In-Line Transmission Parameters Synchronization Protocol for Hitless Optical Coherent Communication

Alexandre GOUIN
Nokia Bell Labs
Villarceaux, France
alexandre.gouin@nokia.com

Arnaud DUPAS
Nokia Bell Labs
Villarceaux, France
arnaud.dupas@nokia.com

Patricia LAYEC
Nokia Bell Labs
Villarceaux, France
patricia.layec@nokia.com

***Abstract*— This paper proposes a protocol to share and synchronize transmission parameters between a transmitter and a receiver of two optical transponders to allow for hitless communication. It has been validated on a 100G real-time platform in terms of latency, adding only 12ns to packet round latency, and in terms of frame loss.**

Keywords—protocol, reconfiguration, hitless, optical communication, FPGA

I. Introduction

With the advent of Software Defined Networks (SDN) and the introduction of Elastic Optical Networks (EON), metro network operators now have the possibility to change at any moment different parameters of the optical line to adapt them to their needs. Whether a new equipment has been installed, a degradation in transmission quality has been detected or a restoration path has been created, the transmitter can be ordered to change its parameters, such as baud rate, modulation format and wavelength. Operators can also have access to more advanced features such as probabilistic shaping and it is expected that networks become more and more programmable in the future, and this will help for the increasing need in data rates and low latency induced by the introduction of 5G. But a change of parameters can temporarily bring down or degrade the communication. In [1], commercial equipment needs 35s to switch from QPSK to 16QAM while real-time FPGA prototype needs approximately 1s [2]. Even if a transmitter can reconfigure itself and prevent packet loss during the process [3, 4], on the receiver side, it may also be beneficial to make some changes to align with the new transmission parameters and attain maximum reception performance. Additionally, not all receivers can adapt to all changes in transmission, so it is important to prevent any possible incompatibilities. This consideration will consequently allow reconfiguration operations between different generation of transmitter and receiver or a multi-vendor scenario.

In this paper we propose a new solution that uses the data line between two transponders, so that they can share with one another their transmission parameters and reconfigure themselves while maintaining hitless communication, i.e. with no packet loss. We also show an experimental set-up on a FPGA-based programmable platform to handle the protocol operations.

II. In-Line Synchronization Protocol

We propose a new protocol for the synchronization of transmission parameters using the data path between two transponders. It was designed with two features in mind: to allow for hitless communication and to prevent any incompatibilities between transmitters and receivers. The sequence of operations is represented in Figure 1.

Whenever the transmitter of an optical transponder is asked to change one of its transmission parameters, by the action of a centralized network controller for example, it sends a message to the receiving transponder by adding a few bytes into the data stream. For instance, it can be located in the ODU (Optical Data Unit) header in the fields specifically made for experimentation, or after the header in the Ethernet frame. In both cases, the messages are required to be kept short: only two bytes are available per ODU header, and it is not desirable to insert too much data into the Ethernet traffic.

The first message in the sequence of operations is the Request (RQST) message (Fig. 1 Step 2). This message contains the parameters that will change and to which value. It also contains the number of training frames that will be sent by the transmitter after the change of parameters if needed. Training frames are meant to help the receiver algorithms to converge to new set of parameters or delay enough the resumption of communication while the receiver is reconfiguring itself. The parameters and their values are written in the message as shared identifiers that are known inside the network and compatible with the SDN control plane list of parameters. When the receiver detects the message from the transmitter, it prepares the acknowledge (ACK) message (Fig 1. Step 3), where it says if it can handle the change of parameter or if it needs more time to prepare itself after the reconfiguration. The answer is sent back to the first

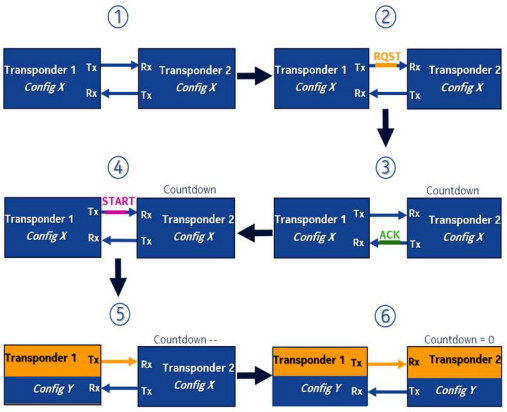

Fig 1. Sequence of operations for our proposed protocol.

978-3-903176-21-8 © 2020 IFIP

transponder, also via the data path. When both entities have agreed on parameters to change and the needed training frames, the transmitter sends a specific short START message (Fig. 1 Step 4) when it begins its reconfiguration. This START message can notify the receiver if multiple previously acknowledged parameters are reconfigured at once. During the reconfiguration time, the incoming frames are stored in memories and are to be redistributed as the traffic starts again. When the change of parameters is completed, the previously requested training frames are sent (Fig. Step 5), and then normal operations resume.

This protocol ensures the hitless property of the communication and that we have maximum compatibility between two equipment. The choice of doing these steps using the data line has been made by the fact that the network controller is oftentimes at very different distance from the network transponders and they use multiple layers of protocol that can take time to process, making complicated a tight synchronization. We believe that this protocol will allow for more reliable and more frequent reconfiguration across the whole network, from metro and DCI networks to core networks with longer transmission distances. It will give more freedom for the operators to adapt their equipment to new situations.

III. Experimental Setup

To realize a real-time implementation of our protocol we designed a system, illustrated in Fig. 2, comprised of a Spirent network analyser and two Xilinx Ultrascale+ FPGA boards with a MicroBlaze [5] processor with the FreeRTOS [6] operating system. All 100G ports are equipped with QSFP28 (Quad Small Form-factor Pluggable) modules. Standard 100G MAC are used in the FPGAs. The Spirent generates 100G Ethernet frames that are processed and transferred from one FPGA to the other, then back to the Spirent. We developed logic modules to handle the protocol operations inside the FPGA. The two logical modules are an Inserter, that inserts inside the incoming traffic a message, described in section 2, according to a set of parameters transmitted by the processor, and a Detector, that detects and stores the messages that are

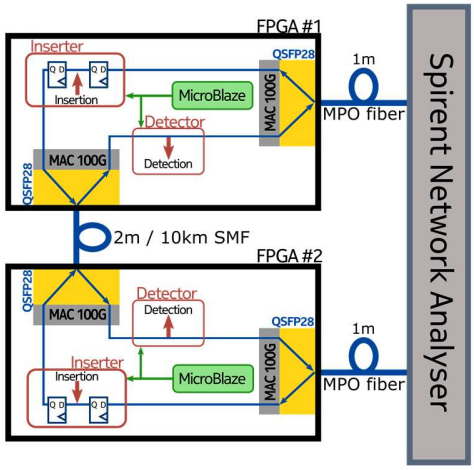

Fig 2. Schematic representation of the real-time implementation of our system

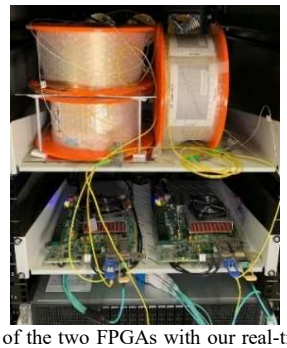

Fig 3. Photograph of the two FPGAs with our real-time implementation separated by 10km single mode fiber

found on the data path. With the help of AXI-4 protocol, the processor is able to manage and control the modules. It can also send and receive various messages and alarms from and to an SDN control plane. As the message insertion takes place between two 322MHz clocked shift registers, we can expect our logical modules to add a mere 12ns in round trip time :

$$2 * \left(2 * \frac{1}{322\ MHz}\right) = 12{,}4\ ns$$

IV. Measurements and Validation

To characterize our design, we measured the latency of our system and compared it to a reference design which is the same as our real-time implementation but without the Inserter and Detector logical modules (the parts in red in Fig. 2).

In the first experiment, the two FPGAs are connected to each other with 2m SMF (Single Mode Fiber) and are connected to the Spirent network analyzer with 1m MPO fiber. With the Spirent, we generate 10s of 100G Ethernet traffic at varying port loads and frame sizes, respectively from 10 to 100% and from 128 to 1518 bytes. Our system also inserts 32 bits messages into the traffic at the rate of 100 messages per second. The message size has been chosen to be as small as possible and still be able to carry enough information for our protocol. The quantity of messages per second has been chosen to insert a significant amount of messages during the generation of traffic, but we do not expect real-life applications to insert as many messages in the traffic, as reconfigurations of transmission parameters are still an exceptionnal process. After that we analyze the outcoming traffic from our systems and the results in latency, measured as a round trip time (RTT), and in frame loss, mesured as parts per thousand frames received with errors by the analyzer, are in Fig. 4 and 5 respectively.

A second experiment is realized with 10km of SMF fiber between both FPGA boards, each equipped with one QSFP28-LR module (see photo in Fig. 3). The Spirent and our inserter behave just as the previous experiment. The 10km fiber length has been chosen as it is the maximum distance for our QSFP28-LR modules. The results for latency and frame loss are compiled in Fig. 6 and 7.

In the graphs in Fig. 4, if we take for example the 512 bytes frame size and we compare the difference in round trip time for loads at 50% and 100% we have for the reference

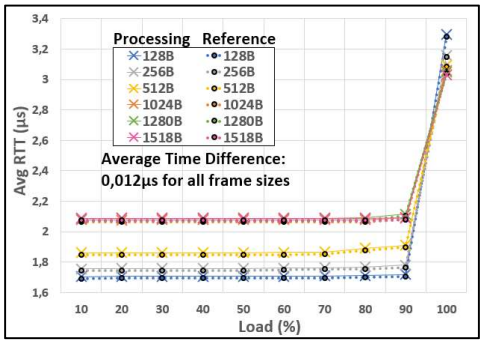

Fig 4. Average Round Trip Time (RTT) in µs for multiple frame sizes in function of port load in %. The two FPGAs are separated by 2m SMF

	128 bytes	256 bytes	512 bytes	1024 bytes	1280 bytes	1518 bytes
Reference	0.012	0.009	0.01	0.009	0.013	0.048
With Processing	0.013	0.01	0.012	0.01	0.016	0.052

Fig 5. Frame loss in ‰ for multiple frame sizes at 100% port load with 2m SMF separating the two FPGAs. Frame loss is null for all other loads

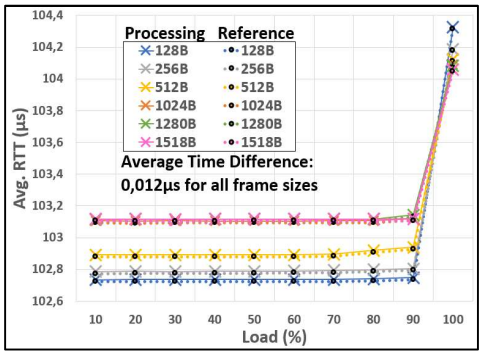

Fig 6. Average RTT in µs for multiple frame sizes in function of port load in %. The two FPGAs are separated by 10km SMF.

	128 bytes	256 bytes	512 bytes	1024 bytes	1280 bytes	1518 bytes
Reference	0.013	0.011	0.013	0.011	0.015	0.051
With Processing	0.013	0.01	0.012	0.01	0.016	0.052

Fig 7. Frame loss in ‰ for multiple frame sizes at 100% port load with 10km SMF separating the two FPGAs. Frame loss is null for all other loads.

design a RTT of 1.85µs and 3.082µs respectively, and for the design with the processing 1.862µs and 3.094µs respectively. If we select the same parameters when the two FPGAs are separated by 10km SMF (Fig. 6) we have a round trip time of 102.8µs and 104.112µs for the reference, and 102.892µs and 104.124µs for the processing design. In these cases the RTT difference is 12ns and is unaffected by the distance and the port load. In average, the reference design and the design with processing have 12ns of RTT difference for all frame sizes, with a standard deviation of 1ns. This measurement is in line with how we implemented our system in section 3. Frame loss is comparable between the two designs in both 2m and 10km fiber separation as the difference is very small, for example

```
a) <17:04:46.113> Sending Request message...
   <17:04:46.120> Returning to Idle Task...
   <17:04:46.160> !!! Message Catched !!!
   <17:04:46.183> Received ACK message
   <17:04:46.188> Returning to Idle task...
   <17:04:47.752> Sending Countdown message...
   <17:04:47.755> Waiting for the end of reconfiguration...
   <17:04:47.759> Reconfig Done signal received !
   <17:04:47.759> Returning to Idle Task...

b) <17:04:46.124> !!! Message Catched !!!
   <17:04:46.146> Received Request message
   <17:04:46.149> Preparing and sending answer...
   <17:04:46.153> Returning to Idle task...
   <17:04:47.756> !!! Message Catched !!!
   <17:04:47.778> Received Start message
   <17:04:47.786> Countdown started !
   <17:04:47.786> Returning to Idle task...
```

Fig 8. Extract of processor logs, when FPGA#1 (log a) instanciates a change of transmission parameter and sends messages to inform FPGA#2 (log b). Timestamping from serial communication software.

0.01‰ and 0.012‰ for the reference design and the design with processing respectively for a frame size of 512 bytes at 100% load, and is likely to come from our measurement precision. We can conclude on the frame loss by saying that it is coming from the hardware shortcomings at 100% port load and not from the implementation of our protocol.

In Fig. 8 are processor logs detailing how our two FPGAs behave while carrying out the operation of our protocol as in Fig. 1. The Spirent generates constant 100G Ethernet traffic and we simulate that we are interfacing our FPGA#1 (see Fig. 2) with a transmitter that can reconfigure itself in 30µs and send to our real-time system a signal to indicate that he finished his reconfiguration, and we simulate that FPGA#2 is interfacing itself with a receiver capable of reconfiguring himself in the time requested by FPGA#1. Messages are sent and received correctly, and the protocol is carried through to completion. Timestamping is added as an indication, as it depends on the processor speed and code optimization.

CONCLUSION

A protocol for synchronization of transmission parameters for hitless optical communication using the data path between two transponders has been proposed and a real-time implementation using FPGAs and a 100G Ethernet link has been validated experimentally, adding only 12ns of latency and no noticeable frame loss. We believe that this protocol will be useful as it allows for more frequent dynamic reconfiguration of the network's equipment thanks to the prevention of traffic drop. Our next goal is to demonstrate the usefulness of our protocol by interfacing ourselves with commercial hardware.

REFERENCES

[1] Y. Zhou, et al., "Field Trial Demonstration of Real-Time Optical Superchannel Transport up to 5.6 Tb/s Over 359 km and 2 Tb/s Over a Live 727 km Flexible Grid Optical Link Using 64 GBaud Software Configurable Transponders," in J. Lightw. Technol., vol. 35, no 3, 2017

[2] S. Yan, et al. "Demonstration of Real-Time Modulation-Adaptable Transmitter," in Proc. ECOC, 2017

[3] A. Dupas et al., "Elastic optical interface with variable baudrate: Architecture and proof-of-concept" IEEE/OSA Journal of Optical Communications and Networking, vol. 9, no. 2, pp. A170–A175, Feb. 2017.

[4] F. Cugini *et al.*, "Benefits of Active Stateful PCE for Flexgrid Networks," in *Optical Fiber Communication Conference (2014), paper Th3I.1*, 2014

[5] www.xilinx.com/microblaze

[6] www.freertos.org