Design & Implementation of PTP Boundary Clocks on P4-programmable systems

Joydeep Pal, Sruthi G S, T V Prabhakar, and Chandramani Singh
Indian Institute of Science, Bengaluru, India
{joydeeppal, sruthig, tvprabs, chandra}@iisc.ac.in

***Abstract*—Precision Time Protocol (PTP) is an important pillar of IEEE 802.1Q Time-Sensitive Networking (TSN) standard to realize deterministic networking. TSN requires tight time synchronization using PTP and thus requires nanosecond resolution timing control of switch resources. Current solutions lack the performance and flexibility to provide an integrated solution. This paper presents P4PTP, a modular design for PTP Boundary Clocks on P4-programmable targets. It is validated on the P4 reference software switch (Bmv2) and a Netronome Agilio SmartNIC. Two implementation approaches have been explored - a complete data plane solution and a hybrid solution combining data plane packet processing and control-plane computation by developing a P4Runtime controller, and their performance are evaluated. The achieved synchronization precision supports requirements of various TSN use-cases. Comparisons are made to a standard PTP-supported COTS switch and to an Intel NIC. Portability and scalability are addressed by running the solution on 1000 Bmv2 targets and using a centralized controller to control the switches.**

***Index Terms*—PTP, P4, P4Runtime, TSN, SmartNIC**

I. Introduction

Tactile Internet (TI) applications require deterministic networks capable of providing bounded latency, low jitter and zero congestion loss. Time-Sensitive Networking (TSN), defined by the IEEE 802.1Q Working Group, is an approach towards enabling such networks. TSN is a set of standards designed with the aim of realizing deterministic networks over the Layer 2 Ethernet standard. The key components of TSN include mechanisms for traffic management and time synchronization.

The Time-Aware Shaper (TAS), standardized in IEEE802.1Qbv-2018 [1], is designed specifically to guarantee bounded latency and low jitter to TSN traffic (known as Scheduled Traffic), even in the presence of non-TSN traffic (known as Best Effort traffic). It ensures a green-channel for Scheduled Traffic (ST) on a single TSN node by means of controlling its egress port. Specifically, it executes the opening and closing of queue gates at the port according to a predefined periodic time schedule known as a Gate Control List (GCL). To ensure an end-to-end deterministic path, a GCL is computed and executed at precise times for each node, contingent on precise time synchronization across nodes. Therefore, time synchronization is essential to ensure a consistent global view of time. It enables applications such as industrial machine vision systems and Internet of Things (IOT) systems, to make global time-based decisions on packets. Although the Network Time Protocol (NTP) provides millisecond synchronization offsets, it is unsuitable for applications requiring microsecond and nanosecond precision.

Precision Time Protocol (PTP), standardized in IEEE 802.1AS-2020 [2], is a network protocol specifically designed to meet such stringent nanosecond-level synchronization requirements in TSN. The quality of synchronization also depends on the timestamp precision obtained from the clocks. Therefore, PTP obtains timestamps from the hardware clock which resides on the NIC interface. Although software timestamping can be used when hardware support is unavailable, it cannot provide high precision. The nodes synchronize their clocks by exchanging periodic time-stamped PTP packets and executing a clock adjustment algorithm. By leveraging hardware timestamping, robust clock adjustment algorithms, and network configuration techniques, PTP achieves high-precision time synchronization, meeting the requirements of various time-sensitive applications. An integrated TSN solution with TAS and PTP introduces complex interactions between scheduling mechanisms and time-sensitive traffic prioritization. There is a cyclic dependency of TAS on PTP i.e. TAS depends on tight time synchronization across TSN to ensure deterministic path to PTP packets. However, PTP itself is responsible for accurate synchronization. An unbounded synchronization accuracy compounds with time, and the TSN ceases to work. Therefore, a well-designed TSN performance is tightly coupled to the performance of the deployed time synchronization [3]. We survey the literature for existing TSN solutions that support TAS and PTP standards and examine their performance.

Existing TSN solutions are available either in the form of fixed-function Commercially off-the-shelf (COTS) devices with proprietary designs or in the form of software stacks to support distinct TSN modules on Network Interface Cards (NIC). However, there is a lack of COTS offerings that incorporate all the necessary modules. These designs are proprietary and fixed-function and hence do not offer the flexibility to implement custom algorithms and features. In contrast, software-based TSN modular solutions exist such as *linuxPTP* [4] and Time-Aware Priority Shaper (TAPRIO), which are Linux kernel-based modules to implement PTP and TAS respectively. These offer good amount of flexibility for experimentation and customization by using their configuration parameters and modification of their source codes.

978-3-903176-74-4

However, these often fall short in delivering the performance required by real-world applications, particularly in terms of synchronization accuracy, reliability and supported data-rates [3]. TAPRIO uses system time to execute the GCL and hence suffers from kernel's variable execution times. However, although it supports an offload feature that utilises the NIC hardware clock to execute GCL, very few high-end NICs support this feature. Therefore, our survey reveals that existing TSN solutions are inadequate for real-world deployments. Thus, there is a need for hardware-assisted, portable, open-source solutions that integrate PTP designs with TAS [5].

Owing to the gap between fixed-function COTS and flexible software modules, has led to a growing interest in programmable networking accelerators which are programmable using Programmable Protocol-independent Packet Processor (P4) language, such as P4-programmable smartNICs. P4 is a domain-specific language used to program the data plane of a network device, and such devices are known as P4 targets. This enables novel use-cases, such as in-network computing. Further, it allows network designers to create a single portable program for any P4 target, without the need to know the target's architecture-specific details or its native programming language, such as VHDL. The targets can be configured from a central logical controller running in the control plane on a local or remote host. The P4 organization also defines a target-agnostic P4Runtime protocol to communicate between a control plane and a P4-based data plane as shown in Figure 1. Therefore, P4 smartNICs offer programmability combined with smartNIC-specific features. Their combination can be explored to deliver significantly better real-world performance characteristics than what is achievable by relying solely on a software solution with a regular NIC as discussed above.

In this paper, a design of a PTP Boundary Clock switch in the data plane using the flexibility of the P4 language is explored to achieve precise time synchronization for TSN. The implementation is demonstrated on two P4 targets - the official P4 reference switch known as the Behavioral Model Version 2 (Bmv2) and a Netronome Agilio CX 2x10GbE SmartNIC. The Netronome smartNIC [6] is chosen for the following reasons: (1) it is equipped with networking accelerator components and (2) its ability to be programmed using P4. Using P4 for our design makes it suitable for integration with other P4-based TSN modules such as TAS and IEEE 802.CB Frame Replication and Elimination for Reliability [7].

The novelty of this work not only leverages the flexibility of P4, but also makes the implementation easily portable on COTS Ethernet switches that support P4. Our work demonstrates that, by utilizing P4's architecture and the advanced capabilities of SmartNICs, we achieve dual objectives of (1) precise synchronization and (2) flexible integration with other network functions such as TSN. With the availability of rich metadata from the SmartNIC using P4, a fine-grained control is gained over key parameters such as cycle times and queue

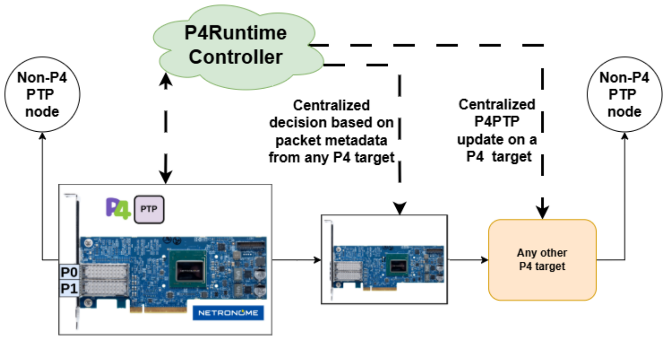

Fig. 1: P4PTP updates and configures any P4 target from a logically centralized P4Runtime-based controller, while it can make centralized decisions based on packet metadata received from each node

gate timing, which are critical for the stringent timing requirements crucial for ensuring deterministic behaviour in TSN applications. Different application domains such as Industrial machine vision systems and smart grids require varying levels of synchronization performance [8]. Hence, our approach provides a robust and adaptable solution for dynamic TSN configurations, where PTP timestamping ensures both precise synchronization and effective control of gate timing for GCL execution. Compared to traditional embedded systems, our design offers significant performance advantages, facilitating efficient network resource management and streamlined switch operations within the 802.1Q TSN framework.

In this paper, we present the following contributions:

- Design and implementation of P4-based PTP Boundary Clock for TSN that works at line-rate. The module's working is demonstrated using BmV2 reference switch and the Netronome Agilio SmartNIC.
- Proposed and validated a P4Runtime-based master controller. The controller architecture supports scalable end-to-end synchronization and management of P4 targets.
- Developed a PTP synchronization task distribution scheme for packet processing and offset computation between data and control planes. The goal is to achieve deterministic packet processing latencies that improves timing accuracy.
- Developed a P4-based tool to perform quantitative analysis of timing-critical parameters such as clock drift, processing latency and end-to-end delays.

The paper is organized as follows. Section II provides an overview of the related work. Section III describes the design implementation of P4PTP. Section IV provides background information on the P4 targets of BmV2 and Netronome Agilio smartNIC. Section V describes the evaluation and comparison of P4PTP with a standard PTP implementations.

II. Related Work

Operating in user space, *linuxPTP* provides kernel support to obtain hardware timestamps directly from a NIC's PTP

Hardware Clock (PHC). The kernel-based software timestamping introduces microsecond-order jitter. In the context of TSN, the TAPRIO scheduling module relies on system time to execute TAS GCLs. Rezabek et al. [9] have reported co-existence issues for *linuxPTP* and TAPRIO, and studied workaround solutions.

Nojavan et al. [10] present a PTP implementation on a Bmv2 switch to support TSN protocols in 5G applications. The optimization is to obtain timestamps closer to the edge and improve the synchronization accuracy to $\pm 60us$. Their work addresses the challenges of incompatible timestamp formats between PTP and P4 targets, and slow control plane processing. Our design explores processing packets in the data plane using the P4 language, while computing offset in the control plane running locally at the edge using the P4Runtime protocol.

Kannan et al. [11] present DPTP, the first work on implementing nanosecond-level synchronization in the data plane. A custom time synchronization protocol is implemented on a Barefoot Tofino ASIC switch. While the protocol leverages the timestamps from various points in the packet path, it is not compatible with the IEEE PTP standard. Our implementation is based on well-established PTP standard to ensure its long life-time, wide support and commercial feasibility. Since the traditional way of processing PTP packets in the control plane is slow, a data plane solution is explored.

Other time-synchronization protocols such as White Rabbit [12], IRIG-B and DTP [13] require support from fixed-function hardware. Therefore, they are limited in terms of scalability. HUYGENS [14] provides a time synchronization method for data centres. The algorithm builds upon PTP messages and uses linear regression to estimate the offset. However, the applicability of its synchronization approach is limited to end-hosts.

TABLE I: Literature survey for P4PTP

Literature	Standards-compliant	Hardware
DPTP [11]	✘	✔
LinuxPTP	✔	✘
Nojovan et al. [10]	✔	✘
White Rabbit [12]	✘	✔
Huygens [14]	✘	✘
P4PTP	✔	✔

Table I shows the differences between the literature and our paper. It highlights the compliance of the work with IEEE PTP standards and the presence of a hardware implementation. P4PTP is the only work in this list which has both features.

III. P4PTP DESIGN AND IMPLEMENTATION

This section describes the modular design and implementation of P4PTP. Designing the modules of the PTP Boundary Clock for a P4 target, requires breaking down the time synchronization process into key modules of packet processing functions and clock adjustment functions. The implementation approach is to develop these functional modules and the associated challenges are described in Section III-A. The key modules developed for P4PTP are described in Section III-B. Figure 2 describes the relationship between the components of P4PTP. There are three major blocks which include the master controller, the edge controller and the P4 data plane. The master controller performs a one-time setup of PTP boundary clocks on the switches, by programming the switches and the table rules. The edge controller running the switch's control plane receives digests periodically from the data plane, computes the offset, and uses this to synchronize the data plane. The data plane receives the PTP packets, performs a set of actions depending on the PTP message type, sends the timestamps as digests to the edge controller and forwards the packets.

The packet processing function requires extracting the PTP message type from packet headers and accordingly perform an action. This alludes to the Match-Action Table (MAT) design in P4.

A. Challenges in implementation

This section describes the key challenges encountered in the implementation of P4PTP due to the use of the P4 language to implement PTP and the strategies to address them. They are enlisted below in decreasing order of implementation difficulty.

- Handling timestamp operations: The computations on PTP timestamps involve operations such as addition and subtraction. These timestamps comprise of seconds and nanoseconds, which complicate performing operations on the combined entity. Specifically, the complete timestamp operation requires handling the underflows, overflows and negative values of each entity along with maintaining required precision. Programming languages such as C (as used in *linuxPTP*) or Python have abstractions to hide the complexities of these operation, where it checks for multiple cases and accordingly modify the operation. A similar approach in P4 is challenging as it does not have these abstractions and does not support operations such as division, modulo and loops. To address this, a computation module is developed as described in Section III-B.
- Hardware timestamps: PTP requires hardware timestamps from the edges of the switch. However, the P4 pipeline exists in the middle, which complicates its native integration. To overcome this, a scheduling policy is developed in the P4 data plane to ensure accurate PTP measurements over the switch. This is described in Section V. This challenge was initially considered as least challenging. However, we came across the limitations of timestamps from the targets during implementation depending on the P4 architecture.
- PTP context awareness: In PTP, multiple packets *(Sync, FollowUp, Delay Request, and Delay Response)* are received in order at different times. For brevity, these are abbreviated as *(sync, fup, dreq, dresp)*. The timestamps for each synchronization round must be retained until the final PTP packet is received. Subsequently, the stored

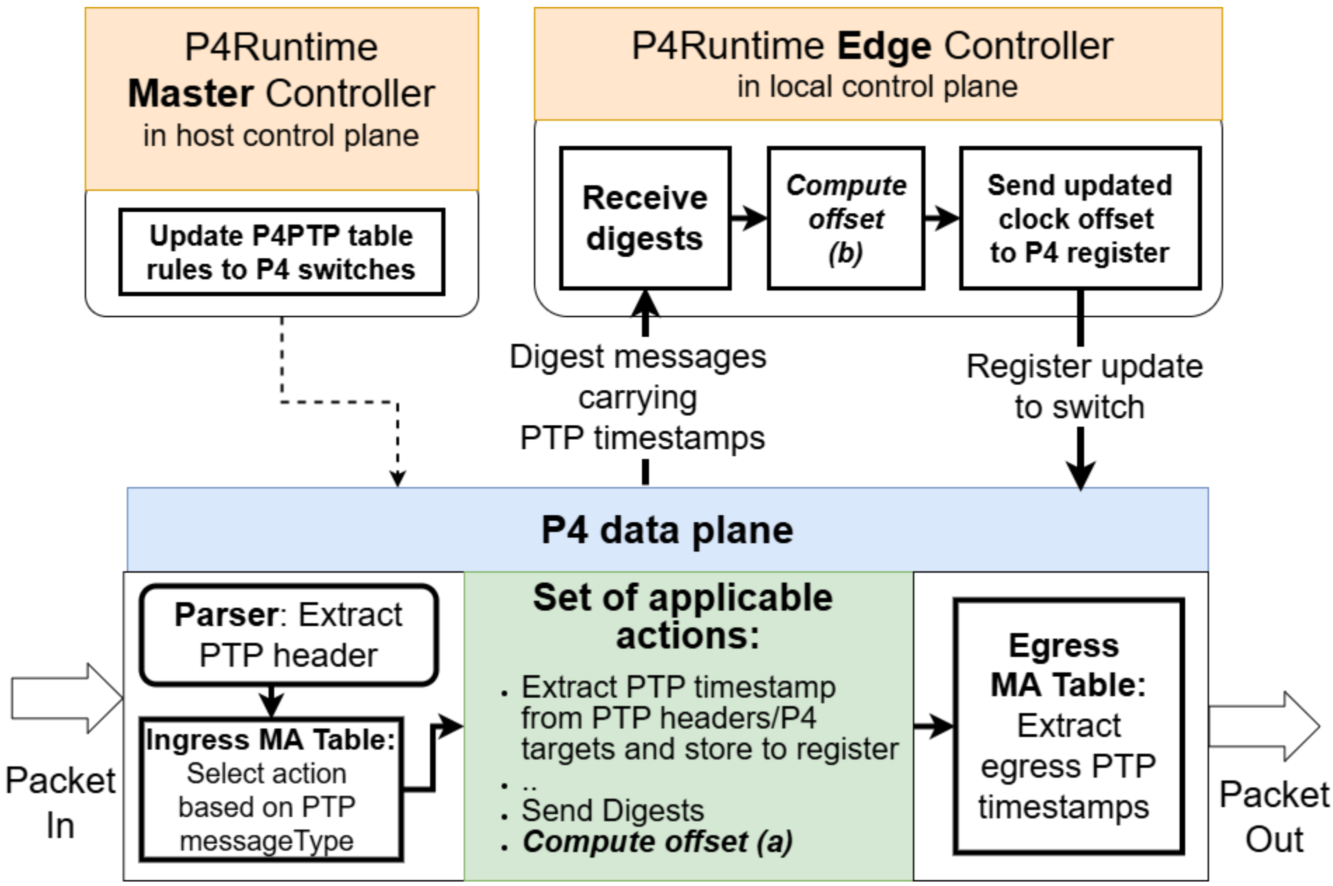

Fig. 2: Schematic to describe the relationship between P4PTP components

information is processed and discarded. However, P4 pipelines are designed with a run-to-completion model i.e., a packet and its metadata exist in the pipeline until it is completely processed. This limits its ability to perform PTP. To address this challenge, a PTP management system is developed which includes parser and header definition to understand PTP messages and usage of stateful elements such as P4 register arrays to enable persistent storage of packet-related data over time. A detailed discussion of this unit is provided in Section III-B.

- P4Runtime controller support: To manage multiple P4 switches from a logically centralized point, a P4Runtime-based controller is developed. It runs in the control plane and communicates with the data plane to manage the network of switches.
- Edge Computing: A P4Runtime controller running at the switch edge, is developed to process small digest messages which carry P4 metadata. In edge computing scenarios, having a local controller on the P4 switch for quick computation of PTP synchronization can reduce the dependency on centralized controllers. This localized processing can lead to faster and more reliable synchronization, especially in distributed network environments.
- Harmonising timestamp formats: The timestamps obtained from PTP messages and P4 targets must be compatible to perform PTP operations. Currently, there exists a standardized timestamp format for PTP but not for P4 targets. Therefore, timestamp conversions requiring operations such as division and multiplication are not supported in P4. A timestamp conversion module is developed using bit-shift operations to preserve accuracy depending on application requirements.
- Portability: P4PTP is general enough to support the PTP over Layer 2 IEEE802.1AS as well as over Layer 4 IEEE1588. Essentially, the Ethernet-based IEEE802.1AS headers required to support Layer 2 PTP can be easily replaced with IEEE1588 headers, ensuring compatibility with PTP standards. Other network protocols can also be seamlessly supported.
- End-to-end synchronization: P4PTP also supports the target-agnostic P4Runtime protocol. Our implementation not only works for switches but also supports routers or NICs that support P4. To make it work over the Internet, we implement the PTP header over IP headers such that network operators can integrate it with their own P4-supported equipment. Thus, we enable seamless end-to-end synchronization over integrated op-

erator networks in a Local Area Network (LAN) and a Wide Area Network (WAN) based on any Layer 2 and Layer 3 such as MPLS and VLAN.

```
action process_ptp_sync_action(macAddr_t ethsrc,
ip4Addr_t ipsrc,bit<64> clockid) {
        ptp_timestamp.write(1, (bit<64>)
standard_metadata.ingress_global_timestamp);
        hdr.ethernet.srcAddr = ethsrc;
        hdr.ipv4.srcAddr = ipsrc;
        hdr.ptp.clockIdentity = clockid;
}
table process_ptp_packets {
    key = {
        hdr.udp.dstPort: exact;
        hdr.ptp.messageType: exact;
    }
    actions = {
        process_ptp_announce_action;
        process_ptp_sync_action;
        process_ptp_followup_action;
        process_ptp_delayreq_action;
        process_ptp_delayresp_action;
        process_ptp_others_action;
        drop;
        NoAction;
    }
    size = 15;
    default_action = NoAction;
}
```

Fig. 3: P4 code snippet for Ingress table and an associated action

B. Key modules of P4PTP

P4PTP comprises functions for synchronizing a downstream node (functions as a timeTransmitter or timeT), synchronizing to an upstream timeTransmitter (functions as a timeReceiver or timeR), handling protocol operations, timestamping, and clock adjustment. The time taken by the synchronization process includes the time taken from generating a Sync message to adjusting the clock. To characterize this process, the key modules are identified, and their execution times are measured in the evaluation. The modules are described as follows.

PTP hierarchy update: PTP nodes exchange Announce messages to elect the GrandMaster clock using the Best Master Clock Algorithm (BMCA). A hierarchy of timeTs and timeRs is established. P4PTP facilitates BMCA among other external nodes by processing these messages, updating them with their own clock identity, and then forwarding them to other nodes.

PTP timeTransmitter and timeReceiver: PTP Boundary Clocks consists of a timeTransmitter and a timeReceiver (abbreviated as timeT and timeR respectively). The timeT unit is responsible for generating and transmitting timestamped PTP messages, that include the switch's identity, to the timeR. In P4PTP, this unit is designed to reuse incoming PTP messages by modifying their headers to incorporate switch identities. This approach of reusing messages in the data plane eliminates the need to drop incoming messages and generate new ones, thereby reducing latency and increasing throughput for PTP packets. For instance, compared to the traditional *linuxPTP* implementation, which generates new messages, our method demonstrates significant performance improvements. To ensure modularity, P4 MATs are utilized and the switch-specific rules in the table are dynamically updated using the centralized controller as shown in Fig. 1. Towards this, two MATs are implemented, one each in the ingress block and egress block. The ingress one is used to add switch identities to the incoming messages and record ingress timestamps while the egress one extracts egress timestamps of Sync message and adds it to the subsequent *fup* message.

The timeR unit receives these PTP messages from the timeT and uses them to synchronize the PTP node on which it is present. The global persistent P4 registers are used to maintain state information. P4PTP uses P4 MATs, one each in ingress and egress block to handle timestamping of PTP messages. In addition, the unit requires obtaining the *dreq* packet from the *fup* packet. Since the original packet is used by the timeT unit residing in the same switch, a P4 cloning operation is used to clone the incoming *fup* message, and this clone is converted to the *dreq* packet and sent back. This approach ensures efficient handling of PTP messages and maintains synchronization accuracy. Since the tables in both units match on PTP message types and perform complementary actions, they are combined into a single table. Figure 3 shows P4PTP's code snippet consisting of the ingress MAT with actions corresponding to each type of PTP packet. The action to process a Sync packet is also shown.

Finally, this unit is able to access the four extracted timestamps, which may be incompatible. The timestamps require compatibility to be used for the PTP computation. Therefore, a conversion module is designed and described in Section III-B and then used for computing propagation delay and offset and described in Section III-B.

Handle timestamp format mismatch: Compatibility between timestamps is ensured by converting either the internal timestamp format to PTP format or vice-versa. Implementation experience on the Netronome and Bmv2 targets shows that the former method has two advantages: (1) it requires code modifications at a single location (instead of multiple places) to cater to the various internal timestamp formats of distinct P4 targets and (2) a P4 operation designed for operating on the universal PTP format, provides a deterministic computation time and the use of a fixed memory, while that designed for each format leads to a decrease in efficiency and an increase in memory usage depending on the internal timestamp format. For example, the code complexity, memory usage and time required for processing Bmv2's microsecond timestamps are higher.

Arithemtic compute modules: This P4 module performs the mathematical operations of addition or subtraction on two timestamps provided as input and provides the resultant timestamp. The seconds and nanoseconds are handled in these timestamps using a single operation. This operation handles the corner cases of overflow and underflow, as well as negative numbers by leveraging the two's complement bit operations executed on P4 registers, completely within 12 lines of P4 language. A single 64-bit P4 register is used to save a timestamp instead of two 32-bit registers of data-type signed integer storing seconds and nanoseconds.

The P4 logic uses the input register without requiring additional memory, resulting in optimized code with fast deterministic execution time and low memory usage. It spans twelve lines, including seven fixed lines and a maximum of five conditional lines. This module is used by P4 functions which compute and apply time offsets. P4 has limitations on directly checking individual bits using array indexing. We use bitwise operations to check the sign bit.

Computation of offset and propagation delay: The module is developed to compute the propagation delay and the synchronization offset in real-time. Two implementations are presented as illustrated in Figure 2 that are executed in the: (a) data plane and (b) control plane. While the measurement concept remains consistent, the primary difference lies in the execution plane. These are described below.

- **In Data plane:** The solution is written in P4 where its capabilities are leveraged to achieve fine-grained control over packet processing and timestamping directly within the data plane. Since it does not allow complex mathematical operations, such as division, the delay and offset equations are rearranged to simplify bit operations as follows.

$$\text{Delay: } [(t_4 - t_1) + (t_3 - t_2)]/2$$
$$\text{Offset: } [(t_2 + t_3) - (t_1 + t_4)]/2$$

  The advantage of these expressions is that each computation becomes a simple addition or subtraction. The mathematical operations required for the operations are delegated to the arithmetic computation module III-B.
- **In Control plane:** In this solution, PTP computations are offloaded to a controller running in the control plane. Timestamps required for the computations are transmitted by the data plane in short digest messages. The controller continuously listens for these digests, extracts timestamps, and performs the computations. Finally, a register update command updates the switch's offset register.
  The entire process must be completed in real-time to prevent loss of synchronization. However, the variable distance between the master controller and a switch leads to varying communication delays. To address this, the P4Runtime framework is utilized, which allows multiple controllers to run in parallel and select an active controller among them. Thus, the logically centralized slow master controller managing the network remains unchanged, while a controller is designed to run at each switch's edge for real-time compute operations. This edge controller,is built using a combination of custom bash and Python scripts. In summary, an edge controller running in the local control plane is used for computing time offsets, rather than relying on a control plane offered by the master controller's host.

This design choice is driven by the interval of PTP synchronization messages, which ensures timely and accurate synchronization. In both cases, the computed signed offset value is written to a designated 64-bit register on the switch. The execution time performance of these solutions is studied in the Evaluation section.

Clock synchronization unit: A switch's function depends on the clock timings and an abrupt modification may lead to unexpected behaviour by affecting critical operations. Hence, a switch's internal clock should not be modified. Therefore, a technique is designed to apply the offset correction only to the functions that explicitly require it, such as those that modify outgoing timestamps and execute GCLs. This technique does not require disturbing the original clock of the switch. This paper explores two designs for this unit. The first design involves executing it in the data plane, while the second design is a hybrid approach which involves processing packets in the data plane and executing the unit in the control plane running locally at the edge using the P4Runtime protocol. In the second design, the time taken for these units are measured.

IV. P4-PROGRAMMABLE TARGETS

P4 targets are networking systems which are programmable using P4. This paper demonstrated P4PTP on two P4 targets: (1) the P4 reference switch known as the Behavioral Model Version 2 (Bmv2) software switch, and (2) a Netronome smartNIC which are described below:

A. Netronome smartNIC hardware switch

Netronome Agilio CX 2x10GbE is a two-port 10GbE P4-programmable SmartNIC equipped with hardware accelerators such as 48 packet processors, 60 Flow Processing Cores (also known as MicroEngines) and a 3-level hierarchical Traffic Manager. This section outlines the key features enabled due to integration of the Netronome smartNIC with P4, which facilitate the deployment of P4PTP:

- Fast processing in the data plane instead of the control plane: A P4 pipeline is defined as the sequence of packet processing blocks, which define how a P4 target processes packets. The compute power on the SmartNIC enables us to leverage the parallel execution of these pipelines on packet processing cores of the smartNIC called MicroEngines (MEs). Therefore, this enables PTP packets to be processed in the data plane in microseconds instead of the slow control plane which may take 100s of milliseconds.
- Programmable blocks of a P4 pipeline: The v1model defines a P4 pipeline comprising blocks in the order of a Parser, an Ingress control, Traffic Manager (TM), an Egress control and a Deparser where apart from the fixed-function TM, all blocks are programmable. The Bmv2 software switch, a widely used P4 target for testing and developing P4 pipelines, is based on v1model.

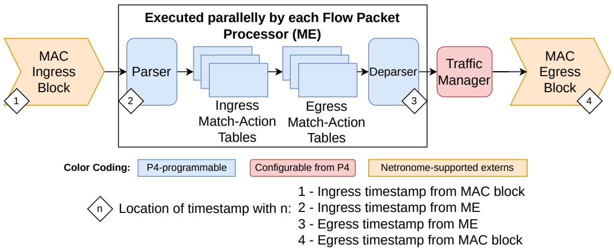

Fig. 4: P4 architecture of Netronome smartNIC

However, Netronome follows a modified version of the v1model where the TM operates after the Egress block as shown in Fig. 4.

- Fixed-function Traffic Manager: The smartNIC TM is configurable from P4 and comprises eight hardware queues and a 3-level hierarchical scheduling and shaping. PTP packets encounter variable queueing delays in the TM due to its queueing, scheduling and shaping policies. Therefore, the egress timestamp (i.e. the timestamp at which packets are going to egress from the switch) for the PTP packet needs to be obtained after the TM policies are applied. However, due to the P4 architecture of the smartNIC where TM is after the P4 pipeline, the egress timestamp is not directly available to P4.
- Hardware timestamps: Timestamps are available directly from the smartNIC's Media Access Control (MAC) block. Each incoming packet is timestamped at the switch ingress, and this timestamp is made available to the P4 program as part of the packet's metadata. This ensures an accurate ingress timestamp for the PTP packet. The egress timestamp of the packet is captured in the MAC block before transmission. However, Netronome does not make it available to P4. Instead, it writes it to the outgoing packet in a user-defined packet location. These timestamps are 64-bit wide with the 32-bit MSB and 32-bit LSB representing seconds and nanoseconds respectively, with a resolution of 1 ns. It represents the time elapsed since the smartNIC was powered on.
- Tiered memory architecture: Netronome provides different types of memory that have varying access latencies. These are available to P4 depending on the locality of initialized memory. For example, variables defined in P4 block exists until the packet exists in the block, while a P4 metadata variable is available as long as the packet exists in the pipeline. P4 makes available a persistent global stateful memory called P4 registers. These are utilized to maintain shared state information. P4PTP optimizes the use of memory.

B. Bmv2 software switch:

The Bmv2 is a reference P4 software switch for prototyping P4 programs and validating functional behaviour. It faces inherent performance limitations due to being implemented in software, incurring software-related latency variations. In our experimental setup, it is built from the source with performance-related optimizations to demonstrate the highest possible performance achievable by P4PTP.

V. Evaluation and Results

The time synchronization accuracy using PTP is characterized by two aspects: the quality of timestamps obtained and the processing of PTP packets in the switch. The synchronization performance of P4PTP is monitored using network delay measurements and system-related metrics such as processing delay and execution times of P4PTP modules. Systems with P4 data planes can provide deterministic latencies, leading to precise time synchronization. The performance of P4PTP is evaluated on the two P4-programmable systems: Bmv2 (for scalability) and Netronome Agilio smartNIC (for hardware performance analysis) and the experimental setup for the two are described as follows.

Experimental setup - Scalability using P4Runtime: An experiment was conducted to study the time taken to achieve network-wide synchronization of P4PTP switches. Figure 5 illustrates the experimental setup, which comprises 'n' number of Bmv2 switches within the Mininet network emulator. A linear topology is considered between the PTP end-hosts. P4PTP is configured on each switch with a table rule update by the master controller to setup the switch's identity, for taking part in the PTP synchronization. An edge controller running at each switch is responsible for obtaining logs and handling offset computation. The synchronization is initialized by the PTP grandmaster on host 'h1', with the initialization time recorded at this point. The time taken for the edge controller on the last switch to show the synchronization logs is recorded. Table II shows the time difference measurements conducted for varying number of switches to synchronize, and this increases O(n) where 'n' is number of switches. This demonstrates the scalability of our proposed method.

TABLE II: Time difference measurements conducted for varying number of switches

Number of switches	Synchronization Time (ms)
10	11.546
50	65.5
100	126.76
200	257.55
500	677.76
1000	1626.48

Experimental setup for Netronome smartNIC: The developed P4PTP code is then ported to a Netronome smartNIC to evaluate the performance in real-world conditions. The synchronization performance of P4PTP running on Netronome smartNIC is compared to that of standard hardware-software PTP BC implementations. Figure 6 shows the experimental setup to measure and compare the synchronization performance between multiple devices under test (DUT). The DUTs include: (1) a Netronome smartNIC running P4PTP, (2) a PTP-supported COTS TSN switch from

Microchip (KSZ9477S) [15] running an optimized *linuxPTP*, (3) a standard industry-grade dual-port Intel X520-DA2 10GbE NIC running *linuxPTP* with hardware timestamping and (4) an Intel NIC running linuxPTP with software timestamping. The DUT host is equipped with a Z690 motherboard, running Ubuntu 18 OS with kernel version 4.15.

PTP packets are generated on a second host that is equipped with a dual-port Intel NIC and Ubuntu 22 OS. The NIC's *eth0* interface is configured as the grandmaster using *linuxPTP*, while *eth1* becomes timeR. This configuration is possible because the Intel NIC supports hardware timestamps independently on each interface. In summary, PTP runs the timeT on *eth0*, while the timeR on *eth1* adjusts its clock or the system time based on the configuration.

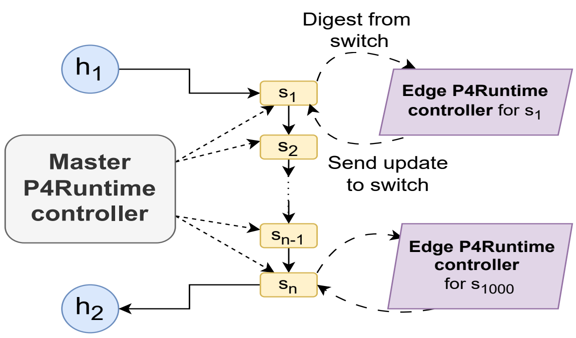

Fig. 5: Experimental setup consisting of 1000 Bmv2 switches running P4PTP. A P4Runtime master controller loads the P4 program and sends a rule update to initialize PTP on all switches, while an edge controller, running on the control plane of each switch, computes offsets and applies the correction to the switches.

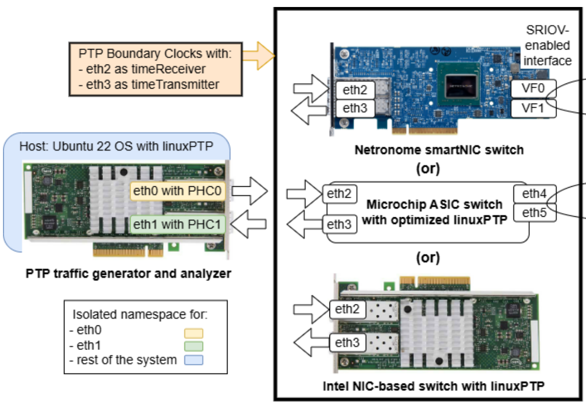

Fig. 6: Experimental setup for PTP Boundary Clocks on DUT. The DUTs are Netronome SmartNIC running P4PTP, Microchip switch, Intel NIC running *linuxPTP* with h/w and s/w timestamping.

A. Execution times for P4PTP modules

The contribution of the execution times for each P4PTP module to the synchronization is characterized. Let the execution times of *sync*, *fup*, *dresp*, Arithmetic and Conversion modules be $t_s, t_{fup}, t_{dresp}, t_a, t_{conv}$. The total execution time for a synchronization round, denoted by t_{round} is given by:

$$t_{round} = t_s + t_{fup} + t_{dresp} + t_a + t_{conv} \quad (1)$$

Table III shows the measurements for the modules specific to timeR on the two P4 targets, using the hybrid approach. The measurements are from the data plane except for the compute module which is executed in the control plane, and are an average of 5 rounds. The average latency incurred by the arithmetic unit in the data plane approach is $\approx 200us$ while that by the hybrid approach is 300us. This makes it sufficient for synchronization intervals over 300us.

In summary, the execution time for a complete PTP synchronization round when messages are processes using P4 is $\approx 300us$, while it takes $\approx 500ms$ when messages are processed using *linuxPTP* on an Intel NIC.

B. Synchronization performance

Since a deterministic delay is critical in maintaining a clock offset in the order of nanoseconds, the packet latency in the P4 pipeline in the SmartNIC is measured and found to be a consistent value of 400ns. Figure 7 shows the clock offset measurements of P4PTP. The x- and y-axes represent the timestamp t_1 and the computed clock offset, respectively. The t_1 required for this computation was made available after suitable modification of the *linuxPTP* source code. The output logged by the arithmetic unit in the smartNIC was available on the edge controller and provided the clock offset. To generate PTP packets, *linuxPTP* was configured to perform 32 synchronization rounds in a second. The measurements were conducted for 10 min with two switches. For these values, it is observed that P4PTP with its hybrid computation approach provides offset values ranging between $\pm 400ns$, in the absence of background traffic. The offset variation is due to the variation from the MAC timestamps, which cannot be accounted for because of the non-availability of the egress hardware timestamp. The results from Netronome are compared with other DUTs with the experimental setup shown in Figure 6. The Microchip COTS switch demonstrates an offset measurement of $\pm 100ns$. This performance is due to it being an ASIC. Finally, the offset achieved with the Intel NIC is in the order of us.

In TSN, the PTP packets travel in-band with other packets. Hence, P4PTP's synchronization on the smartNIC is evaluated under varying network load, where the iperf tool generated and analyzed background traffic by utilizing the SRIOV-enabled VF interfaces. Since the egress timestamp is not available to the switch, the MAC processing delay is characterized in Netronome. It is observed that the synchronization is maintained for up to 1 Gbps of background traffic beyond which the SmartNIC loses synchronization. This is

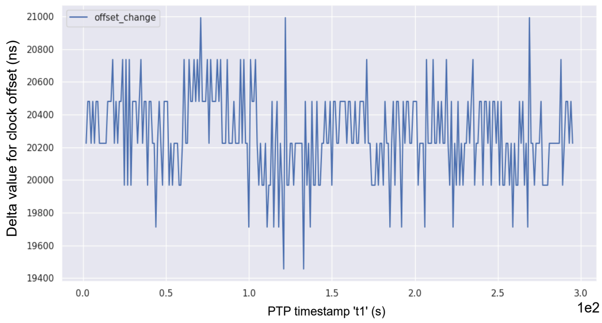

Fig. 7: Time synchronization offset of P4PTP

due to the variability in latency of PTP packets when interspersed with high background traffic at the ingress interface. To observe the effect of modifying the outgoing bandwidth, offset values were measured for 1, 5 and 10 Gbps. The values remain consistent for 5 and 10 Gbps, while in the case of 1 Gbps, packet loss is observed due to outgoing bandwidth being lower than the incoming data rate leading to loss in synchronization. Furthermore, synchronization performance was not affected by modifying SmartNIC resources.

TABLE III: Execution time of P4 modules

Module	Netronome (ns)	Bmv2 (ms)
Announce	177	0.843-1.62
Sync	278	0.91-0.97
FollowUp	1693-1921	1.60-2.05
DelayResp	1289-1441	0.95-1.36
Arithmetic (control plane)	733	0.2-0.3
Conversion	NA	1.6-2.6

C. *Characterization of clock drift of DUT clocks*

PTP periodically initiates message exchanges to compute offsets and synchronize clocks. This time period, known as the update interval, is determined at runtime by the required synchronization precision and the clock variation over time i.e. clock drift. Hence, to characterize the quality of DUT clocks, an experiment was conducted to measure their drift with respect to the reference grandmaster clock on the Intel NIC using the experimental setup shown in Figure 6. To obtain these measurements, the clocks are configured to disable the synchronization, i.e., they run freely. The experiments were conducted for a duration of 5 minutes, with an update interval configured to be 1 second and the results are tabulated in Table IV. The configuration for each DUT is as follows: (1) P4PTP: The offset computation module is enabled while the clock synchronization module is disabled. The hybrid approach is used to obtain logs from the controller. (2) Microchip switch: It has a limitation in providing custom metadata; therefore, approximate results were logged and (3) Intel NIC with h/w and s/w timestamping: The 'free-running' option is enabled on the modified *linuxPTP*. It is observed that the minimum drift is exhibited by the Intel NIC, and the reason may be attributed to the grandmaster NIC being the same model with similar clock properties. The maximum deviation in the drift is for the host system time as expected.

TABLE IV: Mean and std. dev. of clock drifts of DUTs

DUT	Mean (us/s)	Std. dev. (us)
Intel NIC with h/w timestamps	0.42	0.24
Host system time	51.23	40
Microchip TSN switch	5.3	0.08
P4PTP on Netronome	53	4.35

VI. Discussion and Conclusion

PTP traditionally operates in the slow control plane, which leads to varying processing delays in the order of milliseconds. This paper explored a data plane approach to building a PTP Boundary Clock for TSN on P4 targets, to provide nanosecond-level synchronization. The SmartNIC provides deterministic delay and has sufficient hardware resources to process packets at line rate, and therefore facilitates the synchronization of a large number of PTP hosts. In PTP, since each message type has an action and variable number of operations, their execution times are different. The design and implementation of P4PTP has yielded a platform-independent, scalable and a real-time solution for nanosecond synchronization. The design is open-source and available in our github repository. An experiment was conducted on a real-world testbed between two cities which are separated by 400kms [16] and P4PTP running at the two edges. While *linuxPTP* achieved synchronization in $\approx 9s$, the current implementation converged 4x faster in $\approx 2.3s$.

Acknowledgments

Partly supported by the MeitY, Government of India (SP/MITO-20-0006), the Centre for Networked Intelligence (CNI, a Cisco CSR initiative) and Boeing ANRC project fund at Indian Institute of Science.

References

[1] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," IEEE, Tech. Rep., 2018, iSBN: 9781504449298. [Online]. Available: https://ieeexplore.ieee.org/document/8403927/
[2] "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," ISBN: 9781504463416. [Online]. Available: https://ieeexplore.ieee.org/document/9120376/
[3] T. Kovácsházy and Z. Fodor, "Linuxptp Servo Optimization for Network Adapters with High-Precision Oscillators," in *2023 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. London, United Kingdom: IEEE, Sep. 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10296994/
[4] N. T. Foundation, "PTP Linux module," https://linuxptp.nwtime.org/install/, [Online; accessed July-2021].
[5] A. Wiesner and T. Kovacshazy, "Portable, PTP-based Clock Synchronization Implementation for Microcontroller-based Systems and its Performance Evaluation," in *2021 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. NA, FL, USA: IEEE, Oct. 2021, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9615250/
[6] "Netronome agilio cx smartnics." [Online]. Available: https://www.netronome.com/products/agilio-cx/

[7] S. K. Rana, H. Verma, J. Pal, D. Choudhary, T. V. Prabhakar, and C. Singh, "Enhancing Reliability of Scheduled Traffic in Time-Sensitive Networks using Frame Replication and Elimination," in *2023 IEEE 29th International Symposium on Local and Metropolitan Area Networks (LANMAN)*. London, United Kingdom: IEEE, Jul. 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10189416/

[8] M. Levesque and D. Tipper, "A Survey of Clock Synchronization Over Packet-Switched Networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2926–2947, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7509657/

[9] F. Rezabek, M. Bosk, G. Carle, and J. Ott, "TSN Experiments Using COTS Hardware and Open-Source Solutions: Lessons Learned," in *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. Atlanta, GA, USA: IEEE, Mar. 2023, pp. 466–471. [Online]. Available: https://ieeexplore.ieee.org/document/10150312/

[10] A. Nojavan, B. Pontikakis, F.-R. Boyer, and Y. Savaria, "5G Fronthaul in Modular P4: eCPRI Protocol Processing and Precise BMv2 Timestamps for PTP-1588," *IEEE Access*, pp. 1–1, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10857294/

[11] P. G. Kannan, R. Joshi, and M. C. Chan, "Precise Time-synchronization in the Data-Plane using Programmable Switching ASICs," in *Proceedings of the 2019 ACM Symposium on SDN Research*. San Jose CA USA: ACM, Apr. 2019, pp. 8–20. [Online]. Available: https://dl.acm.org/doi/10.1145/3314148.3314353

[12] M. Lipinski, T. Wlostowski, J. Serrano, and P. Alvarez, "White rabbit: a PTP application for robust sub-nanosecond synchronization," in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. Munich, Germany: IEEE, Sep. 2011, pp. 25–30. [Online]. Available: http://ieeexplore.ieee.org/document/6070148/

[13] V. Shrivastav, K. S. Lee, H. Wang, and H. Weatherspoon, "Globally Synchronized Time via Datacenter Networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1401–1416, Aug. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8734740/

[14] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, "Exploiting a Natural Network Effect for Scalable, Fine-grained Clock Synchronization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 81–94. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/geng

[15] "Microchip cots tsn switch - ksz9477s." [Online]. Available: https://www.microchip.com/en-us/product/ksz9477

[16] J. Pal, D. Choudhary, N. K. Gnani, T. V. Prabhakar, C. Singh, H. K. Atluri, and A. Paventhan, "Towards a TSN-DetNet Intercity Testbed for Tactile Cyber-Physical Systems," in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Vancouver, BC, Canada: IEEE, May 2024, pp. 1–6, iNFOCOM CNERT 2024: The 11th International Workshop on Computer and Networking Experimental Research using Testbeds. [Online]. Available: https://ieeexplore.ieee.org/document/10620882/