Jitter Compensation Mechanism for Dynamic Deterministic Networks

Guillaume Soudais[1,2], Tarik Graba[2], Yves Mathieu[2], Sebastien Bigo[1]
(1) Nokia Bell Labs, 91620 Nozay, France, (2) Télécom Paris, Institut Polytechnique de Paris, 91120 Palaiseau, France
guillaume.soudais@nokia.com

Abstract: We compensate jitter between any two unsynchronized endpoints by tracking their clocks and re-creating flows by retiming packets. After implementation over FPGA, we achieve ~10ms synchronization setup time with no more than 70ns jitter. © 2023 The Authors

1. Introduction

Application developers are watching with interest the deployment of 5G edge clouds, excited by the prospect of new time-critical services, enabled by the efficient cooperation of multiple nearby machines (e.g. servers, objects, robots). This prospect calls for multiple adaptations to the infrastructure, including an augmented optical network with ultra-low deterministic jitter, dynamic enough to turn up/tear down flows almost as quickly as in today's typical data centers.

The Time Sensitive Networking (TSN) task force has developed several technologies to fulfill this goal. In order to achieve ultra-low jitter (e.g. sub-µs), tasks are scheduled in a fully synchronized network. But scheduling and synchronizing the whole network can be difficult and takes time, especially when the number of devices and nodes grows. Reports in [1] and [2] extend the scalability of TSN, by interconnecting time sensitive devices through an optical bus. This bus is either dedicated or bridged though legacy switches, respectively. The advantage of this bus is that the intermediate network elements do not need to be synchronized. However, it requires a jitter compensation mechanism at the egress, which deserves further investigations.

A common approach is to store packets at the egress of the bus in order to absorb latency variations and exit with the same interpacket gap as at the bus ingress. A first team simulated this approach [3] while neglecting the ingress/egress clock frequency mismatch (up to 100ppm according to IEEE 802.3 standards), which can produce a prohibitive latency drift of up to 1µs every 10ms. The team in [4] made sure that the egress clock is always running faster than the ingress clock in their testbed, which is a strong constraint for the practical deployment of multi-hop systems. In [2], we coped with the clock mismatch by continuously estimating the clock frequency difference in order to prevent latency drift, achieving <100ns jitter. However, the method had a relatively (>64ms) slow set-up time and its resistance to strong jitter was not assessed. In this paper, we investigate two alternative methods of jitter compensation. We propose to track the relative evolution of the clocks using two different types of tracking filters, then to re-create the flows by retiming the packets. We benchmark the methods experimentally over FPGA platforms and demonstrate 50x faster set-up time than in [2] and up to x30 reduction of jitter.

2. Network scenario

While our methodology applies to any link connecting unsynchronized endpoints, we choose the modular approach of [2] as reference scenario, where the endpoints are connected over an optical bus with a strict latency upper bound. The bus can be engineered in multiple ways, including over legacy networks, by establishing a private VLAN (or bridge) or by giving preemption rights to the time-sensitive packets, as defined by the 802.1QBu standard, across all switches of the path. To contain jitter and achieve deterministic performance, two add-on cards (called DDX in [2]) are inserted at both ends, performing time-division multiplexed (TDM) aggregation/disaggregation of Ethernet flows.

Figure 1a shows our experimental testbed based on real-time FPGA platforms, where we implemented two 10Gbit/s add-on cards interconnected via the optical bus. To mimic the behavior of this bus, we designed an emulator with programmable latency and jitter, replicating a variable length of fiber and a variable jitter from all switching

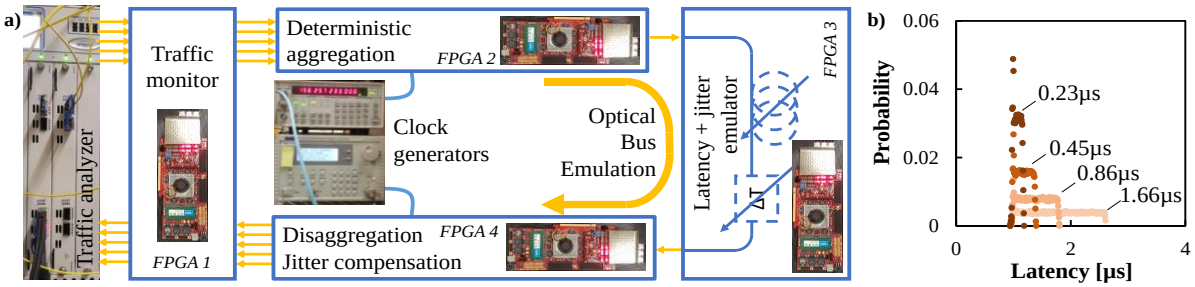

Fig. 1 a) Experimental setup of deterministic optical bus, b) Measured distribution of emulated latencies (jitter)

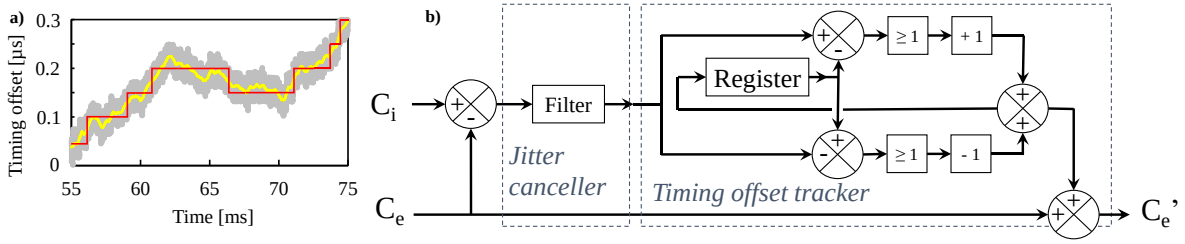

Fig. 2 a) Tracking the offset evolution between ingress and egress counter, b) Offset tracker box diagram

elements along the path, whether optical or electrical. Jitter is computed randomly over ranges from 0µs to 27µs, while 3µs is a typical upper bound per switch with path reservation or frame preemption in commercial hardware. We report in Fig. 1b the measured latency probability density functions (PDF) showing a uniform distribution over the preset ranges, with an overemphasis of the smallest jitters, but not exceeding 10% of the set of random draws.

3. Principles of our jitter compensation mechanisms

At the end of the bus, jitter compensation is required to equalize packet arrival times. As in [2]–[4], we store the arrival time a_n of the n^{th} packet at the ingress and we recreate the flow as follows: we release the first packet at time $b_0=a_0+L_{max}$, L_{max} being the latency upper bound of the bus and we release any n^{th} packet at time $b_n = b_{n-1}+(a_n-a_{n-1}) = b_0+(a_n-a_0)$, thus preserving the delay between packets at the egress as it was at the ingress of bus. This method allows for zero jitter, where jitter is defined as in RFC3393 [5], as the packet delay variation between two consecutive packets. However, we believe that the definition of [5] is not sufficient to guarantee determinism. Latency must also be kept bounded over the flow duration, for which we use here 10ms as reference. We therefore rely on the maximum jitter as quality indicator, defined as the maximum delay variation between any two packets over a 10ms sliding window.

To faithfully recreate the delays between packets, we must account for the frequency difference between the ingress and egress cards. Time t in the ingress card is represented by the counter C_i and incremented at the frequency f_i, $C_i = f_i*t$. In the egress card, the same happens at frequency f_e with counter $C_e = f_e*t$. Thus $C_i = d*C_e$, where $d = f_i/f_e$ is the ratio of frequencies, called drift in this paper. When using time stamps at the ingress and egress from these counters, it is important to correct the egress time to account for the drift between the two clocks. If A_n and B_n are generated from the counters and associated with time a_n and b_n, then $B_n-B_0 = d*(A_n-A_0)$. Depending on which clock runs faster, the packet flow is either dilated or contracted. To provide a constant latency, we track the drift between the counters due to the frequency mismatch. We send periodic timestamps in the packet headers and compare them at the egress card with the time of arrival of the packets. As the drift has relatively slow variations, we propose to use a low pass filter to filter out the drift from the higher frequency jitter. We compare two filter types: (1) a first-order Infinite Impulse Response (IIR) filter which requires no memory of the past recordings and provides the most compact implementation; (2) a Moving Average (MA) filter. The iterative tracking process starts when the first packet (time slot) of a flow is detected. The filters are initialized with x_0, equal to the average of the first 32 samples, a sample being the difference between the time stamp and the arrival time measured at egress (also called timing offset) of one packet.

The IIR filter is described by equation $y_n = a*x_n+(1-a)*y_{n-1}$, where x_n is the n^{th} sample of timing offset, y_n is the filter output, a is the weight where the new sample is affected by the previous one. In the particular case of a constant drift, with s as packet periodicity, $x_n = d*s*n$, then the filter becomes $y_n = d*s*n - d*s/a(1-exp(-a*n))$ where the second term represents the lag, which directly influences the setup time of the filter before its output is stable. The moving average filter is described by equation $y_{n+1} = y_n+(x_{n+1}-x_{n-M})/M$ where M is the number of timing offset samples where averaging is performed. To decrease its lag, we suppose $x_{n-M} = x_0$ when $n-M<0$. In the particular case of a constant drift, $x_n = d*s*n$, then $y_n = d*s*n*(n-1)/2M$, when $n<M$, and $y_n = d*s(n - (M-1)/2)$, when $n\geq M$. With both filters, theory predicts a lag during the setup time of the bus, that decreases with the packet periodicity. Note that the inverse weight $1/a$ and M play a similar role. Hence their magnitudes can be used as one single parameter to compare filters. Decreasing them decreases the setup time but affects latency stability.

We then reconstruct a stepwise approximate of the ingress timing reference C_i into C_e', based on the filter output as schematized in Fig. 2a. We compare the current filter output y_n to the filter output of the previous packet y_{n-1}, stored in a register (Fig. 2b). When the difference is greater than +1 (resp. -1), we increment (resp. decrement) the local counter to catch up with the remote one. After this tracking process, the corrected egress counter is running at the same frequency as the ingress counter, hence the delay between packets is faithfully recreated.

4. Experimentation

We have made an implementation of both filters in an FPGA platform. A 1Gb.s^{-1} flow produced by a traffic generator is passed in a dedicated traffic monitor, also in FPGA, which extracts the latency per packet [6] with high precision down to the clock period (Fig 1a), here 6.4ns. The packets are framed by the ingress time-division multiplexed

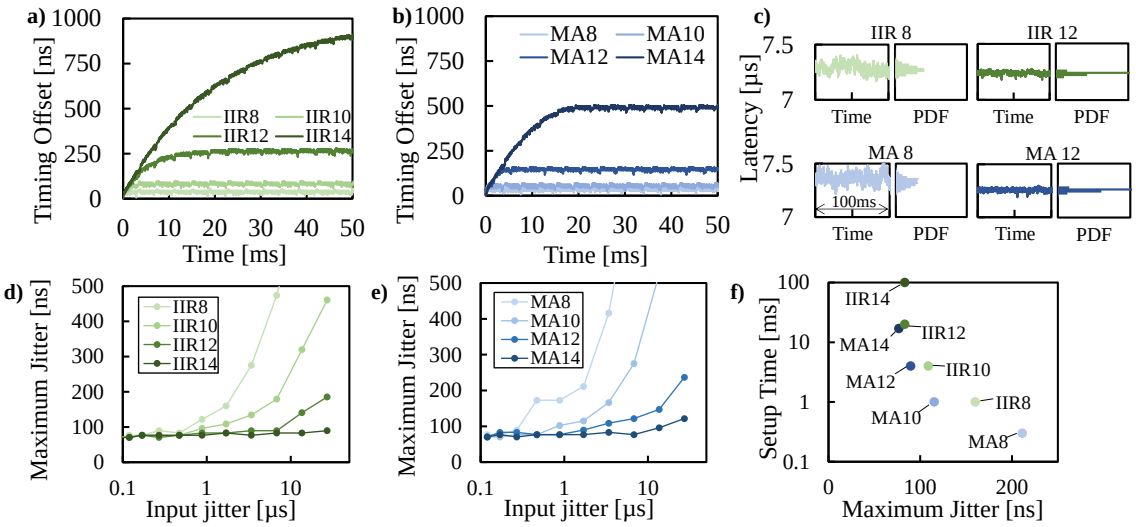

Fig. 3 Evolution of the timing offset during bus setup, without jitter emulation a) with IIR filter - xx in IIRxx is $log_2(1/a)$ b) with MA filter - xx in MAxx is $log_2(M)$ c) Latency recordings and probability density function over 10min with 1.8µs emulated jitter in bus; d) Maximum client jitter versus bus jitter with IIR filter e) Maximum client jitter versus bus jitter with MA filter, f) Filter trade-offs

aggregator and injected into the optical bus emulator. The traffic is then disaggregated, jitter compensated and fed back to the traffic monitor. We initialize the bus ingress and egress clocks at 50ppm frequency difference by clock generators of 1ppm precision, so as to ensure sizeable drifts in the experiment. We vary the filter parameters, namely the inverse weight $1/a$ for the IIR filter or the number of averaged timing offset samples M for MA filter to 2^8, 2^{10}, 2^{12}, 2^{14}. In Fig. 3a and 3b we show the timing offset evolution between the two devices with no added jitter from the emulator (<50ns from the devices themselves). The curve gradually saturates with exponential growth, which causes the lag predicted by theory. Unsurprisingly, we notice a faster convergence when the filter parameters are smaller. However, the moving average filter reaches stability no later than when the buffer is full (M symbols * 1.17µs packet periodicity), whilst the IIR filter takes four times longer to stabilize. We now increase artificially the jitter by x36, to 1.8µs. Fig. 3c displays the efficiency of our jitter compensation mechanism. The excursion of latency over 100ms is well contained, as well as the standard deviations of the latencies over 10min, whereas the 50ppm frequency mismatch would have created 5µs latency variation over 100ms without tracking. Both filters exhibit similar jitter performance, which improves when inverse weight $1/a$ and M grow. In Fig. 3d and 3e we measure the maximum jitter experienced by the flow, as a function of the filter type and jitter in the bus. When the inverse weight parameter $1/a$ coincides with the number of moving average samples M and increases, the residual jitter is found to increase moderately faster with MA than IIR filters. Overall, when we increase the bus jitter to 27µs, we report up to x30 reduction of jitter, particularly when the filter parameters are large (2^{14}). These trends are confirmed in the PDFs of Fig. 3c. In Fig. 3f we summarize the work above by highlighting the trade-off between residual jitter and set up time. Choosing a moving average filter provides shorter rise time and enable faster deterministic route creation than an IIR filter for accelerated distributed computing [7]. The setup synchronization times of Fig. 3f, ranging from 0.5ms to 100ms should be benchmarked with the expected duration of computer flows in edge data centers. Examples of previous reports indicate that map and reduce processes could take 23ms average time to complete in a fully synchronous data center [7], while flows in today's asynchronous data centers could typically last 10ms median duration time [8]. These figures show the importance of considering not only jitter but also fast dynamics when selecting the appropriate technologies for efficient deterministic operation of edge networks.

Conclusion: With our two techniques of jitter compensation at the egress of an optical bus, we demonstrated up to 20x and 50x, respectively, faster synchronization set-up time than in past reports, and x30 jitter reduction factor down to 70ns. This rapid setup time enables rapid reconfiguration of deterministic routes in an unsynchronized network.

References

[1] K. Christodoulopoulos *et al.*, 'Enabling the Scalability of Industrial Networks by Independent Scheduling Domains', OFC 2020
[2] N. Benzaoui *et al.*, 'DDX Add-On Card: Transforming Any Optical Legacy Network into a Deterministic Infrastructure', ECOC 2021
[3] J. Joung *et al.*, 'Zero Jitter for Deterministic Networks Without Time-Synchronization', *IEEE Access*, vol. 9, pp. 49398–49414, 2021.
[4] W. Lautenschlaeger *et al.*, 'Prototyping Optical Ethernet—A Network for Distributed Data Centers in the Edge Cloud', *J. Opt. Commun. Netw.*, vol. 10, no. 12, p. 1005, Dec. 2018.
[5] C. Demichelis *et al.*, 'IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)', RFC Editor, RFC3393, Nov. 2002.
[6] G. Soudais *et al.*, 'Per Packet Distributed Monitoring Plane with Nanoseconds Measurements Precision', ECOC 2021, Bordeaux
[7] S. Sahoo *et al.*, 'Introducing Best-in-Class Service Level Agreement for Time-Sensitive Edge Computing', ECOC 2021, Bordeaux,
[8] T. Benson et al. "Network Traffic Characteristics of Data Centers in the Wild," in Proceedings of the 10th ACM SIGCOMM, 2010.