08-09-2022

White Paper: Timestamping and Clock Synchronisation in P4-Programmable Platforms

Grant Agreement No.:	856726
Work Package	WP6
Task Item:	Task 1
Document Type:	White Paper
Dissemination Level:	PU (Public)
Lead Partner:	GARR
Document ID:	GN4-3-22-49EC10
Authors:	Tomáš Martínek (CESNET), Mauro Campanella (GARR), Federico Pederzolli (FBK), Joseph Hill (UvA), Marinos Dimolianis (NTUA), Damian Parniewicz (PSNC)

The research leading to these results has received funding from the European Union’s Horizon 2020 research and innovation programme under Grant Agreement No. 856726 (GN4-3).

Abstract
This white paper investigates the capabilities of current P4-programmable platforms in terms of timestamps and clock synchronisation, and evaluates their readiness for emerging In-band Network Telemetry (INT) monitoring applications. It provides a summary of INT, introduces the concepts of time synchronisation, and presents a review of current P4 platforms’ capabilities, including the practical implications for INT.

Table of Contents

Table of Figures

Table of Tables

Executive Summary

Ongoing tests within the GN4-3 project by Work Package 6 Network Technologies and Services Development, Task 1 Network Technology Evolution, about using In-band Network Telemetry (INT) have highlighted the usefulness and relevance of precise time synchronisation of clocks in network nodes. Such synchronisation simplifies the evaluation and comparison of the time of events in different nodes, their absolute order in time, and the computation of metrics that rely on accurate time differences, such as one-way packet delay between nodes in the networks. Furthermore, the current INT capabilities permit tagging each packet with a timestamp, even in high-capacity links (100 Gbps), suggesting that time between participating nodes should be synchronised with high precision, in the order of microseconds or nanoseconds. The accuracy required will depend on the specific use case.

This white paper first provides a summary of INT, introduces the basic concepts and components of time synchronisation, and then presents its use in the case of INT, including a review of current capabilities in hardware elements. Hardware support is required to reach sub-millisecond accuracy.

Setting time on networked devices is typically done with Network Time Protocol (NTP), which, if configured well, may be accurate down to the millisecond level. For higher accuracy, the Precision Time Protocol (PTP) has been defined as a standard tool to distribute time information from high-precision sources and to be used where higher precision synchronisation is required. However, PTP requires specific hardware support, and is not (yet) widely deployed.

The review also highlights the complexity generated by the lack of specification in the INT standard for time representation in INT headers, with a combination of time-of-day and fast counter. Hardware platforms should also signal when their counter wraps and provide the functionality of synchronisation using PTP.

The effort is supported by the GN4-3 project in Work Package 6, Task 1.

1 Introduction

With the transition to high-speed fibre-optic communication, the development of ubiquitous high-speed networks, and the migration of applications to the cloud, the demands on network infrastructure have increased. The emphasis is now often on gaining the lowest possible communication latency. To meet these requirements, it is necessary to have not only a robust and distributed infrastructure but also effective tools for managing and monitoring it.

A promising technology for gathering detailed, real-time information from network and traffic is In-band Network Telemetry (INT), which can report the characteristics of switches and individual packets and flows traversing them. The effectiveness of this technology relies on individual monitoring probes distributed along with the network infrastructure that runs inside selected switching elements. In contrast to other technologies, INT is designed to run directly at the data-plane level and does not burden the switching process itself. At the same time, INT tries to offer the user as much flexibility as possible, being able to monitor selected flows or their specific parameters.

Such flexibility of detailed and selective monitoring per specific use case is enabled with the use of P4-programmable network software or – preferably – hardware solutions. To provide adequate performance in one-way delay and delay variation (jitter) measurements, it is also required that all measurement nodes are synchronised using a standard definition of the time [Time_Def_Stds], such as GPS [GPS] or Coordinated Universal Time [UTC], as accurately as possible.

The aim of this paper is to investigate the capabilities of current P4 platforms in terms of timestamps and clock synchronisation, and to evaluate their readiness for emerging INT monitoring applications. In their review, the authors have included not only hardware switch platforms but also platforms based on smart network interface cards (SmartNICs) or platforms operating at different software levels. They also focused on the key features in which these platforms differ and their impact on practical applications.

Section 2 provides a short summary of In-band Network Telemetry, and Section 3 gives an introduction to time synchronisation concepts. Section 4 considers the practical implications of time synchronisation for INT. Section 5 provides a summary and conclusions. Appendix A provides a summary of the technical features of platforms for INT and time synchronisation.

The effort is supported by the GN4-3 project in Work Package 6 Network Technologies and Services Development, Task 1 Network Technology Evolution.

2 In-Band Network Telemetry

In-band Network Telemetry (INT) is a promising technology for network traffic monitoring exploiting the paradigm of data plane programming (DPP). It exploits the benefits of being able to program hardware devices through the P4 language [P4_Language] to extract the local computed information from the device and record it directly inside the packets of interest as they traverse the network.

INT is complementary to streaming telemetry, with which network devices such as routers, switches and firewalls push data, usually averaged, related to the network's health. INT monitoring can focus in real time on a single flow or event according to the associated P4 code.

Each INT-enabled network node can fulfil one of three distinct functions on the path of the packets or flow being monitored:

Function	Description
Source	The initial measurement node. Inserts an additional header in selected (possibly all) packets, containing the information that will be forwarded by subsequent INT-enabled nodes (e.g., timestamp of packet arrival/departure, occupancy of local queues, etc.).
Transit	Any INT-capable intermediate node traversed by a packet carrying an INT header. May add a new header, or extend a header inserted into the packet by the source node by adding the local values of requested information.
Sink	The last node along the monitored path. Removes the INT headers inserted by the source node and by any transit nodes, adds its local information, and forwards that data to a collector, where the information is stored for further analysis.

Table 2.1: Functions of INT-enabled network nodes

A more detailed overview of INT is available in the WP6 T1 white paper *In-band Network Telemetry Tests in NREN Networks* [WP_INT_Tests].

Timestamps can be used to measure, for example, one-way delay and IP packet delay variation (IPDV) [RFC_3393], both for the end-to-end path and for any span between INT nodes. Queue occupancy information allows the user to further identify the likely location and causes of any delay experienced. Based on this information, network engineers or administrators may implement corrective measures, e.g. the quality-of-service (QoS) performance might be tuned at congested points in the network in order to prioritise time-sensitive flows. The impact of such measures can then be assessed in detail and in real time.

The INT specification v1.0 [INT_SPEC1.0] and subsequent versions [INT_SPEC2.0], [INT_SPEC2.1] suggest that the Ingress timestamps should be recorded as soon as possible after the packet enters an INT-enabled network element (in fact, in the reference implementation, it should be recorded as metadata before even starting to parse the headers), and similarly the Egress timestamp ought to be recorded as late as feasible to accurately record the full time the packet spends in the device. Hardware support is needed to add timestamps with sub-millisecond precision, especially at current line rates of multiple tens or hundreds of Gigabits per second (tens of thousands or millions of packets per second).

However, to measure precisely delay (time spent in the network by a packet to reach its destination) and jitter/IPDV (variation in delay of uni-directional, consecutive packets between nodes in a network), it is essential that individual nodes insert accurate timestamps into transiting packets and that such timestamps are synchronised with the global time.

3 Clock Synchronisation

Synchronising clocks on networked devices implies a set of procedures which ensure that multiple, often geographically distributed clocks – each providing a local clock signal for a network node – share a consistent view of the time. The original source of time, an atomic clock, for example, is usually capable of an accuracy of picoseconds (10^{-12} of a second).

Clock synchronisation is anything but trivial: individual nodes must maintain and update their internal (local) time using a hardware timer controlled by oscillators (clock crystals). These clock crystals are not very accurate, causing the local clock to gradually drift from the global time (local clock drift). The frequency of these oscillators is also dependent on local environmental changes such as temperature, vibration, etc., causing local clock jitter.

The measuring nodes have to deal with all these issues, request precise time information (from a trusted node), and adjust their local time repeatedly during normal operation, ideally at least multiple times per minute (and proportionally to their clock drift and desired accuracy). A more detailed description of these problems, including the solutions used, is given in the following subsections.

3.1 Single Clock Behaviour

Behaviour of a single clock is determined by the clock drift (time shift) from the nominal clock, clock jitter and its adjustments in time, as described in this subsection.

3.1.1 Drift

An INT measuring node (source, transit, or sink) needs to implement a mechanism for storing and maintaining precise time information to insert timestamps into packets. One of the simplest ways to implement such a mechanism is to connect the source of the clock signal, the so-called oscillator, to a counter, which increments its value in each clock cycle. The block diagram of such a circuit and the waveform showing the relationship between the clock signal and the value of the counter is presented in Figure 3.1.

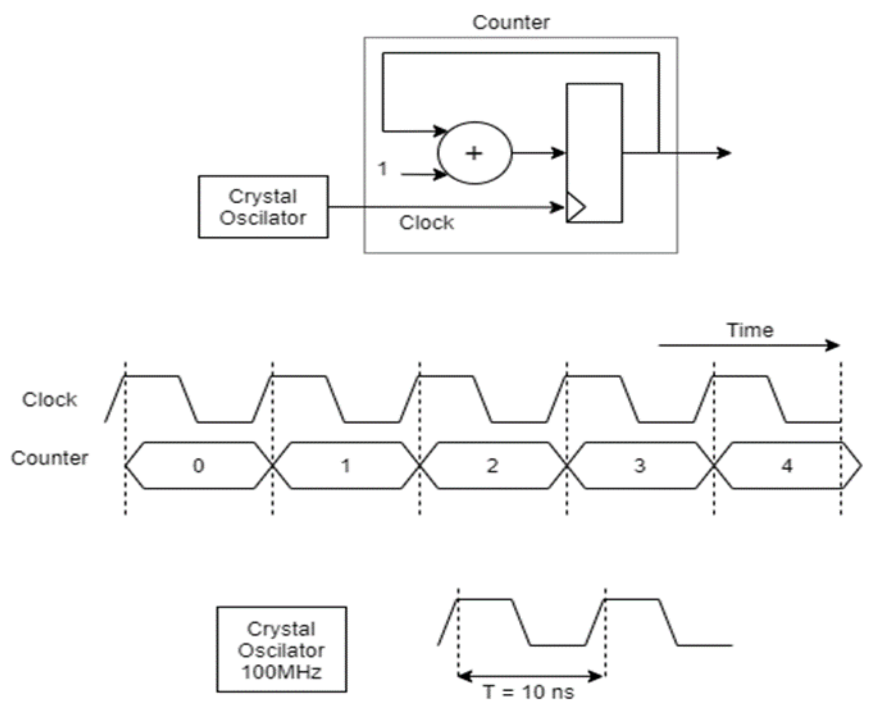

Figure 3.1: Time storage and maintenance using a simple counter (top); block diagram of the circuit and the waveform showing the relationship between clock signal and value of the counter (bottom)

The value of the counter can then, for example, represent the number of ticks since the boot time of the node (connection of the supply voltage, system boot, etc.). To relate this number of ticks to an actual amount of time – for example, the number of seconds since the system was started – it is necessary to multiply the value of the counter by the length of the clock signal period of the oscillator used. Note that the length of the clock signal period can be calculated as T = 1 / f. If the frequency of the oscillator is 100 MHz and 5 ticks have elapsed since the start of the node, then the corresponding time interval equals 10 ns * 5 = 50 ns.

Storing and maintaining information about the current actual, global time can then be realised within this simple hardware circuit by adjusting the initial value of the counter. For example, the amount of time elapsed since the start of the Unix epoch (midnight on 1/1/1970) could be obtained.

However, this simple implementation suffers from several problems. The first is the fact that commonly available clock sources (oscillators) have limited accuracy. In other words, it is not currently possible to mass-produce crystals that generate, for example, a signal with a frequency of exactly 100 MHz. Real clock crystals oscillate at a frequency that is always a little more or less than the nominal one, primarily because of imperfections in the production process. Thus, each crystal has a certain accuracy, and this is defined in PPM (Parts Per Million) units, i.e., by how many parts per million (e.g., microseconds every second) the clock may differ from the correct and expected amount of progress.

The accuracy of commonly available and used oscillators in electronics ranges from 20 to 100 PPM. It is also possible to obtain crystals with higher accuracy of around 1 PPM, but at a significantly higher price.

If an oscillator with an accuracy of 100 PPM is used in the circuit to store and maintain the current time information mentioned above, it would mean that every second, the value of the current time (stored on a given measuring node) can differ by up to 100 microseconds compared with the exact global time (see Figure 3.2). This regular time shift is also referred to as clock drift.

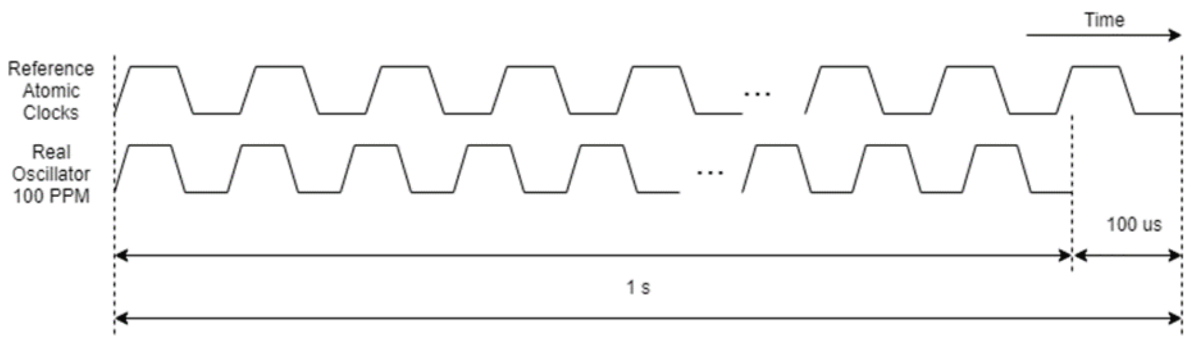

Figure 3.2: The difference between precise and real oscillators

A second limitation of said hardware circuit is the need to convert the value of the counter to time, which requires an additional multiplication operation for every reading. Both of these limitations can be removed by modifying the circuit, as shown in Figure 3.3. The main change is in the generalisation of the counter increment mechanism: instead of adding 1 for each clock cycle, a pre-set or configurable value representing the amount of elapsed time is added to the increment register.

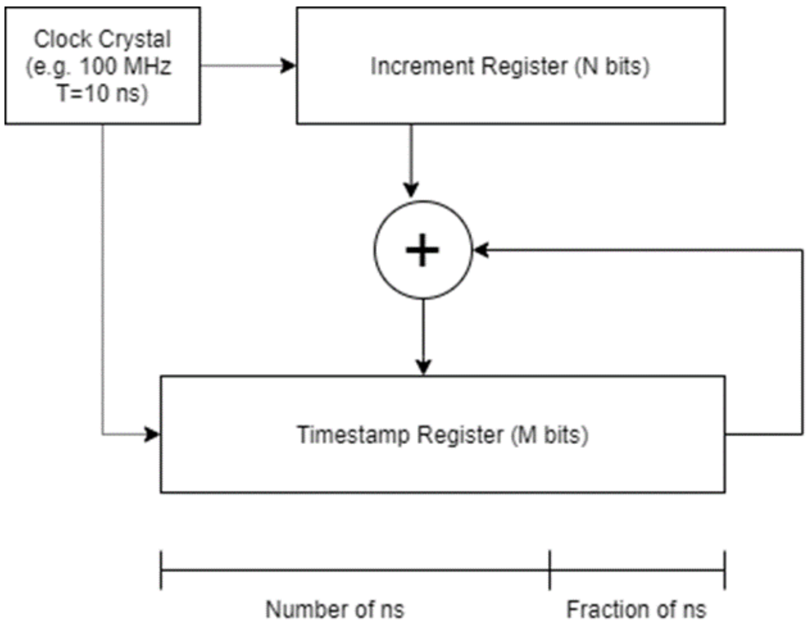
Clock Crystal (e.g. 100 MHz T=10 ns)
Increment Register (N bits)
+
Timestamp Register (M bits)
Number of ns
Fraction of ns

Figure 3.3: Advanced circuit for storing and maintaining local time – clock drift and counter conversion to time solved using configurable increment register

If this value is chosen (or dynamically adjusted) appropriately, it can eliminate both clock drift and the need to convert the value of the counter to time. Consider, for example, an oscillator with a frequency of 100 MHz (T = 10 ns). If the value 10 is stored in the increment register, then the value of the counter will directly represent the time in ns. In addition, consider the inaccuracy of an oscillator of 100 PPM, when the actual time of the clock signal period will be T * 10^{-4} ns shorter (frequency is higher). Then to compensate for clock drift, the value of the increment register must be set to T * (1 - 10^{-4}).

3.1.2 Jitter

The stable deviation of the clock signal period (defined through PPM) is not the only problem of real oscillators. In fact, the period of the clock signal may change dynamically for a particular oscillator (as shown in Figure 3.4). These dynamic changes are usually due to changes in the oscillator's environment, most often temperature. The resulting deviation over time in the length of the clock period is called clock jitter.

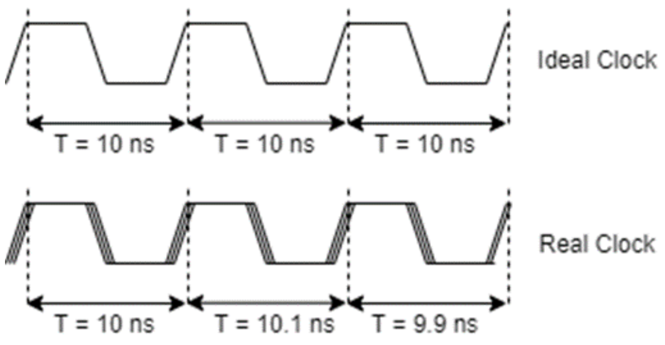

Figure 3.4: Clock jitter – comparison of the ideal and real clock signals

The adverse effects of clock jitter can be partially reduced through a stable oscillator environment, such as an air-conditioned room or box where the measuring node is located. Unfortunately, it cannot be completely eliminated. In practice, this means that the local time of a given measuring node must be periodically compared with the exact time source and synchronised with it.

3.1.3 Adjustment

From the point of view of the measuring node, the update of the local time should be continuous, without sudden jumps to the future or the past. This means that for every two timestamps assigned to successive packets p1 and p2, then it must hold that timestamp (p1) < timestamp (p2), even if a local clock update took place in between these two measurements. Using a simple approach, in which the local time is always overwritten by the precise global time at each time synchronisation, does not guarantee this property, which leads to non-negligible issues for detailed telemetry, such as in-order packets appearing to cross a switch out-of-order, or large (and entirely artificial) jumps appearing in jitter measurements (see Figure 3.5).

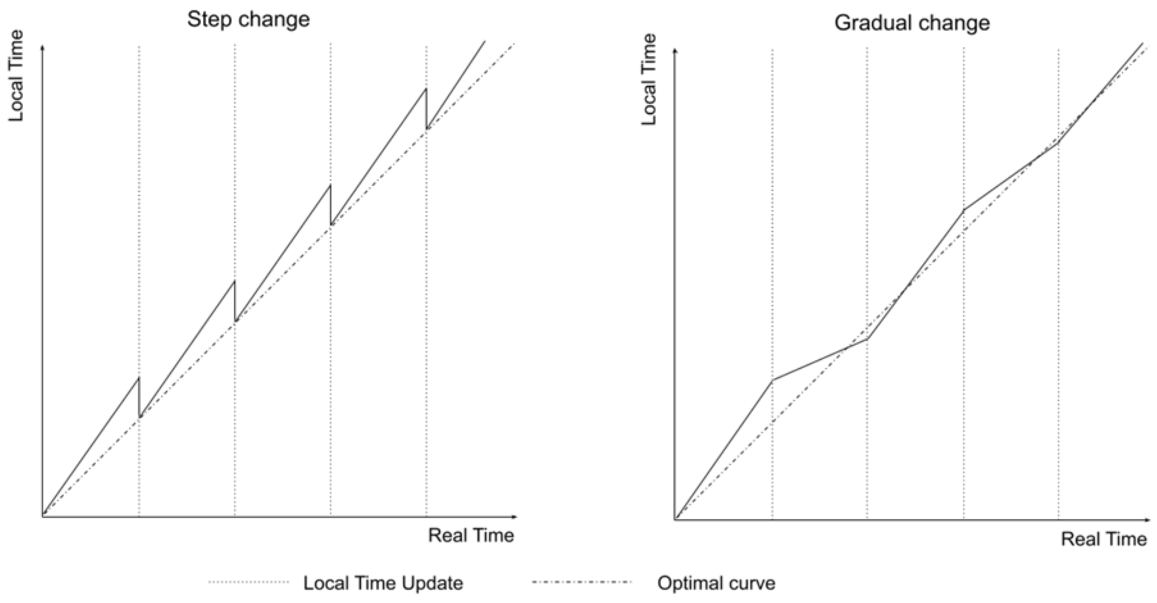

Figure 3.5: Local time update – step vs. gradual change of local time value

In practice, a gradual change of the local time is ensured, for example, by a suitable update of the value of the increment register. Changing this value allows the gradual slowing (or, conversely, acceleration) of a local clock to bring it into and keep it in alignment with the precise global time.

3.2 Getting Precise Time and Its Distribution

There are various sources of precise time. They are usually represented as a hierarchical model consisting of a semi-layered system of time sources. Each level of this hierarchy is termed a stratum, with higher levels within the hierarchy representing higher accuracy of the corresponding time sources: the highest layer, stratum 0, represents atomic clocks [ACW], then stratum 1 corresponds to nodes connected to stratum 0, and so forth. The signal may be received by satellites or radio and, more recently, can be received using an optical fibre.

3.2.1 Time Source from Satellites

One of the easiest ways to get accurate time information for a measurement node is to use an antenna to collect information sent by satellite. The signal provides time and position information used to compute the receiver's position and absolute time at high precision. Known services are Global Positioning System (GPS) and Global Navigation Satellite System (GNSS) [GPS]. This receiver usually collects the information sent by more than one satellite, each containing a stratum 0 time source (atomic clock). The receiver also generates a Pulse per Second (PPS) signal, which represents a short pulse (signal goes high for a short period of time) generated once per second. The accuracy of this signal generation is in the order of tens of nanoseconds [PPS] depending on receiver quality and also on connection with the external antenna.

The high accuracy of the PPS signal can then be used by the measuring node to synchronise its local clock. Through the accurate PPS signal, the deviation of the local time from the exact time can be measured periodically (e.g., every second) and the increment register value can be adjusted appropriately. This is referred to as the clock servo algorithm.

3.2.2 Protocols to Distribute Time

Despite its simplicity, the use of GPS has several drawbacks, including:

- The GPS receiver must be connected to an outdoor antenna that has a direct line of sight to various satellites. This can pose challenges, especially when measurement nodes are located in server rooms and basements of buildings.
- Limited scalability – if there are multiple measurement nodes within a local network requiring clock synchronisation, then connecting each of them individually to GPS receivers is complex and expensive.

To overcome the scalability problem in particular, specialised network protocols have been designed and implemented in the past to distribute precise time information over LAN/MAN networks. The basic idea of these protocols is to minimise the number of nodes connected and synchronised through GPS and to distribute the precise time information from these (master) nodes to the remaining (slave) nodes through the available computer network. Examples of these protocols that have also become a standard are Network Time Protocol (NTP) [RFC 1059] and Precision Time Protocol (PTP) [IEEE 1588-2008]. Each of these is described below.

3.2.2.1 *Network Time Protocol (NTP)*

NTP was one of the first protocols to be defined for synchronising nodes in a computer network. The first specification was released in 1988 (RFC 1059) and the fourth version, NTPv4 (RFC 5905) [RFC 5905], is currently in widespread use.

Clock synchronisation is realised through the NTP protocol, which is based on the client-server model, where clients (nodes requiring clock synchronisation) poll one or more NTP servers (precise time sources). Based on the information obtained from the server(s), the clients calculate their own time offset against the precise time source and adjust their local clocks.

Communication between the client and the server takes place as follows (see Figure 3.6):

1. The client sends a packet with a request to the server and records the timestamp at the moment of packet transmission, T1 (client time domain).
2. The server receives the packet and records the timestamp of the moment of its arrival, T2 (server time domain).
3. The server then sends a response to the client that includes both T1 and the timestamp of the moment the outgoing packet leaves the server, T3 (server time domain).
4. The client receives the response and records the timestamp of its arrival, T4 (client time domain).

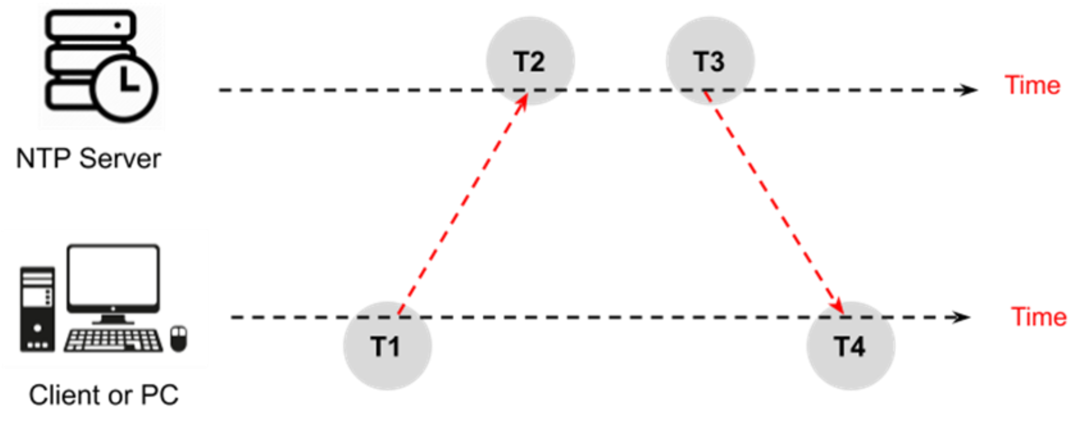

Figure 3.6: Clock synchronisation using Network Time Protocol (NTP)

Based on the obtained timestamps T1, T2, T3 and T4, the client calculates the mean path delay between client and server as:

$$\text{MeanPathDelay} = [(T2-T1) + (T4-T3)] / 2$$

Based on the mean path delay, the client can calculate the offset of its local (inaccurate) time according to the server's (precise) time and then adjust its local clock accordingly.

$$\text{TimeOffset} = \text{Server Time} - \text{Client Time} - \text{MeanPathDelay}$$
$$\text{TimeOffset} = T2 - T1 - [(T2-T1) + (T4-T3)] / 2$$
$$\text{TimeOffset} = [(T2-T1) - (T4-T3)] / 2$$

The accuracy of local clock synchronisation via NTP depends on the network environment in which it is deployed, and several factors affect that accuracy. The MeanPathDelay calculation assumes equal delay in both directions of communication; the delay between client and server can vary, depending on the load on the network elements along the path; finally, timestamps are assigned at the software level on the client and server side, which may add non-negligible and non-constant delays. Thus, the variable delay between the arrival of a packet at the physical server/client interface and the assignment of a timestamp after its reception in software is also reflected in the MeanPathDelay and TimeOffset calculations.

That said, in typical R&E networks, where there are well-provisioned backbones and network equipment and campus/site networks with gigabit or better networking to server systems, it is not unusual to see synchronisation within or below a millisecond or two. Data observed from perfSONAR deployments support this observation. However, if the network environment is not so good, perhaps with congestion, high jitter, asymmetric paths, poorly specified network hardware, or even poorly chosen NTP servers, the synchronisation accuracy might rise to in the order of a few milliseconds [Wang 2008].

3.2.2.2 *Precision Time Protocol (PTP)*

The PTP protocol (IEEE 1588 standard, latest version 2019 [IEEE 1588-2019]) is a more recent alternative to NTP that attempts to remove its disadvantages and improve clock synchronisation accuracy. The basic idea based on measuring the mean path delay and calculating the time offset between client and server is retained. One of the main changes, however, concerns the way the T1 to T4 timestamps are captured. Whereas in NTP they were sampled at the software level, in PTP they are assumed to be measured at the hardware level, as close as possible to the physical network interface. This step eliminates the variable packet transmission delay between the physical network interface and the software.

Measuring timestamps on incoming packets is relatively easy to implement at the hardware level. The timestamp is taken at the time the packet transitions between the PHY and MAC layers and is attached to the packet. For outgoing packets, the situation is much more challenging, as it is not easy to simultaneously add a timestamp to the packet just sent and recalculate checksums at different protocol layers. PTP solves this situation by sending the timestamp through the Follow_Up downstream packet.

The overall communication protocol diagram is shown in Figure 3.7. There are also several other changes from the NTP protocol:

- The server and client nodes have been renamed to Master (or also Grand-Master) and Slave (or also Ordinary Clock).
- Communication is not initiated by the Ordinary Clock node but by the Grand-Master, which sends (via multicast or unicast) a Sync message at regular intervals to measure the path delay in the direction from the Grand-Master to the Ordinary Clock.
- In case the Ordinary Clock wants to adjust its local clock relative to the Grand-Master, it responds by sending a DelayRequest message, which in turn is used to measure the path delay in the direction from the Ordinary Clock to the Grand-Master.
- The last T4 timestamp obtained on the Grand-Master side is then sent towards the Ordinary Clock in a separate DelayResponse message.

Based on the knowledge of timestamps T1 to T4, the Ordinary Clock then calculates MeanPathDelay and TimeOffset according to the same equations as for NTP.

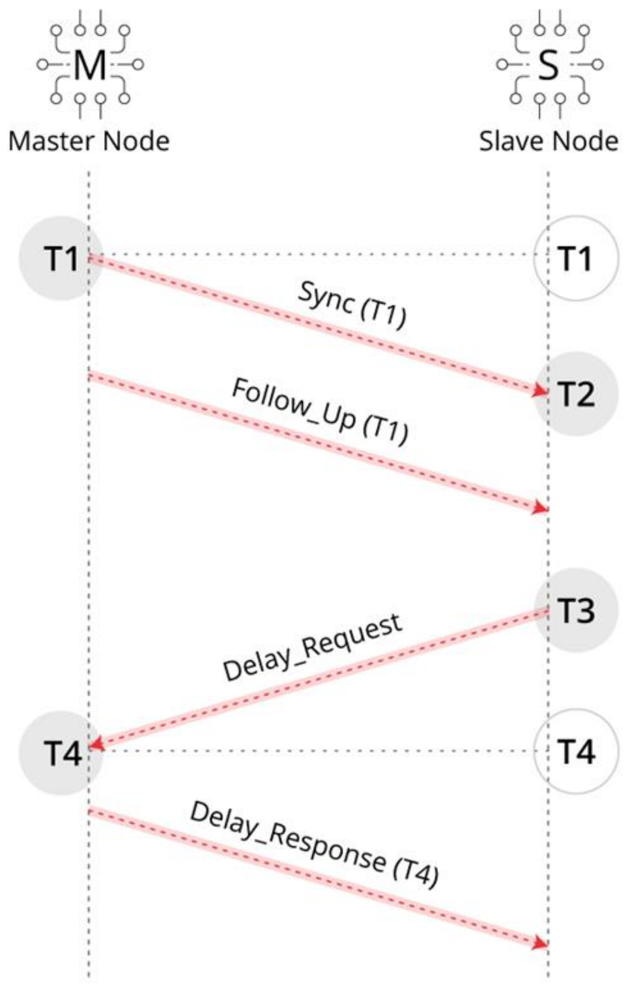

Figure 3.7: Clock synchronisation using Precise Time Protocol [NTPF]

Sensing timestamps at the hardware level is not the only benefit of PTP. As mentioned above, the accuracy of the MeanPathDelay measurement and TimeOffset calculation between an Ordinary Clock and Grand-Master is also negatively affected by the variable delay of switches that are placed along this path. This delay usually varies depending on the load on the switches and may be different for each direction of communication. One way to address this problem (defined by the standard IEEE 1588) is to set the switches to End-to-End Transparent Clock (TC-E2E) mode. In this mode, the switches detect Sync and DelayRequest messages on their ports, measure the time these messages take to pass through the switch (called residence time) and increment the correctionField entry inside these packets by the residence time value (see Figure 3.8). Ordinary Clock nodes can then easily calculate how much of the measured path delay is the variable part (spent inside switches) and refine the calculation for TimeOffset.

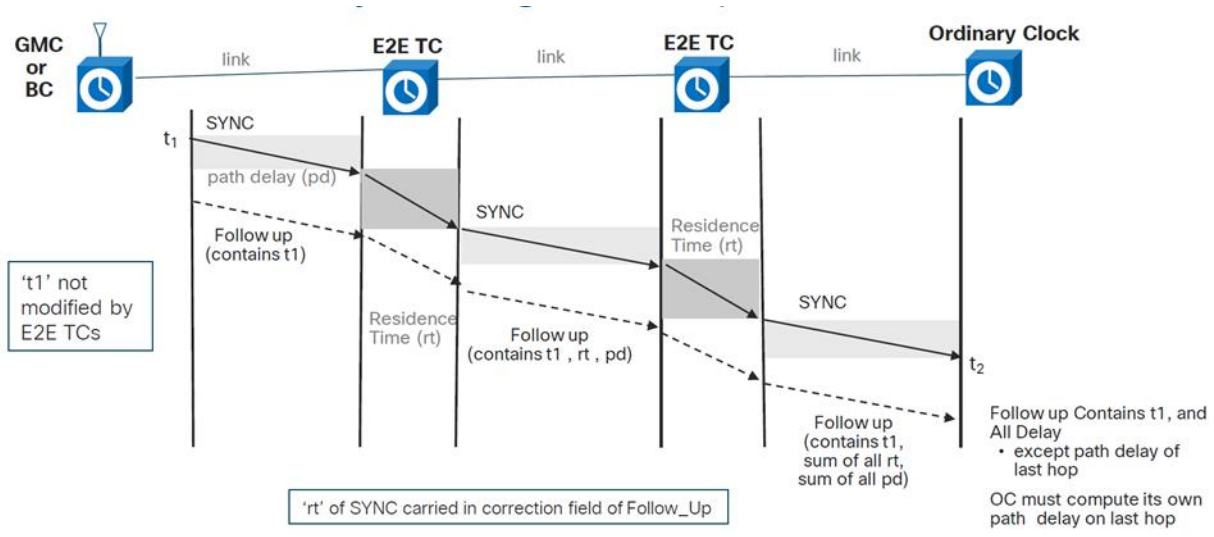
GMC or BC
E2E TC
E2E TC
Ordinary Clock
link
link
link
t_1
SYNC
path delay (pd)
Follow up (contains t1)
't1' not modified by E2E TCs
SYNC
Residence Time (rt)
Follow up (contains t1 , rt , pd)
Residence Time (rt)
SYNC
t_2
Follow up (contains t1, sum of all rt, sum of all pd)
'rt' of SYNC carried in correction field of Follow_Up
Follow up Contains t1, and All Delay
• except path delay of last hop
OC must compute its own path delay on last hop

Figure 3.8: Clock synchronisation using switches in End-to-End Transparent Clock mode [Mitchell_2019]

Variable switch delays are not the only problem. If the local network has a large number of Ordinary Clock nodes, then the Grand-Master node may be overloaded and must respond to incoming DelayRequest messages from all these Ordinary Clocks. One solution (defined by the IEEE 1588 standard) is to set selected switches on the path to the Grand-Master node to Boundary Clock (BC) mode. In this mode, the switch very precisely synchronises its internal clock with the Grand-Master node, creating an alternative source of accurate time within the LAN. It starts generating Sync messages on its (Slave) ports towards the Ordinary Clocks by itself. At the same time, it responds to DelayRequest messages itself and no longer sends them towards the Grand-Master.

Implementing the Boundary Clock mode inside switches is not easy. The switch must be equipped with a temperature-stabilised oscillator, ensure clock drift elimination, and also implement the PTP Best Master Clock (BMC) algorithm to select the best time source to synchronise against. Therefore, as an alternative to the Boundary Clock mode, the IEEE 1588 standard also defines the so-called Peer-to-Peer Transparent Clock (TC-P2P) mode. Similar to TC-E2E mode, under TC-P2P, Sync messages are sent from the Grand-Master node towards all Ordinary Clocks. For these Sync messages, the switches calculate the residence time and update the correctionField entry. The main difference lies in the handling of DelayRequest messages from the Ordinary Clock side. These messages are not sent towards the Grand-Master, but the switch in TC-P2P mode responds to them itself. To be able to respond correctly, it must maintain information about the current path delay between itself and the Grand-Master. It obtains this information through its own sequence of pDelayRequest and pDelayResponse messages with the Grand-Master node (see Figure 3.9). Note that when computing the path delay, it is not necessary for a switch in TC-P2P mode to have its local clock fully synchronised with the Grand-Master node and only the so-called syntonisation is sufficient.

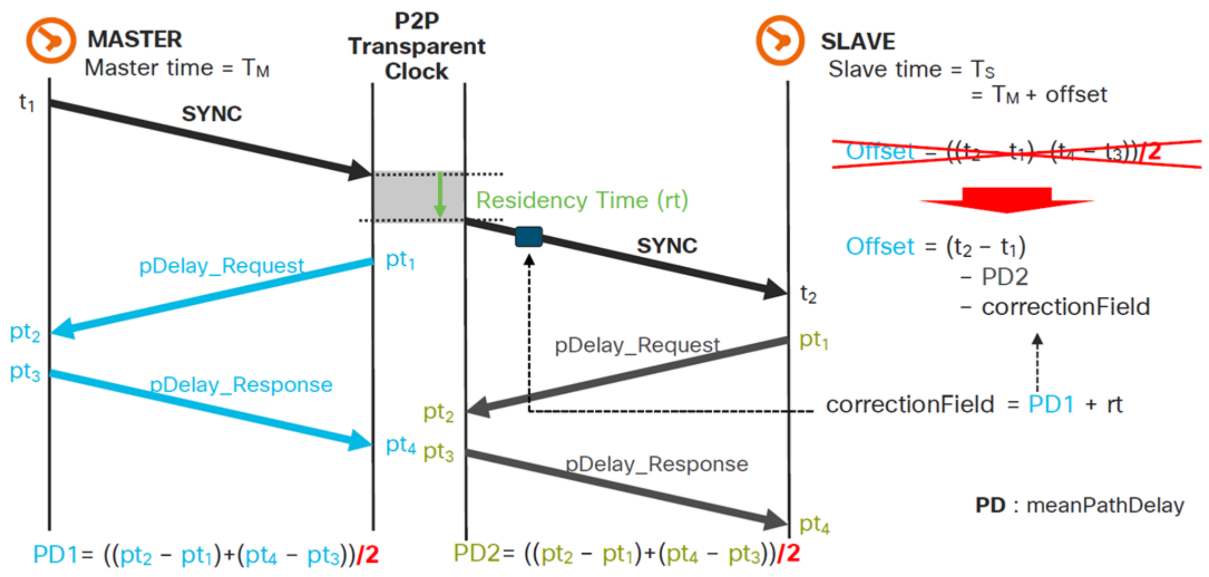

Figure 3.9: Clock synchronisation using switches in Peer-to-Peer Transparent Clock mode [Mitchell_2019]

Switches in TC-P2P mode can also be cascaded to further eliminate communication with the Grand-Master node. In addition, Boundary Clock, TC-E2E and TC-P2P switches can be combined in various ways within the LAN/MAN architecture, depending on the specific requirements of the operator. A complete description of these modes and their features, advantages and disadvantages is beyond the scope of this document and can be found, for example, in [Mitchell_2019].

By applying the above techniques, it is possible to achieve clock synchronisation accuracy in the order of a few microseconds or even sub-microsecond. However, hardware-level support is required not only on the Grand-Master and Ordinary Clock side but also in the switches that are along the path. This is discussed below.

Hardware Support for PTP Synchronisation

While PTP offers improved clock synchronisation and accuracy compared to NTP, it comes at the cost of requiring specific support in network device hardware.

At the end-node level (Ordinary Clocks), IEEE 1588 support requires network interface cards (NICs) to be able to:

1. Assign a timestamp to the selected outbound and inbound packets (Sync and DelayRequest) as close to the physical network interface (preferably between the MAC and PHY layers) as possible.
2. Maintain the local time that is further used as a source of timestamps, using, for example, the implementation based on a circuit with increment register and its setting option (as described in Section 3.1.1 Drift and Figure 3.3).

Full IEEE 1588 support can be efficiently implemented in conjunction with software. While the hardware identifies PTP packets and assigns timestamps to them, the software performs detailed processing, maintains a history of measured time offsets, and implements complex algorithms for increment register control and adjustment (clock servo algorithm).

In addition, by defining a generic interface at the operating system and network card driver level, it is possible to decouple the specific hardware implementation and create universal software to process the PTP protocol over any network card supporting this interface. Within the Linux operating system, these are specifically the ptp4l and phc2sys software tools [Ténart_2020].

PTP support at the switch hardware level is a bit more comprehensive compared to Ordinary Clocks. The standard defines several modes of switch operation (TC-E2E, TC-P2P, and Boundary Clock). The TC-E2E mode is the simplest in terms of implementation and does not place such high demands on the accuracy of the local clock. It is sufficient if the switch is able to identify PTP packets, calculate the residence time and edit the correctionField entry, including updating checksums. Similarly, TC-P2P mode support requires the switch to additionally periodically generate and process pDelayRequest and pDelayResponse messages and maintain the current delay value on the path to the Grand-Master node. The most challenging is Boundary Clock (BC) mode, which requires the switch to synchronise the local clock autonomously, process messages from Ordinary Clocks, and implement the Best Master Clock (BMC) algorithm.

While PTP support at the NIC level has become common and readily available, the situation is worse for switches. Switches with PTP support are rarely found on the market and are more expensive. Further, to achieve the highest synchronisation accuracy, PTP needs to be supported on as many switches as possible along the path between the Grand-Master and Ordinary Clocks nodes. It is not mandatory for all switches along the way to support PTP. However, each non-PTP switch reduces synchronisation accuracy because it is unable to calculate the residence time and update the correctionField in PTP messages. For the end node (Ordinary Clock), this complicates the TimeOffset calculation compared with the exact source.Therefore, PTP support is rarely encountered in current computer networks.

This situation could significantly improve in the future with the proliferation of P4-programmable switches, where the basic TC-E2E mode can be implemented quite easily by simply changing the P4 program.

3.2.3 The Choice between GPS, NTP and PTP

The choice of a particular technology for obtaining accurate time information depends on various factors. However, the most important one is the specific use case and its synchronisation accuracy requirements. This paper focuses on In-band Network Telemetry (INT) technology and its application for the purpose of measuring the quality of traffic flows. INT technology itself can be applied to different use cases and for some of them it requires the synchronisation of INT nodes (source, transit and sink).

As an example, for one-way delay (OWD) measurement, consecutive packets must have different timestamps on the source and destination nodes.

$$\text{SrcTstamp}(p_n) < \text{SrcTstamp}(p_{n+1})$$
$$\text{DstTstamp}(p_n) < \text{DstTstamp}(p_{n+1})$$

This requirement can be satisfied by a suitable implementation for local clock adjustment (see Section 3.1.3).

It is also mandatory that the time of delivery of a packet to the destination node is always higher than the time of emission from the source node.

$$\text{SrcTstamp}(p_n) < \text{DstTstamp}(p_n)$$

The actual communication delay between the source node and the destination node is expressed as:

$$\text{Delay}(p_n) = \text{DstTstamp}(p_n) - \text{SrcTstamp}(p_n)$$

Both the source and destination nodes are subject to clock synchronisation error:

$$\text{SrcTstamp}(p_n) + \text{SrcError} < \text{DstTstamp}(p_n) + \text{DstError}$$

In the worst case, the maximum source and destination node error may be added and exceed the "real" delay:

$$2 \times \text{MaxError} > \text{Delay}(p_n)$$

With regard to the NTP protocol, as stated in Section 3.2.2.1, the accuracy of synchronisation is dependent on the network environment – the network properties and configuration – in which the protocol is used. In R&E networks, which are usually well-provisioned, high speed, well-configured and well-specified, the accuracy achieved can be within or below a millisecond or so.[1] In poor network environments, e.g. with congestion, high jitter, asymmetric routing and poor NTP installation, it may rise to the order of a few milliseconds.

To measure communication delay using INT within a LAN or MAN, INT with precision lower than a few milliseconds, nodes synchronised via PTP or GPS should be used.

Accuracy requirements can be further tightened based on the requirements of the application being monitored. For example, when monitoring remote videoconference calls, the end-to-end latency requirement is below 100 ms, including possible video compression and decompression at the end nodes. This typically leaves tens of ms for the network transmission itself [Baldi_2000]. In the Low Latency (LoLa) application [LoLa], interactivity is preserved only when delay is below 30 ms. Similarly, the accuracy requirements for applications from the 5G area (self-driving cars, augmented reality, smart city cameras, IoT sensors) and from the edge/cloud computing area (gaming, AR/VR experience, media/CDN) often mandate a maximum communication latency below 10 milliseconds [IEEES].

The above suggests that some use cases would greatly benefit if the measurement nodes were synchronised using PTP or local GPS, even if both technologies require hardware support. PTP should be preferred due to its very good scalability and the requirement for only one or two stratum zero sources. The use of GPS implies positioning an antenna with a clear view of at least 3 to 5 satellites, which is not easily achievable in many environments.

[1] This level of accuracy is particularly remarkable given that one-way delay in fibre between neighbouring countries within Europe is typically in the range of 10–20 ms and between continents is around 100 ms [VER].

4 Practical Considerations

In the work for this paper, the authors have focused on the analysis of available P4 platforms and their capabilities in terms of timestamp sensing and clock synchronisation. Not only software platforms were tested (e.g., BMv2, T4P4S, p4c-dpdk, p4c-ebpf, p4c-ubpf and p4c-xdp) but also hardware platforms based on SmartNIC (e.g., P4 to NetFPGA, Netcope P4 and Netronome) or P4-programmable switch architectures (e.g., Intel Tofino). A detailed description of findings is given in Appendix A. This section gives a brief overview of them, including practical implications for the development of applications such as INT. The parameters considered are:

- Timestamp formats, ranges, and their meanings.
- Where in the P4 architecture/model the timestamps are captured/inserted.
- Where in the INT Platform the timestamps are actually captured and inserted.
- The source of the timestamps.
- How the time source is synchronised.

4.1 Timestamp Formats, Ranges, and their Meanings

Different platforms use different timestamp formats and ranges. The most commonly used format is where the stored value represents the number of time units (microseconds, nanoseconds) or units derived from the clock signal source used (e.g., 1/156 MHz = 6.4 ns) that have elapsed since the last counter reset.

The platforms also differ in the number of bits reserved for storing the timestamp (usually 32, 48 or 64 bits). The number of bits automatically defines the maximum value of time units that can be stored and the point at which the timestamp overflows (wraps around) and the count starts again.

There are also differences in the start of the measured period (epoch). While some platforms have this moment adjustable, others define it fixed (e.g., midnight 1/1/1970) or relative to the moment of system start or switch start.

Platform	Resolution [bits]	Meaning	Epoch
BMv2	48	number of microseconds	switch start
Tofino v1	48	number of nanoseconds	adjustable
Netcope P4	64	number of nanoseconds	adjustable

Platform	Resolution [bits]	Meaning	Epoch
NetFPGA	64	number of units derived from the clock signal source (8 ns or 6.4 ns)	since the FPGA boot time

Table 4.1: Examples of platforms and their timestamps

When creating a custom application using timestamps, these specific characteristics have to be taken into account. When combining timestamps from different platforms, the situation is even more complex because some differences in timestamp format may be reconciled at the P4 program level (using arithmetic operations) while others can be resolved at a higher level (e.g., a control plane application).

Specifically, INT does not define a specific format for the timestamps. It only defines the size of the space to store them. INT specification 1.0 [INT_SPEC1.0] acknowledges the complexity:

> *"The exact meaning of the following metadata (e.g., the unit of timestamp values, the precise definition of hop latency, queue occupancy or buffer occupancy) can vary from one device to another for any number of reasons, including the heterogeneity of device architecture, feature sets, resource limits, etc. Thus, defining the exact meaning of each metadata is beyond the scope of this document. Instead we assume that the semantics of metadata for each device model used in a deployment is communicated with the entities interpreting/analysing the reported data in an out-of-band fashion."*

It seems that it could be beneficial if platforms used the same timestamp format as the standardised time synchronisation protocols (NTP or PTP). Unfortunately, even these protocols do not share the same or compatible timestamp format with each other.[2]

When implementing an INT application, the timestamp format compatibility problem may be addressed in several ways:

1. Base all the INT nodes on the same platform, thus using the same format and timestamp range.
2. Convert the timestamp to a uniform format already at the level of individual INT nodes. This may be feasible, especially for open-source software-based platforms where this change can be added. For hardware-based platforms, on the other hand, this approach may be quite complex or even impossible.
3. Use native timestamps within individual INT nodes and perform their conversion to a uniform format at the INT collector level. This approach is easy to implement; however, it may place an excessive burden on the end node performing the conversion and storing the data.

[2] NTP defines the timestamp as 64 bits, where the upper 32 bits represent the number of seconds and the lower 32 bits represent the number of fragments within a second. For PTP, 80 bits are used, where the upper 48 bits represent the number of seconds and the lower 32 bits represent the number of nanoseconds within a second.

4.2 Where in the P4 Architecture/Model the Timestamps Are Captured/Inserted

Different platforms use different P4 architectures/models. The specific model then defines at which points in time the timestamp is assigned to the packet. Two approaches are typically encountered:

1. The timestamp is assigned to the packet when it transits through specific parts of the architecture, typically when it enters the ingress and egress pipeline; this applies to v1model, Portable Switch Architecture (PSA), or Tofino Native Architecture (TNA). The P4 program, in this case, does not affect the timestamp acquisition moment.
2. The timestamp is obtained based on a call to an external function; applies to SimpleSumeSwitch, uBPF model, XDP model. The P4 program directly defines when the timestamp should be taken and stored in metadata.

The specific requirements of the application being developed will suggest which is the most suitable model. For example, an INT application requires that packet timestamps be assigned on entry to and exit from the switch as close to the physical interface as possible. From this perspective, platforms based on v1model, PSA, or TNA are best since a timestamp is assigned to the incoming packet before it enters the pipeline (standard metadata for the ingress part). In addition, some hardware platforms are able to assign a timestamp to a packet even earlier, i.e., when the packet transits between the PHY and the MAC block (e.g., Tofino or Netcope P4). For outgoing packets, the timestamp corresponding to the packet entering the egress part of the model is usually used. To increase accuracy, this timestamp can be adjusted in the P4 program by the expected time spent in the egress pipeline.[3] Although some architectures are able to take a timestamp even when leaving the Deparser/MAC block (TNA), it can usually no longer be inserted into the packet currently being sent, and therefore it cannot be used for the INT protocol.

Choosing a model where the timestamp is obtained through a call to an external object (SimpleSumeSwitch, uBPF model, or XDP model) may permit a more accurate measurement of the moment when the packet leaves the pipeline. On the other hand, the accuracy of measuring the packet arrival may be lower, depending on the rules for calling the external block.

For platforms that use only the ingress pipeline (e.g., NetFPGA, Netcope P4), it is necessary to also use the ingress timestamp for outgoing packets, and possibly increase it by the expected time spent in the pipeline. The pipeline then ends with a Traffic Manager block, which may add unpredictable delay to the packet, reducing the accuracy of the INT measurement.

Finally, it should be noted that not all platforms actually implement timestamping (e.g. T4P4S, p4c-dpdk, and p4c-ebpf). Some of the backend compilers do not support it and leave timestamps unset in the packet metadata. If it is an open-source project, this functionality can only be added by additional in-house coding.

[3] The accuracy of the timestamp correction depends on the complexity and variability of the processing in the egress pipeline. The resulting accuracy of the timestamp can be affected in the order of units to tens of nanoseconds.

4.3 Where in the INT Platform the Timestamps Are Actually Captured and Inserted

For P4-programmable hardware devices, such as SmartNICs and switches (Tofino, NetFPGA, Netcope P4, Netronome), the model used usually corresponds to the actual circuit architecture. Timestamps are then assigned, for example, between the PHY and the MAC block for incoming packets and when entering the egress pipeline for outgoing packets.

Software-based platforms (BMv2, T4P4S, p4c-xdp, p4c-ebpf, p4c-dpdk, etc.) emulate the P4-pipeline function in software, including timestamp scanning for incoming and outgoing packets. The time between the arrival of a packet on the physical interface (PHY) and the capturing of the timestamp (in the software) is not included in the timestamp. Similarly for outgoing packets. Depending on the system architecture and other factors, this delay can vary from microseconds to milliseconds and not be stable. Platforms that operate on lower software layers (e.g., XDP and eBPF) or platforms that bypass the kernel network stack entirely (e.g., T4P4S, p4c-dpdk, and p4c-ubpf) have the potential to achieve better measurement precision.

Measurement inaccuracies on software-based platforms could be partially avoided by using support for hardware timestamps at the NIC level (see Hardware Support for PTP Synchronisation in Section 3.2.2.2). Many of them can scan timestamps between the PHY and MAC layers for each incoming packet. Assigning timestamps to packets sent in hardware is more complicated as the captured timestamp must be inserted into the same outgoing packet and the checksums updated (see description of the 1-step procedure in Section 3.2.2.2). Unfortunately, none of the current P4-programmable platforms yet supports the possibility of hardware timestamps at the level of NICs.

In the case of an INT application, it is necessary to meet high requirements for the accuracy of measuring the moments of arrival and departure of the packet to/from the INT node, preferably during the transition between the PHY and MAC layers. This can currently only be met by hardware-based platforms (Tofino, SmartNICs). However, if one of the software-based platforms is used, then DPDK-based backends can be recommended, where kernel bypass is used, or XDP, where processing is still done at the driver level.

4.4 The Source of the Timestamps

When developing an application, it is important to understand the source of the timestamps and its accuracy. For software-based platforms, the source of time is usually the system time or a time derived from it. For hardware devices, it is an internal increment register-based timer (see Section 3.1.1, Figure 3.3). In both cases, the accuracy of the time source is affected by two main factors: the stability of the oscillator used and the synchronisation method.

If conventional clock crystals are used, their frequency may change abruptly based on changes in ambient temperature. It is therefore recommended to use a temperature compensated crystal oscillator (TCXO) or at least a thermostatic crystal oscillator (OCXO). Therefore, if high measurement accuracies are needed, it is strongly recommended to verify the accuracy and stability of the crystals used on the hardware motherboards before selecting a specific platform.

4.5 How the Time Source Is Synchronised

One of the most important factors affecting the accuracy of a time source is how it is synchronised. If the time source is not synchronised at all, then its accuracy gradually deteriorates over time due to clock drift and clock jitter (see Sections 3.1.1 and 3.1.2). The time source should therefore be synchronised by at least one of the commonly used methods (NTP, PTP, or GPS).

For all software-based platforms where system time is used as the time source, it is relatively easy to activate system time synchronisation using the NTP protocol. If the system also has a network card with hardware support for PTP, and other devices on the path support it, then the accuracy of system time synchronisation can be further improved using standard software tools such as ptp4l and phc2sys [Ténart_2020]. In this way, even for software-based platforms, relatively high synchronisation accuracies can be achieved.

In the case of hardware-based platforms, it is again possible to use standard time synchronisation tools using NTP or PTP if the manufacturer adds the necessary functionality and interface. However, for the platforms tested so far, the authors have not encountered such a standard interface, and manufacturers often choose their own (closed) implementation.

For INT applications where high measurement accuracy is required, it is recommended to synchronise the time source using PTP or GPS. Unfortunately, PTP support is not yet sufficiently widespread and implemented at the switch level. Similarly, synchronisation using GPS has its own problems (e.g., connecting an external antenna or backup). In such cases, NTP can be used as an alternative, but the quality of synchronisation may well not be sufficient for the specific application.

5 Conclusions

This white paper has surveyed existing P4-programmable platforms and explored their capabilities in terms of handling timestamps and clock synchronisation.

Currently, there are a number of different P4-programmable platforms, executable at different software levels or hardware-accelerated, where the P4 pipeline is implemented at the SmartNIC level or by core silicon in the switch.

The paper reports that existing platforms vary considerably in terms of support for timestamps. The differences are not only in the size and format of timestamps but also in the timing of when a timestamp is assigned to a packet during ingress and egress processing.

The accuracy of the timestamp is significantly affected by the exact time source used and the way it is synchronised. All these parameters need to be taken into account, and the capabilities of the specific platform need to be permitted to reach the requirements of the application.

An analysis of the available platforms shows that many of them do not yet support native timestamps and clock synchronisation (this applies especially to software-based platforms) and need to be programmed.

For all the SmartNIC-based platforms studied, timestamping is implemented, but the way their source is synchronised differs. While for some, there is no synchronisation at all, for others, only proprietary NTP-based synchronisation or an external PPS signal is used (Netcope P4).

An interesting case with support for timestamps and clock synchronisation is Intel's Tofino chip-based platform, which supports PTP synchronisation, including its distribution in Transparent Clock and Boundary Clock modes. The only drawback of this platform is the 48-bit timestamp (representing the number of nanoseconds), which overflows on average once every three days.

In the future, support for timestamps and clock synchronisation can be expected to improve on available platforms. For example, it would be very beneficial to standardise in the aforementioned platforms the format, scope, and interpretation of timestamps. Furthermore, evolving from NTP to a more accurate way of clock synchronisation based on PTP or newer is hoped for. Finally, it would also be useful to add interface support to hardware platforms towards standard and open software tools (such as ptp4l and phc2sys) and make the clock synchronisation process transparent and simple.

Appendix A P4-Programmable Platforms and Support for Timestamping/Clock Synchronisation

The original version of the P4 language, labelled v14, defined not only how to describe the programmable parts of the switch, such as the Parser and Match-Action Tables, but also the underlying/reference architecture of the target device, referred to as the “abstract switch model”. With the advent of version 16 of the language, however, important changes have been made. The language definition is strictly separated from the target architecture (targets). The specification focuses only on describing the basic programmable parts such as Parser, Match-Action Tables, and Deparser. The way these programmable parts are integrated within the target architecture, and how they are interfaced with fixed blocks, are left to the device manufacturer. Any specific functions are decoupled from the language using so-called extern blocks.

This step has enabled a significant extension of the P4 language not only to the switch area but also to SmartNICs or other network devices with architecture different from conventional switches, including pure software platforms based on different technologies (DPDK, eBPF, XDP, TC). The manufacturer of such P4-programmable devices has to provide the user not only with the specification of the architecture used but also with the backend part of the P4 compiler, which is used to translate the P4 code into the target platform.

The following sections give brief descriptions of the existing platforms and backend parts of compilers that are currently available. The descriptions focus primarily on their support in terms of timestamps and clock synchronisation. They discuss whether and in which parts of the architecture a timestamp can be obtained for a packet. They also consider the source of these timestamps and how it is synchronised.

The platforms are divided into the following categories:

- Software-based platforms.
- P4-programmable SmartNICs.
- P4-programmable switches.

A.1 Software-Based platforms

A.1.1 BMv2

Behavioral model version 2 (BMv2) is a reference P4 software switch meant to be used as a tool for developing, testing, and debugging P4 data plane and control plane software written for them [BMv2]. It is based on the v1model [V1M], which corresponds to the original "abstract switch model" of P4.14 and allows easier translation of original P4.14 programs to P4.16. The v1model is made up of the Parser, Ingress Pipeline, Traffic Manager, Egress Pipeline, and Deparser blocks (see Figure A.1). Within BMv2 it is implemented as a "simple_switch" program.

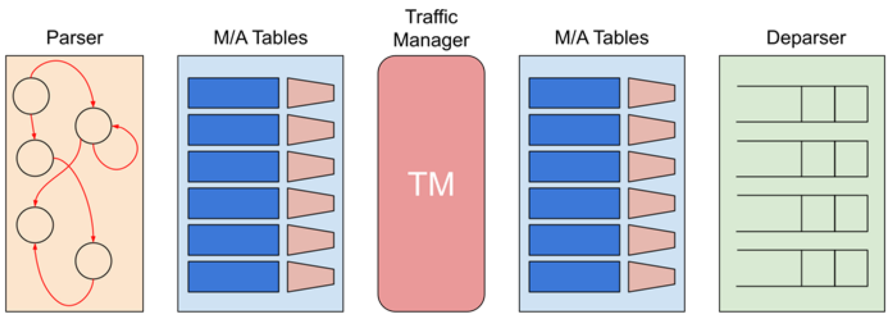

Figure A.1: v1model architecture

For completeness, a new generic switch architecture specification called Portable Switch Architecture (PSA) has also been created in P4.16 and is still in the draft stage [PSA]. BMv2 will implement it as a program called "psa_switch". Compared with the v1model, PSA is extended with a custom Parser and Deparser block in both the Ingress and Egress Pipeline (see Figure A.2).

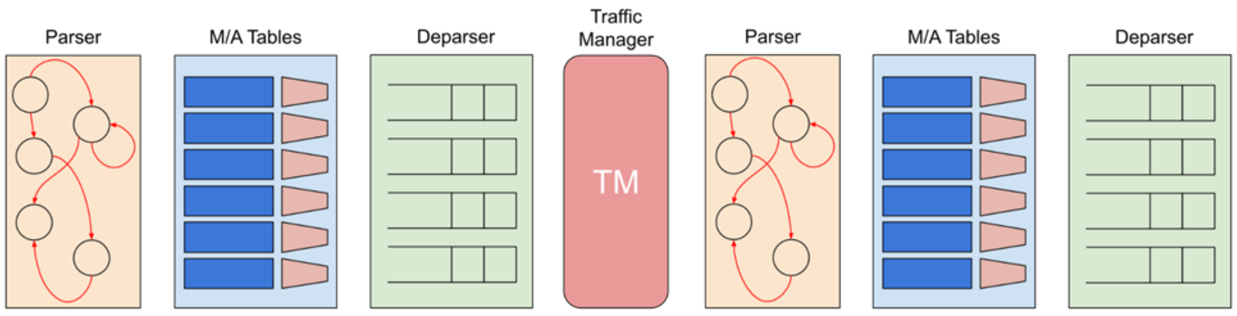

Figure A.2: PSA architecture

A.1.1.1 *Mapping P4 Architectures to Hardware*

It is important to understand how these architectures map to real hardware. In all software-based platforms, the hardware is usually a computer equipped with a network card with one or more network interfaces. Incoming packets enter the network card through the network interface and are subsequently transferred through the DMA to Host RAM. The CPU retrieves the packets from Host

RAM and performs processing. If the packet is also to be sent at the end of processing, it is stored back in a specific area of Host RAM. From there, the network card reads it using DMA transfers and then sends it via the network interface (see Figure A.3).

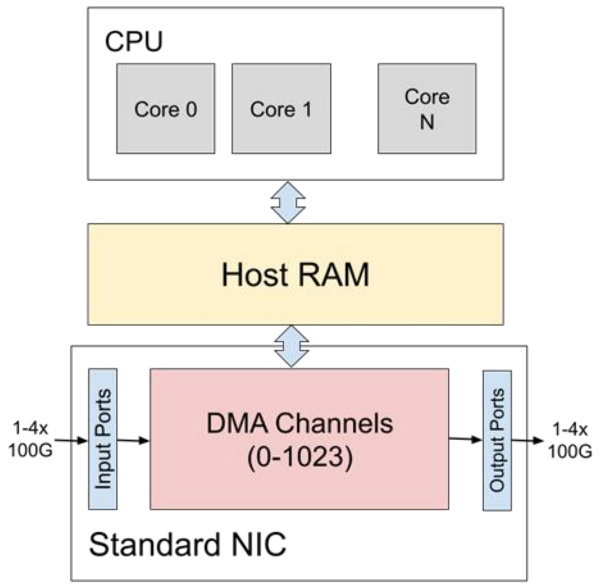

Figure A.3: Hardware architecture

Packet processing can be accelerated, if necessary, by distributing network flows to independent CPU cores. For this purpose, dedicated network cards with support for Receive Side Scaling (RSS) technology and independent DMA channels are required [RSS].

CPU cores run not only the packet processing applications but also the operating system. In conventional computer systems, packets are first processed through the network card driver and network stack inside the operating system kernel (kernel space) and then passed to the application (in user space). In the case of BMv2, the architecture of the v1 (or PSA) model and the generated P4 program is fully simulated at the application level in user space (see Figure A.4).

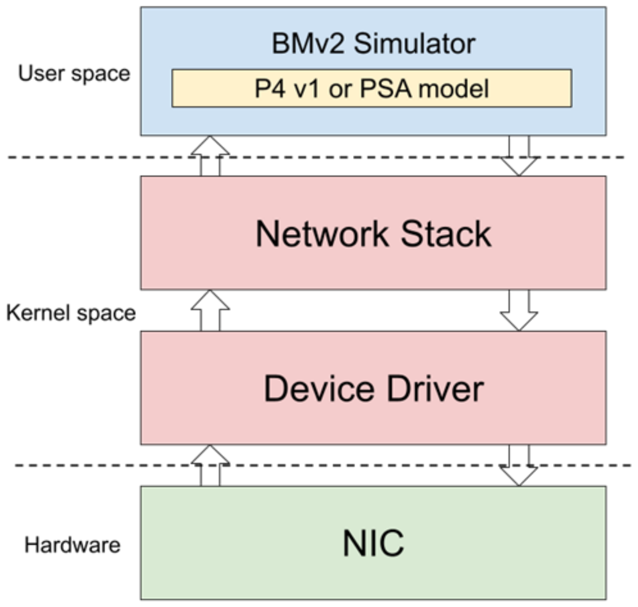

Figure A.4: BMv2 software architecture

A.1.1.2 *Timestamps and Clock Synchronisation*

With the v1model:

- When a packet arrives at the Ingress Pipeline, it is assigned a 48-bit timestamp indicating the number of microseconds since the simple_switch program started. Similarly when a packet enters the Egress Pipeline. Both timestamps are made available to the control program through standard metadata called ingress_global_timestamp and egress_global_timestamp.
- Within the BMv2 model, the source of the timestamps is a local timer measuring the number of microseconds since the start of program execution. If the user requires better global synchronisation, it is recommended to modify the BMv2 source code and use, e.g., system time, NTP, PTP, or HUYGENS [BMv2_TimeSync] [HUYGENS].

With PSA:

- Similar to the v1model, packets are given two timestamps when the packet enters the Ingress and Egress Pipelines. According to the specification, the Ingress timestamp should be assigned to the packet as close as possible to the packet's entry into the device itself, but no later than the entry into the Parser block. The specification does not define the number of bits to represent the timestamp and leaves it up to the manufacturer. The specification also does not require any specific way of synchronising the timestamp source.

Although BMv2 is described as a multi-threaded program, its processing capabilities through multiple cores are limited, and the overall performance is in the order of Gbps [BMv2_Performance]. Note: The goal of BMv2 was not to achieve high performance but to provide a quality simulator for P4 program developers, and this goal has been met.

The latency of a packet passing through the overall platform is variable and ranges from tens to hundreds of microseconds, depending on how the receiving and transmitting parts are implemented, the use of interrupts, and other factors. The actual passage of a packet through the kernel network stack is around 2 x ~26 us [Ahmad_2020].

Throughput and latency problems are addressed by competing P4 compilers using e.g., DPDK, eBPF, or XDP technologies.

A.1.2 T4P4S

Translator for P4 Switches (T4P4S) is another open-source software switch [T4P4Sg]. It converts P4 language source code into Data Plane Development Kit (DPDK) application code [T4P4S]. The v1model is used as the target architecture. One of the main advantages of the DPDK application is that the kernel network stack is bypassed completely during packet processing, and instead of the regular kernel device drivers, their alternatives in user space are used, called Poll Mode Drivers (see Figure A.5). This allows faster packet processing (lower latency) and also higher throughput. T4P4S thus forms an interesting alternative to BMv2, able to achieve throughput in the order of tens of Gbps [T4P4S].

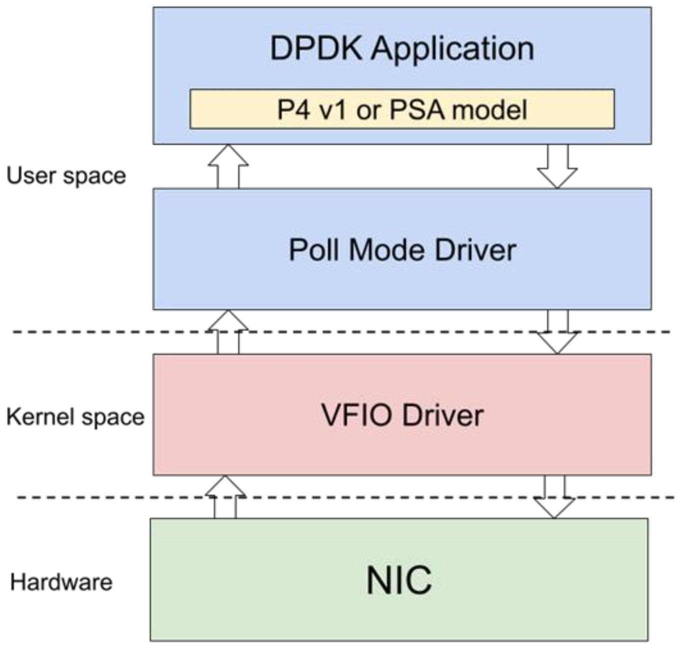

Figure A.5: T4P4S software architecture

A.1.2.1 *Timestamps and Clock Synchronisation*

Unfortunately, timestamps have not been implemented in the current version. According to the v1model definition, standard metadata includes timestamps, but the compiler keeps them uninitialised.

A.1.3 p4c-dpdk

The p4c-dpdk is an alternative open-source DPDK backend [p4c-dpdk] [p4c-dpdkg] with a similar software architecture to T4P4S (see Figure A.5). It translates the P4.16 programs to DPDK API to configure the DPDK software switch (SWX) pipeline [SWX]. Similar to T4P4S, it translates the P4 program to a representation that conforms to the DPDK SWX pipeline and generates the "spec" file to configure the DPDK pipeline. So, the output of the compiler is not the native DPDK code but the configuration that is interpreted through the DPDK pipeline. PSA is used as the target architecture.

A.1.3.1 *Timestamps and Clock Synchronisation*

Similar to T4P4S, the timestamps defined in PSA architecture have not been implemented, and the current version of the compiler keeps them uninitialised to zeros.

A.1.4 p4c-ebpf

This backend translates the P4.16 source code into an extended Berkeley Packet Filter (eBPF) program (a subset of C language) [eBPF]. If the resulting program passes strict verification, it can be run inside the Linux kernel, e.g., as a user-programmable packet filter within the kernel's Traffic Control (TC) subsystem. This approach allows P4 programs to run at the kernel level to achieve lower processing latency (see Figure A.6).

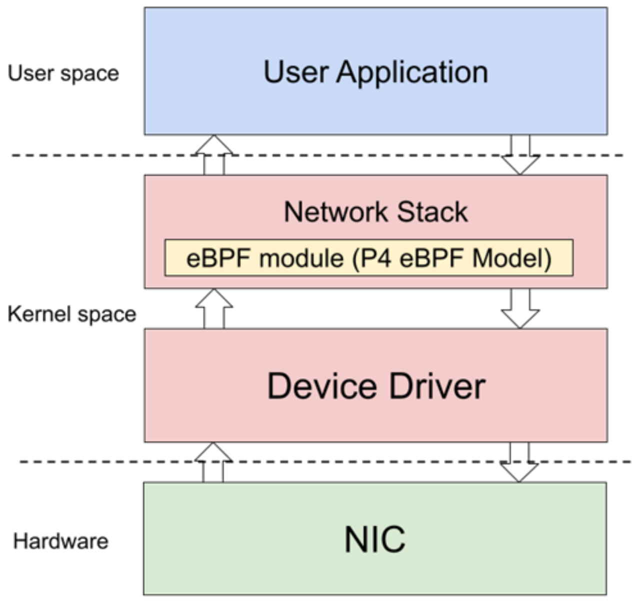

Figure A.6: eBPF software architecture

The target architecture is the eBPF model (see Figure A.7), which is composed of programmable Parser and Match-Action blocks [EBPFm]. Since eBPF is used only in the form of a packet filter, the architecture does not contain a Deparser, and the primary output of the pipeline is a boolean true/false value defining whether or not to drop an incoming packet.

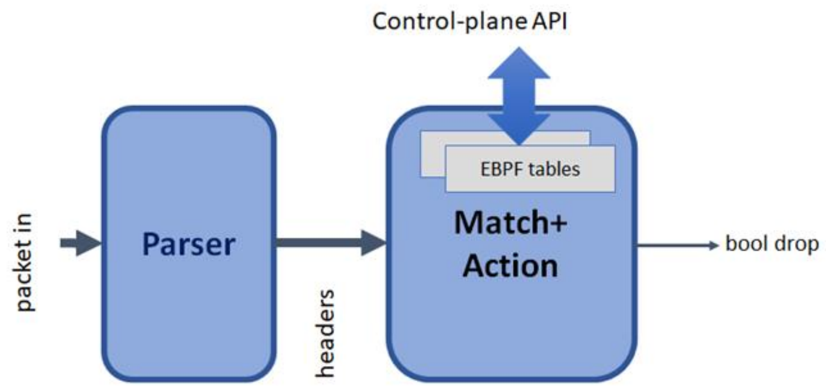

Figure A.7: eBPF model architecture

A.1.4.1 *Timestamps and Clock Synchronisation*

Unfortunately, there is no support for timestamps or clock synchronisation in this model.

A.1.5 p4c-xdp

XDP (eXpress Data Path) is a technology that allows the eBPF program already at the device driver level to run and incoming packets to process before they enter the kernel network stack (see Figure A.8) [XDP]. In contrast to the eBPF filter, which is part of the TC subsystem, packets can also be edited in XDP and forwarded to specified output ports. In addition, placing XDP at the device driver level further reduces packet processing latency.

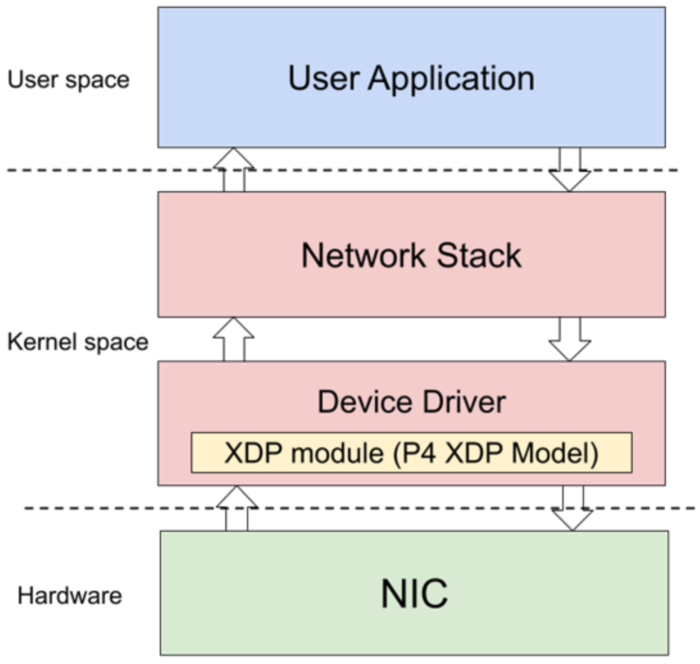

Figure A.8: XDP software architecture

The p4c-xdp backend, like p4c-ebpf, translates the P4.16 source code into an eBPF program. However, the XDP model (see Figure A.9) is used here as the target architecture, which also includes a Deparser block [XDPm]. Thus, it is also possible to modify the input packets, including encapsulation/decapsulation.

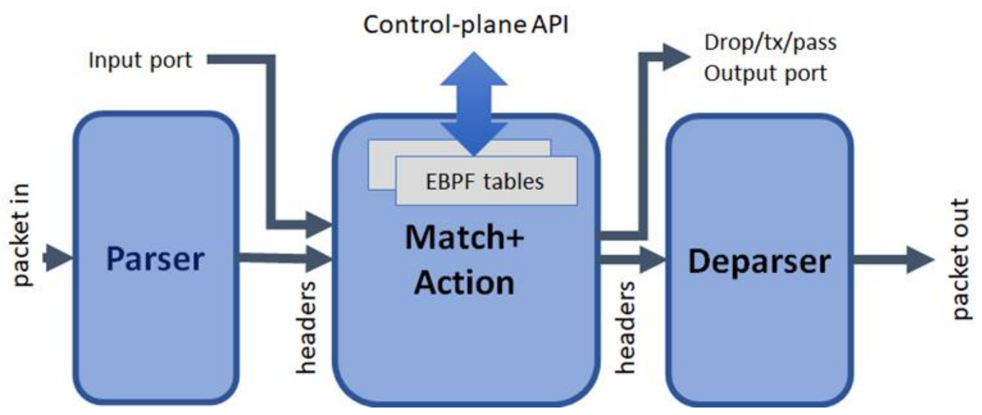

Figure A.9: XDP model architecture

A.1.5.1 *Timestamps and Clock Synchronisation*

In contrast to common switch architectures such as v1model or PSA, timestamps are not available as part of the metadata but can only be obtained by calling the external bpf_ktime_get_ns() function, which returns a number indicating the number of nanoseconds since system boot [KTIME]. Despite the fact that this function returns a 64-bit number, the XDP model so far only uses the lower 32 bits, and thus it overflows approximately once per 4 seconds.

A.1.6 p4c-ubpf

For completeness, there is also uBPF (userspace BPF) and the corresponding p4c-ubpf backend [uBPF]. This is an alternative to eBPF, where the generated program is executed (interpreted) in user space. Compared with eBPF or XDP, there are no such strict restrictions on, e.g., stack size, and debugging of the generated program is also more friendly. uBPF can then be used in any user space application, including the DPDK environment, to implement packet filters.

The target architecture of the p4c-ubpf backend is the uBPF model (see Figure A.10), which includes a Deparser, and it is almost identical to the XDP model (except for some changes at metadata level) [uBPFm]. It is thus possible to modify input packets, including encapsulation/decapsulation.

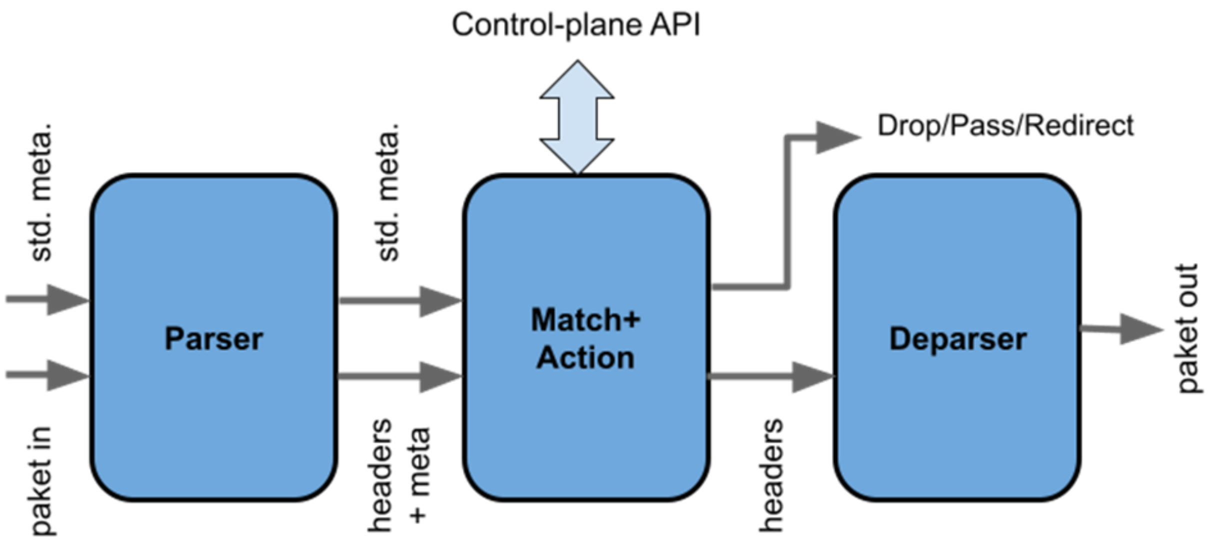

Figure A.10: uBPF model architecture

A.1.6.1 *Timestamps and Clock Synchronisation*

Similar to XDP, timestamps are not available as part of the metadata but can be obtained by calling an external function ubpf_time_get_ns(), which returns a 48-bit number indicating the number of nanoseconds.

A.2 P4-Programmable SmartNICs

A.2.1 FPGA-Based

A.2.1.1 *P4 to NetFPGA*

P4 to NetFPGA represents a workflow and development environment for compiling P4 programs on the NetFPGA SUME board which provides four SFP+ ports [NetFPGA]. The development environment is built around the P4-SDnet compiler [SDNet] and the SDnet data plane builder from Xilinx, i.e., a licence for the Xilinx Vivado design suite is needed. Custom external functions can be implemented in a hardware description language (HDL) such as Verilog and included in the final FPGA program. This also allows external IP cores to be integrated as P4 externs in P4 programs. The P4 to NetFPGA toolchain supports P4.16 based on the SimpleSumeSwitch architecture [NetFPGAg].

The SimpleSumeSwitch model (see Figure A.11) consists of three programmable blocks – the Parser, Match-Action Tables, and Deparser – and one fixed Traffic Manager block. It is an ingress-pipeline-only architecture, which is particularly suitable for the SmartNICs domain where a higher number of ports and switching between them is not expected.

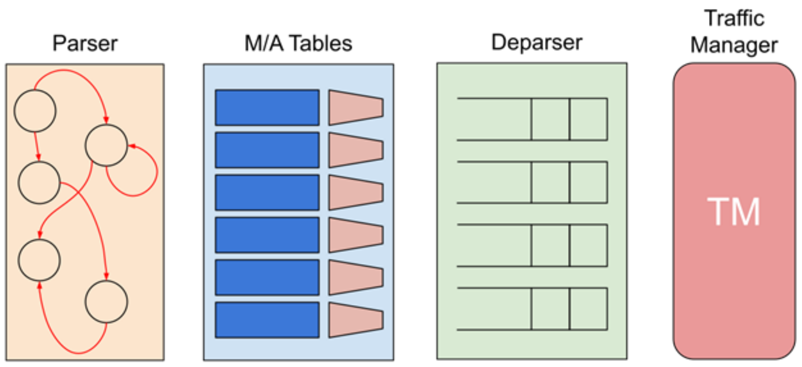

Figure A.11: SimpleSumeSwitch architecture

Timestamps and Clock Synchronisation

Timestamps are not part of the standard metadata but can be obtained through a dedicated external block. The timestamp source is a 64-bit counter running at the frequency of the input Ethernet interface (125 MHz for 1 Gbps or 156 MHz for 10 Gbps). The timestamp, therefore, indicates the number of ticks in units of 8 ns or 6.4 ns, depending on the interface used.

In the default (simple) implementation, there is no clock drift correction [NetFPGAT1]. In later versions of the NetFPGA platform, the Direct Digital Synthesis technique (equivalent to the advanced circuit in Figure 3.3) is used for correction, with the possibility of adjusting the incremental constant and reducing clock drift. However, the incremental constant setting is implemented based on experimental comparison with another clock signal source, which reduces the resulting accuracy and ability to fully eliminate clock drift [NetFPGAT2].

In the current version of the platform, there is no synchronisation capability using NTP or PTP. While there is work focused directly on implementing IEEE 1588 PTP for the NetFPGA card [NetFPGAPTP], this is not directly part of the default development environment, and its availability may be limited.

A.2.1.2 *Netcope P4*

Netcope P4 is a commercial cloud service that creates FPGA firmware from P4 programs [Netcope_P4]. As a target platform, Netcope P4 supports FPGA boards from Netcope, Silicom, and Intel that are based on Xilinx or Intel FPGAs. The v1model is used as the target architecture for compiling P4 code but without egress pipeline support. This model is therefore architecturally very similar to SimpleSumeSwitch, but the structure of the standard metadata is more like v1model.

Timestamps and Clock Synchronisation

The architecture used assigns a timestamp to each incoming packet when it enters the MAC layer. This is then inserted into the P4 pipeline as part of the standard metadata called ingress_timestamp. Since the model does not contain an egress pipeline, there is no timestamp corresponding to the packet entering the egress part.

The timestamp itself is 64 bits in size and represents time in nanoseconds. A hardware timer is used as the timestamp source, with the possibility of modifying the initial value and incremental constant (see Figure 3.3). The synchronisation of the time source (calculation of the incremental constant) is done either based on the Pulse Per Second (PPS) signal, fed from the GPS receiver to a special card input, or through the NTP protocol using Netcope proprietary software running on the host CPU. Time synchronisation via the IEEE 1588 PTP protocol is not yet supported within the Netcope development environment.

A.2.2 NPU-Based

Network processing units (NPUs) are software-programmable ASICs that are optimised for networking applications. They are usually part of standalone network devices or SmartNICs.

A.2.2.1 *Netronome*

Netronome network flow processing (NFP) silicons can be programmed with P4 or C [Netronome]. A C-based programming model is available that supports program functions to access payloads and allows P4 externs to be developed. The Agilio P4C SDK consists of a toolchain including a backend compiler, host software, and a full-featured integrated development environment (IDE). All current Agilio SmartNICs based on NFP-4000, NFP-5000, and NFP-6480 are supported. The v1model is used as a reference architecture for P4 programs.

Timestamps and Clock Synchronisation

The NFP platform assigns two different timestamps to each packet and includes them in intrinsic_metadata structure: (i) ingress_global_timestamp – a 64-bit value representing the time the packet entered the MAC layer, and (ii) current_global_timestamp – a 64-bit value representing the current global timestamp with respect to the time within the MAC block. For both timestamps, the most significant 32 -bits provide the time in seconds while the least significant 32 bits provide a number of nanoseconds [NetronomeTS]. The time is initialised when the host machine is powered on.

At the time of writing (August 2022), the authors do not know what the source of the time is, whether it is possible to reduce its clock drift or whether it can be synchronised via NTP or PTP protocols.

A.3 P4-Programmable Switches

A.3.1 Intel/Barefoot Tofino-Based Switches

The Intel/Barefoot Tofino ASIC implements the so-called Tofino Native Architecture (TNA), which comprises identical ingress and egress pipelines, including a configurable Parser, up to 12 processing stages, and a configurable Deparser in each, on each end of the switching matrix [Tofino]. Note that this architecture is extremely similar to that of PSA, but the resources of each pipeline are shared among a small number of physical ports, and with this being a representation of a hardware circuit, the processing stages cannot grow as much as desired but are constrained by the available resources in the ASIC. Compared with PSA, it provides support for advanced device capabilities, including a richer set of externs such as RegisterAction extern, low-pass filters, weighted random early discard externs, powerful hash externs that can compute CRC based on user-defined polynomials, ParserCounter, and others.

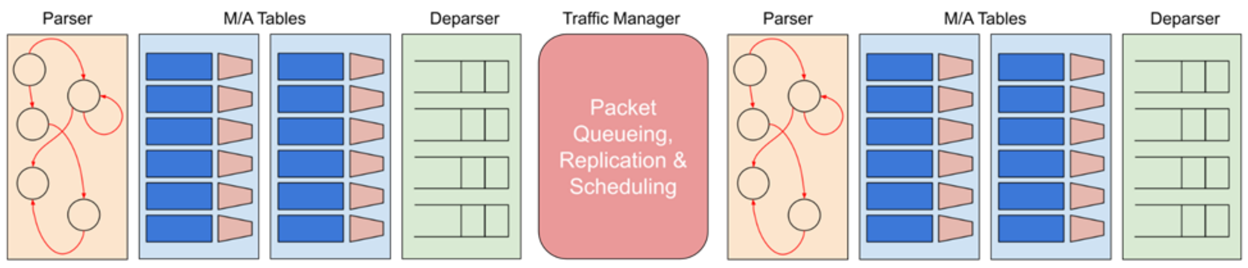

Figure A.12: Tofino native architecture

Timestamps and Clock Synchronisation

Tofino chips have their own internal clock, used in conjunction with a configurable offset and step value to eliminate clock drift and maintain an internal data structure counting nanoseconds from a certain epoch. The timestamps derived from these global clocks have a resolution of 48 bits and are captured in different parts of the pipeline. First, a timestamp is assigned to the packet at the transition between the PHY and MAC layers of the physical ingress port and stored as part of the ingress intrinsic metadata structure. The second timestamp is assigned to the packet when it enters the ingress pipeline, and it is stored as part of the ingress intrinsic metadata from the Parser. The last timestamp is assigned to the packet when it enters the egress pipeline, and it is stored as part of the egress intrinsic metadata from the Parser.

In addition to these three timestamps, it is possible to use additional information about the time a packet spends inside a Traffic Manager block (when transitioning between ingress and egress pipelines). The user gets this information in the form of enq_tstamp (time taken when the packet is enqueued) and deq_timedelta (time delta between the packet's enqueue and dequeue time) entries, which are stored as part of the egress intrinsic metadata. In contrast to the 48-bit primary timestamps, these entries are 18 bits long.

Tofino chips are also fully prepared to support clock synchronisation via the IEEE 1588 PTP protocol. As mentioned above, a timestamp can be taken for any incoming packet when it enters the MAC layer. On the other hand, the timestamp at the time an outgoing packet transitions between the MAC and PHY layers can only be recorded for selected outgoing packets. The user must specify this intent by setting the capture_tstamp_on_tx bit within the egress intrinsic metadata for the output port. Then the captured timestamp is readable from the control plane side.

The switch architecture is also ready to support Transparent Clock (End-to-End, Peer-to-Peer) and Boundary Clock modes. The correctionField and UDP checksum can be updated when sending selected event message packets (Sync, DelayReq). The user specifies this intent by setting the update_delay_on_tx bit within the egress intrinsic metadata for the output port. Subsequently, the user must also supply information about the offset of the UDP checksum and correctionField entries, including the new correctionField value, via the PTP metadata structure.

Although the format of the hardware timestamps differs from the IEEE 1588 standard (48 bits counting a number of nanoseconds vs. 80 bits containing a number of seconds and number of nanoseconds within a second), it can be expected that the necessary conversions will be performed at the control plane level since the frequency of PTP messages is in the order of units per second.

A.3.2 Others

Gradually, other manufacturers of P4-programmable switches are appearing. One example is the Spectrum 2 switch from NVidia/Mellanox. The P4 architecture of this target differs from common models such as PSA (see Figure A.13). It offers up to 5 programmable blocks – one Parser and four Control blocks (two in the ingress and two in the egress portion of the switch). The individual Control blocks typically differ in terms of P4 programmability [Mellanox]. Unfortunately, at the time of writing, the authors do not have detailed information about the capabilities of this architecture from a timestamps and clock synchronisation perspective. Similarly, there may be other manufacturers of P4-programmable switches and their architectures for which the authors do not have sufficient information.

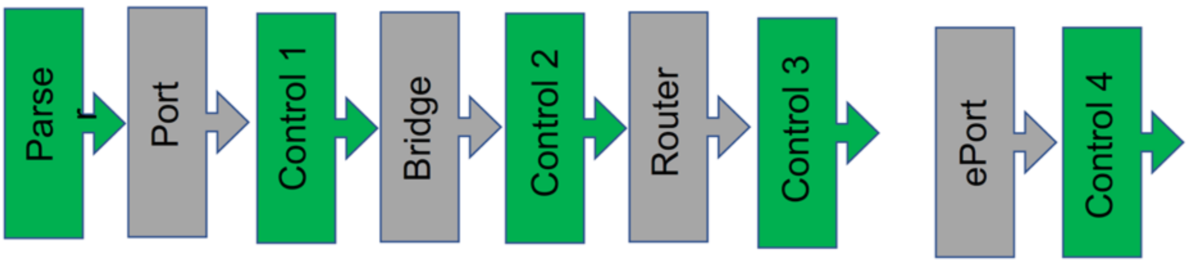

Figure A.13: Spectrum 2 architecture [Mellanox]

References

[ACW]	https://en.wikipedia.org/wiki/Atomic_clock
[Ahmad_2020]	M. Ahmad & A. Rizvi, "DPDK for ultra low latency applications", Userspace Summit, 2020, [Online]. Available at: https://www.youtube.com/watch?v=Rro5fl0sD0M, (Accessed: August 2021)
[Baldi_2000]	M. Baldi and Y. Ofek, "End-to-end delay analysis of videoconferencing over packet-switched networks," in *IEEE/ACM Transactions on Networking*, vol. 8, no. 4, pp. 479–492, Aug. 2000, doi: 10.1109/90.865076
[BMv2]	https://github.com/p4lang/behavioral-model
[BMv2_Performance]	Performance of bmv2 [Online]. Available at: https://github.com/p4lang/behavioral-model/blob/main/docs/performance.md, (Accessed: August 2021)
[BMv2_TimeSync]	The BMv2 Simple Switch target [Online]. Available at: https://github.com/p4lang/behavioral-model/blob/main/docs/simple_switch.md, (Accessed: August 2021)
[DPTP]	Pravein Govindan Kannan, Raj Joshi, Mun Choon Chan, "Precise Time-synchronization in the Data-Plane using Programmable Switching ASICs", SOSR '19: *Proceedings of the 2019 ACM Symposium on SDN Research*, DPTP Github https://github.com/praveingk/DPTP
[eBPF]	https://github.com/p4lang/p4c/tree/main/backends/ebpf
[eBPFm]	https://github.com/p4lang/p4c/blob/main/backends/ebpf/p4include/ebpf_model.p4
[GPS]	https://en.wikipedia.org/wiki/Satellite_navigation
[HUYGENS]	Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum and Amin Vahdat, "Exploiting a natural network effect for scalable, fine-grained clock synchronization". In *Proceedings of NSDI*, 2018. Available at: https://www.usenix.org/conference/nsdi18/presentation/geng
[IEEE_1588-2008]	*IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Instrumentation and Measurement Society, ISBN 978-0-7381-5400-8, 2008
[IEEE_1588-2019]	*IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Instrumentation and Measurement Society, ISBN 978-1-5044-6341-6, 2019
[IEEES]	https://spectrum.ieee.org/breaking-the-latency-barrier

[INT_SPEC1.0]	*In-band Network Telemetry (INT) Dataplane Specification* v1.0, https://github.com/p4lang/p4-applications/blob/master/docs/INT_v1_0.pdf
[INT_SPEC2.0]	*In-band Network Telemetry (INT) Dataplane Specification* v2.0, https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_0.pdf
[INT_SPEC2.1]	*In-band Network Telemetry (INT) Dataplane Specification* v2.1, https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf
[KTIME]	https://github.com/iovisor/bcc/blob/master/docs/reference_guide.md#3-bpf_ktime_get_ns
[LoLa]	https://lola.conts.it/
[Mellanox]	https://opennetworking.org/wp-content/uploads/2020/04/Itzik-Ashkenazi-Slide-Deck.pdf
[Mitchell_2019]	Albert Mitchell, *Precision Time Protocol – deep dive and use cases*, BRKIOT-2517, CISCO, 2019. Available at: https://www.ciscolive.com/c/dam/r/ciscolive/us/docs/2019/pdf/BRKIOT-2517.pdf
[Netcope_P4]	Netcope Technologies, a.s., *Netcope P4 User Guide (v 4.6)*, February 24, 2020
[NetFPGA]	https://cs344-stanford.github.io/lectures/Lecture-3-dev-tools.pdf
[NetFPGAg]	https://github.com/NetFPGA/P4-NetFPGA-public/wiki/Workflow-Overview
[NetFPGAPTP]	S. S. W. Lee, T. Lee and K. Li, “NetFPGA based IEEE 1588 module for time-synchronized software-defined networking”, *2016 6th International Conference on Information Communication and Management (ICICM)*, 2016, pp. 253–259, doi: 10.1109/INFOCOMAN.2016.7784253.
[NetFPGAT1]	G. Antichi, S. Giordano, D. J. Miller and A. W. Moore, “Enabling Open-Source High Speed Network Monitoring on NetFPGA”, *2012 IEEE Network Operations and Management Symposium*, 2012, pp. 1029–1035, doi: 10.1109/NOMS.2012.6212025.
[NetFPGAT2]	G. Antichi, D. J. Miller & S. Giordano, “An Open-Source Hardware Module for High-Speed Network Monitoring on NetFPGA”, 2010 [Online]. Available at: https://www.cl.cam.ac.uk/research/srg/netos/projects/netfpga/workshop/eurodev2010/antichi-wire/antichi.pdf
[Netronome]	https://www.netronome.com/products/smartnic/overview/
[NetronomeTS]	J.-O. Andersson, “Offloading INTCollector Events with P4” (Dissertation), 2019. Retrieved from: http://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-74508
[NTPF]	https://www.incibe-cert.es/en/blog/ntp-sntp-and-ptp-what-time-synchronization-do-i-need
[P4_Language]	https://p4.org
[p4c-dpdk]	C. Dumitrescu, A. Bas, “Enabling P4 in DPDK”, 2019 [Online]. Available at: https://www.youtube.com/watch?v=uI29_q-SoPU (Accessed: August 2021)
[p4c-dpdkg]	https://github.com/p4lang/p4c/tree/main/backends/dpdk
[PPS]	Xiaoji Niu, Kunlun Yan, Tisheng Zhang, Quan Zhang, Hongping Zhang & Jingnan Liu. (2014). “Quality evaluation of the pulse per second (PPS)

	signals from commercial GNSS receivers". *GPS Solutions*. 19. 141-150. 10.1007/s10291-014-0375-7.
[PSA]	https://p4.org/p4-spec/docs/PSA-v1.1.0.html
[RFC_1059]	RFC 1059 *Network Time Protocol (Version 1) Specification and Implementation* https://datatracker.ietf.org/doc/rfc1059/
[RFC_3393]	RFC 3393 *IP Packet Delay Variation Metric for IPPM* https://datatracker.ietf.org/doc/html/draft-ietf-ippm-ipdv
[RFC_5905]	RFC 5905 *Network Time Protocol Version 4: Protocol and Algorithms Specification* https://datatracker.ietf.org/doc/rfc5905/
[RSS]	https://docs.microsoft.com/en-us/windows-hardware/drivers/network/introduction-to-receive-side-scaling
[SDNet]	https://www.xilinx.com/support/documentation-navigation/development-tools/software-development/sdnet.html
[SWX]	https://doc.dpdk.org/guides/prog_guide/packet_framework.html?highlight=swx%20pipeline#the-software-switch-swx-pipeline
[T4P4S]	P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel and S. Laki, "T4P4S: A Target-independent Compiler for Protocol-independent Packet Processors", *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1–8, doi: 10.1109/HPSR.2018.8850752.
[T4P4Sg]	https://github.com/P4ELTE/t4p4s
[Ténart_2020]	Antoine Ténart, "Precision time protocol (PTP) and packet timestamping in Linux", Embedded Linux Conference Europe, October 2020. Available at: https://static.sched.com/hosted_files/osseu2020/c5/tenart-timestamping-and-ptp-in-linux.pdf
[Time_Def_Stds]	https://www.ipses.com/eng/in-depth-analysis/standard-of-time-definition/
[Tofino]	https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html
[uBPF]	https://github.com/p4lang/p4c/tree/main/backends/ubpf
[uBPFm]	https://github.com/p4lang/p4c/blob/main/backends/ubpf/p4include/ubpf_model.p4
[UTC]	https://en.wikipedia.org/wiki/Coordinated_Universal_Time
[V1M]	https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4
[VER]	https://www.verizon.com/business/terms/latency/
[Wang_2008]	Lei Wang, J. Fernandez, J. Burgett, R. W. Conners and Yilu Liu, "An evaluation of network time protocol for clock synchronization in wide area measurements", *2008 IEEE Power and Energy Society General Meeting – Conversion and Delivery of Electrical Energy in the 21st Century*, 2008, pp. 1–5, doi: 10.1109/PES.2008.4596234.
[WP_INT_Tests]	DPP White Paper: *In-band Network Telemetry Tests in NREN Networks* https://www.geant.org/Resources/Documents/GN4-3_White-Paper_In-Band-Network-Telemetry.pdf

[XDP]	W. Tu, F. Ruffy and M. Budiu, “Linux Network Programming with P4”. In Linux Plumbers’ Conference 2018, Vancouver, Canada, 2018. Available at: http://vger.kernel.org/lpc_net2018_talks/p4c-xdp-lpc18-paper.pdf
[XDPm]	https://github.com/vmware/p4c-xdp/blob/master/p4include/xdp_model.p4

Glossary

API	Application Programming Interface
AR/VR	Augmented Reality / Virtual Reality
ASIC	Application Specific Integrated Circuit
BC	Boundary Clock
BMC	Best Master Clock
BMv2	Behavioral model version 2
BPF	Berkeley Packet Filter
CDN	Content Delivery Network
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
DPDK	Data Plane Development Kit
DPP	Data Plane Programming
eBPF	extended Berkeley Packet Filter
FBK	Fondazione Bruno Kessler
FPGA	Field Programmable Gate Array
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HDL	Hardware Description Language
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
INT	In-band Network Telemetry
IoT	Internet of Things
IP	Internet Protocol
IPDV	IP Packet Delay Variation
IPPM	IP Performance Metrics
LAN	Local Area Network
LoLa	Low Latency
MAC	Media Access Control
MAN	Metropolitan Area Network
NFP	Network Flow Processing
NIC	Network Interface Card
NPU	Network Processing Unit
NREN	National Research and Education Network
NTP	Network Time Protocol
NTUA	National Technical University of Athens

OCXO	Oven Controlled Crystal Oscillator
OWD	One-Way Delay
P4	Programming Protocol-independent Packet Processors – a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packets
PHY	Physical
PPM	Parts Per Million
PPS	Pulse Per Second
PSA	Portable Switch Architecture
PTP	Precision Time Protocol
QoS	Quality of Service
R&E	Research and Education
RAM	Random Access Memory
RFC	Request for Comment (IETF)
RSS	Receive Side Scaling
SDK	Software Development Kit
SFP+	enhanced Small Form Factor Pluggable
T	Task
T4P4S	Translator for P4 Switches
TC	Traffic Control
TC-E2E	End-to-End Transparent Clock
TC-P2P	Peer-to-Peer Transparent Clock
TCXO	Temperature Compensated Crystal Oscillator
TNA	Tofino Native Architecture
uBPF	userspace Berkeley Packet Filter
UDP	User Datagram Protocol
UTC	Coordinated Universal Time
UvA	University of Amsterdam
VFIO	Virtual Function Input/Output
WP	Work Package
WP6	GN4-3 Work Package 6 Network Technologies and Services Development
WP6 T1	WP6 Task 1 Network Technology Evolution
XDP	eXpress Data Path