CCF Transactions on Networking (2020) 3:51–65
https://doi.org/10.1007/s42045-020-00029-8

REGULAR PAPER

OpenTSN: an open-source project for time-sensitive networking system development

Wei Quan[1] · Wenwen Fu[1] · Jinli Yan[1] · Zhigang Sun[1]

Received: 9 February 2020 / Accepted: 13 May 2020 / Published online: 5 August 2020

Abstract
Time-sensitive networking (TSN) is a promising technique in many fields such as industrial automation and autonomous driving. The standardization of TSN has been rapidly improved by the IEEE 802.1 TSN working group. Currently, it has formed a comprehensive standard system with a wide range of choices. However, there is a large gap between TSN standards and application specific TSN systems. Designers need to determine the required TSN standards and standard implementation methods based on the application's transmission performance and reliability requirements. Therefore, an easy-to-use developing platform for rapid TSN system prototyping and evaluation plays a vital role in the application of TSN technologies. This article mainly introduces OpenTSN, an open source project that supports rapid TSN system customization. This project has three features, which are SDN-based TSN network control mechanism, time-sensitive management protocol and time-sensitive switching model, for building an efficient TSN system. OpenTSN opens all the hardware and software source codes so that designers can quickly and flexibly customize the TSN system according to their own needs, maximizing the reuse of existing code and reducing the customization complexity. With this project, two FPGA-based prototyping examples with star and ring topology are presented at the experimental section. The experiment results show that the synchronization precision of the entire testing network is under 32 ns and the transmission performance matches the theory analysis of the testing Cyclic Queue and Forwarding based TSN network.

Keywords Time sensitive networking · Open source · System customization · Switch architecture

1 Introduction

In many distributed hard real-time and safety-critical application domains, such as automotive and industrial control applications, the current proprietary bus-based networking technologies are reaching their limits in supporting the increasing communication bandwidth requirements (Yang et al. 2019). To cope with these emerging requirements, Ethernet has a great potential to replace field buses for the ever-increasing bandwidth and high compatibility. However, the nondeterministic queuing delay and packet loss impedes the switched Ethernet from providing deterministic forwarding service. The deterministic forwarding service has strict requirements on latency, packet loss, and delay variation (jitter) during the packet transmission, which is highly desirable for strict real-time and safety-critical applications (Nasrallah et al. 2019).

To empower standard Ethernet with deterministic capability, Time Sensitive Networking (TSN) is proposed as a new paradigm which introduces new time-based features on Ethernet devices. Currently, TSN Task Group has published comprehensive standards and drafts on time synchronization, flow control, flow management, and flow integrity according to the classification method in Nasrallah et al. (2018). These published and ongoing standards have formed a comprehensive standard system with a wide range of choices for designers to implement a TSN system. TSN has a wide range of application areas, and each TSN application has its specific requirement. The selection and implementation of TSN

✉ Zhigang Sun
sunzhigang@nudt.edu.cn

Wei Quan
w.quan@nudt.edu.cn

Wenwen Fu
fuwenwen16@nudt.edu.cn

Jinli Yan
yanjinli10@nudt.edu.cn

[1] College of Computer, National University of Defense Technology, Deya road No. 109, Changsha, Hunan, China

standards are highly related to its applications. Designers need to determine the required TSN standards and standard implementation methods based on the application's transmission performance and reliability requirements.

Taking the industrial automation as an example, control loops are basic elements for industrial automation systems, including programming logic controller (PLC), sensors and actuators . It is essential for control loops to guarantee the roundtrip latency from control application to device. Accommodating with other unpredictable data in the same network increases the difficulty of deterministic latency for control loops. TSN can make it practical by using a serial of TSN standards like IEEE 802.1 AS to provide a general synchronization mechanism on lay 2 network, IEEE 802.1Qch to provide a micro-second level per-hop latency and end-to-end delay jitter and IEEE 802.1CB to increase the system reliability.

However, there is a large gap between TSN standards and application specific TSN systems. Even though there are several ASIC solutions like Broadcom BCM53 serials for TSN switches. It is still hard to fulfill the gap between the diverse TSN application requirements and TSN standards. As these ASIC solutions suffer from the flexibility problem. Besides that, as there is no ASIC solutions for TSN end-system until now, it is impossible to implement an entire TSN network system by using these ASIC chips. To overcome this problem, TTTech and Intel propose a preliminary idea of using FPGA customization and configurability to develop optimized TSN devices and systems. In this article, we present OpenTSN, an open-source TSN System developing project which enables a fast customization of FPGA-based TSN system to achieve this goal. Our work not only realizes the preliminary idea presented in Time-Sensitive Networking, but also extends it by providing basic design methods for implementing an application-specific TSN system. By using this project, designers can rapidly customize their TSN systems on FPGAs from different level, e.g. system level, device level and module level with maximal reuse of standard modules.

The OpenTSN project contains two hardware components TSNSwitch and TSNNic, and one software component TSNLight. TSNSwitch and TSNNic represent FPGA-based switches and network adapters enabled with deterministic transmission capability. TSNLight is a software TSN network controller that controls the underlying TSN devices in a centralized mode. With these components, designers can build a basic TSN system by customizing the corresponding components according to their application requirements.

Our proposed OpenTSN has three features: (1) a SDN-based TSN network control mechanism based on the centralized control model of 802.1Qcc. With this mechanism, the TSN controller can provide static traffic management for periodic time-sensitive flows and dynamic traffic management for stochastic flows; (2) a time-sensitive management and control protocol designed for layer 2 TSN networks. This protocol is simple and efficient compared to network management protocol like NETCONF (Schonwalder et al. 2010). It takes the advantages of the deterministic transmission ability of a TSN network to achieve a reliable network control and management; (3) a time-sensitive switching model. This model decomposes the processing pipeline into TSN-related and TSN-unrelated modules with a unified communication interface, which makes designers easy to extend new TSN features on a TSN switch. The above features are the implementation foundation of OpenTSN. Currently, a star topology TSN network system and a ring topology TSN network system have been built in OpenTSN to demonstrate the ability of OpenTSN.

The rest of this paper is organized as follows. The motivation is introduced in Sect. 2. The OpenTSN overview is presented in Sect. 3. Section 4 describes the main components in OpenTSN. In Sect. 5, prototyping examples and related experimental results are presented, followed by the related work in 6 and conclusion in Sect. 7.

2 Motivation

TSN has various application areas, such as Audio Video Bridging (AVB Systems), mobile fronthaul networks, industrial automation, automotive networks, utility, etc. The deterministic data transmission requirements may vary widely among these TSN applications in terms of bandwidth, reliability, latency and jitter (i.e. delay bound). For instance, the bandwidth requirements of augmented and virtual reality (AR/VR) applications are thousands of time larger than that of industrial control applications. The reliability of a high-voltage distribution network is 3 orders of magnitude higher than that of a medium-voltage distribution network. With regard to the requirements of latency and jitter, the end-to-end latencies should be on the order of a few microseconds to a few milliseconds for industrial applications (Wollschlaeger et al. 2017). Applications like power grid system have very tight delay bounds, e.g., only a few microseconds, while others have more relaxed delay bounds up to a millisecond (Nasrallah et al. 2018). To accommodate these diverse deterministic transmission requirements, 11 technique standards have been published by the IEEE 802.1 TSN task group (TG) in the last 10 years. The published standards involve the main features of a TSN system including synchronization, latency, reliability and resource management. And the TSN standards will be expanded by many ongoing projects in the near future. With these standards, designers are able to develop their TSN systems using the standard that best suits the application requirements. For example, traffic shaping alone has covered 4 standards, including three published standards IEEE 802.1Qav, IEEE 802.1Qbv, IEEE 802.1Qch and one ongoing project IEEE P802.1Qcr (2020), which provides developers with a variety of choices.

However, there is a large gap between the diverse TSN application requirements and TSN standards, which makes it difficult for designers to customize their TSN system on demand. Even though profile standard[1] has been or is being specified by the 802.1 TSN TG for some of the targeted application areas to describe which TSN standards to use and how. The TSN system implementation details such as network topology, device buffer and queue size, time synchronization precision and so on need to be determined by designers. The selected TSN standards should be carefully customized according to the application requirements as the universal solution provided in a standard may not the best solution for the targeted problem.

To fulfill the gap between the diverse TSN application requirements and TSN standards, several ASIC-based solutions have been proposed. As TSN pioneers, Broadcom, Marvell and NXP, have released a series of TSN switching chips supporting typical TSN standards. However, the biggest problem of ASIC-based solutions is flexibility. These commercial TSN chips are normally developed by a Bottom-up method without considering specific application requirements. Therefore, the resource partitioning for tables, queues and buffers in these chips is fixed. And the fixed resource partitioning may be over-provisioning or under-provisioning for the target TSN application. Any feature adjustment like the extension for a new TSN standard means a redesign of the chip. To overcome this problem, TTTech collaborated with Intel and published a white paper about how FPGA customization and configurability can be used to develop optimized TSN devices and systems for your application. They conclude four benefits, which are re-programmability, workload consolidation and acceleration, I/O flexibility, functional safety and security, for implementing TSN on FPGAs. However, this study only provides a preliminary idea without implementation details. Therefore, flexible solutions to fulfill the gap between the diverse TSN application requirements and TSN standards are urgently needed to make TSN technology practical.

3 OpenTSN overview

OpenTSN aims to provide an open-source TSN system developing project that can help developers rapidly customize their TSN systems on FPGAs from different level, e.g. system level, device level and module level. This open-source project provides not only the source code of each components of a TSN system, but also the design documents for each components. These documents introduce the architecture, interfaces, workflow, design parameters of modules and so on. Designers can develop new modules using the defined interfaces and assemble them with modules provided in OpenTSN to achieve a module level system development. As modules developed in OpenTSN provide several parameters for customization, such as the synchronization frequency and the master/slave configuration of the time synchronization module, designers can easily do a device level reconfiguration on the developed components in OpenTSN. A system level TSN network development is achieved by selecting the proper number of corresponding OpenTSN components according to the specific system level requirements like topology.

To give an overview, the design principles, architecture and key features of OpenTSN will be introduced in this section.

3.1 Design principles of OpenTSN

Before the development of OpenTSN, several design principles had been established for a better use of this open-source project by TSN system designers. These principles mainly concentrate on the simplicity, flexibility and extensibility of a TSN system, which are explained as following.

3.1.1 An overall TSN system solution

Different with the ASIC solutions targeted for TSN switches, OpenTSN provides FPGA-based hardware solutions for both TSN switches (TSNSwitch) and adapters (TSNNic), and a TSN network controller (TSNLight) for the control and management of a TSN network built in OpenTSN. Besides that, an experimental testbed is built in OpenTSN based on a programmable network platform embedded with a xilinx FPGA. With this experimental testbed, designers can evaluate their developed TSN network system. OpenTSN also provides a few demonstrations of typical TSN applications to show how a TSN network system can be built. OpenTSN tries to make the customization of a TSN network system as simple as possible.

3.1.2 Modular design supporting flexible customization and extension

OpenTSN uses a modular design for both the hardware devices and software controller to support the flexibility and extensibility of system customization. For the implementation of hardware devices, the platform-unrelated logic takes a modular design. It contains TSN-related modules

[1] A TSN profile selects features, options, configurations, defaults, protocols, and procedures of bridges, end stations, and LANs to build a bridged network for the given TSN application.

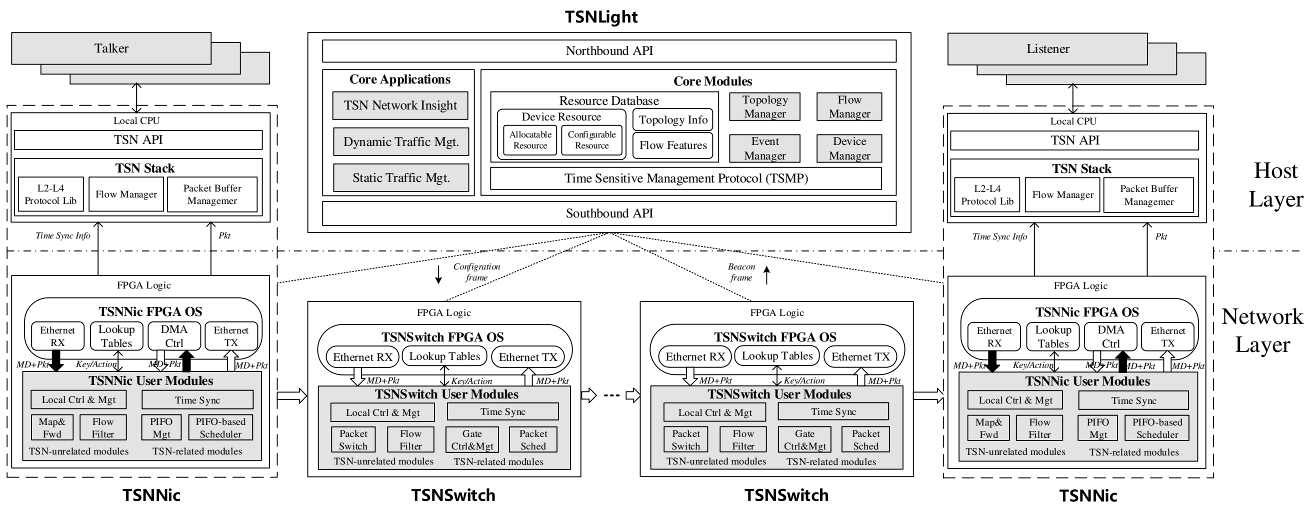


Fig. 1 OpenTSN architecture

and TSN-unrelated modules. These modules constitute a complete TSN packet processing pipeline with the open data communication interface. Therefore, designers can extend their own modules or assemble the existing modules to achieve their specific functions. For the TSN controller implementation, core software modules are provided to cooperatively provide basic services for high-level applications. Based on these core modules, different application can be developed for specific application requirements. Users can either add their specific functions in the core modules or develop new applications based on the existing core modules.

3.1.3 Platform and application independent system design

To increase the flexibility for cross-platform migration, OpenTSN concentrates on the core techniques of TSN. For TSNSwitch and TSNNic, OpenTSN divides the hardware implementation into two parts, platform-related and platform-unrelated logic. The platform-related logic takes charge of the input/output processing like Ethernet and PCIE for the platform-unrelated logic. Between the platform-related and platform-unrelated logic, OpenTSN uses open interfaces for data and control information exchange which will be introduced at the demonstration section. Under this partition, a cross-platform migration can be achieved by only replacing the platform-related logic under the defined interfaces. Besides that, OpenTSN adopts an application-independent system design by and providing parameterized hardware modules which are not limited to specific applications and an application unrelated TSN controller which seperates the implemenation of a TSN network and the application-related management of this network.

3.2 OpenTSN architecture

OpenTSN mainly includes three basic components, i.e. TSNSwitch, TSNNic and TSNLight as illustrated in Fig. 1, for building a TSN system. TSNSwitch and TSNNic represent switches for switching data between ethernet ports and network adapters for adapting data between local CPU and the network. Both of them are enabled with deterministic transmission capability. TSNSwitch is composed of pure FPGA logic where TSNNic includes both FPGA logic and end-system supports on the local CPU. OpenTSN provides general TSN-unrelated modules, e.g. local management, packet parsing, forwarding and switching, and TSN related modules like time synchronization, traffic shaping for implementing the hardware of TSNSwitch and TSNNic. To facilitate module assembly, a unified communication interface is defined between modules for data delivering by adding a MetaData (MD) to each packet. With these user modules, designers can rapidly build a deterministic packet transmission pipeline for their FPGA-based TSN devices. Around this pipeline, a platform-related logic, namely FPGA OS, is provided for IO processing.

Besides these two hardware components, OpenTSN provides TSNLight to control the underlying TSN devices. TSNLight is a software TSN network controller that enables a centralized controlling of the target TSN network. It provides core modules like events, topology and device management. Based on these fundamental modules, TSNLight can handle different flow transmission requirements with different core applications and monitor the state of the running network. Between TSNLight and underlying TSN devices, network management and control information is carried by a time sensitive management protocol (TSMP). TSMP is designed for OpenTSN to control FPGA-based

TSN devices. The details of each components in OpenTSN will be explained in the following section.

Currently, OpenTSN supports IEEE 802.1AS, IEEE 802.1 Qci, IEEE 802.1 Qav, IEEE 802.1 Qbv, IEEE 802.1Qch for implementing FPGA-based TSN devices and IEEE 802.1Qcc in the TSN controller. These standards are provided as modules with the unified interface for reusage. The grey parts in Fig. 1 can be extended or changed by TSN system designers. For example, new standards can be extended by designers to rebuild a new FPGA-based TSN device. To improve the efficiency of the overall network resources, new flow scheduling algorithms can be added into TSNLight and new applications can be developed using the northbound interface provided by TSNLight.

3.3 Key features of OpenTSN

This subsection mainly introduces the key features of OpenTSN. The implementation details of OpenTSN will be introduced in the next section.

3.3.1 SDN-based TSN network control mechanism

OpenTSN uses a centralized control model proposed in IEEE 802.1Qcc. In this centralized control model, network devices including TSNSwitch and TSNNic are configured by the controller as shown in Fig. 1. For user management, information of talkers and listeners is statically described using configuration files according to the definition of user/network configuration information in IEEE 802.1Qcc. With this configuration files and the statically abstracted underlying network resources, the TSN network controller TSNLight can analyze the corresponding configuration parameters and configure the target network. Besides such static network configuration, it also provides dynamic network management similar to a SDN controller like Floodlight (Floodlight controller 2017). With such a SDN-based control mechanism, TSNLight can dynamically allocate network resource for flows that are unknown to the network in advance when a table-miss event is triggered.

Under the above mentioned features, the TSN network controller is able to manage different types of network flows. We divide flows into three categories, i.e. time sensitive (TS) flows, rate-constrained (RC) flows and best-effort (BE) flows. TS flows represent flows requiring deterministic transmission. Packets of this type of flow are generated periodically and must arrive at the destination within the deadline with ultra-low jitter and packet loss. RC flows are flows with a preserved bandwidth allocation and BE flows only need a best-effort service provided by the TSN network. Under this definition, the features of TS and RC flows are predetermined while the BE flows are generated dynamically. To facilitate the debugging and diagnosis of a customized TSN network, TSNLight provides the ability of collecting network statistics periodically and sampling flow packets on specific network devices through the control channel.

3.3.2 Time sensitive management protocol

OpenTSN uses TSMP (Time Sensitive Management Protocol) as the communication protocol between the TSN controller and devices. TSMP works on the second layer of the OSI network model as illustrated in the grey part of Fig. 2. It is encapsulated in a PTP/PCF frame if IEEE 802.1AS (IEEE 802.1AS Standard 2015)/AS6802 (AS6802 Standard 2016) is used for time synchronization in the underlying TSN network. This protocol contains two main messages, Configuration (C) and Beacon (B) which are distinguished by the message type section in a PTP/PCF header. Message C and B represent controlling information encapsulated in configuration packet from controller to device and state and other information encapsulated in beacon packet from device to controller. In each message class, there are multiple subclass messages defined in the TSMP header. In message C, the sub-class messages include configurations for tables and control registers like *switchtable_update*, *synfreq_update*, etc. And message B includes *state_report*, *table_miss*, *packet_sampling*, etc. *state_report* reports all states of tables and statistic registers. *packet_sampling* simples the packets passing though the device. These two TSMP messages are triggered periodically similar to the Beacon messages in IEEE 802.11. *table_miss* contains the packet triggered this table miss event and the corresponding device information.

TSMP provides a reliable transmission guarantee for network management and control information. There are two methods to achieve this goal. One of the methods is mapping TSMP packets (besides *table_miss*) as time-sensitive packets supported in the underlying TSN network. As the TSN network provides a deterministic transmission service for time sensitive flows. The information transmitted between the controller and devices by TSMP can be highly reliable. The other method is using a conformation reply for each configuration. However, this method will reduce the reliability of TSMP compared to the first method.

As can be seen from Fig. 2, compared to the recommended network management protocols like NETCONF and RESTCONF in IEEE 802.1Qcc (IEEE 802.1Qcc Standard 2018), TSMP is more simple and efficient. Protocols like NETCONF and RESTCONF use application-level security such as SSH and HTTP respectively to achieve transmission security. And the reliability is guaranteed by TCP protocol. This means the controller host and network devices need to support a standard TCP/IP stack and complicate high-level protocol parser. However, considering some closed systems with pure FPGA-based

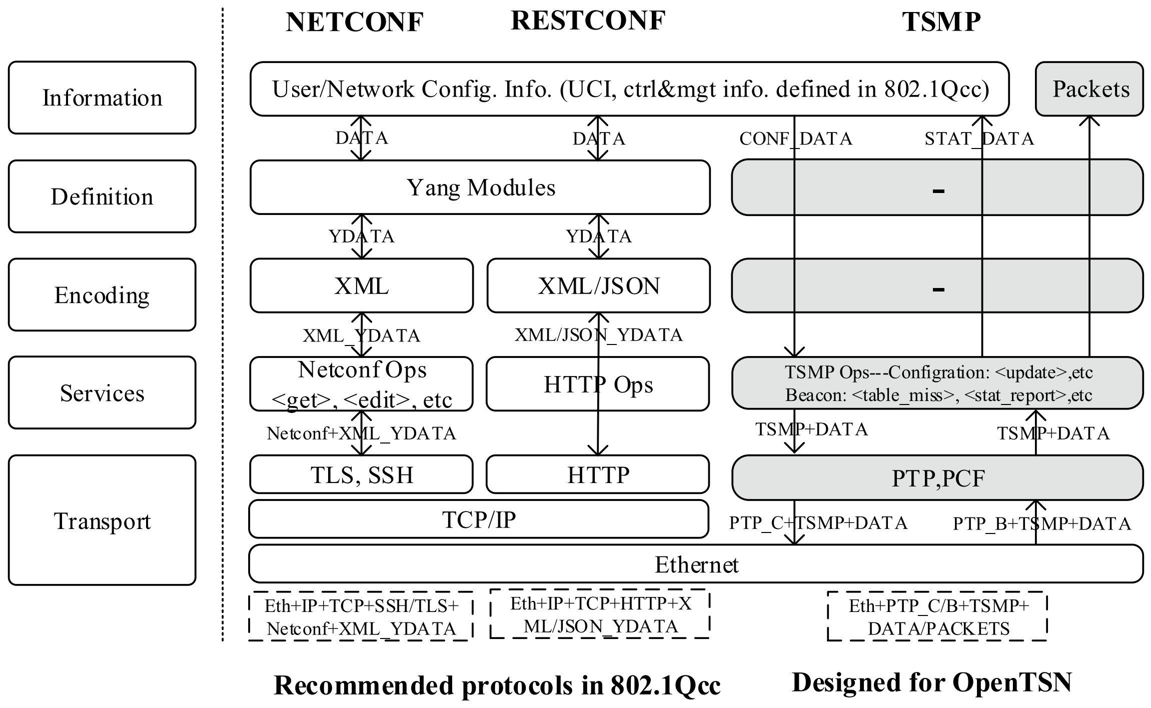

NETCONF
RESTCONF
TSMP
Information
Definition
Encoding
Services
Transport
User/Network Config. Info. (UCI, ctrl&mgt info. defined in 802.1Qcc)
Packets
DATA
DATA
CONF_DATA
STAT_DATA
Yang Modules
-
YDATA
YDATA
XML
XML/JSON
-
XML_YDATA
XML/JSON_YDATA
Netconf Ops <get>, <edit>, etc
HTTP Ops
TSMP Ops---Configration: <update>,etc
Beacon: <table_miss>, <stat_report>,etc
Netconf+XML_YDATA
TSMP+DATA
TSMP+DATA
TLS, SSH
HTTP
PTP,PCF
TCP/IP
PTP_C+TSMP+DATA
PTP_B+TSMP+DATA
Ethernet
Eth+IP+TCP+SSH/TLS+ Netconf+XML_YDATA
Eth+IP+TCP+HTTP+XML/JSON_YDATA
Eth+PTP_C/B+TSMP+ DATA/PACKETS
Recommended protocols in 802.1Qcc
Designed for OpenTSN

Fig. 2 The comparison of TSMP and protocols recommended in 802.1Qcc

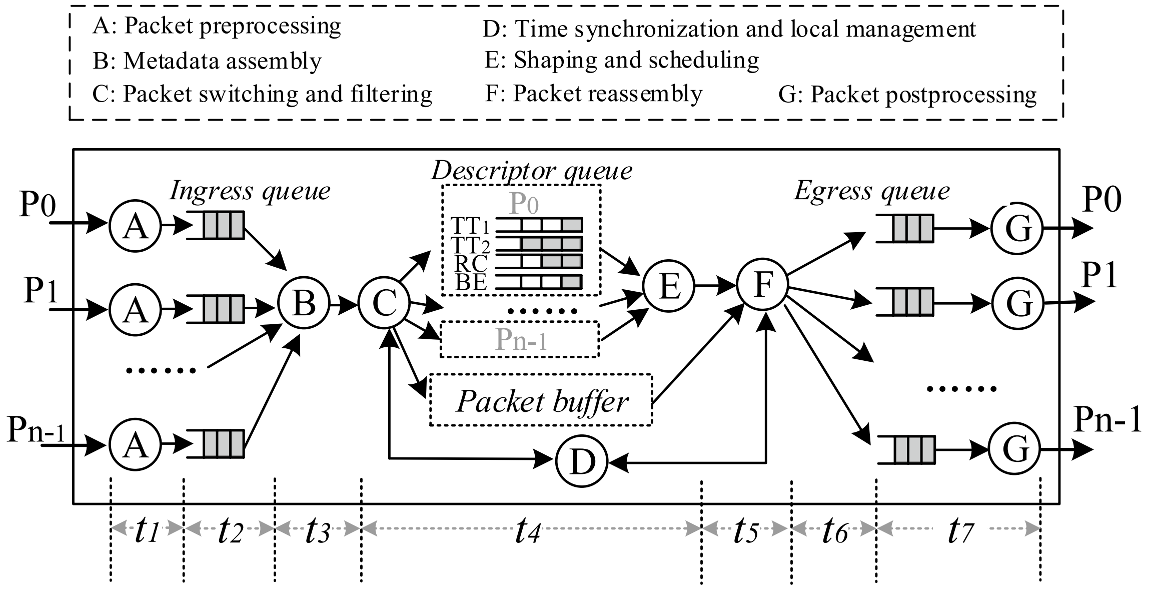


Fig. 3 A time sensitive switching model for TSN switches

TSN network devices, it is hard to support such complicate network management protocols. And the security problem can be ignored in such system. TSMP, on the other hand, can be easily implemented on FPGA. If TSN devices in the target network contain CPUs, these complicated protocols can be extended in OpenTSN.

3.3.3 Time sensitive switching model

The time sensitive switching model is proposed to support deterministic transmission for TSN switches, which contains processing logics (A–G) and storages (Ingress queue, Describer queue, Packet buffer and Egress queue), as shown in Fig. 3. Logic A is in charge of packet preprocessing like

Table 1 Time analysis in TSN switching model

Name	Description	Time consumption FPGA impl. under 125 MHZ
t1	Time for checking CRC, timestamping and deserializing	0.5 µs
t2	Ingress queue delay (upbound)	4.5 µs
t3	Time for filtering, metering and parsing frames	0.2 µs
t4	Including looking up table, writing frames into packet buffer, writing describers into corresponding queues and scheduling describers	St
t5	Submitting metadata	0.2 µs
t6	Time for parsing metadata, shaper, reading frames from packet buffer	0.4 µs
t7	Time for refreshing transparent clock, CRC calculation and deserializing	1 µs

CRC checking, timestamping and so on. As an aggregator, logic B schedules data from multiple queues with a round-robin algorithm and adds a metadata before each packet. Logic C forwards time synchronization and configuration packets to Logic D for global network time synchronization processing and local control/management of modules respectively. And it forwards other packets (TS, RC, BE packets) to packet buffers with their descriptors sent to the corresponding queues. Meanwhile, it controls the enqueue gates and executes flow metering. Logic E controls the dequeue gates and schedules the descriptor from all queues based on the scheduling algorithm. Logic F reads the corresponding packet of descriptor from packet buffers and forwards the scheduled packets to egress queues. Logic G mainly handles CRC calculations, queue rate regulations and transparent clock processing.

In the above processes, traffic shaping and scheduling are critical to achieve deterministic forwarding for TS traffic. Traffic shaping determines at which time slot a packet can enter and leave the corresponding queue. Traffic scheduling determines the output order of packets from different queues in the same scheduling slot. More specifically, it is essential to configure a suitable slot value (d) and storage size. On one hand, the slot value is not only related to the end-to-end transmission latency, which must be abided for meeting the real-time application requirement, but also determines the jitter range. On the other hand, it is necessary to save memory resource by configuring suitable parameters as the memory resource on chip is significantly precious. The parameters of d and storage size are related to the delay and jitter requirements of applications. In the CQF model, there is a clear relationship between d and the transmission jitter (the transmission jitter is 2*d). And the storage size is related to the time period d as will be explained later.

The value of d contains static protection time (Pt) and dynamic time of scheduling (St). Pt contains t_1 to t_3 and t_5 to t_7 as shown in Fig. 3, which can be acquired by multiple experiments in a prototyping system. Table 1 shows the time analysis of each process in our star topology TSN prototyping system of OpenTSN. While St is the dynamic part of d represented by t_4 fluctuating within fixed thresholds, which can be calculated as St=d-Pt. The packet buffer size should be large enough to accommodate all the packets during the time period St. And the depth of descriptor queues are equal to the maximum number of frames that the packet buffer is able to accommodate.

4 Main components in OpenTSN

In current OpenTSN project, there are two hardware components TSNSwitch and TSNNic, and one software component TSNLight as mentioned in Sect. 3.2. The implementation details of these components will be introduced in this section.

4.1 TSNSwitch

In a standard Ethernet switch, ingress processing, switching and egress processing are three common processing steps. These three steps are also required in a TSN switch. Different with a standard switch, a TSN switch needs to add TSN-related features into these steps. In order to achieve the ability of deterministic transmission, key functions like time synchronization, time-based shaping are required on a TSN switch. Time synchronization provides a global uniform time among all TSN switches. Time-based shaping controls the outport enqueue and dequeue time of each packet. These TSN functions should be placed in a proper position in the processing pipeline. For example, time-based shaping should be placed in the egress processing pipeline. Functions like time synchronization should be placed in the ingress processing pipeline as a switching function is required for synchronization packets. In OpenTSN, TSNSwitch is built according to the time sensitive switching model as described in Sect. 3.3.3. Under this model, designers can extend the required TSN standards in the processing pipeline of TSNSwitch.

For implementing a typical TSNSwitch, the packet processing pipeline requires seven stages accroding to the swithcing model. The first and last stage of this pipeline are implemented in the FPGA OS. And for the implementation of other stages, OpenTSN provides six loosely-coupled modules including three TSN-related and three TSN-unrelated modules as illustrated in Fig. 1. The FPGA OS contains logic for Ethernet preprocessing/postprocessing (CRC, timestamping, etc.) and switching tables with an open lookup interface. The three TSN-unrelated modules are explained as following. *Local Ctrl&Mgt* is the local manager of TSNSwitch which is in charge of network management protocol processing, configration and periodic state reporting of local modules. *Packet Switch* is used to lookup the output information from the switching table for each packet with the specified packet fields.[2] *Flow Filter* classifies the input packets with the user-defined packet features and maps them onto different flows for the following traffic policing. About the TSN-related modules, *Time Sync* synchronizes the time among the global TSN network. *Gate Ctrl&Mgt* provides enqueue and dequeue control and traditional queue management under the selected TSN shaping standard. Each queue executes open/close operations at a specified time according to the pre-determined configurations. *Packet Sched* is used to select the packets in which queue should be sent out. And there are shapers limiting the bandwidth of specific queues.

Between modules of TSNSwitch, Metadata as mentioned in Sect. 3.2 is defined to delivery shared information. The MD is composed of *Buf ID*, *Outport*, *Meter ID*, *Queue ID* and *Pkt Len*. The *Buf ID* is used to indicate the index of the address where the packet is stored in the packet buffer pool. *Outport* is used to index the corresponding resource that the packet belongs to. *Meter ID* and *Queue ID* represent the index of a meter and queue that the packet traverses, respectively. With this definition, modules are decoupled so that they can be reused without further modifications or extended with new TSN standards. OpenTSN provides multiple selections for the TSN-related modules to satisfy different TSN transmission requirements. Taking the shaping module for example, IEEE 802.1 Qav, IEEE 802.1 Qbv and IEEE 802.1 Qch have implemented. And many ongoing modules like AS6802 (AS6802 Standard 2016) for time synchronization will be provided in the near future. These modules are built with some static configuration parameters like synchronization precession for *Time Sync* and time slot size for *Gate Ctrl&Mgt*. Designers need to determine these application-specific static parameters for a TSN system customization. When the application scenario changes, users can rapidly rebuild a TSNSwitch by substitute corresponding modules or even only by regulating the related parameters. Therefore, the development effort is greatly reduced.

[2] According to the packet information of MAC header and/or other information such as VLAN header, packets will get the corresponding outport ID, queue ID, meter ID and so on from the switching table for egress processing.

4.2 TSNNic

In OpenTSN, TSNNic works as a network adapters connecting the TSN networks and users and guaranteeing the deterministic communication between them. In this section, the architecture and core data structure would be described in detail.

4.2.1 Architecture of TSNNic

The TSNNic architecture is shown at the left and right side of Fig. 1. Similar to TSNSwitch, the *Network layer* consits of TSNNic FPGA OS and packet processing pipeline in TSNNic User Modules. Compared to TSNSwitch, TSNNic FPGA OS adds *DMA Ctrl* module to transfer packet data between CPU memory and FPGA memory. With regard to TSNNic pipeline, the main difference between TSNSwitch and TSNNic is the TSN-related modules including *PIFO Mgt.* and *PIFO-based Scheduler*. These two modules are used to guarantee that each TSN packet is transmitted at the configured time slot. *PIFO Mgt.* adopts the the Push-In First-Out Queue Structure and First-in First-Out (FIFO) structure to hold the packet descriptors. *PIFO-based Scheduler* selects packets from the PIFO and FIFO structures according to the current network time.

In the *Host Layer*, TSNNic consists of *TSN Stack* and *TSN API*. *TSN Stack* mainly contains three following modules. *Packet Buffer Manager* module allocates dedicated packet buffers from the CPU memory for TS/RC/BE flows. *Flow Manager* stores the registered flow features and submits them to TSNLight for getting scheduling information. *L2–L4 protocol Lib* module provide packet encapsulation and decapsulation services. Besides, the TSN stack synchronizes the CPU time with the underlying hardware logic periodically, which provides precise time for upper application.

The TSN API provides two kinds of APIs for the development of real-time applications. First, the register/unregister APIs are used to register/unregister the flow features during the initialization phase, including the source host, destination host, period, deadline, packet length. Second, the Socket-like APIs provide easy-to-use interfaces for sending and receiving the UDP/IP/raw MAC packets without considering the control of time.

4.2.2 Core data structure and workflow

The design of core data structure in TSNNic plays a crucial role in guarantee the forwarding determinism of TSN flows and support different types of flows synchronously. The core data structures are depicted in Fig. 4.

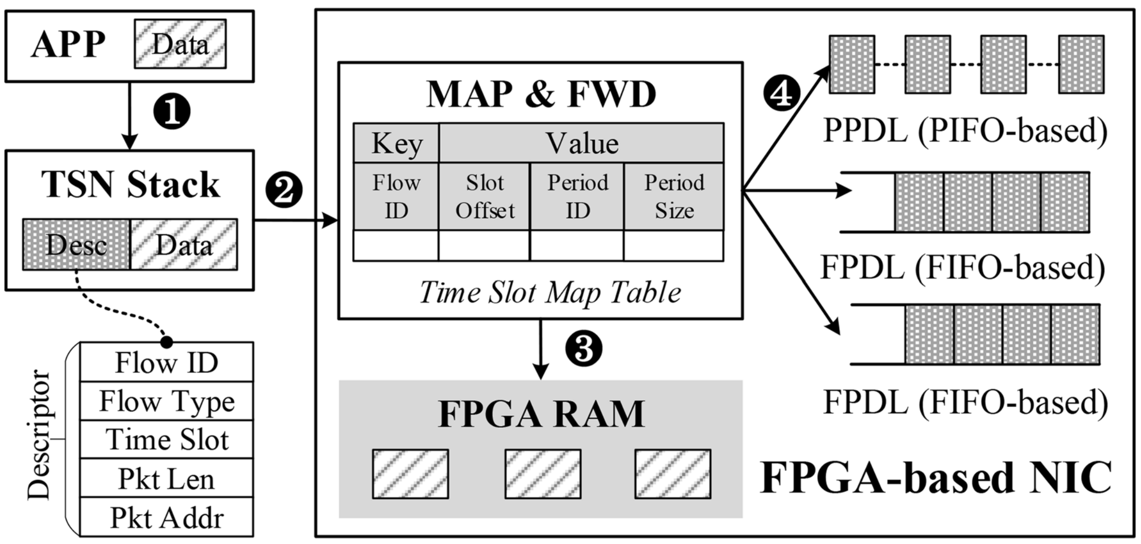


Fig. 4 The core data structure and workflow in TSNNic

(1) Packet descriptor. The descriptor is generated and attached before the packet data by TSN stack. The fileds of descriptor includes *Flow ID*, *Flow Type*, *Time Slot*, *Pkt Len* and *Pkt Addr*. *Flow ID* and *Flow Type* indicates which flow and flow type that the current packet belongs to. *Time Slot* is the specified sending time of this packet. *Pkt len* and *Pkt Addr* is the length and address in FPGA RAM of this packet.

(2) PIFO-based descriptor list (PPDL) and FIFO-based descriptor list (FPDL). PPDL uses the Push-In First-Out Queue (PIFO) (Sivaraman et al. 2016) to organize descriptors of TSN packets while FPDL uses the First-in First-out structure to store packet descriptors. PPDL provides a flexible sorting of packet descriptors according to the sending time of each packet. The packet descriptor is inserted into the FPDL directly without sorting.

(3) Time slot map table. The time slot map table is built to compute the sending time of each TSN packet. The key field is *Flow ID* while the value field contains *Slot Offset*, *Period ID* and *Period Size*. The *Slot Offset* indicates the time slot relative to the start of each period. The *Period ID* is the index of period that the current TSN flow stays and *Period Size* indicates the time interval taken to produce the TSN packets in a round. These value fileds are used to compute the absolute sending time of each TSN packet. *Period ID* value would be updated when the current packet is sent out.

The TX workflow of TSNNic is described as follows. First, the user app generates the application data and submits it into the TSN stack. Second, The TSN stack generates the packet descriptor and attaches it to the packet data. Third, when the *MAP & FWD* module receives packet, it allocates a packet buffer to hold the packet data and writes the buffer address into the packet descriptor. Finally, the *MAP & FWD* module identifies which flow type the current packet belongs to. For TSN packets, it computes the sending time slot with the Time Slot Map Table, followed by enqueuing the packet descriptor into PPDL. While for RC/BE packets, it directly put the descriptor into the corresponding FPDL.

The RX data flow of TSNNic is very simple. When the TSNNic receives packets from the ethernet port, it performs flow filtering to verify the validity of packet. Then, it transfers the packet data into the CPU RAM with *DMA Ctrl* module. Finally, the user application would access this packet at the specified time.

4.3 TSNLight

TSNLight is a centralized network controller which provides a platform for developers to implement various management applications. In TSN networks, how to map the upper application flows onto the underlying TSN-related resources both temporally and spatially is very critical to guarantee the quality of service (QoS). Thus, a global resource abstraction is the foundation for developers to design control applications. TSNLight provides such functions to fully explore the ability of a customized TSN network. And the architecture of TSNLight is depicted in Fig. 1. The TSNLight controller consists of core applitions and core modules. The core modules provide a resource database and related modules to manage these resources. The core applications includes two traffic management applications to allocate the underlying resources for static flows and dynamic flows respectively, and a network monitoring application to analysis the network states like synchronization precision, flow latency and jitter based on the statistics collected from the network. The core applications interact with the core modules with function calls. TSNLight configures the resource and collect the state information of TSN switches and TSN end systems via the southbound API. Besides, a northbound API is provided by TSNLight for high-level applications to control and manage the TSN network.

The *Core Modules* includes the essential functions of TSNLight. *Resource Database* provides a core resource view for the upper scheduling algorithms. There are three types of resource including the topology, flow feature and device resource. *Topology Info* describes all nodes, including switches, end systems, and their connectivity. *Flow Feature* presents the basic attributes of time-sensitive flows, such as source, destination, period, etc. The *Device Resource* is divided into configurable resource and allocatable resource. The configurable resource denotes the configuration information that will be installed into the data plane, including the time slot size, gate control lists. The allocatable resource shows the state of queue resources identified by space (switch and port) and time (time slot). *Flow Manager* manages the features of time-sensitive flows, rate-constraint flows and best-effort flows from the centralized user configuration application. And it provides function interface for the scheduling applications. *Topology Manager* stores the topology info into the resource database. Besides, it periodically checks the state of every link and discovers new nodes. *Device Manager* configures the underlying TSN switches and end system according to the current configurable resources. *Event Manager* classifies the external packets from data plane into different events. The core applications register the required events and the corresponding event handlers. When the event manager receives the registered events, it would execute the related event callbacks. TSMP Protocol provides the encapsulation and decapsulation of the control packets as introduced in Sect. 3.3.2.

In TSNLight, the resource scheduling algorithm is the key to improve the TSN network utilization. Therefore, the design of the scheduling algorithm is highly depend on the TSN standards especially the shaping standard implemented in the target TSN network. Currently, a static resource scheduling algorithm has been proposed in OpenTSN to support a Cyclic Queuing and Forwarding (CQF) (IEEE 802.1Qch Standard 2017) based TSNSwitch and TSNNic. A CQF-based TSNSwitch in OpenTSN only contains 4 queues per port, two for time-sensitve flows, one for rate-constrained flows and one for best-effort flows. And the gate control mechanism of CQF only distinguish odd/even time slots. These implementations reduce the resource scheduling complexity. As page limitations, interested readers can refer to Yan et al. (2020) for further details of this algorithm.

5 Demonstration examples

5.1 Fast prototyping in OpenTSN

To customize a TSN system in OpenTSN, designers need to determine four key properties according to the application requirements. These properties include: (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 of TSN devices would be affected. (2) Flow features. It includes the number of flows and their performance requirements (latency, jitter, bandwidth, packet loss, etc). These influence the shaping standard selection and the size of tables, per-queue depth and packet buffer size of TSN devices. (3) Synchronization precision. It determines the selection of synchronization standards and the implementation platforms (software/hardware), which could affect memory, computing and bandwidth resources. (4) Reliability. It determines the synchronization algorithms (gPTP(IEEE 802.1AS Standard 2015)/AS6802(AS6802 Standard 2016)) and reliability-related standards like frame replication and elimination (IEEE 802.1CB Standard 2017). These application requirements affect the selection of function modules and the resource specification of each module.

OpenTSN can use different FPGA platforms for implementing a TSN prototype by replacing the platform-related code. TSNLight is a software controller which can be executed on a standard Linux system. After the above properties of the target TSN system are determined, FPGA bit files of TSNSwitch and TSNNic can be built by assembling the corresponding hardware modules and downloaded onto the target platform. Related to the time-based shaping standard selected for TSNSwitch and TSNNic, different traffic planning and scheduling algorithms can be adopted in TSNLight's core applications to generate a customized TSN controller. Beside system customization, OpenTSN also supports new standard extension by developing new hardware modules and software modules/applications according to the open interfaces listed in Table 2.[3] For demonstration purpose, a star topology intra-satellite TSN network system and a ring topology industrial control TSN network system have been built in OpenTSN, which will be introduced in details in the next subsection. The currently implemented and ongoing core modules of OpenTSN are shown in Table 3. Notice that, these implemented FPGA modules do not have a one-to-one relationship with the logic modules shown in Fig. 1. The configurations to these modules in a customized

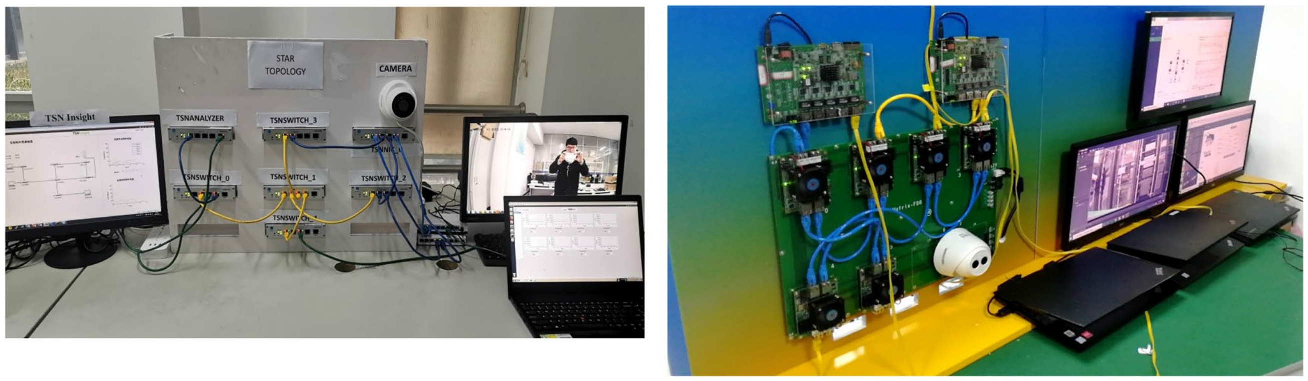
[3] The detailed definition of these interfaces can be found in the design documents in OpenTSN git repository. https://gitee.com/opentsn/openTSN/tree/master.

Table 2 Open interfaces in OpenTSN

Components	Interface	Usage	State
TSNSwitch/Nic	Data transmission	This interface contains five hardware signals for packet transmission between modules	Done
	Metadata	A data structure defined for Control information exchange, embedded in the interface of data transmission	Done
	Lookup table	A lookup interface defined for the forwarding and switching Module, containing Key and Result	Ongoing
TSNLight	Southbound API	An open interface defined for Information exchange between the controller and devices	Done
	Northbound API	An open interface defined for Information exchange between the controller and applications	Ongoing

Table 3 Open modules in OpenTSN

Components	Module	Usage	State
TSNSwitch/Nic	PTP	Time synchronization	Done
	AS6802	Time synchronization	Ongoing
	LCM	Local control and management	Done
	ESW	Ethernet switching	Done
	EMF	Ethernet mapping and forwarding	Ongoing
	EOS-Qav/Qbv/Qch	Qav/Qbv/Qch based filtering, queuing, scheduling	Done
	EOS-PIFO	PIFO based filtering, queuing, scheduling	Ongoing
TSNLight	FM/TM/DM/EM	Flow/topology/device/event management module	Done
	TSMP	TSMP protocol lib	Done
	TSN_Insight	Network monitoring app.	Done
	TSN_STM	Static traffic management app.	Done
	TSN_DTM	Dynamic traffic management app.	Ongoing



(a) A star topology TSN network **(b)** A ring topology TSN network

Fig. 5 OpenTSN demonstrations

TSN system can be found in details in the user manuals in OpenTSN repository.

5.2 Prototyping examples

5.2.1 Typical demonstrations

In this section, we mainly introduce the two typical demonstration examples of OpenTSN as shown in Fig. 5. As the selected demonstration examples given in OpenTSN have different requirements in topology, flow features and reliability. Two obviously different TSN systems need to be customized. According to the corresponding requirement, the prototyping systems are built by assembling the corresponding modules with proper parameters in OpenTSN.

The left side of Fig. 5 shows a star topology TSN network which is normally used as industrial control network, intra-satellite network, etc. In this TSN network, the TSNSwitch and TSNNic are implemented based on Xilinx Zynq 7020 FPGA SoC. For the TSNSwitch, all the processing modules are running on FPGA. With respect to the TSNNic, FPGA is used to implement the time synchronization and time-based packet receiving/sending and the TSN stack and applications run on CPU. The star topology contains five TSNSwitches in total and the core switch has four child nodes. The right side of Fig. 5 shows a typical ring topology TSN network for Ethernet Train Backbone (ETB). Different with the star topology TSN network, the TSNSwitches in this ring topology TSN network are implemented on Altera Arria 10 based FPGA platform with two ports supporting dual-direction deterministic transmission and two general Ethernet ports. For both prototyping system, a packet generator is running on a TSNNic to generate three types of flows (time-sensitive flow, rate-constrained flow and best-effort flow as mentioned before). Each type of flow is generated under the configured packet header and features like bandwidth, period (for time-sensitive flow only), etc. The TSN analyzer in the star topology TSN network is used for experimental supports which adds a timestamp for all TSMP Beacon packets periodically sent from the underlying devices for further analysis by TSNLight. A web camera is applied to the TSN system with a terminal for real-time video display. All the devices are connected with 1Gbps Ethernet cables.

5.2.2 Experimental setups and results

As the above introduced demonstration systems gives similar experimental results with regard to the metrics of synchronization precision and transmission performance. In this article, we select the star topology TSN network to do our experiments. This prototyping system adopts 1588 PTP synchronization protocol and Cyclic Queuing and Forwarding model (CQF) as the core features implemented in FPGA. In our experiments, the features of TS flows that we generate are guided with the IEC 60802 standard. This standard describes the typical flow features in the production cell and line. Here we generate 1024 TS flows and the period of each TS flow is 10 ms. The packet size of each flow is between 64 and 1500 B. In the star topology, the number of hops that all the packets traverse between any end systems is 3.

The experiments include two main parts, synchronization precision and transmission performance. For the first experiment, we test the precision of pure hardware PTP protocol in our testbed. The *TSNSWITCH_0* is selected as the master node and other switches are slave nodes. The period of synchronization is set to 1 ms. The offset of time synchronization between the slave nodes and the master node is shown in Fig. 6a. This test is executed for 200 rounds. The maximal difference is less than 32 ns and most of the results are between 0 and 16 ns. Thus, it demonstrates that the implementation of PTP on our platform is effective. In the second experiment, the transmission performance (latency, jitter and packet loss) is evaluated under different packet size, time slot size and background flows. Since the priority of TS flows is the highest and other flows cannot preempt the bandwidth, the packet loss in all the experiments is 0.

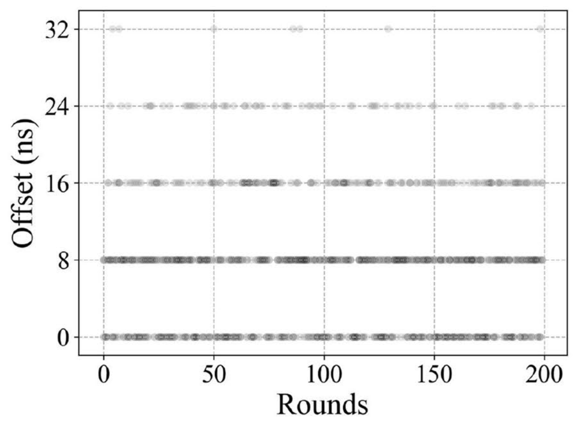
The theoretical latency range of CQF model is between (hop - 1)*slot and (hop + 1)*slot. Thus, the transmission delay is positively related to the number of hops and time slot size. Here we use standard deviation of latency to describe the jitter, which is only positively related to the time slot size. As shown in Fig. 6b, the latency increases slightly as the packet size increases. The reason is that the time for outputting the packet is positively correlated with the packet size. The second case shows the change of latency under different time slot size in Fig. 6c. The average latency and jitter are increased manyfold according to the upper and lower bound of CQF above. 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. The results show that there is no affection on the latency and jitter of critical TS flows in Fig. 6d.

In summary, the tested prototype built in OpenTSN not only can guarantee high precision of time synchronization, but also provide deterministic transmission performance that follows the theoretical results.

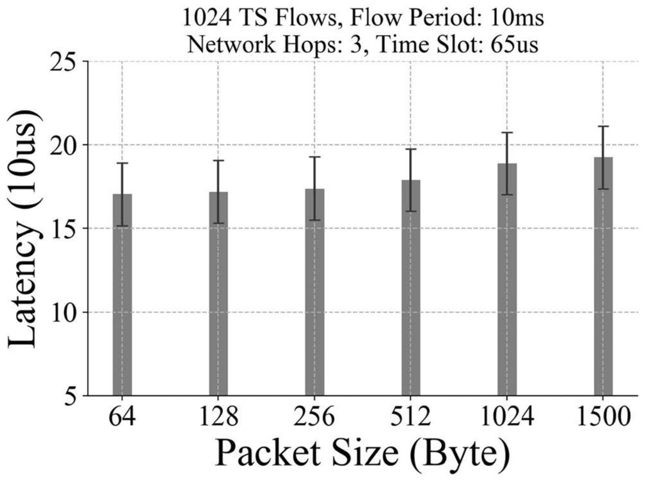
6 Related works

As a newly emerged technique, TSN has received a lot of research interests. However these research works mainly focus on the scheduling algorithms. Only a few works focus on the system-level TSN network development and evaluation. For a system-level TSN network development, the most related work is Time-Sensitive Networking. This work propose an idea for implementing TSN devices and systems using FPGA SoC (Time-Sensitive Networking). However, this study only provide preliminary ideas without implementation details. Our proposed OpenTSN is an FPGA-based solution. Therefore, OpenTSN can get the flexibility advantages of FPGA solutions for supporting new TSN technologies as mentioned in Sect. 2.

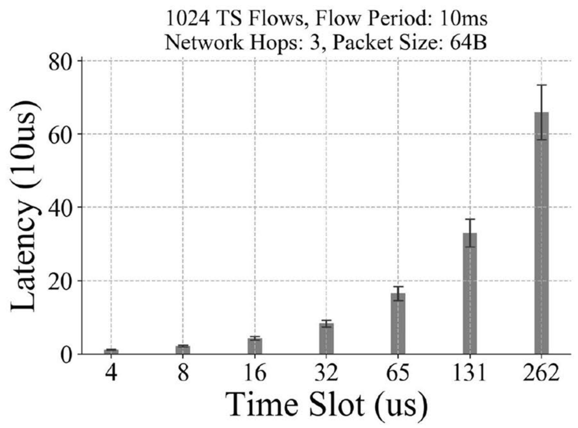
Considering the ASIC-based TSN products, the industry has built some TSN test beds for constructing TSN systems with products from different companies. Huawei and many other companies built the OPC UA over TSN testbed (Opc ua over tsn testbed 2019). The testbed dedicated to provide edge computing technology stacks like OPC UA over TSN in manufacturing and other industrial markets, to ensure that industrial needs and requirements to be best addressed through members' products. Industrial Internet Consortium (IIC) also created two physical instances of the TSN testbed (Iic tsn testbed 2019). One is hosted in North America



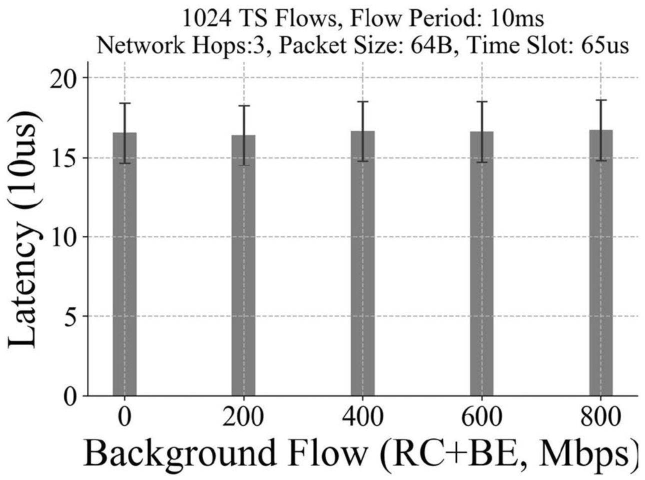
(a) Precision of time synchronization



(b) End-to-End latency under different packet size



(c) End-to-End latency under different time slot



(d) End-to-End latency under different background flows

Fig. 6 Experimental results

the other is hosted in Germany. These testbeds are used for plugfest activities where member companies collaborate to test implementations and interoperability. However, the testbeds can not directly support the deployment of new TSN technologies

There are some simulation based solutions presented in literatures. For example, Jiang et al. (2018) build a TSN simulation model by OMNeT++ containing traffic scheduling (TAS) and time synchronization (gPTP). And the results of simulation model shows there is no influence for time-sensitive traffic with regulating the bandwidth of BE traffic, which is not convincing as the BE traffics transmission may result in the delay of time-sensitive traffic at the point of switching time slot. Meyer et al. (2013) present a module with time-sensitive traffic and AVB traffic for analyzing mutual influence. While the time-sensitive traffic is not in accordance with the one in IEEE 802.1 Qbv. Nevertheless, the insights on CBS is worthy of learning. Pahlevan and Obermaisser (2018) implement TAS and PSFP functions of switches on OPNET framework, using the regular Ethernet workstation as an end station. To acquire better QoS, they design the end station by FPGA with controllable sending time of time-sensitive traffic. Falk et al. (2019) propose Nesting, which is an OMNeT++ simulation model for TSN with two switches under the time-aware and credit-based shaping. Compared to real FPGA based TSN systems, these simulation based solutions have the accuracy drawback of software simulations.

7 Conclusion

To help developers rapidly customize and evaluate a TSN system on FPGAs from different level, this article presented OpenTSN. It is an open source project which contains the necessary hardware and software components. These components are built based on the concluded basic features including a SDN-based TSN network control mechanism, a time-sensitive management protocol and a time-sensitive switching model. Under these properties, components can work together to provide required deterministic transmission. Each component can be customized from different level according to the system requirements like network topology, device buffer and queue size, time synchronization precision and so on. This article analyzed the architecture of each component in details so that designers can fully understand and use OpenTSN. The ability of our proposed OpenTSN has been verified by the demonstrating TSN systems.

Acknowledgements This work is supported by National Natural Science Foundation of China (Grant nos. 61802417, 61601483, 61702538) and National Key Research and Development Program of China (Grant nos. 2018YFB1800505, 2018YFB1800402).

References

AS6802 Standard (2016). https://www.sae.org/standards/content/as6802
Broadcom 53570 TSN chip (2017). https://docs.broadcom.com/docs-and-downloads/53570-PB101.pdf
Floodlight controller (2017). https://1.ieee802.org/tsn/802-1qci
IEEE 802.1AS Standard (2015). www.ieee802.org/1/pages/802.1as.html
IEEE 802.1CB Standard (2017). https://1.ieee802.org/tsn/802-1cb
IEEE 802.1Qav Standard (2018). http://www.ieee802.org/1/pages/802.1av.html
IEEE 802.1Qbv Standard (2015). http://www.ieee802.org/1/pages/802.1bv.html
IEEE 802.1Qcc Standard (2018). https://1.ieee802.org/tsn/802-1qcc/
IEEE 802.1Qch Standard (2017). https://1.ieee802.org/tsn/802-1qch/
IEEE 802.1Qci Standard (2017). https://1.ieee802.org/tsn/802-1qci
IEEE 802.1Qcr (2020). https://1.ieee802.org/tsn/802-1qcr//
IEEE TSN Task Group (2012). https://1.ieee802.org/tsn/
Iic tsn testbed (2019). https://www.iiconsortium.org/time-sensitive-networks.htm
Marvell 88e6390X TSN chip (2019). https://www.marvell.com/switching/assets/LinkStreet_88E6390X_FINAL2.pdf
NXP SJA1105 TSN chip (2019). https://www.nxp.com/products/interfaces/ethernet-/automotive-ethernet-switches/five-ports-avb-tsn-automotive-ethernet-switch:SJA1105TEL
Opc ua over tsn testbed (2019). https://e.huawei.com/uk/news/uk/2019/edge-computing-opc-ua-hannover-messe
OpenTSN Project (2019). https://gitee.com/opentsn/openTSN/tree/master
Overview of TTE Applications and Development at NASA/JSC (2016). https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20160012363.pdf
Time-Sensitive Networking: From Theory to Implementation in Industrial Automation (2019). 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
Traffic types mapping to TSN mechanism (2019). http://grouper.ieee.org/groups/802/1/files/public/docs2019/60802-Hotta-Traffic-Types-Mapping-to-TSN-Mechanism-0119-v01.pdf
Falk, J., Hellmanns, D., Carabelli, B., Nayak, N., Dürr, F., Kehrer, S., Rothermel, K.: Nesting: Simulating ieee time-sensitive networking (tsn) in omnet++. In: 2019 International Conference on Networked Systems (NetSys), pp. 1–8. IEEE (2019)
Jiang, J., Li, Y., Hong, S.H., Xu, A., Wang, K.: A time-sensitive networking (tsn) simulation model based on omnet++. In: 2018 IEEE International Conference on Mechatronics and Automation (ICMA), pp. 643–648. IEEE (2018)
Meyer, P., Steinbach, T., Korf, F., Schmidt, T.C.: Extending ieee 802.1 avb with time-triggered scheduling: a simulation study of the coexistence of synchronous and asynchronous traffic. In: 2013 IEEE Vehicular Networking Conference, pp. 47–54. IEEE (2013)
Nasrallah, A., Balasubramanian, V., Thyagaturu, A., Reisslein, M., ElBakoury, H.: Cyclic queuing and forwarding for large scale deterministic networks: a survey. arXiv:1905.08478 (arXiv preprint) 2019
Nasrallah, A., Thyagaturu, A.S., et al.: Ultra-low latency (ull) networks: The ieee tsn and ietf detnet standards and related 5g ull research. IEEE Commun. Surv. Tutori. 21(1), 88–145 (2018)
Pahlevan, M., Obermaisser, R.: Evaluation of time-triggered traffic in time-sensitive networks using the opnet simulation framework. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp. 283–287. IEEE (2018)
Schonwalder, J., Bjorklund, M., Shafer, P.: Network configuration management using netconf and yang. IEEE Commun. Mag. **48**(9), 166–173 (2010)
Sivaraman, A., Subramanian, S., Alizadeh, M., Chole, S., Chuang, S.T., Agrawal, A., Balakrishnan, H., Edsall, T., Katti, S., McKeown, N.: Programmable packet scheduling at line rate. In: Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16, pp. 44–57, New York, NY, USA, (2016) Association for Computing Machinery
Wollschlaeger, M., Sauter, T., Jasperneite, J.: The future of industrial communication: automation networks in the era of the internet of things and industry 4.0. IEEE Ind. Electron. Mag. **11**(1), 17–27 (2017)
Yan, J., Quan, W., Jiang, X., Sun, Z.: Injection time planning: making cqf practical in time-sensitive networking (2020)
Yang, X., Sun, Z., Li, J., Yan, J., Li, T., Quan, W., Xu, D., Antichi, G.: Fast: enabling fast software/hardware prototype for network experimentation. In: Proceedings of the International Symposium on Quality of Service, pp. 1–10 (2019)

Wei Quan born in 1987. PhD, assistant professor in College of Computer Science and Technology, National University of Defense Technology. His main research interests cover time-sensitive network, software defined network and FPGA design.

Wenwen Fu born in 1994. PhD candidate in National University of Defense Technology. His main research interests include programmable network, data center network and time-sensitive network.

Jinli Yan born in 1993. PhD candidate in National University of Defense Technology. His main research interests include time-sensitive network, data center network and software defined network.

Zhigang Sun born in 1974. PhD, professor in College of Computer Science and Technology, National University of Defense Technology. Chair of FAST opensource project. His main research interests include software defined network, time-sensitive network, network architecture, FPGA design and network security.