Improving the PTP Synchronization Accuracy Under Asymmetric Delay Conditions

Martin Lévesque and David Tipper
School of Information Sciences
University of Pittsburgh
Pittsburgh, PA, USA

***Abstract*—Tight synchronization timing is expected to play a crucial role for the realization of emerging Internet of Things (IoT) high value real-time applications such as smart transportation and smart grid. Most packet-based synchronization protocols require two-way communications delay symmetry for precise accuracy. The Precision Time Protocol (PTP) provides mechanisms to measure and take into account the delay asymmetry problem, such as the residence time measurements, but recommend PTP support at all nodes. In this paper, we consider partial on-path PTP support, where a subset of the nodes are PTP unaware. We propose probing-based mechanisms in order to estimate asymmetry and improve the synchronization performance. The extensive simulation results show that the proposed lightweight per-packet asymmetry mitigation (PPAM) mechanism, aiming at estimating the per-packet delay components, outperforms the other considered protocols mainly at low to mid network loads, while under certain conditions the regular PTP with QoS support provides more stable synchronization accuracy.**

I. Introduction

The emergence of the Internet of Things (IoT) and machine-to-machine (M2M) communications systems will result in a trillion new distributed endpoints expected to be deployed by 2022, according to Cisco [1]. Time synchronization will play a critical role for the realization of certain high value applications, such as smart transportation, intelligent power grid, as well as new medical and financial applications, as reported in a recent technical note published by the National Institute of Standards and Technology (NIST) [1]. In [1], the authors argue that significant research is required in an integrated manner for the realization of time-aware applications, computers, and communications systems (TAACCS), decomposed into the following main areas of research: (1) clock and oscillator design, (2) time and frequency transfer, (3) use of synchronized timing in communications networks, and (4) hardware and software architecture, applications and systems design. In this paper, we focus on the time and frequency transfer area considering time synchronization of devices connected by packet switched networks.

A general solution to providing synchronization among networked devices requiring time alignment is incorporating an accurate source of time (e.g., atomic clock, Global Positioning System (GPS) equipped receiver) in each device. However, equipping each device of this technology would be extremely costly, especially for instance in sensor and actuator devices, which in many applications (e.g., smart grid [2]) are cost and computationally constrained. The most cost-effective way to synchronize clocks on devices connected by a packet based network is to use widely deployed, available, and cheap crystal oscillator based clocks, and adjust them as precisely as possible using a packet-based synchronization protocol such as the Precision Time Protocol (PTP). Here, we focus on PTP version 2, IEEE 1588 [2], which was proposed to provide synchronization of clocks over heterogeneous systems and networks. In synchronization over packet based networks, one of the major hurdles to accuracy is asymmetry in the communications delay between devices. This asymmetry has several sources such as, different routes being used, variable queuing delays, variations in cable length, different link bandwidths, and multiple vendor equipment types. PTP provides mechanisms to measure and take into account the delay asymmetry, such as the residence time measurements at each node. However, the standard recommends PTP support at all nodes in order to achieve high accuracy, thereby significantly increasing the cost. In this paper, we consider partial on-path PTP support, where a subset of the nodes are PTP unaware and propose probing-based schemes to estimate the delay asymmetry. Hence, our approach is expected to be more cost effective and can better support legacy systems and a mix of communication technologies.

This paper is structured as follows. We first overview the Precision Time Protocol (PTP) and related work in Section II. We next describe the main network delay components in Section III, which need to be considered in detail to improve the delay estimations and mitigate the negative effects of having asymmetrical delays. In Section IV, we focus on the time and frequency transfer for TAACCs and propose probing-based mechanisms to improve the synchronization accuracy over low-cost and multi-class networks. We next present numerical results in Section V and conclude the paper in Section VI.

This work was supported by NSERC Postdoctoral Fellowship No. 453711-2014.

Corresponding author: Martin Lévesque, School of Information Sciences, University of Pittsburgh, Pittsburgh, PA 15260 (email: levesque@pitt.edu).

978-1-4673-7595-5/15/$31.00 ©2015 IEEE

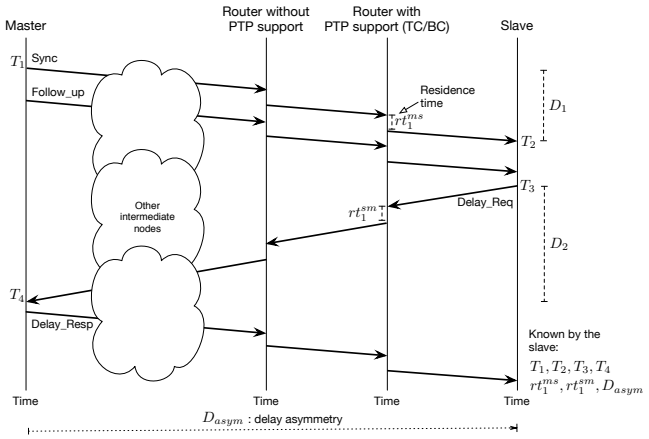


Fig. 1: The Precision Time Protocol (PTP) over a network with a subset of the routers having PTP support. (TC: Transparent clock, BC: Boundary clock)

II. Background and Related Work

PTP was proposed to meet the more precise synchronization requirements of power systems, industrial automation, telecommunications applications, manufacturing systems, and so forth. PTP operates at the application layer with one or more devices acting as a master clock which serves as the time reference and should have reliable source of time (e.g., atomic clock, GPS equipped receiver, etc.). The other devices are termed slaves and synchronize to the master clock. Each PTP node contains one or many ports, and communicates with other PTP nodes in a given network by implementing the PTP synchronization protocol described hereafter.

A. PTP Synchronization Protocol

The main procedures of the PTP synchronization protocol, depicted in Fig. 1, are listed as follow:

- The master node periodically transmits a *Sync* message to the slave nodes containing the sent time T_1.
- A *Follow_up* message which contains T_1 is optionally sent depending on the timestamp processing mechanism.
- When T_1 arrives at a given slave node, the received time T_2 is recorded.
- Next, the slave node then sends a *Delay_req* message to the master clock and T_3 is recorded locally.
- Finally, the master node records the reception time T_4 and sends it back to the slave node in a *Delay_resp* message.

The offset time at a given slave node k is approximated by:

$$\Theta_k = \frac{D_1 - D_2}{2}, \tag{1}$$

where $D_1 = T_2 - T_1$ and $D_2 = T_4 - T_3$ correspond to the downstream (from the master node to slave node) and upstream (from a slave node to master node) delays, respectively. To correct its clock, a given slave node k adjusts its local time t_k to $t_k \leftarrow t_k - \Theta_k$. Note that servo clocks are commonly used in order to frequently update the local frequency.

It is worth noting that Eq. (1) approximates the offset precisely if the messages experience symmetrical delays, that is, if both D_1 and D_2 are equal. However, the presence of asymmetrical delays can significantly degrade the synchronization accuracy.

B. Related Work

Several techniques have been proposed to estimate delay asymmetry in a PTP network , among others: offline delay estimation, servo clock, probing-based mechanisms, timestamp precision improvement, and so forth. In the probing-based approach, which is the focus of this work, probe messages are sent and analyzed, mainly in regard of the sending and receiving events. In [3], the authors extended the PTP protocol by adding probe bursts after the operation of the PTP protocol in order to approximate the upstream and downstream data rates and so the end-to-end delay. Their proposals, resulting in improved synchronization performance, were however only investigated over single hop connections. In a similar way, the authors in [4] proposed to duplicate *Sync* messages to find the delay over multiple intermediate nodes without transparent clock. Also, probing messages were added before and after *Sync* and *Delay_req* messages in [5] in order to detect line utilization. Note that each of these proposals improve the delay asymmetry mainly according to specific delay components, for instance for the transmission delay, queuing delay, and so forth. In the following, we overview the main delay components, which are key notions to improve symmetry.

III. Network Delay Components

Consider a path with a master node m and slave node s, whereby the path is interconnected via a set of nodes noted V. Thus, $s, m \in V$ and we have $n = |V - \{s, m\}|$ intermediate vertices. We define the upstream (from s to m) and downstream (from m to s) delays $D_{u/d}$ as follow:

$$D_{u/d} = \sum_{\forall i,j=i+1} \left(p_i + \Phi(\rho_{ij}) + \frac{L_{ptp}}{C_{ij}} + \Delta_{ij} + \frac{d_{ij}}{\epsilon} \right) + p_{m/s} + \mathcal{A}_{u/d}, \tag{2}$$

where i represents the node index depending on the upstream/downstream direction. The notation of the sum for $\forall i, j = i+1$ means that it is summed over i with j set to $i+1$. Specifically, for the upstream direction D_u, i corresponds to:

$$i = s, v_1^u, ..., v_n^u, \tag{3}$$

and for the downstream direction (D_d)

$$i = m, v_1^d, ..., v_n^d \tag{4}$$

where $v_{1..n}^{u/d}$ represents the intermediate vertices ordered depending whether it is in the upstream or downstream direction, respectively. The variable and fixed delay components in Eq. (2) are described in the following.

- *Processing delay (p_i):* The processing delay p_i at each node i includes the delays experienced in the routing table lookup, processes execution by the operating system, and so forth. At the master and slave nodes, this delay component can be reduced using hardware-based timestamps and we seperate this term from the summation to allow for this. Thus, $p_{m/s}$ represents the processing delay at the last node either in the upstream (m) or downstream (s) direction.
- *Queuing delay ($\Phi(\rho_{ij})$):* Depending on the incoming traffic intensity ρ_{ij} from node i going through node j at a given network interface, variable queuing delays are experienced due to the variable queue length. This delay component is challenging to estimate since it depends on the traffic load. One significant technique to take into account this delay component in PTPv2 is the use of transparent clocks which measure the residence times.
- *Transmission delay ($\frac{F}{C_{ij}}$):* The transmission delay depends on the data rate C_{ij} of a given channel and frame length F. Note that the data rate can vary from one link to the other, which can lead to asymmetrical delays. For example if we have two different data rates 120 and 300 Mbps and given the 126 octets length of the PTP messages (for *Sync* and *Delay_req* messages), then the transmission delays are respectively 8 and 3 μs without including the lower layers overheads (e.g., TCP/UDP, IP, etc.).
- *Per-packet/frame control delay (Δ_{ij}):* In many networks, especially in wireless ones, the frame transmission has a fixed or variable extra control delay Δ_{ij}. For instance, the distributed coordination function (DCF) in WiFi systems has a backoff delay to decrease the collision probability [6], which varies on a per-frame basis and based on each node traffic intensity. Note that WiFi has a timing mechanism standardized in IEEE 802.11v, which is integrated in IEEE 802.1AS as a PTP profile [7].
- *Propagation delay ($\frac{d_{ij}}{\epsilon}$):* The propagation delay, which is the physical time required to transmit a given frame from i to j with distance d_{ij}. Note that $\epsilon = \frac{c}{ir}$, where c corresponds to the speed of light and ir the index of reflection. This delay component is generally less than a microsecond for a distance of few meters, and grows significantly for distances of few kilometers.
- *Network-wide asymmetry delay ($\mathcal{A}_{u/d}$):* In some networks, the upstream and downstream delays are significantly asymmetrical on a network-wide basis. For instance, an Ethernet passive optical network (EPON) has this type of communications behavior [6]. Such a network has a tree-like structure consisting of an optical line terminal (OLT) managing the bandwidth in the network and the optical network units (ONUs) act as end-users. As the OLT broadcasts the frames to all ONUs, no extra control delay is required in the downstream direction (from OLT to ONUs). However, in the upstream direction, ONUs must be scheduled via the OLT and transmit frames as reserved blocks. Such delay difference between the upstream and downstream directions render an asymmetrical delay, noted $\mathcal{A}_{u/d}$. Note that EPON and Gigabit PON are time synchronized networks. IEEE 802.1AS also contains a PTP profile for these networks [7], thus this delay component can be taken into account if the network nodes support IEEE 802.1AS.

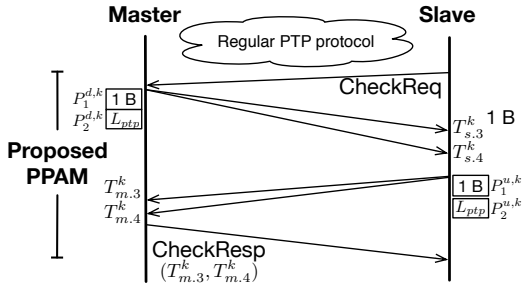


Fig. 2: Per-packet asymmetry mitigation (PPAM) protocol.

Given these different network delay components, we propose a method to improve the PTP performance under asymmetrical conditions without considering PTP hardware support at all intermediate nodes (e.g., boundary/transparent clocks).

IV. Proposed Asymmetry Mitigation Techniques

In the previous section, we described the main network delay components. Recent proposed techniques that take into account the delay asymmetry properties, are probing-based, which consist of sending extra messages after the PTP procedures and estimating the asymmetry in order to correct the offset time.

A. Per-Packet Asymmetry Mitigation (PPAM) Mechanism

In the following, we propose a protocol to correct asymmetry, as depicted in Fig 2. The overall protocol follows a structure similar to Lee's algorithm [3], but is lightweight and aims at also taking into account the per-packet control delays (term Δ_{ij} described in Section III).

We let $k = 0, 1, ..., \infty$ be the PTP iterations, representing PTP protocol executions. $k = 1$ models the first PPAM execution, and $k = 0$ is used to represent the initialization values. The main steps of the protocol are described as follow:

- As in the Lee's protocol, a *CheckReq* message is sent by a given slave node to the master node after the regular PTP protocol.
- Next, the master node begins the probing step, which, in contrast to the block burst transmission mechanism in Lee, consists of generating two consecutive messages $P_1^{d,k}$ and $P_2^{d,k}$ destined to the given slave node. In order to be bandwidth efficient, we set the first message ($P_1^{d,k}$) length to 1 octet and L_{ptp} (126 octets) for the second message ($P_2^{d,k}$) length. At the slave reception of each message, timestamps $T_{s.3}^k$ and $T_{s.4}^k$ are recorded. The rationale behind the use of these message lengths is that

we are interested to measure the transmission duration of a PTP message, therefore if we substract $T^k_{s.4} - T^k_{s.3}$ using these specific message lengths, we obtain the base message transmission duration.

- After the slave node receives both $P^{d,k}_1$ and $P^{d,k}_2$, it similarly generates two consecutive messages $P^{u,k}_1$ and $P^{u,k}_2$ destined for the master node. Notice that, as opposed in Lee, we generate both $P^{u,k}_1$ and $P^{u,k}_2$ one after the other when $P^{d,k}_2$ is received, instead at the reception of each message. By generating $P^{u,k}_1$ and $P^{u,k}_2$ one after the other as $\{P^{d,k}_1, P^{d,k}_2\}$, both directions are measured under similar conditions. Further, when the master node receives $P^{u,k}_1$ and $P^{u,k}_2$, their respective timestamps are recorded as $T^k_{m.3}$ and $T^k_{m.4}$, respectively.
- The protocol is completed following the *CheckResp* message containing $T^k_{m.3}$ and $T^k_{m.4}$ from the master node to the given slave node.

We define the minimum delay at iteration k in the downstream direction as:

$$D^{min,k}_{m2s} = (1-\alpha)\cdot D^k_{m2s} + \alpha \cdot D^{min,k-1}_{m2s} \tag{5}$$

and similarly the minimum delay at iteration k in the upstream direction is:

$$D^{min,k}_{s2m} = (1-\alpha)\cdot D^k_{s2m} + \alpha \cdot D^{min,k-1}_{s2m}, \tag{6}$$

where α is a weighting factor used to smooth out variations. The current delays from the master to slave and slave to master are given by:

$$D^k_{m2s} = T^k_{s.4} - T^k_{s.3} \tag{7}$$

and

$$D^k_{s2m} = T^k_{m.4} - T^k_{m.3}. \tag{8}$$

Finally, we have the following offset equation:

$$\Theta^k_{ppam} = \frac{(D_{m2s,k} - D^{min,k}_{m2s}) - (D^k_{s2m} - D^{min,k}_{s2m})}{2} \tag{9}$$

in order to align both upstream and downstream delays and take into account the transmission and per-packet control delays. Algorithm 1 gives the detailed procedure for estimating the clock offset of a given slave node. A slave clock time t_s can then be updated as follows:

$$t_s \leftarrow t_s - \Theta^k_{ppam}. \tag{10}$$

It is worth noting that the algorithm only requires to maintain the parameter values for k and $k-1$.

Input:
k: Iteration index.
$T^k_{s.3}, T^k_{s.4}$: Timestamps recorded at a given slave node.
$T^k_{m.3}, T^k_{m.4}$: Timestamps recorded at the master node.
$D^{min,k-1}_{m2s}, D^{min,k-1}_{s2m}$: Minimum delays at the previous iteration.
α: Parameter used to smooth out variations of the minimum delays.
Output:
Θ^k_{ppam}: Clock offset of a given slave node.

```
begin
    D^k_{m2s} ← T^k_{s.4} − T^k_{s.3}
    D^k_{s2m} ← T^k_{m.4} − T^k_{m.3}
    if k = 0 then
        D^{min,k}_{m2s} ← D^k_{m2s}
        D^{min,k}_{s2m} ← ·D^k_{s2m}
    else
        D^{min,k}_{m2s} ← (1 − α) · D^k_{m2s} + α · D^{min,k−1}_{m2s}
        D^{min,k}_{s2m} ← (1 − α) · D^k_{s2m} + α · D^{min,k−1}_{s2m}
    end
    Update Θ^k_{ppam} as in Eq. (9).
end
```

Algorithm 1: The proposed PPAM algorithm.

B. Class Probing

The proposed PPAM and Lee protocols require changing the PTP implementation, since the offset equation needs to be modified. As noted in the PTPv2 standard, it is highly recommended to process PTP messages with high priority when no transparent clock is available at intermediate nodes. If all nodes in the network have QoS support allowing processing of PTP messages first, the synchronization accuracy is improved. However, when a given PTP packet arrives at a router, if a low-priority packet (e.g., non-PTP packet) has arrived just before, the PTP packet needs to wait for the termination of the low-priority packet, which can cause non negligible asymmetry as we show shortly in Section V. In order to align the transmission probability of a non-PTP packet, we propose *Class Probing*, which consists of sending a low-priority message prior to sending any PTP message. By doing so, when a PTP message arrives at a certain switch/router, a non-PTP message will be transmitted in both the upstream and downstream directions, thus improving symmetry.

V. Simulation Results

In this section, we investigate the synchronization accuracy under different network asymmetry conditions using our recently developed OMNeT++ PTP simulation framework [1]. The topology we consider, depicted in Fig. 3, contains a master node m, 10 slave nodes, and 2 intermediate routers. All communications interfaces are based on Ethernet with 100 Mbps full duplex capacity. We use wired Ethernet links to simplify the performance comparison. In order to vary the network load, we have two traffic generators (g_u and g_d in Fig. 3) connected to the intermediate routers (r_1 and r_2 in Fig. 3) which exchange messages between each other. Further, to simulate realistic and bursty traffic patterns, the traffic generators follow a Pareto distribution such that the average interarrival time between the generated packets equals

[1]The simulation model is available online: http://www.omnetpp.org/9-articles/software/3729-ptp-plusplus.

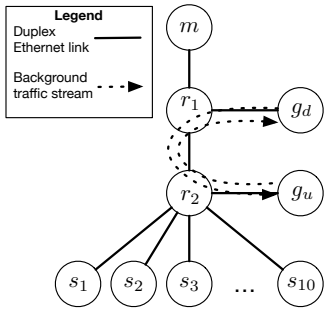
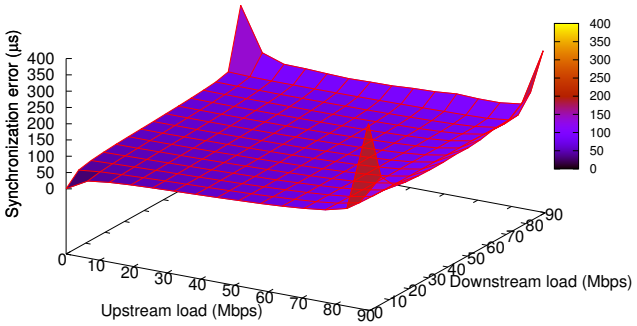


Fig. 3: Considered simulation network topology consisting of a master node m and 10 slave nodes $\{s_1, s_2, ..., s_{10}\}$. Traffic generators g_u and g_d create background traffic packets.

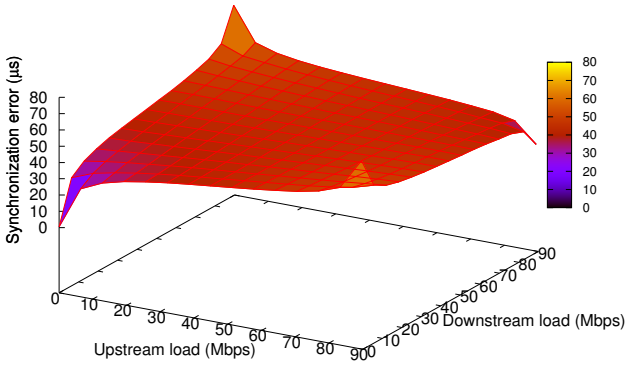
$\mu = \frac{a \cdot b}{a-1}$, where $a = 1.5$ and b are the shape and scale of the Pareto distribution, respectively. Note that μ is varied and b is set accordingly. We also set the clock drift of the slaves (relatively to the master) to 10^{-4} seconds [1] following a normal distribution. Note that for each generated message, the UDP and IP headers are also encapsulated [2]. For each given configuration and scenario, 15 simulations are executed with different random seeds and the average across the simulations is shown.

One of the most challenging asymmetry components is the variable queuing delay, which depends on the network load. In Fig. 4a), we vary the upstream and downstream traffic loads and record the PTP synchronization error without using any asymmetry mitigation technique (e.g., post PTP mechanism, prioritized queuing). When the traffic load is significantly low in the upstream and downstream directions, as expected, the synchronization error is below 10 μs. However, as the load increases, the synchronization error grows, especially when the network has asymmetrical traffic load conditions (e.g., when the upstream load equals 90 Mbps with 0 Mbps downstream load). To improve the performance, we next configure prioritized queues in order to process PTP messages first. It is worth noting that PTP recommends to set the PTP packets with high priority, therefore we consider this improvement mechanism as it is low-cost to deploy. Fig. 4b) shows significant accuracy improvement with priority queues, where the synchronization error varies between 0-80 μs compared to 0-400 μs when not using any QoS. As in the previous results (without using any QoS), when the traffic load is asymmetrical, the synchronization error increases. However, we observe that when priority queues are used at high loads, the synchronization error decreases compared at medium loads. This is due to the fact that at high load, the probability that a transmission of a non PTP packet occurs while a PTP packet arrives at the same time is higher.

[2]The detailed simulation configurations are available online: https://github.com/martinlevesque/ptp-plusplus/tree/master/simulation.



(a) Without QoS.



(b) With QoS.

Fig. 4: Synchronization accuracy with variable upstream and downstream network loads.

A. Class Probing

We show a comparison of using the proposed class probing mechanism (Refer to Section IV-B) vs. using the PTP standard in Fig. 5, whereby we assume that all network nodes have QoS support. We observe a significant accuracy improvement while using the class probing technique from low to high upstream load. Recall that this proposed technique only requires a node to send a low-priority packet prior to any PTP packet in order to align the non-PTP transmission probability.

B. PTP Asymmetry Performance Comparison

We next investigate the proposed PPAM mechanism and compare it to the standard PTP and block burst transmission technique (Lee [3]). In the PPAM mechanism, we set $\alpha = 0.9$ used in Eqs. (5-6). For comparison, we consider multiple parameters affecting the asymmetry delay components, that is, the upstream (λ_u) and downstream (λ_d) traffic loads, upstream capacity (c_u) at intermediate routers, and upstream per-frame control delay (Δ) at the intermediate routers. For the traffic loads ($\lambda_{u/d}$), we consider low ($l = 1\ Mbps$),

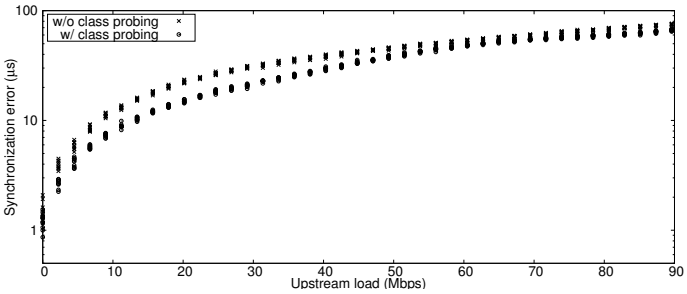


Fig. 5: Improvement of the synchronization accuracy by using the proposed class probing mechanism.

medium ($m = 30\ Mbps$), and high ($h = 90\ Mbps$) traffic loads. Note that those traffic loads are the total generated traffic originating from g_u and g_d. In order to compare under multiple network conditions, we measure the performance for all permutations of these parameter values, as shown in Tables I-II. Note that when $c_u = 1\ Gbps$, an asymmetry occurs since the remainder of the networking interfaces have 100 Mbps capacity. We have shown in bold the best scheme in term of smallest synchronization error for each scenario.

In Table I, we compare the synchronization error without any QoS support. Note that we vary the downstream traffic load only in the downstream direction since the per-frame extra delay (Δ) limits the maximum upstream throughput. We observe that when the asymmetry is mainly caused by the network load, the standard PTP, without using any asymmetry mitigation technique, outperforms the Lee and PPAM mechanisms. However, at low to medium loads, the proposed PPAM scheme outperforms the PTP and Lee synchronization performance.

		$c_u = 100\ Mbps$		$c_u = 1\ Gbps$	
		$\Delta = 0$	$\Delta = 150\mu s$	$\Delta = 0$	$\Delta = 150\mu s$
PTP	$\lambda_d = l, \lambda_u = l$	**2.5**	75.3	14.4	61.5
	$\lambda_d = m, \lambda_u = l$	**31.6**	51.6	44.3	43.4
	$\lambda_d = h, \lambda_u = l$	**353.2**	**289.3**	**362.2**	**298.7**
Lee	$\lambda_d = l, \lambda_u = l$	2.8	24.3	14.6	18.9
	$\lambda_d = m, \lambda_u = l$	**31.6**	74.6	44.1	68.6
	$\lambda_d = h, \lambda_u = l$	3013.1	5461.4	3344.6	5145.5
PPAM	$\lambda_d = l, \lambda_u = l$	2.9	**8.0**	**6.8**	**3.4**
	$\lambda_d = m, \lambda_u = l$	32.3	**40.4**	**36.5**	**33.2**
	$\lambda_d = h, \lambda_u = l$	1229.0	1243.4	1165.7	1198.5

TABLE I: Synchronization error (μs) without QoS support.

We next investigate the considered schemes when all network nodes have QoS support by considering all load permutations of λ_d and λ_u, as depicted in Table II. Again, when the main delay asymmetry component is based on the traffic ($\Delta = 0$ and $c_u = 100\ Mbps$), the standard PTP has the lowest synchronization error compared to using the Lee or PPAM techniques. However, when an asymmetry delay condition occurs based on either the capacity or per-frame delay, the proposed PPAM scheme provides a lower synchronization error, except when $\lambda_u = m, h$ and $\Delta = 150\mu s$. This is due to the fact that the probability of non-PTP frame transmission is high, and thus degrades the synchronization performance. Solutions to overcome this shortcoming will be investigated in future work.

		$c_u = 100\ Mbps$		$c_u = 1\ Gbps$	
		$\Delta = 0$	$\Delta = 150\mu s$	$\Delta = 0$	$\Delta = 150\mu s$
PTP	$\lambda_d = l, \lambda_u = l$	**2.5**	75.3	14.4	61.5
	$\lambda_d = l, \lambda_u = m$	30.9	125.5	14.2	79.0
	$\lambda_d = l, \lambda_u = h$	**74.9**	140.5	13.7	100.8
	$\lambda_d = m, \lambda_u = l$	**31.6**	51.6	44.3	43.4
	$\lambda_d = m, \lambda_u = m$	**39.0**	97.8	43.5	57.1
	$\lambda_d = m, \lambda_u = h$	**50.6**	111.9	42.7	74.1
	$\lambda_d = h, \lambda_u = l$	**75.9**	**19.9**	**89.6**	**23.4**
	$\lambda_d = h, \lambda_u = m$	**55.6**	55.4	**89.3**	**27.4**
	$\lambda_d = h, \lambda_u = h$	25.5	66.9	88.8	**33.1**
Lee	$\lambda_d = l, \lambda_u = l$	2.8	22.4	14.6	20.1
	$\lambda_d = l, \lambda_u = m$	**28.6**	**31.5**	15.1	**17.8**
	$\lambda_d = l, \lambda_u = h$	75.8	**34.5**	16.2	**17.8**
	$\lambda_d = m, \lambda_u = l$	32.3	73.1	43.3	70.3
	$\lambda_d = m, \lambda_u = m$	42.8	**73.8**	44.6	70.4
	$\lambda_d = m, \lambda_u = h$	55.5	**74.9**	46.3	69.8
	$\lambda_d = h, \lambda_u = l$	76.8	129.8	90.2	138.1
	$\lambda_d = h, \lambda_u = m$	64.4	130.6	92.2	136.9
	$\lambda_d = h, \lambda_u = h$	**24.8**	130.8	95.8	138.6
PPAM	$\lambda_d = l, \lambda_u = l$	2.9	**8.0**	**6.8**	**3.4**
	$\lambda_d = l, \lambda_u = m$	32.4	109.6	**6.6**	33.8
	$\lambda_d = l, \lambda_u = h$	83.0	126.6	**7.6**	44.6
	$\lambda_d = m, \lambda_u = l$	32.4	**39.5**	**36.5**	**33.0**
	$\lambda_d = m, \lambda_u = m$	40.1	85.1	**35.8**	**54.0**
	$\lambda_d = m, \lambda_u = h$	58.8	92.1	**37.0**	**61.5**
	$\lambda_d = h, \lambda_u = l$	83.0	55.5	**89.6**	45.3
	$\lambda_d = h, \lambda_u = m$	58.7	**49.6**	**89.3**	53.9
	$\lambda_d = h, \lambda_u = h$	26.3	**47.9**	**88.4**	55.0

TABLE II: Synchronization error (μs) comparison with QoS support.

VI. Conclusions

Emerging IoT and M2M technologies will require tight time and frequency synchronization, which is quite challenging given that their supporting devices must be low-cost and highly secured. In this paper, we first looked at the main network delay components, which are key notions related to the network delay asymmetry problem. We then proposed *class probing* and per-packet asymmetry mitigation (PPAM) mechanisms in the context of partial on-path PTP support, which were shown to provide promising synchronization improvements for a range of scenarios. In future work, we will investigate several topologies with paths having a variable number of transparent clocks at intermediate nodes to study the effect of PTP and proposed mechanisms.

References

[1] M. Weiss, J. Eidson, C. Barry, D. Broman, L. Goldin, B. Iannucci, E. A. Lee, and K. Stanton, "Time-Aware Applications, Computers, and Communication Systems (TAACCS)," NIST Technical Note 1867, NIST, National Institute of Standards and Technology, U.S. Department of Commerce, USA, February 2015.

[2] IEC Technical Committee 65 and IEEE Standards Association (IEEE-SA) Standards Board, "IEEE Standard for a precision clock synchronization protocol for networked measurement and control systems," *IEC 61588:2009(E) IEEE Std. 1588(E)-2008*, pp. C1–274, Feb 2009.

[3] S. Lee, "An Enhanced IEEE 1588 Time Synchronization Algorithm for Asymmetric Communication Link using Block Burst Transmission," *IEEE Communications Letters*, vol. 12, pp. 687–689, Sept. 2008.

[4] S. Schriegel, H. Trsek, and J. Jasperneite, "Enhancement for a Clock Synchronization Protocol in Heterogeneous Networks," in *Proc., IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, pp. 1–5, 2009.

[5] T. Murakami and Y. Horiuchi, "Improvement of synchronization accuracy in IEEE 1588 using a queuing estimation method," in *Proc., IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, pp. 1–5, 2009.

[6] F. Aurzada, M. Lévesque, M. Maier, and M. Reisslein, "FiWi Access Networks Based on Next-Generation PON and Gigabit-Class WLAN Technologies: A Capacity and Delay Analysis," *IEEE/ACM Transactions on Networking (ToN)*, vol. 2, pp. 1176–1189, Aug. 2014.

[7] G. M. Garner and H. Ryu, "Synchronization of Audio/Video Bridging Networks Using IEEE 802.1AS," *IEEE Communications Magazine*, vol. 49, pp. 140–147, Feb. 2011.