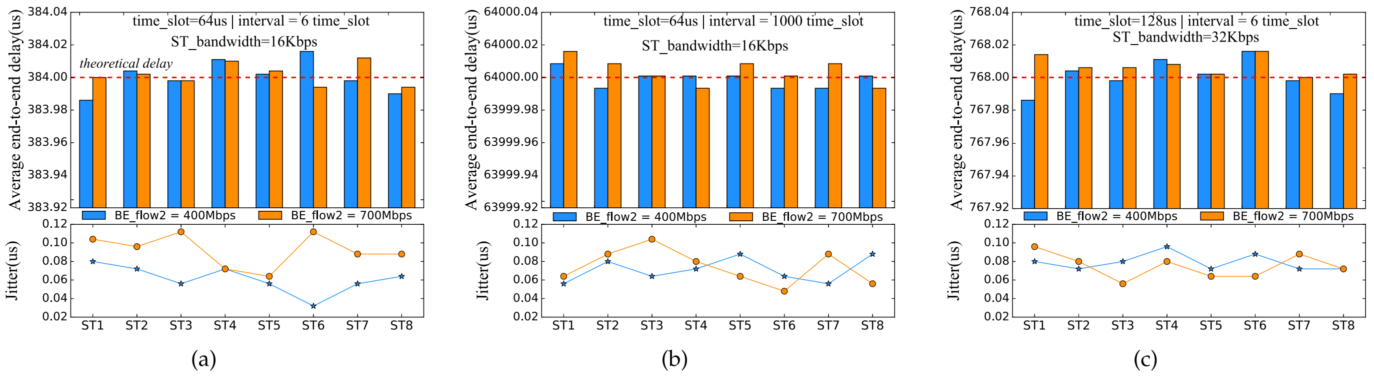


Fig. 9. End-to-End latency and jitter. (a) basis setup. (b) under different interval. (c) under different time-slot and bandwidth of ST flows.

Experiment II: line topology, multicast flows.

Devices Settings. Guaranteeing determinism of multicast ST traffic is crucial for many TSN scenarios, such as triple modular redundancy in aerospace craft and advanced driver assistance system in the vehicle. Thus, we build this experiment to verify the determinism precision that FastTSN provides for multicast ST traffic. As illustrated in Fig. 10, there are three FL-Is to form a linear network. All FL-Is are used as switches, which means FL-Is do not use the Injection and Submission Control functionalities to improve ST determinism. This is consistent with the above scenarios.

Traffic Settings. The ST flow generated by Tester 2 is a multicast flow, which respectively goes through FL-I 2 and FL-I 3 to arrive at Tester 2. Thus, Tester 2 receives two copies of each ST packet (the identical packets are called packet pair). Tester 2 forwards the ST packets to the analyzer as soon as it receives them. The BE flow is used as background flow to demonstrate the influence of BE flow on the determinism of ST multicast flow.

Determinism Evaluation. We first verify the influence of ST packet length and bandwidth on ST determinism. We stop the BE flow and generate ST flows with different packet lengths and bandwidths, 128B and 1Mbp/s, 512B and 5Mbp/s, 1024B and 10Mbp/s. Moreover, we configure this ST flow to transmit from FL-I 2 and FL-I 3 at the same time-slot. After that, we collect the time-point pair for each ST packet pair arrived at Tester 2, and calculate the absolute time difference of each time-point pair. As shown in Fig. 11a, the time difference keeps less than 100ns under different ST packet lengths and bandwidths, which is mainly resulted from clock offset.

Next, we evaluate the influence of background traffic on ST determinism. We add a BE flow whose packet length is 128B and change its bandwidth, 10Mbp/s, 100Mbp/s, 200Mbp/s and 500Mbp/s. As FL-Is do not use the injection and submission control functionalities to improve ST determinism, the BE flow is supposed to increase the time difference of each time-point pair. This is because when an ST packet is planned to transmit, a BE packet may be transmitting at the same physics port, resulting in the ST packet having to wait until the BE packet finishes transmission. This increases the time offset of ST packet sending from FL-I 2, further increasing the time difference of each time-point pair. We collect the time difference of 2000 packet pairs, and the results show the time difference is positively correlated with BE bandwidth. This is because the higher bandwidth of the BE flow is, the greater probability that ST flow is blocked at FL-I 2. As the BE packet length is 128B, transmitting a BE packet from FL-I 2 port whose line rate is 1Gbp/s needs 1μs. Thus, the time difference is generally less than 1us, as shown in Fig. 11b. In order to eliminate the influence of BE traffic on ST transmission determinism, we turn on the guard band mechanism [21], that is, BE traffic is not allowed to transmit within 12 microseconds before ST traffic is planned to be transmitted, and the result is the same as the one shown in Fig. 11a.

7 Use Cases

In this section, we demonstrate two customized TSN chips, HX-DS09 ASIC chip and TZ-TS01 FPGA-based chip.

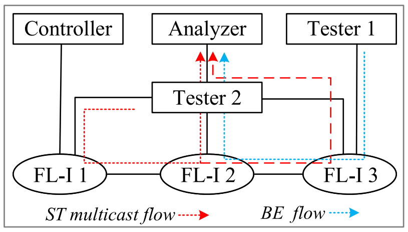
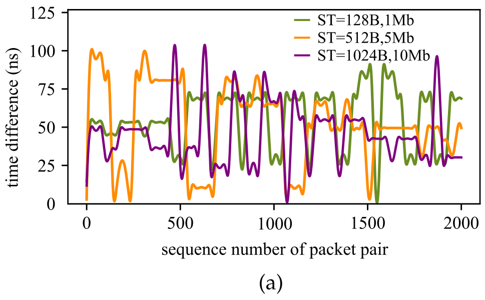


Fig. 10. Topology of experiment II.



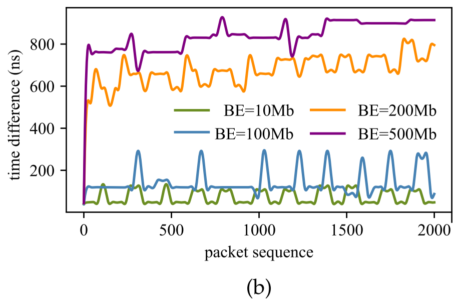


Fig. 11. Time differences of ST packet pairs. (a) under different ST bandwidth and packet length. (b) under different bandwidths of BE flow.

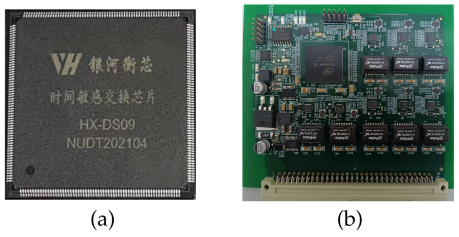


Fig. 12. Two Customized chip based on Fenglin-I. (a) HX-DS09 chip. (b) A FPGA-based TSN chip.

7.1 HX-DS09 ASIC Chip

Further to demonstrate the benefits of Fenglin-I, we customize an ASIC chip named HX-DS09 (as shown in Fig. 12a) based on Fenglin-I to provide deterministic service for time-critical flows among controllers, actuators, and sensors in an industrial control system. The requirements are (i) some actuators need to respond to the duplicate multicast packets. Since the memory resource in the actuators is limited, the actuators at most supports receiving packets continuously for 2μs without any drops. Therefore, HX-DS09 requires 2μs determinism for multicast flows. (ii) The power supply provides 1W power to the original Ethernet chip. In order to substitute the Ethernet low-power chip, HX-DS09 requires less than 1W power consumption.

As the ATC method described in section 5, Customizing HX-DS09 based on Fenglin-I requires the following steps.

Design Function Chains. The target chip requires 2μs determinism for multicast flows, and FastTSN can deliver 1μs determinism for multicast flows with the interference of 128B BE traffic, as illustrated by the results of experiment II. To reduce or eliminate the interference of non-ST packets, frame preemption and slicing functionalities are two options. Because frame preemption involves modifying PHY logic, which results of high implementation complexity. Therefore, we insert slicing and recombining functionalities into Fenglin-I function chains, forming the function chains of HX-DS09. The slicing functionalities cuts each packet larger than 128 bytes into multiple slices smaller than 128 bytes, and the recombining function reassembles these slices back into a complete packet. According to the definition of end-to-end delay in Appendix A, available in the online supplemental material, it is critical to eliminate the influence of BE packets on the injection and submission time of ST packets. Therefore, the HX-DS09 chip inserts the slicing functionality before the FL-tag Encapsulation function and the recombining functionality after the FL-tag Decapsulation function. Moreover, there is no RC traffic in the target application, so we delete the function chain for RC traffic further to form the HX-DS09 TTP, as shown in Fig. 13.

Renew Table Map. In order to reassemble slices back into a complete packet, the recombining module adopts a table to record the sequence number of received slices. The keys of this table are the StreamID and packet sequence, and the value is the sequence number of received slices. Extracting and optimizing this table takes a few hours.

Configure Resource Size. As the recombining functionality reassembles slices at the last hop and the last hop node loads less than 100 flows in the target application, the size of the table in recombining functionality is set to 128.

Develop Function Chains. In this step, we develop the slicing and recombining modules. The input interfaces of the FL-tag Encapsulation module and the output interfaces of the FL-tag Decapsulation module are the standard Ethernet packet interfaces. In order to adapt to those interfaces, both the input and output interfaces of slicing and recombining functionalities are also the standard Ethernet packet interfaces.

Verify Customized Chip. In this step, we build a TSN system similar to experiment II. The verification results show that HX-DS09 is able to deliver 1.2μs determinism, which is mainly composed of the following three parts. (1) Clock offset. As shown in Fig. 9, the clock offset always keeps within 64ns. (2) The jitter of the PHY chip in the HX-DS09 test board. Since the HX-DS09 test board respectively records the sending and receiving timestamps before and after the PHY chip, the jitter of the HX-DS09's PHY chip will interfere with the timestamp values. After testing a single HX-DS09 test board, it is concluded that the round-trip jitter of the HX-DS09's PHY chip keeps within 132ns. (3) Interference of non-ST frames. Since the HX-DS09 integrates the

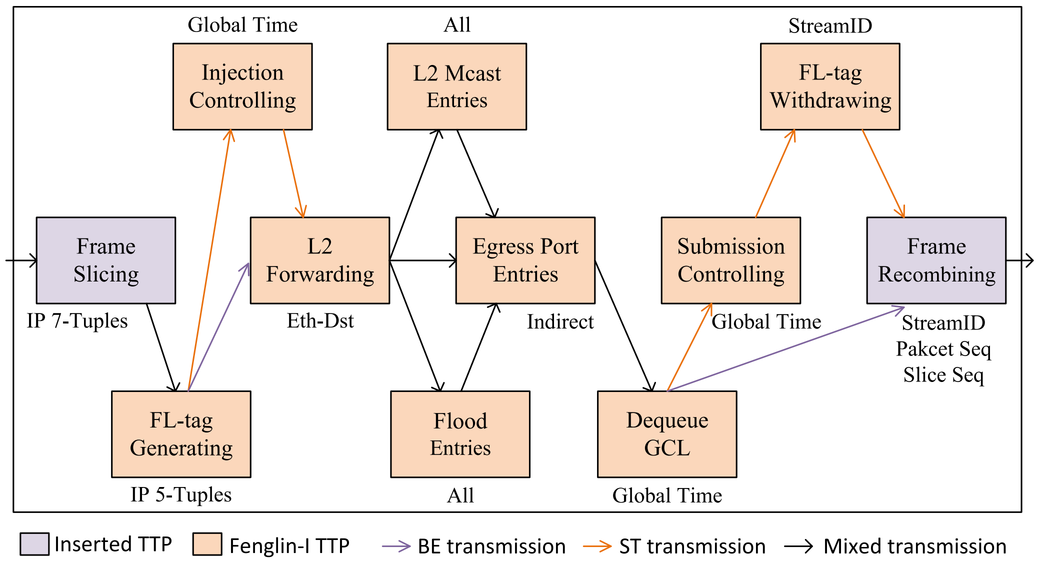


Fig. 13. HX-DS09 chip extends the frame slicing and recombining functionalities based on Fenglin-I chip.

TABLE 2
Code Quantity in HX-DS09

HX-DS09 Modules	quantity	belonging	percentage
LCPU ingress pipeline	3128 lines	FastTSN	84.1%
LCPU egress pipeline	1932 lines		
L2 forwarding	1472 lines		
buffer management	2300 lines		
gPTP	733 lines		
switching ingress pipeline	3036 lines		
switching egress pipeline	4370 lines		
FL-tag Encapsulation	1357 lines		
FL-tag Decapsulation	483 lines		
LCPU or SW interface	1416 lines		
slicing	456 lines	incremental modules	15.9%
recombining	3381 lines		

slicing and reassembling functionalities which cut the frame larger than 128 bytes into multiple frames smaller than 128 bytes, the maximum block time caused by a non-ST frame is 1μs (128B/1Gbps). Moreover, the HX-DS09 adopts 84.1% of code from FastTSN chip database (as shown in Table 2) and achieves ultra-low power (0.5W) under the 130nm process.

7.2 TZ-TS01 FPGA-Based Chip

A third-party customizes an FPGA-based chip named TZ-TS01 (as shown in Fig. 12b) based on Fenglin-I to provide deterministic service for time-critical flows between controllers and sensors in an aircraft network. This scenario (i) requires the single-hop delay keep less than 200μs under the 100Mbps physical interface, (ii) requires end-to-end transmission jitter keep less than 100μs, (iii) support 64 ST flows from 32 nodes.

As FastTSN satisfies these requirements, the designers of TZ-TS01 just delete the Injection Control and Submission Control functionalities, without any increment code, and configure the table size on demand. Although the deterministic accuracy is decreased to tens of microseconds, it still satisfies the determinism requirement. Moreover, the single-hop delay keeps less than 7us under the 100Mbps physical interface.

8 Related Work

Open-Source Hardware. With the rapid increase of new IT technologies, hardware become more and more complicated. In order to reduce the complexity of new technology development, open-source technology has become an essential means of hardware development. RISC-V [10] foundation proposes the instruction set architecture, and some organizations open source code of RISC-V chips. For example, SiFive proposes an open-source RSIC-V chip, Rocket-chip. Users are able to agilely customize a RSIC-V chip by following the specification of the instruction set architecture and referring to the source code. However, The RISC-V chip aims at general purpose computation. Another open-source hardware is corundum (a high-performance FPGA-based NIC) [28], which opens the source-code. corundum project encourages users to directly reproduce the mature products, rather than serving as the foundation of customization. NetFPGA [29] and P4NetFPGA [30] are famous open-source programmable switches. They recommend designers to customize switches by configuring table parameters. However, they do not support time-triggered actions, such as Enqueue and Dequeue GCL functionalities, and thus cannot be used to customize TSN switches.

TSN Chip Design. The commercial off-the-shelf (COTS) TSN chips [4], [5] are developed in a Bottom-up method without considering specific application requirements. It is challenging for users to select a COTS chip that satisfies all their requirements appropriately. Moreover, the resource partitioning for tables, queues and buffers in such chips is fixed. In many cases, the resource partitioning does not adapt to the specific application features very well. As a result, the on-chip memory resource is often under low utilization. Therefore, TSN-Builder [6] decomposes a TSN chip into five core modules and decouples the various resource specifications of each module from the fixed processing logic with a comprehensive abstraction of memory-related resources. Users can configure these resource specifications on demand for customizing a resource-efficient TSN switches. However, TSN-builder lacks the software tools and does not recommend designing new function chains, limiting flexibility for chip customization. Moreover, its forwarding model follows CQF and TAS, facing the challenge I.

Open-Source TSN-Related Project. OpenAvnu project [31] focuses on TSN and AVB systems and realizes the gPTP and SRP protocols, etc. However, it does not realize TSN core functionalities, such as TAS and CQF, etc. OpenTSN project [8], [9] opens a compete TSN system including hardware in data plane and software in control plane to reduce the customization complexity. However, it lacks the detailed high-level abstraction and customization method. Moreover, it also adopts CQF and TAS forwarding model, facing the challenge I.

9 Conclusion

In this paper, we propose an open-source TSN chip to serve as the basis of customizing TSN chips to reduce development overhead. Specifically, the Fenglin-I chip contains high-level abstraction (TTP), software tools, and a real chip (FastTSN) following Fenglin-I TTP specification. To demonstrate the effectiveness of Fenglin-I, we prototype and evaluate Fenglin-I on FPGA-based testbed. We verify that Fenglin-I chip satisfies the common requirements of TSN applications. Moreover, we propose a method named ATC to introduce the concrete steps of customizing TSN chips based on Fenglin-I. Finally, we and a third-party use the ATC method to customize two different chips. The post-customization chips reuse 84.1% and 100% from FastTSN's codebase.

References

[1] A. Nasrallah *et al.*, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Commun. Surv. Tuts.*, vol. 21, no. 1, pp. 88–145, Jan.–Mar. 2019.

[2] J. Farkas, L. L. Bello, and C. Gunther, "Time-sensitive networking standards," *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 20–21, Jun. 2018.

[3] *IEEE Standard for Local and Metropolitan Area Networks-Bridges and Bridged Networks (802.1Q standard)*, IEEE Std. 802.1Q-2018, Jul. 2018. [Online]. Available: https://standards.ieee.org/standard/802_1Q-2018.html

[4] BROADCOM, "BCM53570, 1G/2.5G/10G/25G TSN connectivity switch, (product brief)," 2022. [Online]. Available: https://docs.broadcom.com/doc/53570-PB101
[5] NXP, "SJA1105TEL, five-ports AVB and TSN automotive ethernet switch (product data sheet)," 2019. [Online]. Available: https://www.nxp.com.cn/docs/en/data-sheet/SJA1105.pdf
[6] Y. Jinli *et al.*, "TSN-Builder: Enabling rapid customization of resource-efficient switches for time-sensitive networking," in *Proc. 57th ACM/IEEE Des. Automat. Conf.*, 2020, pp. 1–6.
[7] H. Won *et al.*, "A 0.87 W transceiver IC for 100 Gigabit ethernet in 40 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 50, no. 2, pp. 399–413, Feb. 2015.
[8] W. Quanetal, "OpenTSN: An open-source project for time-sensitive networking system development," *CCF Trans. Netw.*, vol. 3, no. 9, pp. 51–65, 2020.
[9] "OpenTSN team, an opensource project to enable TSN research," 2020. [Online]. Available: https://github.com/fast-codesign/OpenTSN2.0
[10] "RISC-V International, RSIC-V specifications," 2016. [Online]. Available: https://riscv.org/technical/specifications/
[11] N. Mckeownetal ., "DpenFlow: Enabling innovation in campus networks," *Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.
[12] Open Networking Foundation, "OpenFlow table type patterns," 2014. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/04/OpenFlow%20Table%20Type%20Patterns%20v1.0.pdf
[13] J. Sanchez-Garrido *et al.*, "Implementation of a time-sensitive networking (TSN) Ethernet bus for microlaunchers," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 57, no. 5, pp. 2743–2758, Oct. 2021.
[14] Q. Yu, H. Wan, X. Zhao, Y. Gao, and M. Gu, "Online scheduling for dynamic VM migration in multicast time-sensitive networks," *IEEE Trans. Ind. Inform.*, vol. 16, no. 6, pp. 3778–3788, Jun. 2020.
[15] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile augmented reality survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.
[16] B. Cizmeci, X. Xu, R. Chaudhari, C. Bachhuber, N. Alt, and E. Steinbach, "A multiplexing scheme for multimodal teleoperation," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 2, pp. 21:1–21:28, May 2017.
[17] *IEC/IEEE TSN Profile for Industrial Automation*, IEC/IEEE Std. 60802 V0.61," Apr. 2018.
[18] *IEEE Standard for Local and Metropolitan Area Networks–Frame Replication and Elimination for Reliability*, IEEE Std. 802.1CB-2017, Oct. 2017. [Online]. Available: : https://standards.ieee.org/standard/802_1CB-2017.html
[19] *IEEE Standard for Local and Metropolitan area Networks–Bridges and Bridged Networks–Amendment 28*, IEEE Std. I802.1Qci-2017, Sep. 2017. [Online]. Available: https://1.ieee802.org/tsn/802–1qci/
[20] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks - Amendment 34: Asynchronous Traffic Shaping*, IEEE Std. 802.1Qcr-2020, Nov. 2020. [Online]. Available: https://standards.ieee.org/standard/802_1Qcr-202
[21] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 26: Frame Preemption*, IEEE Std. 802.1Qbu-2016, Aug. 2016. [Online]. Available: : https://standards.ieee.org/standard/802_1Qbu-2016.html
[22] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 29: Cyclic Queuing and Forwarding*, IEEE Std. 802.1Qch-2017, 2017. [Online]. Available: https://standards.ieee.org/standard/802_1Qbv-2015.html
[23] *IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications*, IEEE Std. 802.1AS-2020, 2020. [Online]. Available: https://www.ieee802.org/1/pages/802.1as.html
[24] *IEEE Standard for Local and Metropolitan Area Network-Forwarding and Queuing Enhancements for Time-Sensitive Streams*, IEEE Std. 802.1Qav-2009, 2009. [Online]. Available: https://standards.ieee.org/ieee/802.1Qav/
[25] J. Yan, W. Quan, X. Jiang, and Z. Sun, "Injection time planning: Making CQF practical in time-sensitive networking," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 616–625.
[26] Y. Huang, S. Wang, T. Huang, B. Wu, Y. Wu, and Y. Liu, "Online routing and scheduling for time-sensitive networks," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst.*, 2021, pp. 272–281, doi: 10.1109/ICDCS51616.2021.00034.
[27] R. S. Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv gate control list synthesis using array theory encoding," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2018, pp. 13–24.
[28] A. Forencich, A. C. Snoeren, G. Porter, and G. Papen, "Corundum: An open-source 100-Gbps NIC," in *Proc. IEEE 28th Annu. Int. Symp. Field-Programmable Custom Comput. Machines*, 2020, pp. 38–46.
[29] J. W. Lockwood *et al.*, "NetFPGA–An open platform for gigabit-rate network switching and routing," in *Proc. IEEE Int. Conf. Microelectronic Syst. Educ.*, 2007, pp. 160–161, doi: 10.1109/MSE.2007.69.
[30] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The P4NetFPGA workflow for line-rate packet processing," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2019, pp. 1–9.
[31] "OpenAvnu-an avnu sponsored repository for time sensitive network (TSN and AVB) technology," 2020. [Online]. Available: https://github.com/Avnu/OpenAvnu
[32] K. Steinhammer and A. Ademaj, "Hardware implementation of the TimeTriggered Ethernet controller," in *Embedded System Design: Topics Techniques and Trends*. Boston, MA, USA: Springer, 2007, pp. 325–338.
[33] T. Fruhwirth, W. Steiner, and B. Stangl, "TTEthernet SW-based end-system for AUTOSAR," in *Proc. 10th IEEE Int. Symp. Ind. Embedded Syst*, 2015, pp. 1–8.
[34] E. Kyriakakis *et al.*, "A time-predictable opensource TTEthernet end-system," *J. Syst. Architecture*, vol. 108, 2020, Art. no. 101744.

Wenwen Fu is currently working towrd the PhD degree with the National University of Defense Technology. His main research interests include time-sensitive networking, time-triggered ethernet, software defined network, and congestion control in data centers.

Wei Quan is an associate professor with the College of Computer Science and Technology, National University of Defense Technology. His main research interests include cover time-sensitive networking, time-triggered ethernet, and computer architecture.

Jinli Yan is currently working toward the PhD degree with the National University of Defense Technology. His main research interests include time-sensitive networking, time-triggered ethernet, software defined network, and network functions virtualization.

Zhigang Sun is a professor with the National University of Defense Technology, chair of OpenTSN open-source project. His main research interests include deterministic ethernet, software defined network, network architecture, and network security.