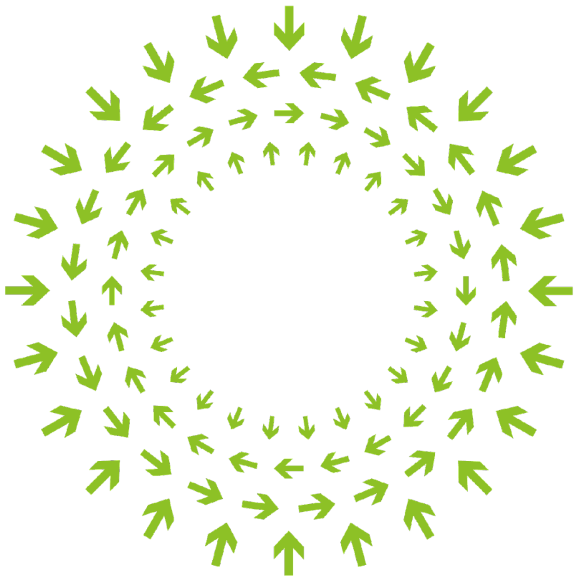

OPEN Compute Project

Simple Precision Time Protocol (SPTP)

1.0

Base Specification v1.1
Effective October, 2023

Author: Meta – Oleg Obleukhov
Meta – Alexander Bulimov
Meta – Ahmad Byagowi

Table of Contents

1 Introduction 3
2 Background 3
3 Method 5
3.1 Delay Request 6
3.2 Sync 6
3.3 Follow Up/Announce 6
3.4 Advantages and Simplifications 7
3.4.1 No negotiation between client and a server 7
3.4.2 Client driven exchange 7
3.4.3 Server fault detection 7
3.4.4 No state 8
3.4.5 Low resource consumption 8
3.4.6 Simple implementation 8
3.4.7 Using existing underlying hardware support 8
3.4.8 Distributed load resistance 8
3.4.9 Denial of Service Attack (DoS) protection 9
3.4.10 Mean path delay always calculated from latest exchange 9
3.4.11 Less flow control 10
3.4.12 No negotiation between client and a server 10
3.4.13 No multicast support 10
4 Results 10
4.1.1 Precision testing with single instance deployment 10
4.1.2 Precision testing with large-scale deployment 11
4.1.3 Performance testing with large-scale deployment 13
5 Discussion 15
6 Conclusion 16
7 Reference 16

Abstract — In this manuscript we describe a simplified implementation of the Precision Time Protocol (PTP) which is called Simple PTP (SPTP). Existing unicast profiles come with various extensions such as subscription, signaling messages, flow control and authentication methods. These extensions come at the price of increased complexity and performance penalties. However, in a typical data center application most of the extensions are unnecessary and often impossible to use. We tested the SPTP implementation at scale and showed that it is functional in comparison to mainstream unicast PTP implementations using G8265.1 and G8275.2 profiles.

Keywords—SPTP, PTP4I, Unicast, Open Time Server, Data center, Time Card, PTPv2, UDP

1 Introduction

In 2002, Precision Time Protocol (PTP) was introduced “as a successor for NTP” [1]. The standard, backed by IEEE 1588, was greatly revised in 2008 to allow wider definition of profiles [2]. The standard is flexible and allows to cover several use cases, such as unicast, multicast, and various negotiation and authentication methods. In this manuscript we will be mostly focusing on PTPv2 unicast UDP profiles.

Mainstream unicast profiles such as G.8265.1 and G.8275.2 have many design features from multicast profiles which are not necessarily useful for unicast. Such properties include various hand shaking messages and a “top to bottom” direction of communication. Service messages such as signaling with CANCEL_UNICAST_TRANSMISSION TLV [3] are designed to balance the client load and shift traffic based on the network conditions.

In the data center environment with the secure and private networks most of the aforementioned capabilities are redundant and create unnecessary complexity. Furthermore, since the introduction of the Open Time Server with the Time Card [4], most of the capacity limitations are no longer applicable.

It is possible to greatly simplify the Two-step PTPv2 exchange by eliminating the entire handshaking mechanism and transferring the packet exchange control to the client. This way we will be able to reduce network, memory, and CPU utilization on the server side. In addition, SPTP completely eliminates state machines which drastically increases reliability and simplifies client and server implementations.

Simple PTP packet exchange is binary compatible with IEEE 1588-2019 and it’s possible to implement using existing mechanisms, leveraging underlying hardware functionality such as hardware timestamping and Transparent Clock.

2 Background

A typical IEEE 1588-2019 two-step PTPv2 unicast UDP flow consists of the following exchange:

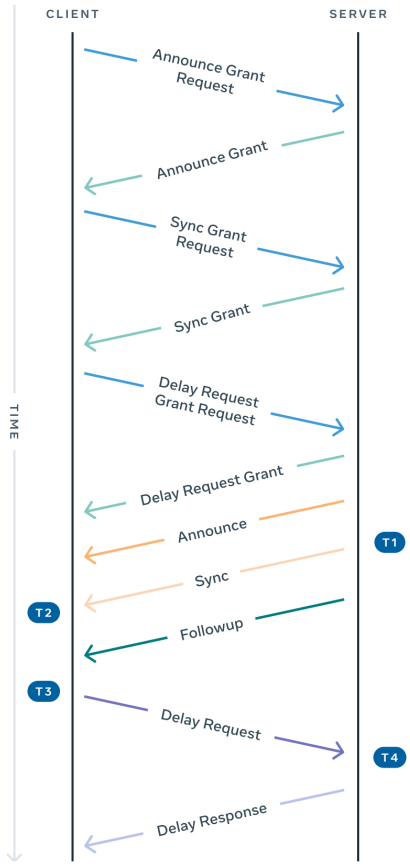

Figure 1. Typical two-step PTPv2 exchange

1. Client sends an Announce request to the server
2. Server responds with a grant or cancellation
3. Client sends a Sync request to the server
4. Server responds with a grant or cancellation
5. Client sends a Delay Request to the server
6. Server responds with a grant or cancellation
7. Server sends a Sync message
8. Server sends a Follow Up message
9. Server sends an Announce message
10. Client sends a Delay Request to the server
11. Server responds with a Delay response

This sequence repeats either in full or in part depending on the negotiation result. The exchange shown in Figure 1 is one of many possible combinations. It may involve additional steps such as grant cancellation, grand cancellation acknowledgements and so on. Depending on the implementation and configuration frequency of these messages may vary. After completing negotiation, the frequency of some messages is usually reduced.

This design allows for a great flexibility especially for less powerful equipment where resources are limited. In combination with multicast it allows to support a relatively large number of clients using very either old or embedded devices.

For example, a PTP server can reject the request or confirm a less frequent exchange if the resources are exhausted. However, this approach comes with the tradeoffs such as:

- Excessive network communication
- Multicast support requirement for large numbers of clients
- Unicast support has a strict capacity limit [5]
- State maintenance on both server and client side
- Individual clients have no control over the communication parameters
- Server driven decision
- These tradeoffs are limiting the data center application where typically communication is driven by hundreds of thousands of clients and multicast is not supported.
- With the introduction of Open Time Server, we can support large quantities of unicast clients operating at individual frequencies and exchange rates.

3 Method

By eliminating the handshake mechanism, we greatly simplify the PTP exchange (SPTP):

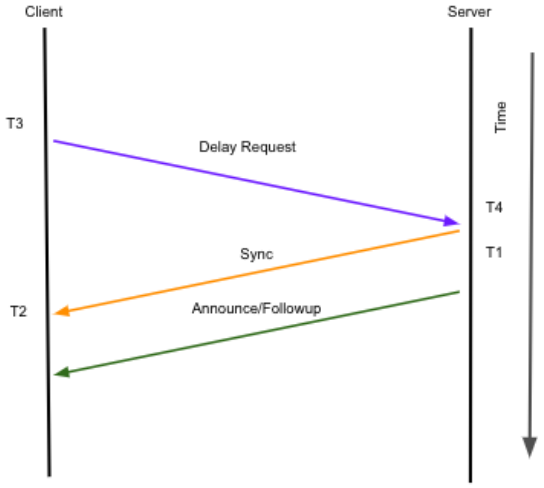

Figure 2. Typical SPTP exchange

1. Client sends a Delay Request
2. Server responds with a Sync
3. Server sends Follow up/Announce

In the flow shown in Figure 2 instead of 11 different network exchanges and the requirement for client and server state machines for the duration of the subscription, there are only 3 packets exchanged and no state needs be preserved on either side. In the simplified exchange, every packet has an important role.

3.1 Delay Request

Delay Request initiates the SPTP exchange. It's interpreted by a server not only as a standard Delay Request containing the correction field (CF1) of the transparent clock, but also as a signal to respond with a Sync and Follow Up packets. Just like in two-step PTPv2 exchange, it generates T3 on the departure on the client side and T4 on the arrival on the server side.
To distinguish between PTPv2 Delay Request and SPTP Delay Request the PTP profile Specific 1 flag must be set by the client.

3.2 Sync

In response to a Delay Request a Sync packet would be sent containing the T4 generated at an earlier stage. Just like in a regular Two-Step PTPv2 exchange Sync packet will generate a T1 on the departure of the server side. While in transit, the correction field of the packet (CF2) is populated by the network equipment.

3.3 Follow Up/Announce

Following the Sync packet an Announce packet is immediately sent containing T1 generated at a previous stage.

In addition, the correction filed from the Delay Request field is populated by the CF1 value collected at an earlier stage.

Announce packet also contains typical PTPv2 information such as Clock Class, Clock Accuracy and so on. On the client side the arrival of the packet generates the T2 timestamp.

After a successful SPTP exchange, default two-step PTPv2 formulas for mean path delay and clock offset must be applied:

$$mean_path_delay = ((T4 - T3) + (T2-T1) - CF1 -CF2)/2 \quad \text{(eq. 1)}$$

$$clock_offset = T2 - T1 - mean_path_delay \quad \text{(eq. 2)}$$

3.4 Advantages and Simplifications

The Simple PTP exchange has certain number of advantages and simplifications.

3.4.1 No negotiation between client and a server

For G.8275.2, various multicast profiles and many other profiles there are several constraints defined such as FOREIGN_MASTER_TIME_WINDOW and FOREIGN_MASTER_THRESHOLD [6] which significantly delay the initial synchronization.
Due to a missing handshake and a short exchange cycle the synchronization time is as little as 1.5 round trip time which leads to a faster restart and ready for use time.

3.4.2 Client driven exchange

Every single client has full control over the speed and quality of the exchange. For example, sending a Delay Request packet every 2 seconds will result in Sync and Follow Up/Announce responses every 2 seconds.
In case the client needs to increase frequency of synchronization, reducing the interval between Delay Request packets will help to achieve this without need of a renegotiation.

3.4.3 Server fault detection

To address reliability and failover concerns, client software implementation supports Best Master Clock Algorithm (BMCA) which may differ depending on a PTP profile. Using this algorithm, a client is able to select the best server based on several different criteria.

In the aforementioned PTP profiles, selection is happening based on the values sent by the server in the Announce message [7] without taking into account the actual synchronization quality such as mean path delay asymmetry or clock offset.

This may lead to a situation where a client is following a faulty server and has no chance of detecting it.

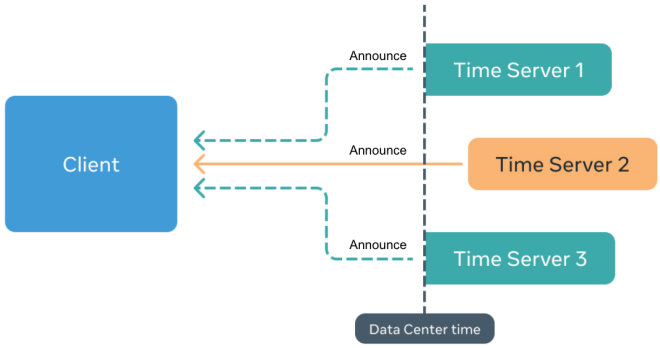

Figure 3. Client following faulty Time Server 2 based on announce

In SPTP, together with all the attributes from the Announce message, the client has access to the path delay and a calculated clock offset after every exchange with every server. And because the exchange is client-driven, the offsets could be calculated at the exact same time. Depending on the SPTP implementation this could be used to detect faulty servers based on quorum or even calculate a mean offset.

3.4.4 No state

Absence of in-memory state machine means both - server and a client could be restarted at any time without an impact on synchronization quality and need of the renegotiation.

3.4.5 Low resource consumption

Due to the absence of a state machine, there is a significant reduction in memory and CPU consumption on the server side. For example, maintaining the states of hundreds of thousands clients could consume significant resources whereas in SPTP it’s simply missing as a concept. Exchanging less packets could also significantly reduce network utilization. This manifests notably when frequency of signaling messages is high compared to SYNC. This facilitates performance improvements and directly impacts the absolute number of clients a time server can process.
The performance gains with SPTP vary significantly depending on the number of tracked time servers.
Most notably in the case of G8265.1 and G.8275.2 a client only receives a SYNC message from one out of N PTP servers. In case of a time server outage and a failover, load on remaining servers will increase proportionally until the last one supports the entire network. This may lead to a cascading failure also known as a domino effect.

In case of SPTP the amount of requests the time server receives will not be impacted by a failover event and will stay constant.

3.4.6 Simple implementation

The SPTP client and a server implementation is significantly simpler due to absence of the state machines transitions between the states.

3.4.7 Using existing underlying hardware support

SPTP is utilizing Delay Request and Sync UDP packets operated on port 319 and therefore automatically gets all the hardware support from the networking equipment such as Transparent Clock and Transmit and Receive hardware timestamps.

3.4.8 Distributed load resistance

Because of the complexity of the PTP profiles it's difficult to implement resilient clients.

With a traditional G.8275.2 subscription model, we observed issues related to a race condition, when server and client experience different load profiles. For example, when the client host has a significantly high load average, reading a Sync message may be delayed due to the process scheduling mechanism. Once the client host recovers, it will read several RX timestamps in a row which will negatively impact a core PI-servo assumption of a fixed and predictable synchronization interval. This could be mitigated in client software by additional extra validation steps which will vary from implementation to implementation and likely introduce a new dependency on a system clock [8].

Since SPTP relies on a client-driven exchange, it doesn't have this race condition by design. If the client doesn't send a Delay Request it will not get a Sync message. For example, during significant CPU utilization delayed packets will be dropped and new exchange initiated with usual frequency.

3.4.9 Denial of Service Attack (DoS) protection

In SPTP exchange the amplification attack ratio is 2.0 which is low and requires significant resources to sustain the attack.

3.4.10 Mean path delay always calculated from latest exchange

In complex multipath environments each packet can transition between different leaves of the network by anywhere from 1 to an infinite number of paths [9]. This means forward and reverse paths are likely going to differ and change with passage of time.

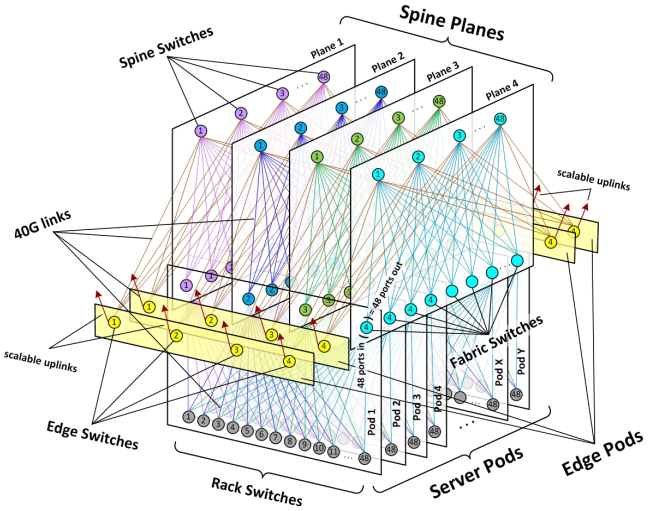

Figure 4. Fabric network layout

To ensure mean path delay calculation with eq. 1 and subsequent clock offset calculation in eq. 2 is correct it's preferred to measure forward and return paths as close as possible. SPTP

guarantees that by design as exchange guarantees all 4 timestamps being refreshed at the same time.

There are also tradeoffs which one needs to be aware of before deciding to use the SPTP:

3.4.11 Less flow control

It's impossible to set different intervals for different types of messages. For example, Sync and Follow Up/Announce are always bound together.

3.4.12 No negotiation between client and a server

Although the server has a right to ignore the Delay Request, there is no communication channel to suggest a reason or propose alternative parameters. For example, implementing client authentication may require introduction of additional exchanges which will neglect some of the performance and simplification gains.

3.4.13 No multicast support

SPTP is unicast only. There is no support for multicast-based communication. Non link-local multicast is not widely supported in datacenter networks and this ensures a simpler software implementation.

4 Results

There are several implementations of the SPTP deployed at large scale. Most notably the SPTP is a default precision time synchronization protocol at Meta.

4.1.1 Precision testing with single instance deployment

Initial deployments to a single client confirmed no regression in the precision of the synchronization:

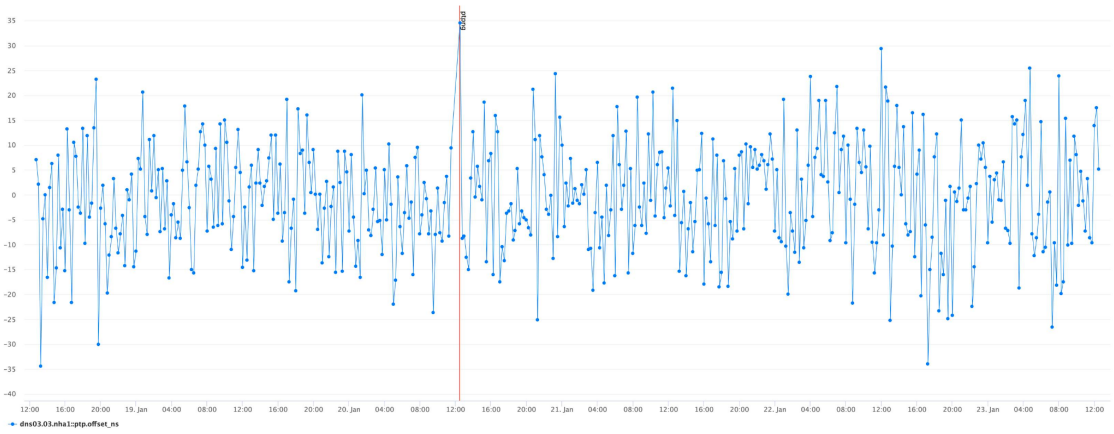

Figure 5. Clock offset after switching from ptp4l and SPTP

This is due to the fact that the same hardware capabilities and the mathematical formulas were used.
This is due to the fact that the same hardware capabilities and the mathematical formulas were used.

4.1.2 Precision testing with large-scale deployment

For the large-scale deployment we compared our own implementation of G.8275.2 and SPTP server software. We used the following validation framework:

Type	Number	Validation
PTP Servers	16	Calnex Sentinel
Transparent Clocks	5000	Calnex Sentinel Calnex Neo
PTP Client	> 100000	SPTP logs Calnex Sentinel

Table 1. large-scale validation metrics

With a large-scale deployment to over 100000 client machines in the data center it’s possible to compare performance of G.8275.2 and SPTP back-to-back.

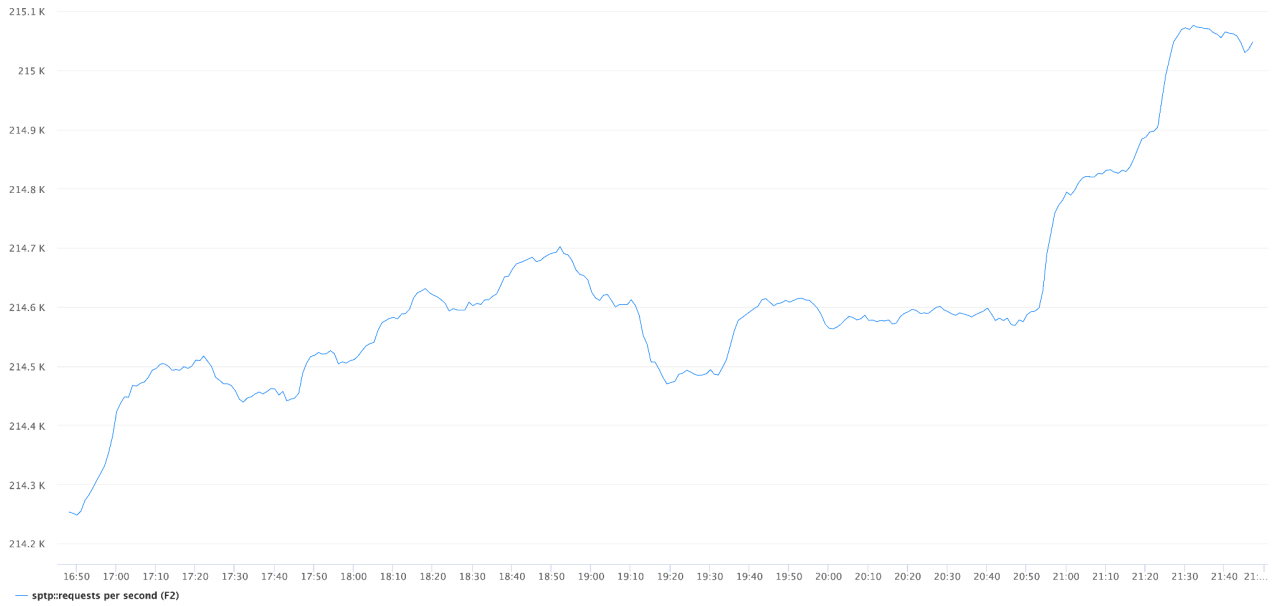

Figure 6. PTP server supporting 215k unicast clients

As confirmed by measurements taken from a large number of clients and reduced into a single time series using a P99.99 aggregation method, G.8275.2 can achieve a very good level synchronization:

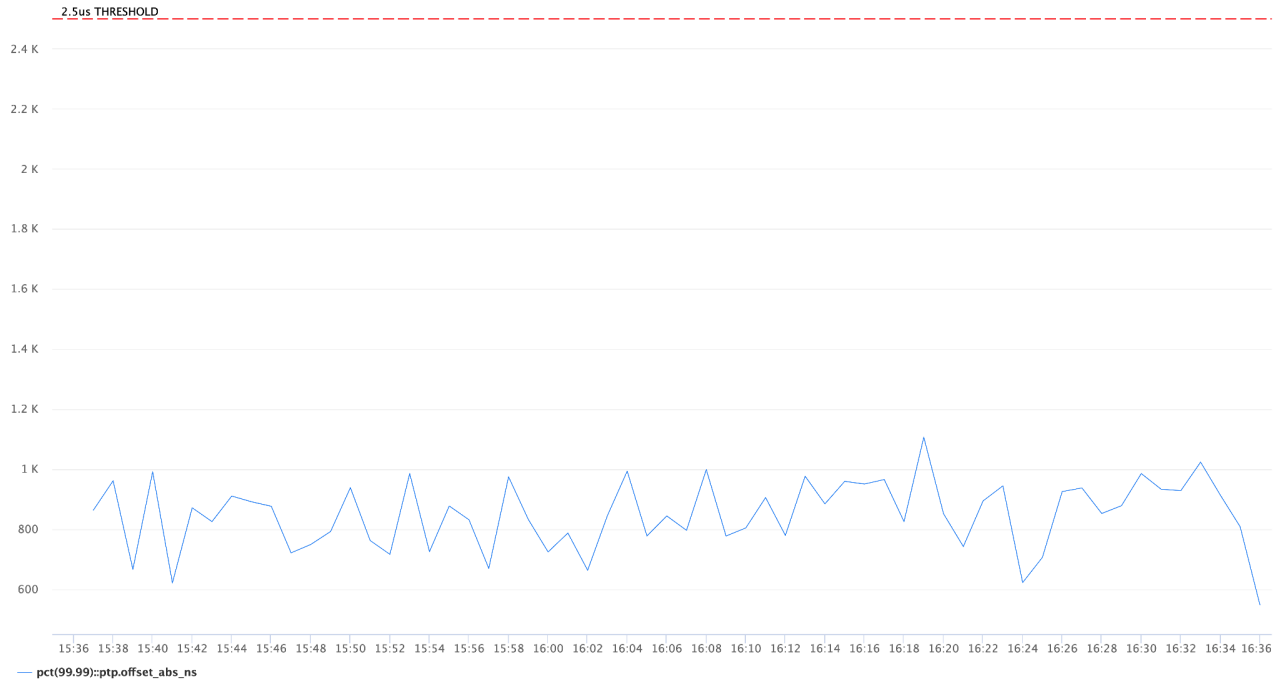

Figure 7. P99.99 offset collected from over 100000 ptp4l (G.8275.2) clients

As seen in Figure 7 the absolute value of P99.99 (10 out of 100000 hosts) clock offset stays under ±2.5 µs budget.

Repeating the same measurement after migration to SPTP produces a very similar result, only marginally different due to a statistical error:

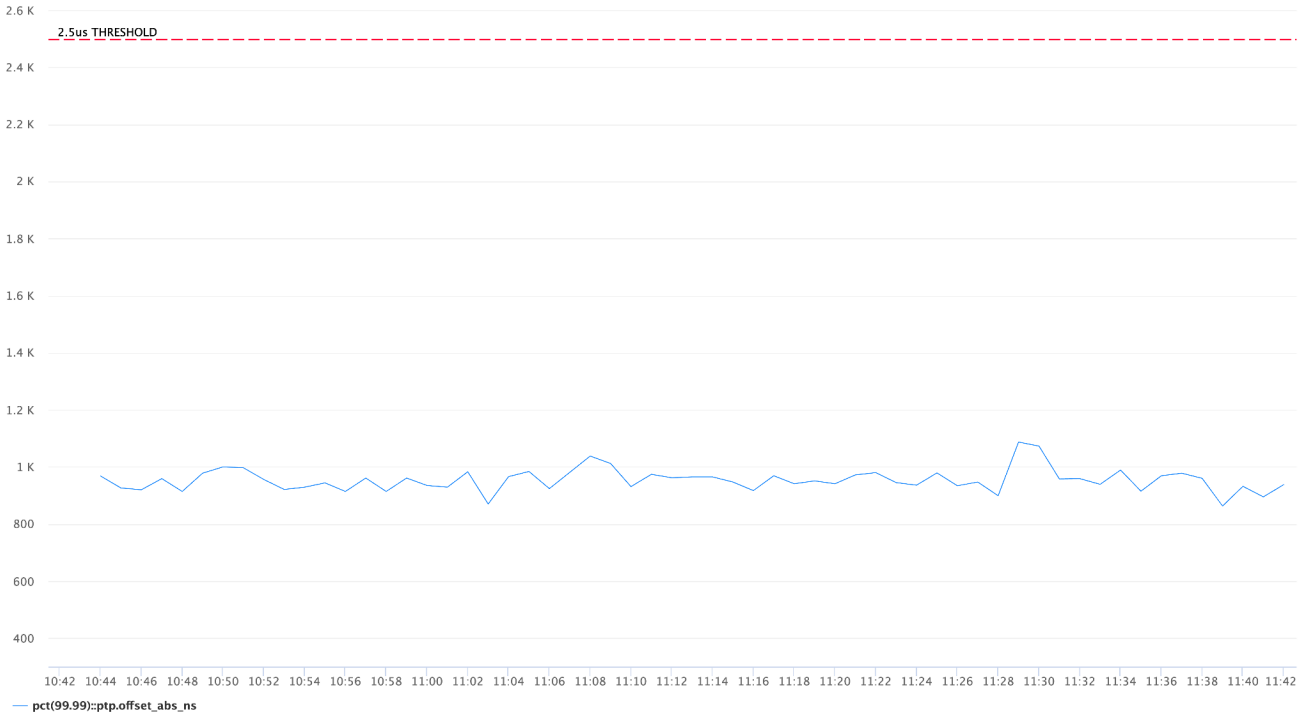

Figure 8. P99.99 offset collected from over 100000 SPTP clients

It is worth pointing out that remaining P99.99 aggregation is simply as a noise reduction filter.

4.1.3 Performance testing with large-scale deployment

With large-scale deployment of our implementations we can confirm resource utilization improvements.

We noticed that due to the difference in multi-server support, the performance gains vary significantly depending on the number of tracked time servers.

For example, with just a single time appliance serving the entire network there are significant improvements across the board. Most notably - over 40% CPU, 70% memory and 50% network utilization improvements:

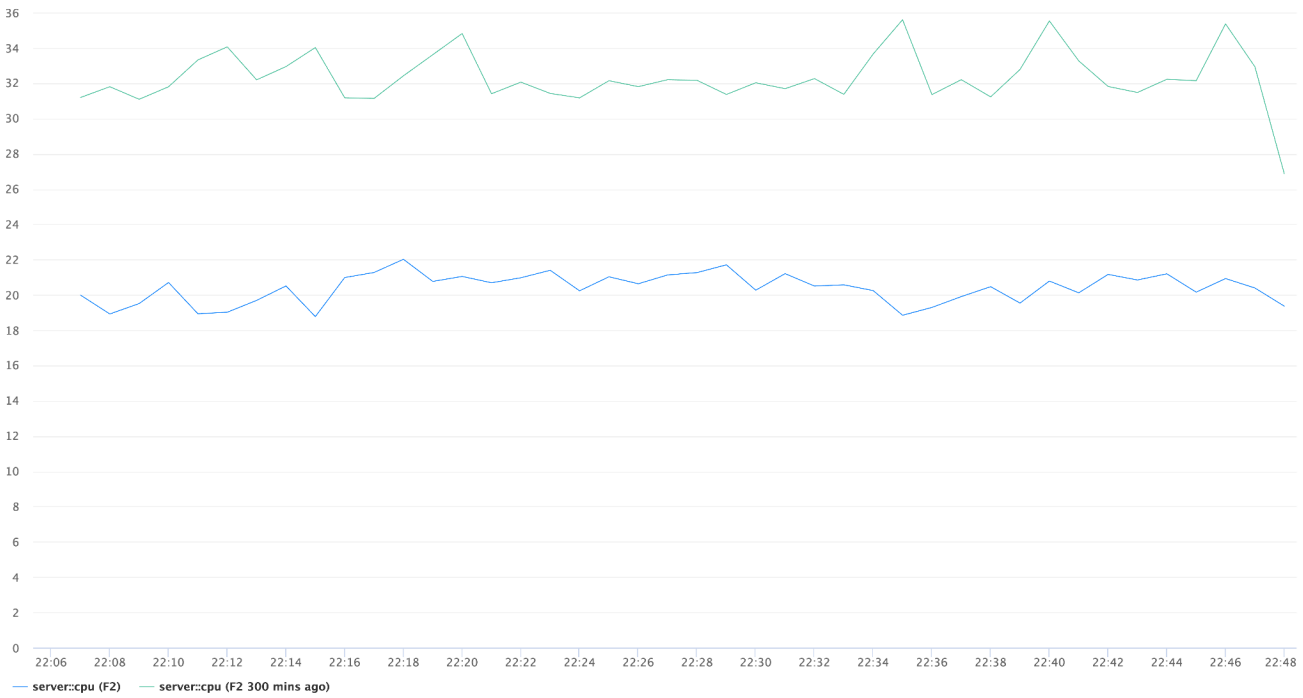

Figure 9. Server CPU utilization pct with ptp4l (green) vs SPTP (blue)

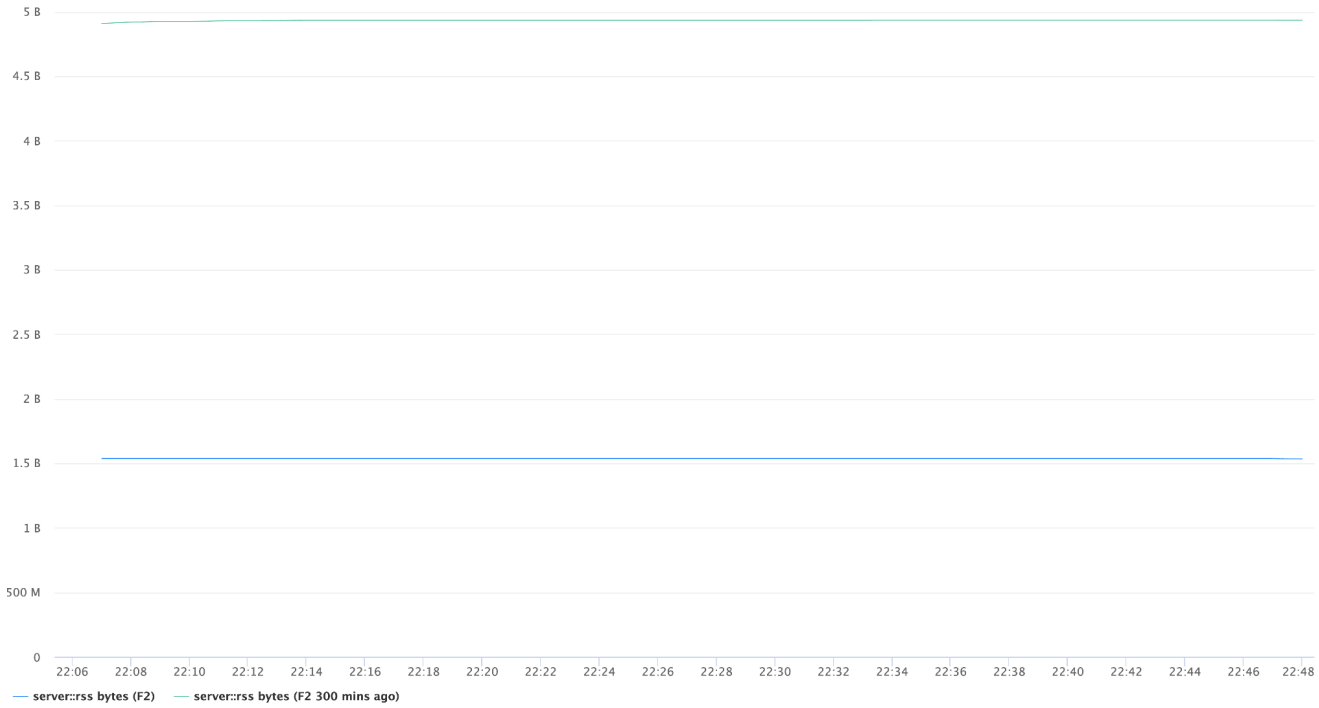

Figure 10. Server memory utilization in bytes with ptp4l (green) vs SPTP (blue)

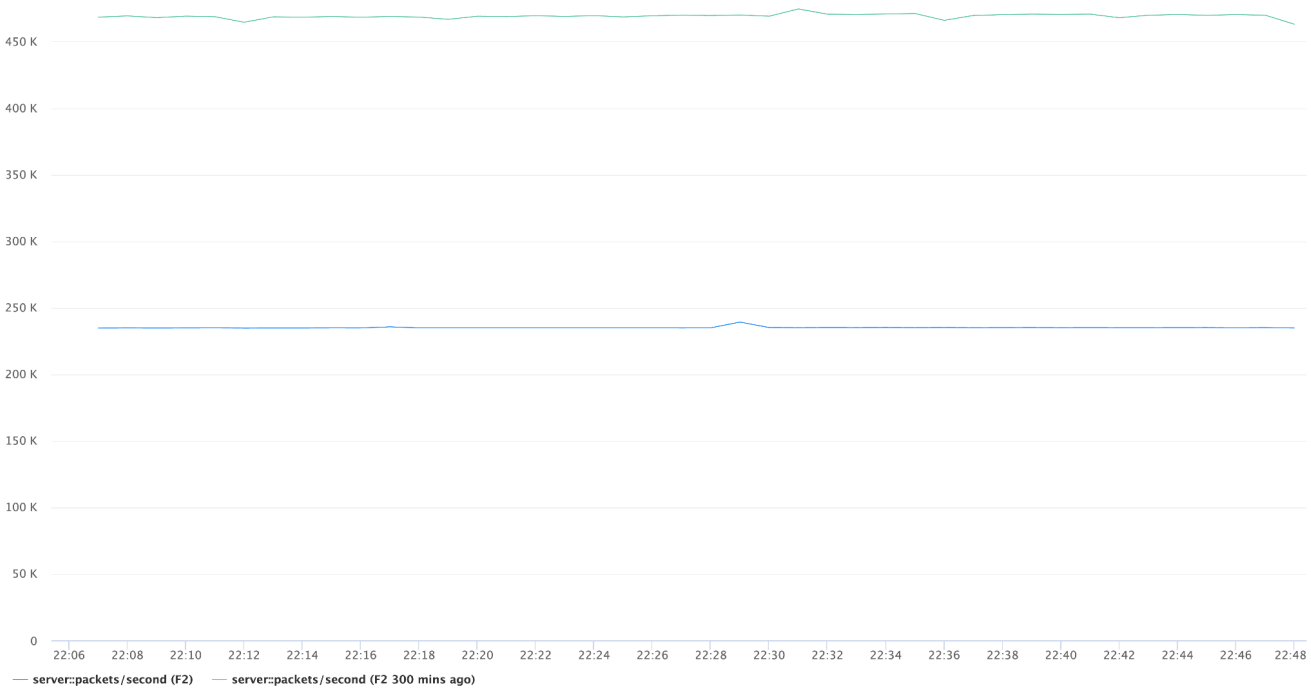

Figure 11. Packets per second with ptp4l (green) vs SPTP (blue)

The magnitude of the performance gains decreases proportionally to the number of configured PTP servers. In our particular case with 4 PTP servers configured, CPU and network utilization are roughly the same, with 50% memory utilization improvement between traditional unicast profiles and SPTP.

However, as mentioned earlier, one needs to allocate resources capable of handling worst case degradation scenarios and in case of G8265.1 and G8275.2 implementations will be on average 50% higher than with SPTP.

5 Discussion

Since SPTP can offer the exact same level of synchronization with a lot fewer resources consumed, we think it's a reasonable alternative to the existing unicast PTP profiles.

On a large data center deployment, it can help to combat frequently changing network paths and save gigabits of network traffic, gigabytes of memory and many CPU cycles.

It will eliminate a lot of complexity inherited from multicast PTP profiles, which is not necessarily useful in the trusted networks of the modern data centers.

It should be noted that, for systems that still require subscription and authentication, SPTP would not be suitable.

Additionally, by removing the need for subscription the ability to observe multiple clocks would be possible. This ability allows us to provide higher reliability by comparing the time sync from multiple sources at the end node.

6 Conclusion

SPTP can offer a significantly simpler, faster and more reliable synchronization. Along with G.8265.1 and G.8275.2 it provides excellent synchronization quality using a different set of parameters. Simplification comes with certain tradeoffs such as missing signaling messages which users need to be aware of and decide which profile is the best for them.

Having it standardized and assigned a unicast profile identifier will encourage wider support, adoption and popularization of PTP as a default precise time synchronization protocol.

7 Reference

[1] Eidson, John (10 October 2005). "IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, a Tutorial". National Institute of Standards and Technology (NIST).
[2] 1588-2019 - IEEE Approved Draft Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems". IEEE. Retrieved 15 February 2020
[3] IEEE Std 1588-2019, section 14.1.1 Table 52.
[4] Byagowi, at al. “Time Card and Open Time Server” ISPCS 2022, Vienna
[5] Microchip TimeProvider 5000 spec
[6] IEEE Std 1588-2019, section 9.3.2.4.5
[7] Oleg Obleukhov, Ahmad Byagowi, “How Precision Time Protocol is being deployed at Meta”
[8] Richard Cochran, ptp4l manual, sanity_freq_limit
[9] Alexey Andreyev, “Introducing data center fabric, the next-generation Facebook data center network”