TSNTag: A Multi-Semantic Flow Identification Mechanism Enabling Full-Custom TSN Design*

Jinli Yan, Wenwen Fu, Wei Quan, Zhigang Sun
College of Computer, National University of Defense Technology
{yanjinli10,fuwenwen16,w.quan,sunzhigang}@nudt.edu.cn

CCS CONCEPTS

• **Networks → Network design principles**.

KEYWORDS

Time-Sensitive Network, Custom, Function Mapping, Flow Tag

ACM Reference Format:
Jinli Yan, Wenwen Fu, Wei Quan, Zhigang Sun. 2020. TSNTag: A Multi-Semantic Flow Identification Mechanism Enabling Full-Custom TSN Design. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20 Demos and Posters), August 10–14, 2020, Virtual Event, USA*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3405837.3411399

1 INTRODUCTION

Time-Sensitive Networking (TSN) emerges as a novel network paradigm by empowering ethernet with deterministic and reliable capabilities. Currently, the TSN Task Group has published comprehensive standards and drafts on time synchronization, flow control, etc [6]. Although TSN has a great potential to be applied in distributed real-time domains (*e.g.,* aerospace, industry control, energy), it is challenging for developers to save resource consumption and cost when designing TSN infrastructures [5].

By comparing the closed and proprietary TSN application scenarios with open data center, we observe that customizing a TSN network is feasible from three aspects. Firstly, there are fewer critical communication requirements, and enhancing interoperability for different TSN products is unnecessary. Secondly, the critical features (*e.g.,* topology, critical flows) are static and predetermined. Thirdly, each scenario is built for a specific purpose, *e.g.,* rocket launching, production line. These scenarios present diversified requirements on application integration, performance targets, etc [7]. Thus, providing an on-demand customization approach can accelerate the implementation and deployment of the TSN technique.

1.1 Motivation

Currently, the standard TSN system design method focuses on providing a full-featured product to adapt multiple scenarios and achieving high interoperability with the products from other vendors. For instance, the Broadcom BCM53570 TSN switching chip [1] supports almost all the essential TSN standards. With such Bottom-up developing mode, the resource partitioning does not adapt to the specific application features very well in most cases and the on-chip memory resource is often under low utilization. To overcome these drawbacks, GMV Inc. and Seven Solutions Inc. propose a light TSN implementation for avionics with Top-down method [2]. They focus on selecting TSN functions and integrating commercial off-the-shelf (COTS) IP cores under specific network workloads (*e.g.,* packet lookup, scheduler, redundancy).

Table 1: Comparison of TSN Design Approaches.

Approach	Design Goal	Technical Features	Flow Identification & Semantic
Standard Design[1]	•Interoperability •Support Multiple Scenario	•Bottom-up •Follow & Support Full-Feature Standards	•Active dMac+VlanTag •Defined in 802.1Qcc[4] • Identity: StreamID
Light Design[2]	• Save Resource & Cost • Adapt to Specific Scenario	• Top-down • Select Functions Under Specific Network Workload	
Full Custom with TSNTag		• Top-down • Select & Map Functions into Compute/Network Resources Under Specific User& Network Workload	• User-Customized Tag • Identity: StreamID, PacketID • Instructions: *e.g.,* QueueID, Path, etc.

Besides, we observe that the application workloads (*e.g.,* data monitor, analysis) also have significant effects on TSN design. For example, some latency-sensitive application workloads require hardware acceleration in the network while some latency-insensitive network workload can be migrated onto the CPU of Endpoint to save network resources. Here an Endpoint is an end system that consists of CPU and network interface card (NIC) and integrates user and network functions. Thus, developers need a full-custom TSN design to map TSN functions into network/user resource and hardware/software resource with in-depth evaluation on the upper network/user workload and underlying resource constraints.

To support highly efficient collaboration between TSN functions partitioned on different resources, it is critical to design a well-adapted flow identification. The 802.1 Qcc standard [4] proposes to carry the flow identification via active destination mac and VlanTag. These 64 bits only represent the StreamID, which is over-provisioning in most cases. FlowTags [3] attaches each packet with the necessary middlebox context and sets tag-related tables in each middlebox for supporting sophisticated and dynamic traffic-dependent action sets. Different from the FlowTags-enabled scenarios that require complex network function chains, TSN-related

*Wei Quan is the corresponding author.

SIGCOMM '20 Demos and Posters, August 10–14, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8048-5/20/08.
https://doi.org/10.1145/3405837.3411399

scenarios do not require high interoperability and focus on deliverying deterministic and reliable transmission. These related works enlighten us to simplify network design further by customizing tag reasonably and avoiding unnecessary lookup operations. Thus, we propose a multi-semantic flow identification mechanism named TSNTag to enable a full-custom TSN design. We compare our design with two existing TSN design approaches above in Table 1.

1.2 TSNTag Overview

There are two principles to be followed in the TSNTag design. Firstly, the flow identification and instructions embedded in user-customized Tag must be transparent to upper-layer applications. It means that all the modifications should only be made on layer 2 and the applications can be migrated seamlessly when the underlying network infrastructures are updated. Secondly, the attached tag should deliver essential control information for simplifying TSN system design and optimizing resource allocation and consumption.

The full-custom TSN design with TSNTag can be divided into two phases. In phase 1, the developers should select on-demand TSN functions and partition them to different resources (*e.g.,* Endpoint, Switch) under multidimensional constraints of resource consumption, performance, etc. In phase 2, the key control information required by each TSN function is extracted for designing a well-adapted tag. In the process, each packet (without standard ethernet head) from applications is appended with a customized tag that carries rich instructions (*e.g.,* QueueID, Path, etc.). The function modules can take corresponding actions according to the instructions without complex lookup and extra processing logic.

2 CASE STUDY

We take a typical application scenario in avionics and industry domains as an example to fully customize a TSN network.

2.1 Scenario Features

Topology and Node. This scenario has four nodes named as switched endpoint (SE) that form a ring topology. It plays the roles of an endpoint and a switch. As an endpoint, each SE adapts user data to TSN network. As a switch, it provides two ports for supporting bidirectional traffic transmission.

Function Requirements. In addition to the representative TSN features (*e.g.,* time synchronization, time-aware shaper), there are two advanced features required to enhance the determinism and reliability: (1) End-to-end frame replication and elimination for reliability (FRER). It duplicates the critical packets at the source and sends them across disjoint network paths. The receiver recovers critical data from duplicated packets. (2) Precise frame injection and submission. It requires precise control of the time for sending a packet to network and applications. For example, the motion control in the industry requires packets to arrive at the actuator within a specified time window for executing in-order operations.

2.2 Full-Custom Design with TSNTag

TSN-enabled SE Design. Since each SE has both the endpoint and switch capabilities, the design for each SE can be divided into three parts, as shown in Fig. 1(a). (1) **Endpoint Tx.** The *Packet Replication* and *Tag Map* function are implemented on CPU because these operations are time-insensitive and can be pre-completed in CPU. Since the *Tx sched* function requires to schedule each critical packet from the buffer to the network at a specified time, this module is partitioned into NIC. The reason that we use buffers instead of queues is because several flows may be aggregated in the same queue, which makes it impractical to control the injection and submission time precisely. (2) **Endpoint Rx.** Similar to Endpoint Tx, the *Rx Sched* function is mapped to NIC for providing precise packet submission. The *Packet Drop* and *Tag Unmap* function are running on CPU because dropping the duplicated packet requires to design complex sequence window. (3) **Switch.** The *Time-aware Shaper* function in NIC selects packets based on the queues with gate control mechnism. The ingress and egress gates attached to each queue are set as open/close state periodically according to the configurations. Besides, each SE provides precise time synchronization as a basic service in NIC.

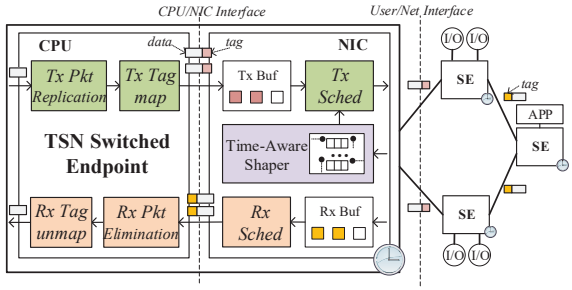

Figure 1: Use Case

Tag Structure. The packet data generated by applications is attached via a matched tag for delivering important instructions between CPU/NIC interface and User/Network interface. The customized tag structure consists of identity and instructions in Fig. 1(b). In addition to **StreamID**, **PacketID** is added to the identity field because the FRER function requires sequence value to determine whether the packet should be dropped. The instructions contain four fields. The **TxBuf** and **RxBuf** represent a specific buffer where each critical packet is stored for precise frame injection and submission. The **QueueID** indicates which queue a packet should be enqueued when traversing an intermediate node. The **Path** carries the routing information for each packet. Since the topology is ring and there are only two network ports, the packet can be delivered into the destination accurately only with the transmission direction, *i.e.*, path can be described as clockwise or anti-clockwise.

ACKNOWLEDGMENT

This work is supported by National Key Research and Development Program of China (Grant No.2018YFB1800505, 2018YFB1800402), National Natural Science Foundation of China (Grant No.61702538, 61802417, 61601483, 91938301), Training Program for Excellent Young Innovators of Changsha (Grant No.kq1905006) and Research Project of National University of Defense Technology (Grant No.ZK 17-03-53, ZK18-03-40).

REFERENCES

[1] Broadcom. 2020. BCM53570 TSN Chip. https://www.broadcom.cn/products/ethernet-connectivity/switching/strataconnect/bcm53570.

[2] Light TSN Design. 2020. An IPCORE for Deterministic Ethernet via TSN Light Implementation: Challenges and Opportunities. https://indico.esa.int/event/323/contributions/5043/attachments/3745/5201/12.30_-_An_IPCORE_for_Deterministic_Ethernet_via_TSN_..._.pdf.

[3] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. 2014. Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA, 543–546.

[4] IEEE. 2020. 802.1Qcc Standard. https://1.ieee802.org/tsn/802-1qcc/.

[5] Intel. 2020. Time-Sensitive Networking: From Theory to Implementation in Industrial Automation. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01279-time-sensitive-networking-from-theory-to-implementation-in-industrial-automation.pdf.

[6] Ahmed Nasrallah, Akhilesh S Thyagaturu, et al. 2018. Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 88–145.

[7] Jinli Yan, Wei Quan, et al. 2020. TSN-Builder: Enabling Rapid Customization of Resource-Efficient Switches for Time-Sensitive Networking, 2020 57th ACM/IEEE Design Automation Conference (DAC). (2020).