PREPRINTS

ECRTS
9-12 July
Paris, France
2013

12th International Workshop on Real Time Networks RTN'13

Paris, France, July 9, 2013
In conjunction with the 25th ECRTS

http://irt.enseeiht.fr/scharbarg/rtn2013.html

Chairs:

Jean-Dominique Decotignie
CSEM
Jean-Luc Scharbarg
Université de Toulouse – IRIT/INPT/ENSEEIHT
Eduardo Tovar
IPP-HURRAY

Workshop Chairs

Program Committee

Advance Program

8:30-9:15	**Registration**
9:15-9:30	**Welcome and opening remarks**
9:30-11:00	**Session 1 - Keynote Talk** *Taking up the challenge of a Co-Design approach for Networked Control Systems in a Wireless Context* Guy Juanole
11:00-11:30	**Coffee Break**
11:30-13:00	**Session 2** *PCI Express as a Killer of Software-based Real-Time Ethernet* Rostislav Lisovy, Michal Sojka and Zdenek Hanzalek *Probabilistic timing analysis of a simple network switch: some preliminary investigations* Michael Short and Muneeb Dawood *On the Gap Between Mathematical Modeling and Measurement Analysis for Performance Evaluation of the 802.15.4 MAC Protocol* Francois Despaux, Ye-Qiong Song and Abdelkader Lahmadi
13:00-14:30	**Lunch**
14:30-16:00	**Session 3** *Calculation of Worst Case Backlog for AFDX Buffers with Two Priority Levels using Trajectory Approach* Naga Rajesh Garikiparthi, Rodrigo Coelho and Gerhard Fohler *MTU Assignment in a Master-Slave Switched Ethernet Network* Mohammad Ashjaei, Moris Behnam, Luis Almeida and Thomas Nolte *Implementing Virtual Channels in Ethernet using Hierarchical Sporadic Servers* Zahid Iqbal, Luis Almeida and Moris Behnam
16:00-16:30	**Coffee Break**
16:30-18:00	**Panel discussion and closing remarks**

Session 1 -Keynote talk

Taking up the challenge of a Co-Design approach for Networked Control Systems in a Wireless Context

Guy Juanole
LAAS-CNRS, Toulouse, France

Abstract

The Networked Control Systems are a very important research area because of their multidisciplinary aspect (Automatic Control, Computer Science, Communication Network). Here we only consider the aspect (Automatic Control, Communication Network) by focusing on Wireless LANs (WLANs) based on a Collision -Free CSMA type MAC protocol (the collision -free property is got by using priorities which allow to transform a situation, which would be a "collision situation " with a strict CSMA type protocol, into a "winner-looser(s) situation"; such a protocol is called the CANlike protocol). The aim of the paper is, by considering several process control applications distributed on a WLAN, to present a co-design ,of the frame scheduling of the frames of the process control applications AND of the controllers of these applications, on the basis of a bidirectional relation between the Quality of Control (QoC) provided by the controllers and the Quality of Service(QoS) provided by the scheduling of the frames on the WLAN (relation QoC $\leftarrow$ QoS) i-e we have both the relation QoC $\rightarrow$ QoS (QoS is QoC driven i-e we have a "Application performances aware dynamic QoS adaptation") and the relation QoS $\rightarrow$ QoC (QoC is QoS driven i-e we have a "Network performances aware dynamic QoC adaptation"). The relation QoC $\rightarrow$ QoS is implemented by considering hybrid priorities for the frame scheduling(it is the dynamic part of these priorities which provides this relation QoC $\rightarrow$ QoS) .The relation QoS $\rightarrow$ QoC is implemented on the basis of the delay compensation method called dominant pole method. Finally we show the interest of the proposed co-design approach in order to have an efficient control system.

Session 2

PCI Express as a Killer of Software-based Real-Time Ethernet

Rostislav Lisový, Michal Sojka, Zdeněk Hanzálek
Czech Technical University in Prague,
Faculty of Electrical Engineering
Technická 2, 121 35 Prague 6, Czech Republic
Email: {lisovros,sojkam1,hanzalek}@fel.cvut.cz

***Abstract*—The time-triggered Ethernet gains in popularity in many different industrial applications. While several hardware implementations exist, software implementations are also very attractive for their price-to-performance ratio. The main parameter that influences the performance of time-triggered protocols is the transmission jitter, which is greater in software implementations.**

In this paper we evaluate one source of transmission jitter occurring in such software implementations – the PCI Express bus, which interconnects the CPU, memory and the network interface card in modern multi-core computers. We show that the contribution of the PCI Express to the transmission jitter of Ethernet frames is significant and is in the same order of magnitude as the scheduling jitter of modern real-time operating systems. PCI Express latency and jitter are evaluated under various loads produced by virtual machines running on dedicated CPU cores. We use the IEEE 1588 feature of the network card for precise timing measurements.

I. Introduction

Ethernet-based networks are becoming more and more popular in industrial communication. This is because it is a historically well proven technology, it offers high bandwidth and many real-time (RT) extensions that make the Ethernet communication deterministic exist. Unfortunately, there is no single universal real-time Ethernet extension. Several companies offer their proprietary solutions [1]. Together with the high price of those solutions, this might be the reason why software-based real-time Ethernet implementations are so popular [2–6].

We investigate the possibility of implementing a software-based real-time Ethernet protocol while utilizing the extensive virtualization capabilities of modern x86 hardware. Our focus is on the commercial-off-the-shelf (COTS) networking and computing hardware, which is gaining in popularity for industrial automation, not only because its favorable price and widespread availability but also because of the familiar environment when used with any of the real-time Linux derivatives.

One way of achieving deterministic medium access is to use *time division multiple access* method employed by the so called *time-triggered* (TT) protocols [7, 8]. TT protocols need to maintain a notion of global time in order to synchronize the transmission in all nodes. One way to achieve this is to use *Precision Time Protocol* (PTP), standardized in IEEE 1588 [9], that allows one to synchronize the time with sub-microsecond precision over the Ethernet. The advantages of TT networks are determinism and trivial evaluation of the worst-case behavior. A disadvantage is inefficient use of available bandwidth, because temporarily unused slots must be either retained in the schedule or complex rules for their skipping must be introduced. In addition, if the used technology exhibits transmission (TX) jitter[1], which is common with software-based solutions, it is necessary to insert large inter-frame gaps that decrease bandwidth utilization even more. Examples of TT protocols are TTEthernet [8], ProfiNet IRT [10] or FlexRay [11].

Some of the drawbacks of TT protocols are mitigated by event-triggered protocols. There, the medium access is controlled by the reception of specific messages from other nodes. For example *Ethernet Powerlink* [5] has a managing node that controls it when so called controlled nodes can access the medium. In *Node Ordered Protocol* [12], the medium access is determined by the predefined order of nodes. Another principle is used in *Avionics Full-Duplex Switched Ethernet* (AFDX) [4], which employs bandwidth limiting to ensure that the network is not overloaded and latencies remain low.

For today's industry, determinism of the network communication is necessary but not sufficient. The efficiency of resource usage is also important but it contradicts the demand for determinism. Therefore, there are attempts to integrate multiple subsystems of different criticality in a single platform to improve the efficiency. This contrasts to the federated principle applied so far, where every subsystem was implemented as a separate node. The examples of modern integrated architectures are IMA [13] in avionics and AUTOSAR [14] in the automotive domain. One of the means for efficient integration of subsystems is the use of *virtualization*. Here, *hypervisors* are used to provide strict separation of independent subsystems [15] allowing one to build a mixed-criticality system.

In the past, it was believed that the biggest source of TX jitter occurring in software implementations of the real-time Ethernet was the operating system's (OS) scheduler [3]. With appropriate hardware and modern RT operating systems, the worst-case scheduling latencies are below $30\,\mu s$ [16]. Nowadays, with the advent of multi-core CPUs, it is possible to dedicate one or more cores for network processing and completely eliminate the non-determinism of the OS scheduler. We have performed this by using a NOVA microhypervisor [17]. We isolate one CPU from all unrelated activities such as timer interrupts. This is not yet possible with standard Linux and their virtualization platforms such as KVM. Moreover, the

[1]*Transmission jitter* is the difference between maximum and minimum deviation from the intended transmission time.

minimalist design of NOVA, together with a small memory footprint of our network processing code, ensures that all the network processing code and related data can be fetched from the CPU's private level-2 cache memory without interference caused by memory traffic from other cores. Additionally, the isolation and fault-containment properties of the NOVA system make it suitable for use in safety-critical environments, which would be impossible for systems such as Linux or RTAI, where a huge amount of code (Linux kernel) runs in the most privileged CPU mode.

We believed that our implementation outlined in the previous paragraph would provide very good performance, especially in terms of TX jitter figures. To our surprise, the real jitter was greater than $10\,\mu s$, which was comparable to other Linux-based solutions found in the literature. Therefore, we decided to investigate the cause of it.

The contributions of this paper are: We evaluate the properties of the PCI Express bus (PCIe), which interconnects the CPU, memory and the network interface card (NIC). We show that the contribution of the PCIe to the TX jitter of Ethernet frames is significant. PCIe latency and jitter are evaluated under various loads produced by virtual machines running on other CPU cores. We use the IEEE 1588 feature of the NIC for precise timing measurements. Our findings are useful for all SW-based real-time protocols implemented on modern x86 hardware. For time-triggered networks our results can be used to determine the proper size of inter-frame gaps. For event-triggered networks, the TX jitter influences the timing precision, which might be an important parameter for many applications.

The paper is structured as follows: After reviewing the related work in Section II, we describe the architecture of modern computers and of our hardware and software used for measurements in Section III. The results of our experiments are presented in Section IV and we conclude with Section V.

II. Related work

Many software implementations of real-time Ethernet exist. Probably, the most well known is RTnet [18]. It is a generic networking stack for RTAI and Xenomai – real-time extensions of Linux. As RTnet is implemented as a kernel module sharing an address space with the Linux kernel, it is not well suited for safety-critical applications.

Grillinger, Ademaj, Steinhammer, *et al.* [3] describe software implementation of the Time-Triggered Ethernet (TTE) implemented in RTAI. The authors evaluated the achieved latencies and jitters and found them in the order of tenths of microseconds. They claim that the main bottleneck of their implementation is the interrupt latency that influences the precision of software packet timestamping and that hardware time stamping would help. In this paper, we show that despite the fact that hardware timestamping is used, the PCI Express causes significant jitter.

Bartols, Steinbach, Korf, *et al.* [19] analyzed the latencies of the TTE hardware by using a Linux kernel with rt_preempt patches. They implemented software-based timestamping of the packets and report that the precision of their measurements is in units of microseconds. Since the system used for their measurement was based on a PCI Express bus, it is questionable whether the precision was really so low. We show that a PCI Express can introduce jitter over $10\,\mu s$.

Cena, Cereia, Bertolotti, *et al.* [20] describe a software implementation of the IEEE 1588 time synchronization protocol based on RTnet. The accuracy of their implementation is assessed by generating a signal on a parallel port of a PC and measuring the properties of that signal. Since the parallel port is connected over a slow LPC bus as detailed in Section III-A, the jitter of the parallel port's generated signal is also influenced by the PCI Express jitter, which can be quite high.

Pure software implementation of the OpenPOWERLINK, open source Industrial Ethernet solution, are described in [5]. Safety-certifiable software implementation of the AFDX stack and the achieved latencies are analyzed in [4]. None of those papers give sufficient details on the CPU-NIC interconnect.

III. Architecture

A. *Today's PC architecture*

The architecture of the modern PC and of many industrial computers is determined by the architecture of the PCI Express (PCIe) bus [21]. The central component called a *root complex* connects the CPU cores with memory controllers and other peripherals (Fig. 1). It is usually integrated on the same chip as the CPU. The other components of the PCIe topology are *endpoints*, which usually represent devices, and *switches*, which are responsible for interconnecting all of the endpoints with the root complex. All those components are interconnected via PCIe links that are formed by one or more lanes. The more lanes the higher bandwidth of the link. N-lane link is denoted as xN. All PCIe communication is packet-based and packets are routed from the root complex via switches to the destination endpoints and back. Since one link can transfer only one packet in one direction at a time, packets may be delayed by waiting for a free link.

Root complex typically has several PCIe links. In PCs, one is dedicated to a graphics adapter, another is connected to a so called *platform controller hub* [22] (or chipset in short). It contains PCIe switch(es) interconnecting different PCIe endpoints and conceivably a bridge to the legacy PCI bus. PCH also integrates other controllers such as USB, SATA, LPC (*low pin count interface* – used to connect legacy peripherals such as a parallel port or a PS/2 keyboard). Those additional controllers appear to the operating system as PCI devices. Besides PCI devices, PCH also contains non-PCI devices such as high-precision event timers (HPET).

Due to the packet-based character of the PCIe communication, sharing of PCI links between devices and several sources of latency in the PCIe communication protocol [23] (e.g. the need for acknowledging received packets), the total latency of PCIe communication can be relatively high compared to an older parallel PCI bus.

B. *Intel 82576 Gigabit Ethernet Controller*

In our experiments, presented in Section IV, we used a modern network interface card (NIC) based on Intel's 82576 Gigabit Ethernet controller. The main reason we chose this

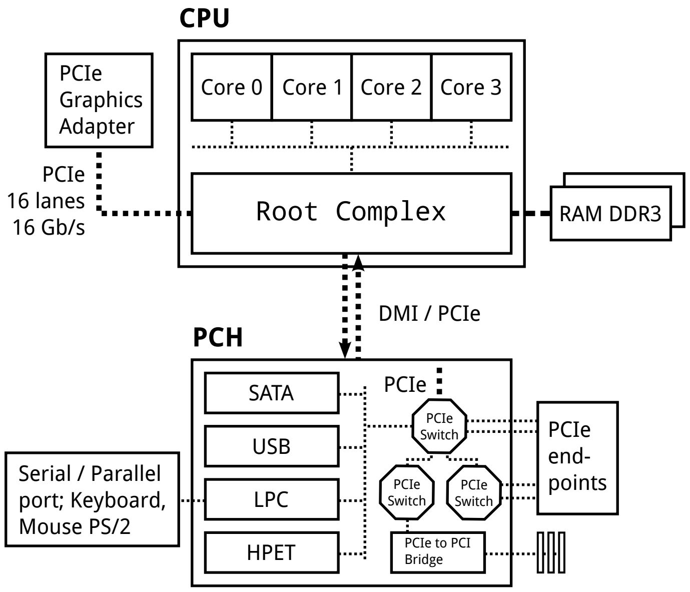


Figure 1. Typical architecture of a modern PC

COTS NIC was the built-in hardware support for the IEEE 1588 standard. This support was used for precise measurements of the PCIe latencies in this paper. The NIC contains two Ethernet controllers but in our experiments we use only one of them.

The key features supporting the implementation of PTP on this device are an adjustable clock and hardware timestamping.

The adjustable clock is implemented as a 64-bit up counter. The readout is possible through two 32-bit registers (the higher half of the value is latched when the lower half is read). The clock is periodically incremented. Both, the period and the increment are configurable. The increment period can be set as a multiple of 16 ns.

The hardware timestamping feature allows one to capture timestamps (i.e. the value of clock described above) of the received PTP packets and of arbitrary transmitted packets. Only one RX and TX timestamp may be stored at the same time in dedicated pairs of 32-bit wide registers. The hardware responsible for timestamping is as close as possible to the PHY circuit, which performs the conversion between logical signals and the physical layer. This ensures a high precision of the captured timestamps, which are taken immediately after transmitting/receiving the Ethernet *Start of frame delimiter*.

C. Software architecture

Our longer-term goal is to build a software-based time-triggered Ethernet stack on COTS x86 computers with a NOVA microhypervisor. While this stack is not yet implemented, we outline its planned software architecture in this section, because it is the same as in our experimental setup for this paper.

The software architecture is depicted in Figure 2. The lowest level consists of a NOVA microhypervisor [17]. It is responsible for hardware resource management and scheduling and it is the only component that runs in privileged processor mode (kernel mode). Its very small trusted computing base (9 kLoC) together with the virtualization support makes it a very interesting solution for safety-critical applications. Note that in NOVA, device drivers are not the part of the kernel.

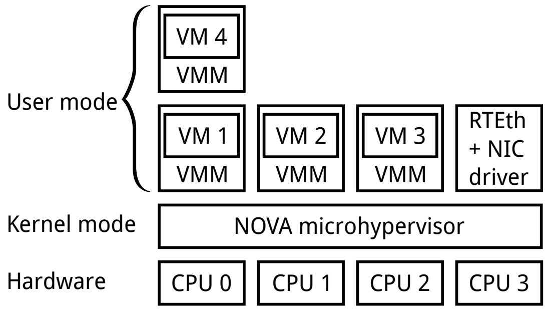


Figure 2. Software architecture of our implementation

NOVA can execute applications in two modes: native mode and virtual machines (VM). *Virtual machine monitor* (VMM) is a NOVA application running in the user mode comprising the native part that emulates the PC platform and the VM part that executes the VM code.

The code implementing the real-time Ethernet functionality is placed in RTEth application running in the user mode. It is a native NOVA application and besides other things, it contains the device driver for the NIC. The application is responsible for managing the transmission and reception of the Ethernet frames according to the predefined schedule. It is important to note, that the application does not touch (i.e. copy) the data to be transmitted or received. The data to be transmitted is stored in the main system memory directly by the application that produces it (e.g. a virtual machine). This application only notifies the RTEth application that the data is ready in the memory and RTEth instructs the NIC to transmit it. A similar principle is applied for packet reception. This is possible because of the use of the shared memory between the RTEth application and its clients. The implementation is simplified by the use of IOMMU.

The RTEth application itself is pinned to one CPU core, which is reserved solely for it. The size of application's code and data is 40 KiB, which means it fits into the CPU's 256 KiB of L2 cache. Note that the kernel code used for inter-process communication and scheduling is less than 2 KiB in size and it fits into the cache together with the application. This means that the application does not suffer from interference caused by memory traffic from other cores in the system. This reduces the execution time jitter of the application and makes its execution more predictable.

D. Testbed setup

The computer used for the evaluation was a common PC computer equipped with an Intel i5-3550 CPU (IvyBridge, 4 cores, no hyper-threading), 4 GiB of RAM and a network add-on card with an Intel 82576 GbE controller (NIC in the following). The NIC is equipped with an x4 PCIe connector.

The computer comprises two PCIe slots. One of them is an x16 slot connected directly to the root complex inside the CPU, the other, an x4 slot, is connected to the chipset (PCH).

```
+-00.0  Intel Corp. Ivy Bridge DRAM Controller
+-01.0-[01]--+-00.0  Intel Corp. 82576 Gigabit Network Connection
|            \-00.1  Intel Corp. 82576 Gigabit Network Connection
+-02.0  Intel Corp. Ivy Bridge Graphics Controller
+-14.0  Intel Corp. Panther Point USB xHCI Host Controller
+-16.0  Intel Corp. Panther Point MEI Controller #1
+-19.0  Intel Corp. 82579LM Gigabit Network Connection
+-1a.0  Intel Corp. Panther Point USB Enhanced Host Controller #2
+-1b.0  Intel Corp. Panther Point High Definition Audio Controller
+-1d.0  Intel Corp. Panther Point USB Enhanced Host Controller #1
+-1e.0-[02]--
+-1f.0  Intel Corp. Panther Point LPC Controller
+-1f.2  Intel Corp. Panther Point 6 port SATA AHCI Controller
\-1f.3  Intel Corp. Panther Point SMBus Controller
```

Figure 3. Logical PCIe topology as shown by `lspci -tv` command

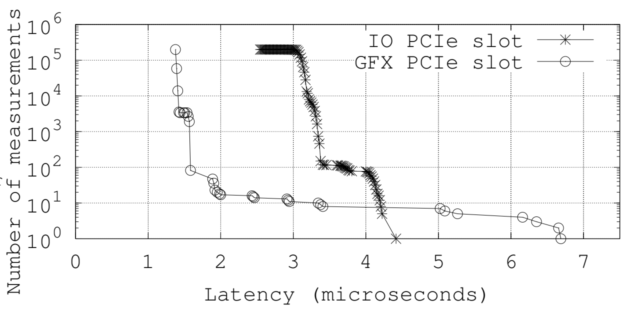


Figure 4. Latency profile of the NIC clock register readout (for NIC in two different PCIe slots). System with no load.

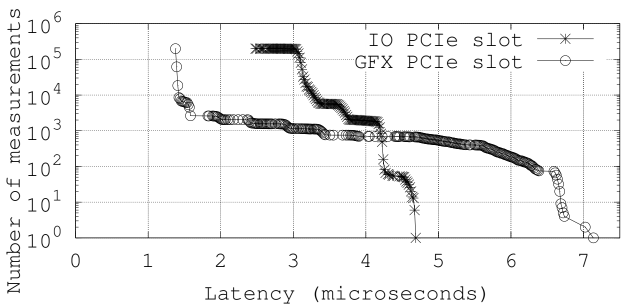


Figure 5. Latency profile of the NIC clock register readout (for NIC in two different PCIe slots). System with CPU load.

For the purpose of this paper we call the former the GFX slot and the latter the IO slot.

Besides experimenting with the NIC, we also utilized the SATA controller to generate interfering PCIe traffic. We connected a common rotating hard drive with a SATA 3.0 interface to one of the on-board SATA ports.

We tried to extract the physical PCIe topology of our system, but it does not provide the relevant PCIe capabilities to do that. The logical PCI topology presented to the operating system is flattened and does not correspond to the physical topology. Nevertheless, Figure 3 shows the output of the `lspci` tool. In this case, the NIC was connected to the GFX slot, which is denoted as PCI bus number 1 (`[01]` in the figure). When the NIC was connected to the IO slot, the corresponding entries appeared on bus 2 (`[02]` in the figure).

In our measurements, we did not identify any anomalies that could be caused by System Management Interrupts, so we did not attempt to eliminate or mitigate them.

IV. Evaluation

This section presents the results of our measurements of the PCI Express latencies. We measured two types of latencies in our experiments: the latency of the NIC clock register readout and the hardware TX latency. The experiments are described in more details in the following subsections.

All measurements were performed under several different loads of the system:

- *No load*: No intentional load was placed on the PCIe or CPU.
- *CPU load*: Three Linux 3.6 VMs running on dedicated CPUs (VM1-3 in Fig. 2) run a Sysbench CPU benchmark[2].
- *Disk load*: One Linux 3.6 VM with direct access to the SATA controller (connected to the PCIe bus) was run on a dedicated CPU. This VM was executing a `dd if=/dev/sda of=/dev/null bs=8M count=1 iflag=direct` command in an infinite loop. The size of the block (8 MB) was chosen on purpose to fit into the on-disk cache. This way, the disk traffic consumes the maximum possible PCIe bandwidth.
- *Combined load*: A combination of disk and CPU load – one VM with disk load and two VMs with CPU load.
- *Disk + serial load*: The same as disk load but the output of the `dd` command (about 100 characters) was sent to the serial line.

Besides changing the load, we also changed the PCIe slot where the NIC was plugged in during the experiment (the GFX and IO PCIe slot).

We present the results of some experiments in the form of a *latency profile*. This is a cumulative histogram of the measured values with a reversed vertical axis in log scale. The advantage over a plain cumulative histogram is that the worst-case latencies are magnified by the log scale (see for instance lower right hand corner of Fig. 4).

A. Latency of NIC clock register readout

In this experiment, we measured the time spent by reading the clock register located in the NIC. The resulting latency was calculated as the difference between the values obtained from two consecutive reads of the whole 64-bit clock register ($t_{clk2}-t_{clk1}$ in Fig. 6). Despite the fact we do not exactly know how big a fraction of the total time was spent in the NIC's internal logic and what was caused by the PCIe latencies, we believe that the measured time represents the lower bound of the communication latencies between the CPU and the NIC.

Figure 4 shows the values measured for the *no load* scenario whereas Figure 5 contains values measured with *CPU*

[2]Available from http://sysbench.sourceforge.net/; the command was `sysbench --test=cpu --cpu-max-prime=999999999 run --num-threads=1`

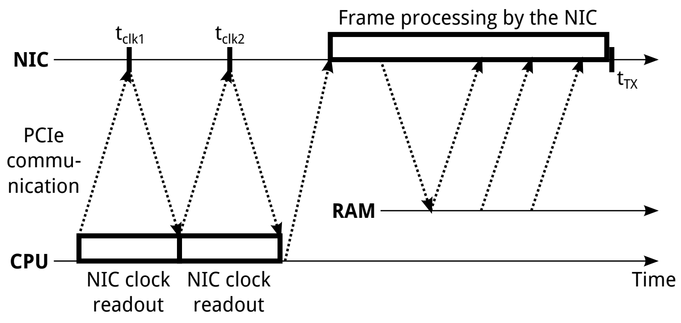


Figure 6. Explanation of the measured latencies

load. It can be seen that there are significant differences in the latency and jitter figures between the PCIe slots. We summarize the measured values in the table below:

	Avg. latency		Jitter	
Slot	*No load*	*CPU load*	*No load*	*CPU load*
GFX	$1.38\,\mu s$	$1.41\,\mu s$	$5.31\,\mu s$	$5.76\,\mu s$
IO	$3.11\,\mu s$	$3.12\,\mu s$	$1.87\,\mu s$	$2.21\,\mu s$

Figure 7 shows the results of the *disk + serial load* scenario for the NIC in the IO slot. There are periodic spikes of increased latency. Although we first thought that the spikes are caused by disk transfers, it turned out that they are brought about by communication over the serial port that we used as a console for the VM. In NOVA, the serial driver uses polling to wait for an empty TX buffer register and this results in a high PCIe bus load. In production systems, polling is avoided whenever possible but sometimes device drivers have to use polling to work around hardware bugs [24].

A careful reader can also identify a small increase in latencies (cca. $0.5\,\mu s$) with a 40 ms period in Figure 7. This was caused by updating the text screen in the VGA video RAM of the integrated graphics adapter. If the VGA is configured in NOVA, the screen of the foreground application gets copied to the VGA memory 25 times per second. A similar increase of latencies can also be seen in Fig. 4. If the external graphics adapter and/or fully graphical mode was used, the latencies could be much worse [25].

B. Hardware NIC TX latency

In this experiment, we measured the time needed by the NIC to start the transmission of a frame. More precisely, we measured the time between the moment when the NIC got the information about a new frame to send (setting the NIC TX descriptor register to point to the ready packet descriptor) and the timestamp captured by the NIC while the frame was being transmitted. During the transmission the NIC autonomously fetches the frame payload from the RAM (via PCIe).

The results presented in this section are valid for 166 byte long frames. When we increased the frame length to 1 KiB, the latency increased by $1.5\,\mu s$ in both the GFX and IO slots.

Figures 8 and 9 show the latency profiles of the TX latencies under different loads in the GFX and IO slots, respectively. The latencies were calculated as a difference between the TX timestamp and the value read from the NIC clock register just before setting the NIC TX descriptor register

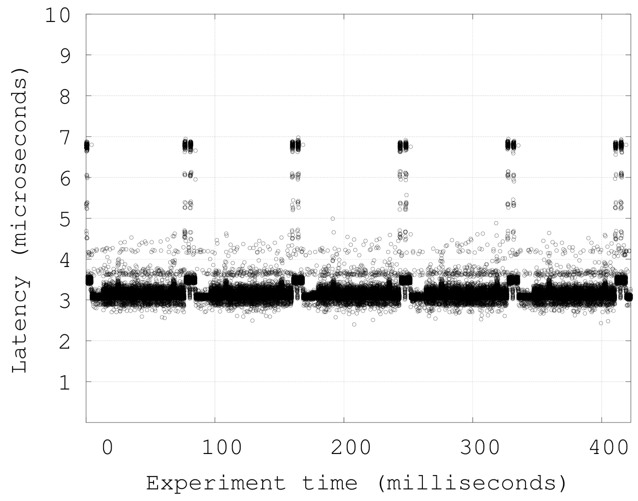


Figure 7. NIC clock readout latency in disk + serial load scenario (NIC was in the IO slot)

($t_{TX} - t_{clk2}$ in Fig. 6). The latencies for the GFX slot and IO slot ranged from $5\,\mu s$ to $14\,\mu s$ and from $8.5\,\mu s$ to $19.5\,\mu s$, respectively.

The distribution of latencies in time is shown in Figure 10. In the depicted experiment, the periods of no load and disk load scenarios were interleaved with a period approximately equal to 60 seconds. It can be seen that the distribution of the increased latencies in time is uniform.

The precision of our measurement method is influenced by the following factors: The end of the measured interval is captured with very high precision (hardware timestamp), but the start of the interval (NIC clock readout) suffers from an error in the range of several microseconds as shown in Section IV-A. If we wanted to decrease the error, we would need another clock synchronised with an NIC clock having a negligible readout time.

It is interesting to see that even a sole CPU load on unrelated CPUs caused big increases in latencies. The reason is that the CPU load makes the Linux scheduler to be invoked frequently. This resulted in about 1500 timer interrupts per second per VM. As NOVA uses HPET timers as a backend for all virtual timers, the communication between the CPU and HPET, located in the chipset, has an influence on the PCIe bus and, therefore, also on the NIC latencies.

The worst latencies were achieved for the *disk + serial load* although the sole disk load exhibits a low average latency. As mentioned above, this is caused by polling in the serial port driver. In summary, the jitter of the PCIe latency is similar for both slots and is approximately $10\,\mu s$.

V. Conclusion

With hardware support for IEEE 1588, it is possible to synchronize NIC clocks with sub-microsecond precision. However, if one wants to schedule the Ethernet traffic in the software, as many popular real-time Ethernet software stacks do, the achieved frame transmission precision is much worse. It is believed that the main cause of the transmission jitter is the

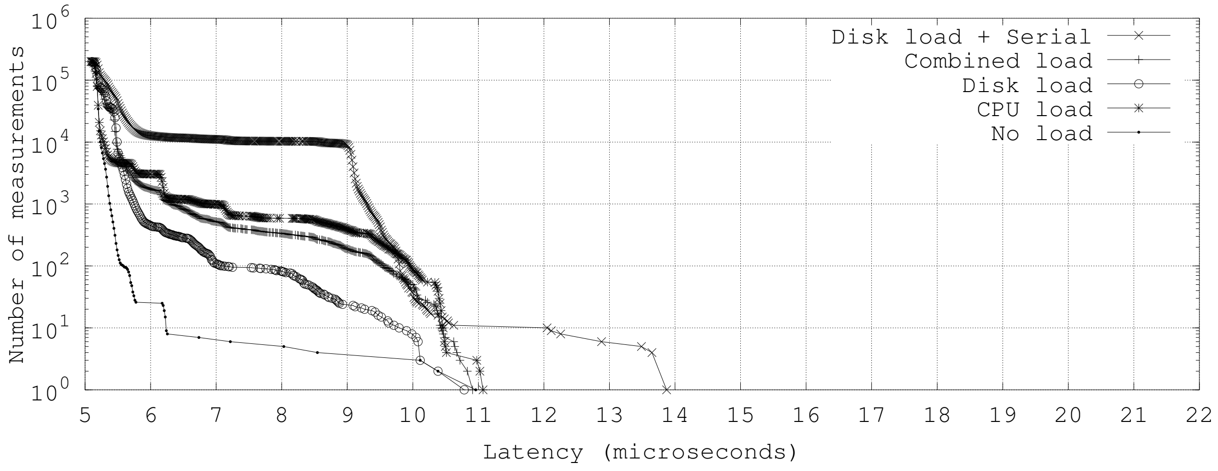


Figure 8. Latency profiles of the frame transmission on an Intel 82576 GbE controller for different PCIe and system loads (an NIC plugged into the GFX PCIe slot)

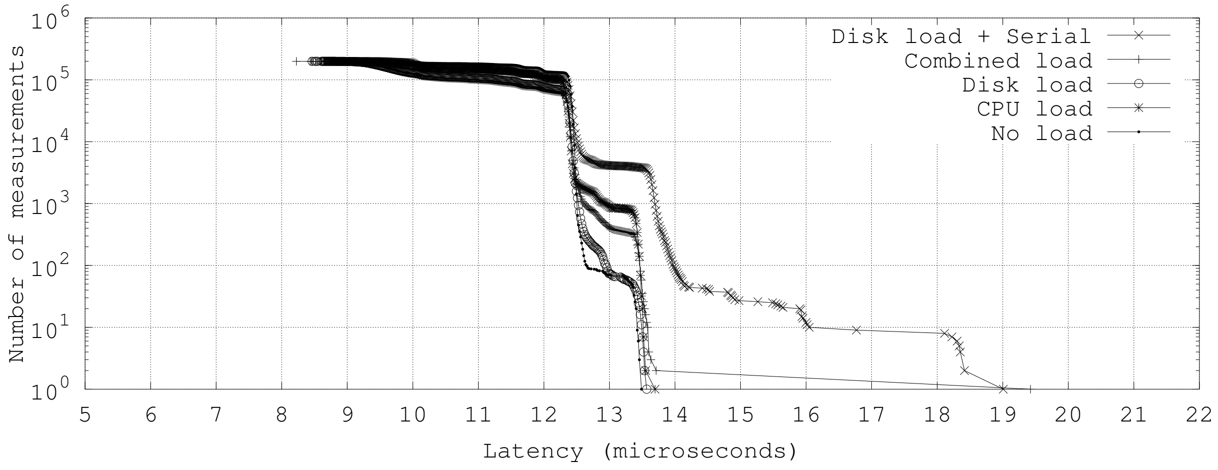


Figure 9. Latency profiles of the frame transmission on an Intel 82576 GbE controller for different PCIe and system loads (an NIC plugged into the IO PCIe slot)

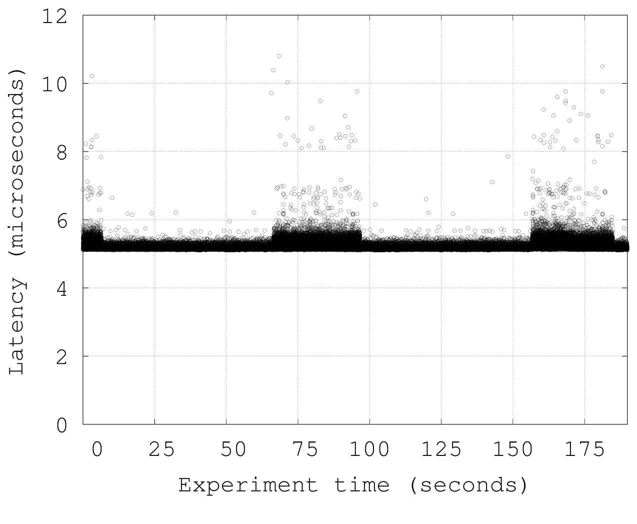


Figure 10. No load and disk load scenarios. Latencies calculated as $t_{TX} - t_{clk2}$.

scheduler of the operating system. In this paper, we identified another often neglected source of the transmission jitter, which is the PCI Express bus. We measured its contribution to the overall transmission jitter and found it to be around $10\,\mu s$. This value is in the same order of magnitude as the scheduling jitter of modern real-time operating systems.

As for our future work, we will look at improving the PCI Express induced jitter by using PCI Express QoS features such as isochronous virtual channels mentioned in [21]. We could not use them in this work, because our NIC does not support them. We plan to use the NetFPGA [26] platform to experiment with those.

Acknowledgment

The authors would like to thank Zoltan Fodor from Intel for providing the network cards for experiments.

The research leading to these results received funding from the ARTEMIS Joint Undertaking under the grant agreement n^o 295354 and from the Grant Agency of the Czech Republic under the Project GACR P103/12/1994.

References

[1] M. Felser, "Real time ethernet: standardization and implementations", in *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*, 2010, pp. 3766–3771. DOI: 10.1109/ISIE.2010.5637988.

[2] J. Kiszka, B. Wagner, Y. Zhang, and J. Broenink, "RTnet – a flexible hard real-time networking framework", in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, (Catania, Italy), Sep. 12–22, 2005. DOI: 10.1109/ETFA.2005.1612559.

[3] P. Grillinger, A. Ademaj, K. Steinhammer, and H. Kopetz, "Software implementation of a time-triggered ethernet controller", in *Factory Communication Systems, 2006 IEEE International Workshop on*, IEEE, pp. 145–150. DOI: 10.1109/WFCS.2006.1704143.

[4] I. Khazali, M. Boulais, and P. Cole, "AFDX software network stack implementation—practical lessons learned", in *Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th*, IEEE, 2009, 1–B. DOI: 10.1109/DASC.2009.5347574.

[5] J. Baumgartner and S. Schoenegger, "POWERLINK and real-time Linux: A perfect match for highest performance in real applications", in *Twelfth Real-Time Linux Workshop, Nairobi, Kenya*, 2010.

[6] J. Loeser and H. Haertig, "Low-latency hard real-time communication over switched Ethernet", in *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, 2004, pp. 13–22. DOI: 10.1109/EMRTS.2004.1310992.

[7] A. Ademaj and H. Kopetz, "Time-triggered ethernet and IEEE 1588 clock synchronization", in *Precision Clock Synchronization for Measurement, Control and Communication, 2007. ISPCS 2007. IEEE International Symposium on*, IEEE, 2007, pp. 41–43. DOI: 10.1109/ISPCS.2007.4383771.

[8] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered ethernet (TTE) design", in *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, IEEE, 2005, pp. 22–33. DOI: 10.1109/ISORC.2005.56.

[9] J. C. Eidson, *Measurement, Control, and Communication Using IEEE 1588*, 1st. Springer Publishing Company, Incorporated, 2010, ISBN: 184996565X, 9781849965651.

[10] Z. Hanzalek, P. Burget, and P. Sucha, "Profinet io irt message scheduling with temporal constraints", *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 3, pp. 369–380, 2010, ISSN: 1551-3203. DOI: 10.1109/TII.2010.2052819.

[11] FlexRay Consortium *et al.*, "Flexray communications system", *Protocol Specification Version*, vol. 2, 2005.

[12] L. Chanjuan, N. McGuire, and Z. Qingguo, "A new real-time network protocol-node order protocol", in *Proceedings of eleventh real-time Linux workshop*, Open-Source Automation Development Labs, 2009, pp. 105 –109.

[13] C. Watkins and R. Walter, "Transitioning from federated avionics architectures to Integrated Modular Avionics", in *Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th*, 2007, DOI: 10.1109/DASC.2007.4391842.

[14] AUTOSAR, *Specification of operating system*, R4.0 rev 3, Nov. 2011. [Online]. Available: http://www.autosar.org/download/R4.0/AUTOSAR_SWS_OS.pdf.

[15] C. Baumann, T. Bormer, H. Blasum, and S. Tverdyshev, "Proving memory separation in a microkernel by code level verification", in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, 2011, pp. 25–32. DOI: 10.1109/ISORCW.2011.14.

[16] C. Emde. (Oct. 2010). Long-term monitoring of apparent latency in preempt_rt Linux real-time systems, OSADL, [Online]. Available: https://www.osadl.org/fileadmin/dam/articles/Long-term-latency-monitoring.pdf (visited on 04/2013).

[17] U. Steinberg and B. Kauer, "NOVA: a microhypervisor-based secure virtualization architecture", in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10, Paris, France: ACM, 2010, pp. 209–222, ISBN: 978-1-60558-577-2. DOI: 10.1145/1755913.1755935.

[18] J. Kiszka. (Apr. 3, 2013). RTnet, [Online]. Available: http://www.rtnet.org/.

[19] F. Bartols, T. Steinbach, F. Korf, and T. C. Schmidt, "Performance analysis of time-triggered ether-networks using off-the-shelf-components", in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, IEEE, 2011, pp. 49–56. DOI: 10.1109/ISORCW.2011.16.

[20] G. Cena, M. Cereia, I. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "A software implementation of IEEE 1588 on RTAI/RTnet platforms", in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, 2010, pp. 1–8. DOI: 10.1109/ETFA.2010.5640955.

[21] PCI Special Interest Group, *PCI Express Base Specification, Revision 2.1*. PCI-SIG, 2009.

[22] Intel, *Intel® 7 series / C216 chipset family platform controller hub (PCH) datasheet*, 326776-003, Jun. 2012. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/7-series-chipset-pch-datasheet.pdf (visited on 04/2013).

[23] K. Yogendhar, V. Thyagarajan, and S. Swaminathan. (May 21, 2007). Realizing the performance potential of a PCI-Express IP, [Online]. Available: http://www.design-reuse.com/articles/15900/realizing-the-performance-potential-of-a-pci-express-ip.html.

[24] Freescale Semiconductor, *MPC5200 (L25R) errata*, Rev. 5, ATA interrupt is not affected by FIFO errors, Dec. 2011, ch. 2.1. [Online]. Available: http://www.freescale.com/files/32bit/doc/errata/MPC5200E.pdf (visited on 04/2013).

[25] M. Cereia, I. Bertolotti, and S. Scanzio, "Performance of a real-time ethercat master under linux", *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 4, pp. 679–687, 2011, ISSN: 1551-3203. DOI: 10.1109/TII.2011.2166777.

[26] NetFPGA. (2013). NetFPGA, [Online]. Available: http://netfpga.org/ (visited on 04/2013).

Probabilistic timing analysis of a simple network switch: some preliminary investigations

Michael Short and Muneeb Dawood
Electronics & Control Group / Technology Futures Institute,
Teesside University,
Middlesbrough, UK.
{m.short, m.dawood}@tees.ac.uk

Abstract**—*In this paper the timing properties of a single switch LAN are analyzed. The analysis is based upon previous work to analyze the synchronous busy period of a message set assuming FIFO buffering is employed. However in this paper the real-time traffic (periodic and sporadic) is also subject to random interference from other frames, and a probabilistic stance is taken. Some interesting observations are made based upon our initial investigations, and preliminary algorithms are proposed to estimate the worst case queuing delays at source nodes and switch output ports assuming some knowledge of the (mean) interference levels are known. The work was principally motivated by the need for easy-to-apply probabilistic timing analysis in some building and home automation implementations, which are described in the paper; it may also be applicable to certain industrial contexts. Future work will consider more complex and less restrictive scenarios including multiple switch hierarchical networks.*

Keywords—Stochastic timing analysis; Network switch; Home and building automation; Smart grids; Traffic models.

I. INTRODUCTION

Recent trends have seen an increase in the use of packet-switching technologies for the implementation of simple distributed (and possibly embedded) networks for sensing and control applications [1][2]. Providing guarantees of timely delivery in packet-switched networks is a complicated problem, as the worst-case delays incurred across multiple hops in the network must be derived. Of course it is possible to employ packet scheduling techniques, e.g., Earliest Deadline First (EDF) [3]. This can simplify the overall analysis problem, and by having appropriate admission controls a flexible yet predictable network may be implemented.

On the other hand, in some applications (e.g. home and building automation), the use of specific packet scheduling algorithms may not be practical since many standard packet-switching network components and technologies only support simple First Come First Served (FCFS) scheduling. Previous work has examined the timing properties of networks scheduled using FCFS under the assumption that traffic is implemented as a number of periodic streams [1][2]. For reasons discussed in [2], it can become very complex when applying techniques such as network calculus when traffic is periodic and FCFS is employed. In this paper, we wish to analyze the timing properties of simple packet switched LANs (using techniques similar to those of [2]) in which the real-time traffic is periodic/sporadic but also experiences random interference from other frames. Since the interference is random, probabilistic timing guarantees are appropriate; in this article we report some initial investigations and findings. The remainder of the paper is structured as follows. In Section II, our motivation is discussed. Section III presents our assumptions on the network topology and traffic models, and Section IV discusses our preliminary work on probabilistic calculation of queuing delays and buffer sizes. Section V discusses areas for future improvements.

II. MOTIVIATION: ICT ARCHITECTURES FOR ENERGY POSITIVE NEIGHBORHOODS

Loosely speaking, an Energy Positive Neighborhood (EPN) is one in which more energy is produced with renewable energy sources than is consumed at the same area (on the average) over a large enough period of time. To achieve the goal of an EPN, there are many socio-economic and technical issues to be dealt with, and the authors are currently involved in the EU-funded IDEAS[1] project to investigate solutions to some of these problems. In this paper, we are interested in the non-functional (timing) requirements that an information and communication technology (ICT) based decision support system for residents, businesses and energy providers in an EPN is likely to possess. In this respect, a key technological concept required to enable an EPN is the 'smart grid': a distribution network that not only allows for the physical transfer of energy (usually electricity but not necessarily so), but also supports ICT interfaces that enable information exchange related to energy supply/demand availability and pricing. The main components and network infrastructure that we envisage will be required for an EPN is as shown in Fig. 1.

At the heart of the infrastructure is the EPN service provider – this is a not-for-profit organizational and regulatory body that plays a similar role to the Independent System Operator (ISO) in traditional electricity generation and distribution markets [4]. However, the EPN service provider also extends the duties of the ISO (principally safety and availability of the network and management of Energy Trading (ET) activities) into several key areas to help achieve the main goal of the EPN.

[1] http://www.ideasproject.eu/

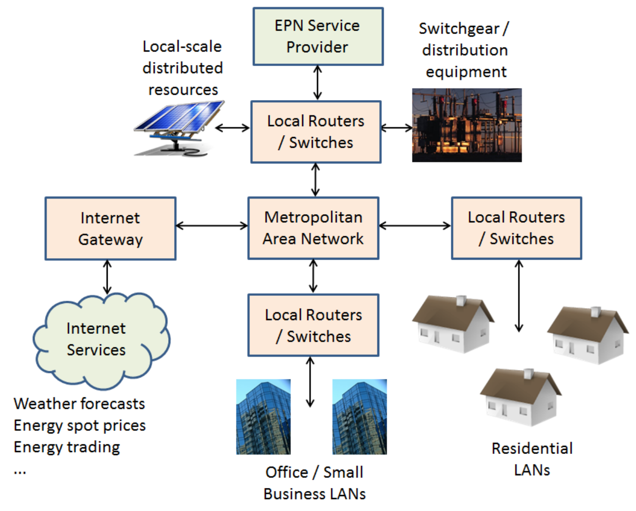


Fig. 1. Envisioned overview of the network infrastructure for an EPN.

These key areas include the maximization of renewable energy resources through both reactive and predictive optimization of the energy transfer within the distribution network (e.g. generator settings), Supply and Demand Prediction (SDP) and Demand Side Management (DSM). The EPN service provider may have to communicate with multiple Energy Companies (ECs), equipment for energy generation and distribution, residential homes and small businesses, and also wider Internet services such as weather forecasting and interactions with national energy markets. With respect to the criticality and timing constraints for traffic within the envisioned EPN infrastructure, three required classes of service can be identified:

1) Hard Real-Time: This class of traffic relates mainly to the management of critical energy generation and distribution infrastructure; although the traffic volume may be fairly low, high levels of predictability and reliability are required. The level of criticality is high because potentially dangerous overload (underload) conditions in network segments may occur, and predictability is needed to react to undersupply or over demand situations by transiently decreasing (increasing) generation or storage capacity [4][5]. In emergency situations, activation of switch gear can be used to physically disconnect equipment from the distribution grid.

2) Soft Real-Time: This class of traffic relates mainly to DSM, SDP and ET; the traffic volume may be relatively high during periods of peak energy demand or availability, and predictability is desired but not mandatory (decision support messages for customers relating to a peak demand ending at 13:00 are not useful if delivered at 13:05) [5]. Although DSM and ET can effectively help to manage undersupply or over demand situations, it is not always effective and cannot be relied upon as the only solution.

3) Near/Non-Real-Time: This class of traffic relates mainly to the acquisition of data (e.g. from smart electricity meters) to support billing and customer energy visualization activities. The main issues here are not related to timing, but are principally related to the large data volume and security (e.g. see the work of [6]). As such they are not considered further in this paper.

For implementing the critical hard real-time infrastructure communications, IEC 61850 [7] is currently the most widely used international standard for automation of power generation and distribution equipment (generators of all kinds, sub-stations, switchgear and energy storage). The goal of the standard is to define specifications such that intelligent electronic devices (IED's) from different vendors are interoperable and can communicate directly with each other without using a gateway. IEC 61850 communications are based on switched Ethernet, bringing obvious benefits such as low costs and high throughput levels. An obvious limitation of standard Ethernet is that it does not provide the required timing guarantees in more general situations, and the standard is principally designed for implementation on simple (isolated) LANs within single installations to achieve the required predictability. Although there are some extensions to support intra-station and station-control center communication, this is mainly aimed at non-critical applications such as remote monitoring. To bridge between multiple stations and extend the overall automation network into the wider EPN, the IEEE 802.1 Audio Video Bridging Standard (AVB) [8] and Time-Triggered Ethernet (TTE) protocols [9] are currently under investigation to provide the required timing and bandwidth guarantees within an EPN as part of the IDEAS project.

However in this paper, we are not especially concerned with these aspects of the EPN and our work in this area will be reported in future. With reference to Fig. 1, we are interested in the non-critical, soft real-time elements of the infrastructure; specifically, the timing behavior of the network segments that are used to extend the EPN functionality into residential and office/small business LANs. The functionalities to be extended are principally DSM and generation regulation services through non-critical automation. In most situations it will not be practical (or even feasible) to employ specialized networking equipment across all hops in these situations – in all likelihood, the principal use for these LANs and interconnects will be to implement regular Internet connectivity in houses and offices, and in most cases even primitive traffic shaping techniques may not be possible. A typical representation of the infrastructure within a domestic residency within the EPN is shown in Fig. 2. Via a series of routers/switches, EPN messages are transferred between the EPN backbone and each residence. The LANs are implemented by a simple switch or router that directs EPN traffic either to or from an automation interface, an energy measurement interface or HMI devices. In some cases, the automation interface may have a KNX or other low-level fieldbus for device connection (as is shown in the figure); alternatively, automation devices may connect directly to the central switch and form part of the LAN. As these automation and measurement interfaces are required only at the lowest levels of the infrastructure (and may already be present in new builds), extensive changes to existing network infrastructure are not required. As such, we will assume that the network switches/routers will be simple FIFO-based devices employing store-and-forward operation.

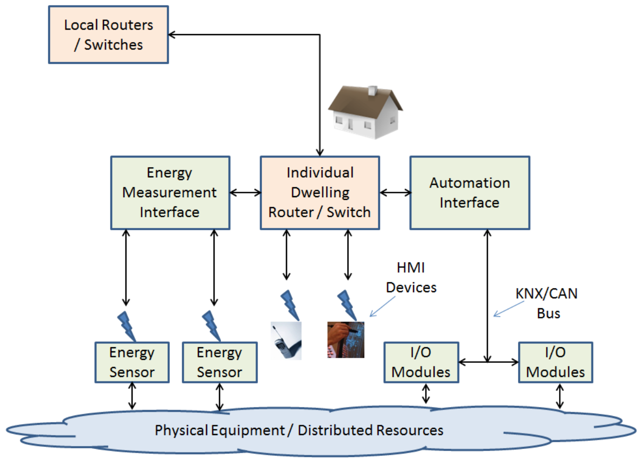


Fig. 2. Envisioned network infrastructure for residential dwellings.

We now give a *specific* motivational example as to exactly why we believe these elements of the EPN require probabilistic real-time analysis. Recall that one of the roles of the EPN service provider is to balance energy supply and demand and regulate the power flows within the EPN and the net power flow across the EPN boundary. If there is an imbalance in the net flow at some specific time, in the sense that the power into the EPN (say 75 MW) is larger than the pre-contracted amount (say 70 MW), then one of two actions can happen. The first is that the area ISO will activate regulation power on behalf of the EPN to restore the imbalance; this is a very costly process [4]. The second is that the generators within the EPN up (or down) regulate their outputs to eliminate the imbalance internally. The latter is clearly the preferable option if the costs are lower; however, the costs to up (or down) regulate large power plant at short notice are still excessive. On the other hand, the costs to regulate a distributed renewable resource (solar panel, wind turbine) are negligible if the environmental conditions are suitable, but the change in power output is low per resource (e.g. 10 KW).

Therefore the cost-minimizing action is normally to send dispatch instructions to a large amount of these distributed resources, the sum of whose outputs can impact the net power flows. These dispatch instructions will typically be transmitted once every 10/15 minutes and need to be delivered with low latency to be effective. As regular Internet and other traffic coexist with these dispatch instructions (and switch buffers are of finite size), no 100% guarantees of latency or successful delivery can be obtained. However, it is desired that around 99.9% of these dispatch instructions be successfully delivered on time; for a 1 MW requested change in net power output, a delivered change of 0.999 MW within about 60 seconds is more than sufficient. As such, we would like to obtain worst-case transmission and switch buffer size bounds that are 99.9% accurate given some knowledge of the random traffic characteristics of the LAN or LAN segments under analysis. In this preliminary work we will first make quite a lot of simplifying assumptions and restrict and then attempt to make some initial progress towards a means to obtain such estimates.

III. Network And Traffic Model

A. *Single-Switch Network Section*

The first restriction that we will make is that the residential LAN infrastructure consists of a number of stations connected via a single network switch or router, as shown in Fig. 3. We assume that if a router is present, it only carries out very simple low-level routing and so may be effectively treated as a standard FIFO-buffered switch. We will also assume that the switch is homogenous, i.e. that the incoming/outgoing bit rates of each port are identical. Although this is a very simple network to consider, it provides a representative starting point. Such a network may occur in a small-scale home or building automation installation as discussed in the previous Section, or alternately may be applicable to small-scale industrial applications where process automation and control traffic may co-exists with other (higher-level) traffic that is best described by random attributes. We assume that time is discrete and occurs in integer multiples of a global clock which has a resolution equal of δ (typically this would be the homogenous network bit-time). For simplicity, we assume henceforth that $\delta = 1$. We assume that the network segment to be analyzed consists of N_s stations connected to the single switch, which has N_p active ports such that $N_s \leq N_p$. We assume that the system to be implemented is described as a number N_c of virtual channels, with each channel mapping a logical path between a source and destination station through the switch.

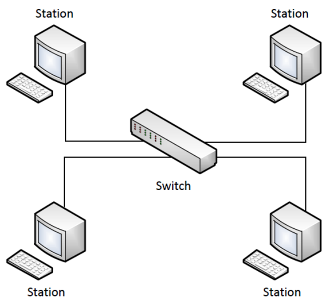


Fig. 3. Example of a single switch network such as is under consideration.

Using similar terminology as in [1] and [2], in Fig. 4 the total end-to-end delay for any virtual channel i (denoted as $T_{e2edelay,i}$) is comprised of several sub-sources of delay as shown in Fig. 4:

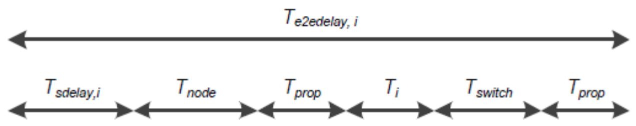


Fig. 4. Sources of delay in the single-switch network.

The components making up the $T_{e2edelay,i}$ delay are as follows: $T_{sdelay,i}$ represents the worst-case delay at the source node, and is principally due to queuing whilst awaiting access to the NIC. T_{node} represents the worst case latency for a frame in the head of the queue to leave the source node (e.g. due to non-preemption). T_{prop} represents the propagation delay over the physical link (we assume that connection cables are of identical length; an assumption which is easily lifted if required). T_i represents the worst-case delay at the switch, and is principally due to buffering whilst awaiting access to the output port. Finally, T_{switch} is the worst case latency for a frame in the head of the queue to leave a switch/port. In this paper, as in [1] and [2], we are principally concerned with determining the source node delay $T_{sdelay,i}$ and the switch delay T_i. In the analysis that follows, we are more closely follow that developed in [2] as the work of [1] – whilst being simpler in its formulation than – makes some pessimistic assumptions (e.g. that all stations may simultaneously transmit to any other station) and restricts key parameters (e.g. that periods are all greater than the worst-case transmission delay). For the remainder of the discussion, we assume that time (which is represented by t) is continuous, real-valued and non-negative. Next, we outline the deterministic and random traffic models that are employed in this preliminary study.

B. Periodic/sporadic traffic

For periodic/sporadic channels, let each channel $\tau_i \in \Gamma$ be represented by the 4-tuple:

$$\tau_i = (S_i, D_i, T_i, C_i) \tag{1}$$

In which S_i is the (integer) source station identifier and D_i is the (integer) destination station identifier. For simplicity, assume that station identifiers are identical to the port numbers (i.e. station 1 is connected to port 1, station 2 is connected to port 2, and so on). $T_i \in \aleph^+$ represents the period/minimum inter-arrival time of the channel and $C_i \in \aleph^+$ is the worst-case transmission time of any message frame generated by the channel (each invocation of the channel is called a *message frame* or simply *frame*). Let the k^{th} frame generated by channel i be denoted as $\tau_{i,k}$. Successive frames generated by sporadic channels are always separated by *at least* T_i units of time; successive frames generated by periodic channels are always separated by *exactly* T_i time units. Since it is known that the worst-case manifestation of a sporadic message stream is the pattern in which the minimum inter-arrival times are always adhered to (and the stream effectively becomes periodic) [3][4], periodic/sporadic streams will henceforth be referred to as simply periodic for ease of exposition. For periodic streams, the worst-case cumulative workload generated by stream i (denoted as w_i) in the interval $[0, t)$ can be calculated using [2]:

$$w_i(t) = \left(\left\lfloor \frac{t}{T_i} \right\rfloor + 1 \right) \cdot C_i \tag{2}$$

C. Random traffic

Many types of network traffic are essentially random in nature; it is well known that in that some circumstances frame inter-arrival times are well-modeled as exponential or geometric distributions [10][11]. Let us assume that each source station sends and receives random traffic, however this random traffic does not have a specific destination (in the case of traffic generated by a node) or a specific source (for traffic received by a node). In addition, we assume that the payload length is unknown, and make the assumption that any random frames processes by node i will always have length $\leq \hat{C}_i \in \aleph^+$ (typically, $\hat{C}_i$ will be set to the worst-case payload length allowed by the protocol for all nodes). The final assumption that we make is that the system is not closed, in the sense that random traffic in the network is just sent and received by the stations connected to the switch; an external gateway may be present, hence the total mean traffic sent and received by all nodes is not necessarily equal. Let the inter-arrival times of the random traffic generated by node i be geometrically distributed with a mean $\bar{T}_i^+ \in \Re^+$, and the inter-arrival times of traffic received by node i also be geometrically distributed with mean $\bar{T}_i^- \in \Re^+$. Then for each station i, the parameters of the random traffic can be described by the 3-tuple $\theta_i \in \Pi$:

$$\theta_i = \left(\bar{T}_i^+, \bar{T}_i^-, \hat{C}_i\right) \tag{3}$$

Given a set of stations, virtual channels and a probability $R \in [0.5, 1)$, we are interested in obtaining tight probabilistic bounds on the worst-case transmission delay that each channel may experience. Also, we are interested in determining the required source and switch buffer sizes such that probability that these timing or buffer bounds become violated is guaranteed to be $\leq (1-R)$. In order to determine this information efficiently, we require a means to upper bound the expected number of packet arrivals for a random traffic stream in some interval of time. Since time is discrete and the distribution of arrival times assumed to be geometric, the probability of a packet arrival at each individual time step (p) is equal to $1/\bar{T}$, where $\bar{T}$ is the mean inter-arrival. If the number of packet arrivals occurring in t consecutive time-steps is given by the variable X, then X follows a Binomial distribution with parameters t and p. To obtain the upper bound with confidence probability R, we therefore seek to evaluate the R^{th} quantile of X; since obtaining the exact quantile requires a (non-trivial) iterative search over the Binomal distribution function [10], we shall instead use the following upper tail quantile inequality that was recently proven:

Theorem 1: Let $\eta(t, p, R)$ represent the R^{th} quantile of a Binomially distributed random variable comprising t identical and independent Bernoulli variables, each having and individual probability of success $p \in (0, 1)$. Then defining the quantity $C(R) = \sqrt{-2\ln(1-R)}$ for $R \in [0.5, 1)$ an easily computable and asymptotically tight upper bound on $\eta(t, p, R)$ is given by:

$$\eta(t, p, R) \leq \left\lceil tp + C(R)\sqrt{tp(1-p)} + \frac{C(R)^2}{6} \right\rceil \tag{4}$$

Proof: Short & Proenza 2013 (see [11], Corollary 1).

When carrying out a timing analysis, expression (4) will normally have to be evaluated many times for different values of t. Since we assume that both $p = 1/\bar{T}$ and R are known (as mentioned, we typically take $R = 0.999$), to simplify the repeated computation of (4) for a particular link an easy simplification is to first calculate the two quantities $C_1 = \sqrt{-2\ln(1-R)\,(1-p)}$ and $C_2 = -\ln(1-R)/3$. A bound on the outgoing workload generated by random traffic originating in station j in the interval [0, t) (denoted as w_j^+) for a confidence probability R is then obtained as:

$$w_j^+(t) = \left\lceil \frac{t}{\overline{T}_j^+} + C_1\sqrt{\frac{t}{\overline{T}_j^+}} + C_2 \right\rceil \cdot \hat{C}_j \tag{5}$$

This is a simple closed-form expression which is easily computed for any of the input parameters in constant time; note that the probability that the actual workload exceeds that computed by (5) in the specified interval is formally guaranteed to be ≤ (1-R) [11]. The worst-case incoming workload for station j (denoted as w_j^-) can be obtained from (5) with appropriate replacement of $\overline{T}_j^+$ with $\overline{T}_j^-$ and appropriate adjustment of C_1. Expression (5) seems simple enough to allow an adaption of the 'busy period' analysis methods developed by Fan et al. in [2] to be adapted to the case of stochastic traffic; however there are several points that require attention prior to developing a suitable analysis.

IV. Busy Period Analysis

A. *Analysis of periodic streams*

For purely periodic streams, several key results were established in [2]; in the context of the current work, the two main points that were proven were as follows. When determining the delays due to FIFO buffering of frames at the source stations, the synchronous arrival case (all periodic streams arrive simultaneously at t = 0) is the worst possible. The worst case queuing delay (which occurs at t = 0) is finite and is given by:

$$DS_i = \sum_{\substack{\tau_j \in \Gamma \\ S_j = i}} C_j \tag{6}$$

if and only if the utilization of the outgoing channel from station i is not overloaded, i.e. has a utilization bounded as follows:

$$\sum_{\substack{\tau_j \in \Gamma \\ S_j = i}} \frac{C_j}{T_j} \le 1 \tag{7}$$

Where the summation limits of (6) and (7) are such that every valid channel in the set Γ which a source station identifier of j are included in the summation. Secondly, when determining the delay due to FIFO buffering of incoming frames at the output port i of a single switch, the synchronous arrival case (again in which all periodic streams arrive simultaneously at t = 0 in each of the source nodes) is the worst possible. The worst case queuing delay in this situation is found during the length of the initial busy period, which can be obtained as the smallest (positive) solution of the following equation (which is iterated from t = 0):

$$WP_i(t) = \sum_{\substack{\tau_j \in \Gamma \\ D_j = i}} w_j(t) = t \tag{8}$$

and has finite length if and only if the utilization of the incoming channel from station i is not overloaded, i.e. has a utilization bounded as follows:

$$\sum_{\substack{\tau_j \in \Gamma \\ D_j = i}} \frac{C_j}{T_j} \le 1 \tag{9}$$

Clearly, for the network model presented in the previous Section, if there is no random traffic then these techniques would suffice to determine the worst-case delays. When random traffic is included in the analysis, however, it is not immediately obvious the extent to which these results are still relevant. In the next two Sections, the focus will be upon delay analysis in the source and switch output ports.

B. *Source node queuing delay*

Firstly, we have observed that worst-case queuing delay in a source node no longer occurs at t = 0 due to the non-periodic nature of the workload function (5); this is highlighted by the following simple example. In Fig. 5, we compare the source node FIFO queue size $Q(t)$ of a single periodic channel with {T = 10, C = 5} with a channel experiencing only random traffic with {$\overline{T}^+$ = 10, $\hat{C}$ = 5}. A confidence probability R = 0.999 was used in the latter; the constants C_1 and C_2 required for (5) were computed as 3.526 and 2.303 respectively. The plot shows the comparison between t = 0 and t = 170, the point in time in which the queue size for the random traffic drops to zero indicating the end of the busy period. Clearly in the latter case the queue size is first increasing from $Q(0)$ = 15 and first peaks at the maximum value of $Q(28)$ = 32, before starting to decrease (non-monotonically) at t = 33. This indicates that it is likely to also need to employ busy period analysis in the source nodes when random traffic is present, unlike in the purely periodic case.

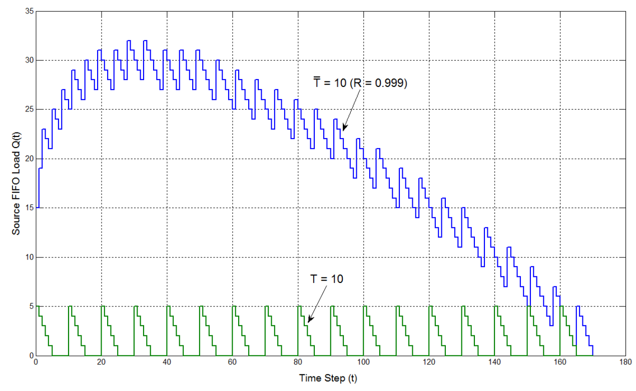


Fig. 5. Comparison of FIFO delay in a source node.

For the workload arrival function (5), it was shown in [11] that the workload is non-decreasing in t and eventually approaches (but does not exactly converge upon) the mean workload $t\hat{C}/\overline{T}^+$. This implies that the worst-case queuing delay can be obtained by examining the synchronous busy period for periodic tasks under the assumption that the random traffic also starts to arrive at t = 0. However, we first need to consider under what conditions the resulting busy period will have a finite length; a necessary condition is clearly that the utilization of the channel (including the *mean* utilization $\hat{C}/\overline{T}^{+}$ of the random traffic) does not exceed unity. Unfortunately this condition is not sufficient, as choosing an example in which there is no periodic traffic and any $\hat{C} = \overline{T}^{+} > 0$ with $R > 0.5$ can easily be verified using (5). Observing that the source traffic can be modeled by the summation of two queues (*Geo*/*D*/1 and *D*/*D*/1 in Kendall's notation [12]), it is known that such a link has a finite busy period if and only if the link utilization is strictly less than one [12]. This result also holds despite the observation that the inequality (5) is not exact due to Theorem 2 in [11], which has shown that the relative overestimation error in expression (4) vanishes for large values of *t*.

However, when the total link utilization is close to unity the length of the synchronous busy period becomes too large to analyze in a reasonable time and is sensitive to the choice of *R*. Indeed this seems to be much worse that in the purely periodic case, in which the busy period is always limited by the least common multiple (*lcm*) of the channel periods even for high utilizations. By limiting the effective allowable channel utilization to be less than some upper limit $U_M \approx 0.99$ results in tractable behavior for reliability levels in our range of interest. Therefore, in order to determine the worst-case latency the following general procedure can be used. Defining the total outgoing cumulative workload for station *i* at time *t* as $W_i^+(t)$:

$$W_i^+(t) = \left(\sum_{\substack{\tau_j \in \Gamma \\ S_j = i}} w_j(t) \right) + w_i^+(t) \tag{10}$$

Then if the busy period has length *L* the outgoing queue size $Q_i^+(t)$ for $t \le L$ is easily computed as $W_i^+(t) - t$. To find the worst case delay, one finds the extrema of the function $W_i^+(t) - t$ subject to $0 \le t \le L$. As it is assumed that time is discrete, then a simple iterative scheme may be used to solve the problem in a straightforward fashion with time complexity $O(N_s\ L)$ and space $O(N_s)$. Pseudo-code for such an algorithm is shown in Fig. 6.

```
01 INPUT(i, R, Γ, Π, δt, Ns);
02 t:=0;
03 Q:=0;
04 IF $\left( \sum_{\substack{\tau_j \in \Gamma \\ S_j = i}} \frac{C_j}{T_j} \right) + \frac{\hat{C}_i}{\overline{T}_i^+} > U_{Max}$ RETURN(∞);
05 DO:
06    W:= $\sum_{\substack{\tau_j \in \Gamma \\ S_j = i}} w_j(t) + w_i^+(t)$;
07    Q:=MAX{Q,(W-t));
08    t:=t+1;
09 WHILE((W-t)>0);
10 RETURN(Q);
```

Fig. 6. Algorithm to determine the maximum queuing delay for a single source station.

The operation of the algorithm may be briefly described as follows. Lines 2 and 3 initialize the variables *Q* and *t*, representing the worst-case queue size and time respectively. Line 4 checks the channel utilization and returns signaling an error if it is overloaded. There then follows a loop between lines 5 and 9, which terminates only when the cumulative workload $W_i^+(t)$ (represented by the variable *W*) is ≤ to *t*, indicating the presence of idle time and hence the end of the busy period. The workload *W* is updated on line 6 according to expression (17); the worst case queuing found so far is then updated on line 7, and time advanced by the factor δ (= 1) on line 8. The worst-case queuing population is then returned on line 10.

Example 1: Suppose we have the following traffic characteristics in a source node: two periodic/sporadic streams $\{T = 20, C = 3\}$ and $\{T = 30, C = 5\}$ combined with a random outgoing stream $\{\overline{T}^{+} = 10, \hat{C} = 5\}$. Application of the algorithm described above (assuming $R = 0.999$ giving $C_1 = 3.526$ and $C_2 = 2.303$) yields a worst case delay estimate of 69, which occurs at $t = 240$ assuming the start of the busy period at $t = 0$. The end of the busy period in this case occurs at $t = 1131$.

Note that these measures of delay not only represent the latency incurred by a frame when exiting the source node, they also imply a bound on the buffer size required by the node to implement the queue [2]. If the random traffic characteristics have been correctly modeled, then the probability of either of these bounds being violated is guaranteed to be ≤ (1-*R*) due to Theorem 1.

C. Switch output port queuing delay

Turning attention now to a switch output port, the analysis may progress upon the following lines. Given our assumptions upon the network topology (Fig. 3), we may observe that the traffic leaving the switch via a given output port is essentially the input traffic destined for the corresponding station. Since Fan et al. [2] have shown that for periodic streams the synchronous arrival case in each of the source nodes is the worst-possible (note that this considers *only* the traffic to be delivered to this specific output port; other port traffic is omitted from the analysis), and we have that the interference from random traffic is maximized over smaller intervals, let us again define the total incoming cumulative workload for station *i* at time *t* as $W_i^-(t)$:

$$W_i^-(t) = \left(\sum_{\substack{\tau_j \in \Gamma \\ D_j = i}} w_j(t) \right) + w_i^-(t) \tag{11}$$

The analysis to obtain the worst-case delay may then proceed along similar lines as that developed for the traffic leaving a source node, with the following caveat; the rate at which work from the input ports of the switch can be transferred to any single output port is limited by the physical design of the switch [2]. In the case of periodic and random traffic streams operating with a discrete clock having resolution δ, this restriction can be (pessimistically) captured as follows:

Observation 1: In the case where a station does not transmit frames directly to itself (i.e. no direct loop-back), then the worst-case workload that can be transferred to the output queue of any switch port with every clock tick δ is $(N_p\text{-}1)\delta$.

Proof: Assume that during an interval of time having length δ each active port of the switch is busy processing incoming traffic. Consider any output port j. Since there is no loop-back, assuming the worst-case then at most $(N_p\text{-}1)$ ports can have incoming traffic destined to be transferred to port j. In the worst-case all input ports will be busy processing incoming traffic for port j simultaneously, hence the maximum workload transferred to queue j is $(N_p\text{-}1)\delta$.

Again observing that the switch output traffic can be represented by the summation of multiple queues (one *Geo*/*D*/1 queue and one *D*/*D*/1 queue for each channel with this destination port), the busy period will be finite if and only if the output port link utilization is strictly less than one [12]. The same effective allowable channel utilization limit of $U_M \approx 0.99$ results in tractable behavior for reliability levels in our range of interest. Taking these factors into consideration leads to the simple iterative scheme for delay estimation shown in Fig. 7, again requiring time $O(N_s L)$ and space $O(N_s)$. The operation of the algorithm is almost identical to that of Fig. 6, with the main exception that the rate at which the workload W is updated on line 6 is rate-limited to $(N_p\text{-}1)$ per iteration (since we assume $\delta = 1$). Note that this rate-limit does not affect the length of the busy period if $N_p > 1$, since the same total workload is eventually delivered out of the port, but has the effect of modulating (and potentially reducing) the peaks of the queue size.

```
01 INPUT(i, R, Γ, Π, δt, Ns, Np);
02 t:=0;
03 Q:=0;
04 IF $\left(\sum_{\substack{\tau_j\in\Gamma \\ D_j=i}} \frac{C_j}{T_j}\right) + \frac{\hat{C}_i}{\overline{T}_i^-} > U_{Max}$ RETURN(∞);
05 DO:
06    W:=W+MIN((Np-1),(W- $\sum_{\substack{\tau_j\in\Gamma \\ D_j=i}} w_j(t) + w_i^-(t)$));
07    Q:=MAX{Q,(W-t)};
08    t:=t+1;
09 WHILE((W-t)>0);
10 RETURN(Q);
```

Fig. 7. Algorithm to determine the maximum queuing delay at a switch output port.

This latter point related to the effect of N_p is illustrated by Fig. 8 below, in which the queuing delay for the same random traffic model $\{\overline{T}^- = 10, \hat{C} = 5\}$ with $R = 0.999$ employed to create Fig. 5 is displayed for values of N_p equal to 2, 3 and 6. For $N_p = 2$, the queue size is never greater than 1 as the rate of delivery into the buffer is the same as the rate of exit. For $N_p = 3$, the rate of delivery into the buffer is twice the rate of exit leading to a steady increase and a peak of $Q(29) = 30$. As can be seen, for $N_p = 6$ the worst-case queue delay of $Q(28) = 32$ is still achieved, and the overall evolution of the queuing delay begins to approach that shown in Fig. 5. For $N_p > 10$, the evolution becomes identical as the worst-case rate of delivery into the buffer is no longer affected by the limit.

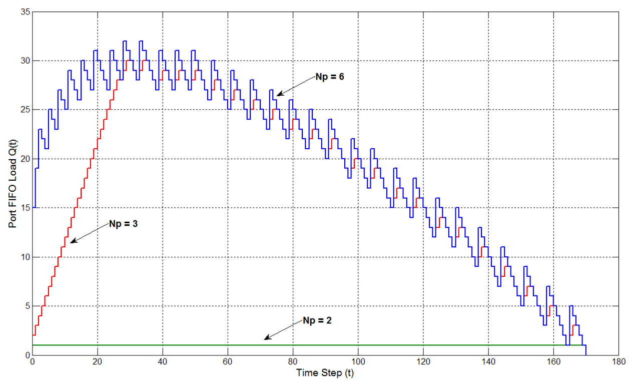


Fig. 8. FIFO delay in a switch output port with random traffic for various values of N_p.

Example 2: Suppose we have the following traffic characteristics in a switch with 4 ports ($N_p = 4$): three periodic/sporadic streams $\{T = 30, C = 2\}$, $\{T = 50, C = 5\}$ and $\{T = 100, C = 10\}$ combined with a random incoming stream $\{\overline{T}^- = 20, \hat{C} = 10\}$. Application of the algorithm described above (assuming $R = 0.999$ giving $C_1 = 3.623$ and $C_2 = 2.303$) yields a worst case delay estimate of 114, which occurs at $t = 312$ assuming the start of the busy period at $t = 0$. The end of the busy period in this case occurs at $t = 1468$.

Note again that these measures of delay not only represent the latency incurred by a frame when transiting through a switch, but they also imply a bound on the buffer size required at the switch output port [2]. If the random traffic characteristics have been correctly modeled, then the probability of either of these bounds being violated is again guaranteed to be $\leq (1\text{-}R)$ due to Theorem 1.

V. Areas For Further Work

There are many areas in which the preliminary work presented in this paper could be improved. Firstly, the algorithms of the previous Section would benefit greatly from being event-triggered as opposed to simply driven by an increasing time variable. The only difficulty in achieving this relates to obtaining accurate estimates of the next arrival of work for the workload function (5); however an expression to predict the next arrival time of an event should be relatively easy to obtain. It is also clear that some pessimism could potentially be removed from the estimation of the port output delay by maintaining a separate FIFO for the incoming workload originating in each source node in Fig 7. Also, to enable more realistic structures of network to be analyzed, it is required to be able to analyze switches in which part (or all) of the incoming traffic originates from other switches and with non-homogenous link transmission rates. One further improvement that is also required is the need to model situations in which the probability of a packet arrival at each time-step is time-varying (e.g. to cater for bursty arrivals of packets). For this latter point, we note that the work in [11] may be adapted to suit this purpose, but we leave this to future work. Finally, we acknowledge that queuing theory (see e.g. [12]) provides an alternate means to obtain probabilistic bounds on the transient sizes of the queues considered in this paper; we also plan to compare our current work against such techniques.

Acknowledgment

The IDEAS Collaborative Project (Grant Agreement No. 600071) is co-funded by the European Commission, Information Society and Media Directorate General, under the Seventh Framework Programme (FP7), Cooperation theme three, "Information and Communication Technologies". The authors wish to acknowledge the commission for their support, the efforts of the partners, and the contributions of all those involved in IDEAS. The authors also wish to express thanks to the anonymous reviewers whose comments helped to improve the paper.

References

[1] K.C. Lee, S. Lee & M.H. Lee, "Worst case communication delay of real-time industrial switched Ethernet with multiple levels," *IEEE Transactions on Industrial Electronics*, Vol. 53, No. 5, pp. 1669-1676, 2006.

[2] X. Fan, M. Jonsson, J. Jonsson, "Guaranteed real-time communication in packet switched networks with FCFS queuing," *Computer Networks*, Vol. 53, pp. 400–417, 2009.

[3] Q. Zhang & K.G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Transactions on Communications*, Vol. 42, Nos. 2/3/4, pp. 1096-1105, 1994.

[4] C. Harris. *Electricity Markets: Pricing, Structures, and Economics*. John Wiley & Sons, 2005.

[5] P. Vytelingum, S.D. Ramchurn, T.D. Voice, A. Rogers & N.R. Jennings, "Trading agents for the smart electricity grid," In: *The Ninth International Conference on Autonomous Agents and Multiagent Systems*, Toronto, Canada, pp. 897-904, May 2010.

[6] T. Mikkola, E. Bunn, P. Hurri, G. Jacucci, M. Lehtonen, M., Fitta, S. Biza, "Near real time energy monitoring for end users: Requirements and sample applications", In: *Proc. of the 2011 IEEE International Conference on Smart Grid Communications*, pp. 451 – 456, 17-20 Oct. 2011.

[7] International Electrotechnical Comission (IEC). *IEC 61850: Communication Networks and Systems in Substations*. International Electrotechnical Comission, Geneva, 2002.

[8] IEEE 802.1 AVB Task Group. *IEEE 802.1 Audio/Video Bridging*. [Online]. Available: http://www.ieee802.org/1/pages/avbridges.html.

[9] TTA-Group. *TTEthernet-Specification*. 2008. [Online]. Available: http://www.ttagroup.org/ttethernet/specification.htm.

[10] N.L. Johnson, A.W. Kemp and S. Kotz. *Univariate Discrete Distributions: Third Edition*. Wiley-Interscience, 2005.

[11] M. Short & J. Proenza, "Towards efficient probabilistic scheduling guarantees for real-time systems subject to random errors and random bursts of errors," In: Proc. 25th Euromico Conference On Real-Time Systems (ECRTS 2013), Paris, France, July 2013.

[12] L. Kleinrock. *Queueing Systems. Vol I: Theory*. Wiley Interscience, 1975.

On the Gap Between Mathematical Modeling and Measurement Analysis for Performance Evaluation of the 802.15.4 MAC Protocol

François Despaux
LORIA, University of Lorraine
Nancy, France
Email: francois.despaux@loria.fr

Ye-Qiong Song
LORIA, University of Lorraine
Nancy, France
Email: song@loria.fr

Abdelkader Lahmadi
LORIA, University of Lorraine
Nancy, France
Email: abdelkader.lahmadi@loria.fr

***Abstract*—IEEE 802.15.4 MAC protocol is the basis of many wireless sensor networks. Several studies have focused on analyzing the MAC layer by means of mathematical models such as Markov chain in order to be able to estimate the protocol performance parameters. Normally, simulation is used in order to validate the accuracy of the model. Unfortunately, real life is not always as easy as we would expect and extra considerations must be taken into account when considering real scenarios. In this paper our objective is to determine the existing gap between theoretical models and measurement analysis in real scenarios. We compare results obtained by an *a priori* accurate mathematical framework modeling the slotted IEEE 802.15.4 MAC protocol with experimental results obtained in Telosb motes and an implementation of the protocol over TinyOS. Comparision was made in terms of average delay. We also present some implementation considerations that needs to be taken into account when designing theoretical models for evaluating the delay in WSN.**

***Keywords*-TinyOS; Slotted CSMA/CA; Delay Evaluation**

I. Introduction

IEEE 802.15.4 protocol stack [1] is the basis of many wireless sensor networks (WSN) and has been proposed for low data rate and low power applications. Understanding the behavior and performance limitation of the protocol is challenging. In this way, a lot of research has been done to evaluate its performance through different methods, including theoretical modeling and simulation analysis. Bianchi's model [2] for the standard IEEE 802.11 under saturated traffic and ideal channel conditions have been extended for modeling the IEEE 802.15.4 MAC protocol. In [5], Park et al. proposes a Markov chain based on [2] for modeling the slotted version of the CSMA/CA mechanism and they give performance results in terms of service time delay and delay for successful packet sent. Misic et al. in [4] proposes a Markov chain approach for modeling the beacon enabled IEEE 802.15.4 MAC protocol considering M/G/1/K queue modeling and superframe with both active/only and active/inactive periods. Expressions for the access delay, probability distribution of the packet service time as well as probability distribution of the queue length are presented. Based on simulation results, both [5] and [4] frameworks seems to be suitable for estimating the delay in slotted CSMA/CA. However, none of the approaches consider the implication of implementing the protocol in real motes such as Telosb and how the underlying Operating System (OS), real hardware interaction and realistic conditions affects the delay estimation. In this paper we present an experimental evaluation of the Slotted CSMA/CA MAC protocol by considering realistic conditions using Telosb motes and an implementation of the protocol over TinyOS and we compare our results with the ones obtained in [4]. We present results in terms of average delay showing the gap between theoretical and empirical approaches and we give some implementation considerations that needs to be taken into account when designing theoretical models for evaluating the delay in WSN in order to have a more accurate model to work with. This paper is organized as follows. In Section II we present an overview of the IEEE 802.15.4 MAC protocol and the Markov chain model presented in [4]. A MAC layer implementation in TinyOS is presented in Section III. Experiments and Results are presented in Section IV and V respectively. A discussion regarding the results is given in Section VI and we conclude our work in Section VII.

II. IEEE 802.15.4 Standard & Markov Chain Model

A. Overview of IEEE 802.15.4 Standard

Misic The main idea of the MAC sub-layer in the IEEE 802.15.4 protocol is to provide an interface between the PHY layer and the higher layer protocols of LR-WPANs. Like the IEEE 802.11 protocol, the standard make use of CSMA/CA as the channel access protocol and it also brings support for contention-free and contention-based periods. Two operational modes are supported, *beacon enabled* and *Non beacon-enabled* modes. In the former beacons are periodically generated by the coordinator to synchronize attached devices. A beacon frame is part of a superframe which also embeds all data frames exchanged between the nodes and the PAN coordinator. In *Non beacon-enabled* devices can simply send their data by using unslotted CSMA/CA and there is no notion of superframe. In our work, we focus in the *beacon enabled* mode which use a slotted version of the CSMA/CA and exponential backoff. Both CSMA/CA mechanism are based on backoff periods where one backoff period is equal to *aUnitBackoffPeriod*$=20$ symbols (sym), $1\ sym = 4\ bits$ and the backoff period boundaries must be aligned with the superframe slot boundaries. Four variables are considered in the CSMA/CA mechanism:

NB: represents the number of times the CSMA/CA algorithm will enter in backoff while attempting the access to the current channel. It is initialized to zero before each new transmission attempt. If *NB* exceeds the limit of *macMaxCSMABackoffs*, the algorithm terminates with channel access failure status and failure is reported to higher protocol layers which can then decide whether to abort the packet in question or re-attempt to transmit it as a new packet.

RT: if a node fails to receive ACK due to collision or acknowledgement timeout the variable is increased by one up to *macMaxFrameRetries* and the packet is retransmitted. If RT is greater than *macMaxFrameRetries* packet is discarded due to the retry limit. This value is initialized to zero.

CW: representing the number of times the CSMA/CA will check the channel availability before starting transmission. The value by default is 2 and is reset to 2 each time the channel is busy and finally,

BE: represents the backoff exponent. Each time the channel is found busy BE is incremented by 1 until it reach the maximum possible value *aMaxBE* which is a constant defined. The slotted CSMA/CA algorithm proceeds as follows.

- *1 - Initialization*: In this step, NB, CW, RT and BE are initialized. BE value is determined based on the *macBattLifExt* parameter. If its value is equal to *true* then BE is initialized as $BE = min(2, macMinBE)$, otherwise, it is initialized as $BE = macMinBE$ where *macMinBE* specifies the minimum of backoff exponent which is set to 3 by default. After the initialization, the algorithm locates the boundary of the next backoff period and goes to step 2.
- *2 - Random waiting delay for collision avoidance*: In this point the algorithm waits a random backoff in order to avoid collisions. The random backoff period is taken from the range of $[0, 2^{BE}-1]$. After the backoff period, it goes to step 3.
- *3 - Clear Channel Assessment*: In this step the algorithm check the availability of the channel. If the channel is found idle the algorithm goes the *idle channel* step, otherwise it moves to the *Busy channel* step. The CCA must be started at the boundary of a backoff period after the expiration of the waiting delay timer.
- *4 - Busy channel*: In this step, the CCA found the channel busy. Then, BE and NB parameters are incremented by 1 and CW is reseted to the initial value. The BE parameter must not exceed the value of *aMaxBE* parameter whose value by default is 5. If NB exceeds the parameter *macMaxCSMABackoffs* then the algorithms finish by throwing a channel access failure status. Otherwise, if NB is lower or equal to *macMaxCSMABackoffs* algorithm jumps to step 2 (*Random waiting delay for collision avoidance* step).
- *5 - Idle channel*: In this step, the channel was found idle. Then, CW parameter is decremented by 1. In case CW reaches zero then the MAC Protocol may start successfully its transmission. Otherwise, the algorithm returns to step 3. The transmission is started if and only if the remaining number of backoff periods in the current superframe is sufficient to handle both the frame and the subsequent acknowledgement transmissions. Otherwise, transmission is deferred until the next superframe. In case of a collision or acknowledgement timeout the algorithm goes to step 6. If packet is transmitted but a collision occurs, the algorithm continues in step 6.
- *6 - Collision Ocurred*: In this step RT is incremented by one. If RT is lower than *macMaxFrameRetries* then variables CW, NB and BE are initialized as in step 1 and algorithm jumps to step 2 (*Random waiting delay for collision avoidance* step). Otherwise, the packet is discarded due to the retry limit.

B. Markov Chain Approach

Authors in [4] present a Markov chain approach for modeling the behavior of the slotted IEEE 802.15.4 MAC protocol. In this approach, the packet queue in the device buffer is modeled as a *M/G/1/K* queueing system. Based on the protocol specification previously presented, authors model the system as a stochastic process
$\mathcal{P}$=$\{$*n(t), c(t), b(t), d(t)*$\}$ where $n(t)$ represents the value of the backoff time counter at time t, $c(t)$ represents the value of NB at time t, $b(t)$ represents the value of CW at time t and $d(t)$ represents the current value of the delay line counter (started if packet is deferred) at time t. Then, a discret-time Markov chain depicting this process is defined. Based on this Markov chain model, authors were able to find the probability distribution of the packet service time, probability distribution of the queue length and both probability distribution and average delay from the moment a packet arrives to the queue until the moment the node receives the corresponding packet acknowledgement. Our purpose is to compare this theoretical average delay with the empirical one. For details in the implementation refer to chapter 3 in [4].

III. MAC LAYER IMPLEMENTATION IN TINYOS

The objective of this section is to present an overview of the TKN154 MAC implementation over TinyOS in order to understand the main components of this module and also to determine how delay is affected by the interaction of each component.

A. TKN154 Module

TKN154 [3] is a platform independent IEEE 802.15.4-2006 MAC implementation for the 2.1 release of the TinyOS execution environment meeting the main tasks of the 802.15.4 MAC protocol such as PAN association and dis-association, slotted and unslotted versions of protocol, beacon transmission and synchronization, among others. Main components and interfaces used to exchange MAC frames between components are shown in Figure 1 and defined in [3]. TKN154 MAC can be divided into three sublayers. The lowest level, the RadioControlP component, manages the access to the radio. Components on the second level represent different parts of a superframe. For instance, BeaconTransmitP/BeaconSynchronizeP responsibles for transmission/reception of a beacon frame, DispatchSlottedCsmaP component manages frame transmission and reception during the CAP. The components on the top level implement the remaining MAC data and management services such as PAN association or requesting data from a coordinator. A component of this level typically provides MAC primitives to the next higher layer. For instance, DataP provides *MCPS-DATA.request* primitive to the next higher layer to send a frame to a peer device. Data frame will be assembled and enqueued in the send queue DispatchQueueP. DispatchSlottedCsmaP will eventually dequeue the frame and manage its transmission. Transmission status will be propagated to higher layer by means of *MCPS-DATA.confirm* event where an appropriate status code will signal whether the transmission was successful or not.
A set of interfaces towards the Radio Driver are also implemented. These interfaces push many time-critical operation from the MAC to the radio driver which are not negligible affecting the packet transmission delay.

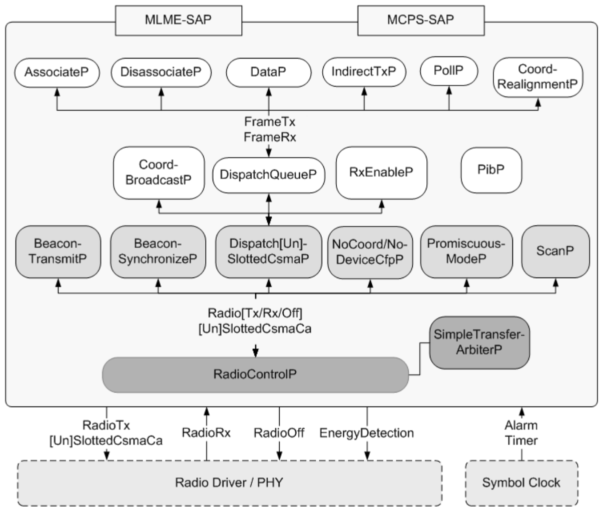


Figure 1: The TKN15.4 architecture: components are represented by rounded boxes, interfaces by connection lines.

B. Delay Analysis within TKN154 Implementation

As we mentioned before and based on simulation results, [4] framework approach seems to be accurate for estimating the delay within the slotted IEEE 802.15.4 MAC protocol. However, implementation considerations were not taking into account and as we will see next, these cannot be omitted if we want to have a suitable mathematical framework to estimates the delay within the protocol. In this subsection we analyse the TKN154 components interaction showing how delay is affected due to implementation issues. In TinyOS there are two threads of execution: tasks and hardware event handlers. Tasks are functions whose execution is deferred. Once scheduled, they run to completion and do not preempt one another. Hardware event handlers are executed in response to a hardware interrupt and also runs to completion, but may preempt the execution of a task or other hardware event handler. In this way, the completion of a particular task or event handler may be delayed due to a hardware interrupt affecting the delay of the whole protocol. Some other issues associated to the Operating System behavior may have effects on the protocol performance. For example, the TinyOS interface to the SPI bus introduces a large processing overhead that reduces the achievable SPI bus transfer rates [6].

1) Extra-delays estimation: In order to estimate extra-delays in the implementation of the MAC protocol over TinyOS we have analysed the execution stack from the moment a packet is generated until the moment the packet is acknowledged (or eventually lost if the number of retries exceeds the predefined threshold). Figure 2 shows a sequence diagram with the most relevant operations within the protocol. The first non-negligible delay presented in the execution is related to the SPI resource request in step 6. This delay together with the CC2420 switch to Rxmode shown in step 7 gives an extra delay of $t_1 = 5ms$. Then a random backoff period and two clear channel assessment (CCA) follows the execution of the protocol. However, both CCA and the random backoff period were taken into account in [4]. Next, a fix delay (step 11) of $t_2 = 2.8ms$ between the end of the second CCA and the invocation of the *transmissionStarted* function was found. Transmission is then delayed $t_3 = 2.8ms$ due to SFD Capture operation (step 13). Finally the packet is sent with a transmission delay shown in step 14. We have also found an extra-delay (step 15) $t_4 = 3ms$ in the coordinator side before sending the acknowledged packet back to the source (step 16). This analysis gives us un idea of the extra-delay due to the protocol implementation that should be taking into account when estimating the MAC protocol delay in TinyOS. Finally, we have also found a random delay t_5 due to deferred packets. Normally, when the remaining time within the CAP period of the current superframe does not suffice to complete the transmission the packet is deferred and will be transmitted at the begining of the next superframe. However, in TinyOS we have seen that in some cases the deferred packet transmission does not start at the begining of the next superframe. Instead, it skips the immediate superframe deferring the transmission to the next one. That means that in this case, packet transmission will be delayed $t_{wait} = t_{curr} + t_{sup}$ where t_{curr} is the remaining time within the CAP period of the current superframe and t_{sup} is the duration of the superframe (in our case a 100% duty cycle is consider so optional inactive period is not taking into account). Considering that in our experimental scenarios, MAC attributes SO and BO were set to five then $t_{sup} = 30720$ symbols meaning that delay in those packets skiping the immediate superframe would be at least 30720 symbols = 0.5 seconds (1 symbol = 16μs).

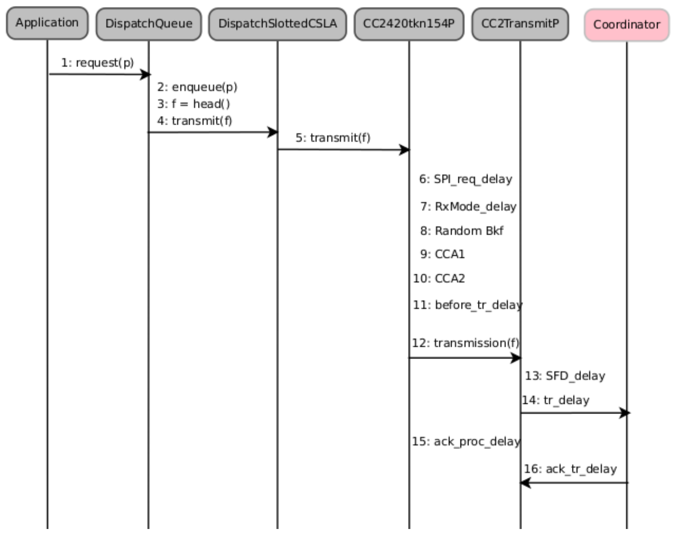


Figure 2: TKN154 Delay Analysis.

IV. Experiments

In this section we present the experiments we have done in order to compare the theoretical results obtained by [4] with real scenarios using an implementation of the slotted IEEE 802.15.4 MAC protocol over TinyOS and Telosb motes. We first start by specifying the main parameters and then we present a set of scenarios we have used in the experimentation.

A. *Parameters & Scenarios*

In order to compare both theoretical and experimental results we have set the same MAC protocol parameters which are shown in Table I. As we can see both BO and SO are set to 5 so no inactive period is considered and then Duty Cycle is 100%. We consider six scenarios which varies in the number of nodes and arrival rate of packet to nodes. The idea is to make a comparition of both approaches in terms of the average delay, that is to say, the average delay

Parameter	Value
Max Frame Retries	3
Max CSMA Backoff	4
Max Backoff Exponent	5
Min Backoff Exponent	3
Battery Life Extension	False
Beacon Order	5
Superframe Order	5

Table I: MAC parameters.

from the moment a generated packet is put in the queue until the reception of the acknowledged for that packet. We consider a star topology where coordinator is situated at the center and devices are located around the coordinator. Distance between devices and coordinator is the same for all scenarios and was set to one meter. The transmission power for each node was set to 0dBm. We know from [7] that, for this transmission power and considering a distance of one meter, the packet reception rate is almost 1 since the transitional region (a region characterized by unreliable and asymetric links with high variance in reception rate) starts at a distance of almost 10 meters. Simulation parameters are summarized in Table II. Packet payload is fixed as 34 bytes for all scenarios. As in [4], packet arrivals to each device follow a Poisson process with mean arrival rate of λ and each node accepts new packets through a buffer with finite size of $L = 2$ packets. All scenarios use the same buffer size. Channel bitrate is 250kbps.

Scenario	Nodes	Traffic Load λ (packets/s)
Scenario 1	2	1
Scenario 2	4	1
Scenario 3	6	1
Scenario 4	2	10
Scenario 5	4	10
Scenario 6	6	10

Table II: Scenario Parameters

V. Results

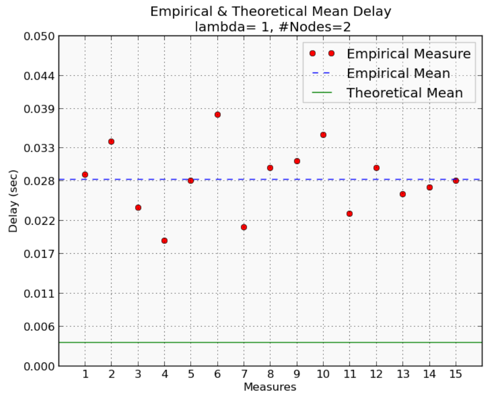
We present now the experimentation results. Our objective is to compare the theoretical and empirical average delay. Misic framework was implemented in Matlab while we use TKN154 implementation of the slotted IEEE 802.15.4 over TinyOS and telosb motes. For each scenario, a total of fifteen measurements were done in telosb motes and then the average delay was found and compared with the average delay obtained by the [4] framework. Table III shows theoretical and empirical average delay for each scenario. Figures 3 and 4 summarize the results of each scenario. Points in graphics represent the empirical measures (a total of fifteen instances of measurement for each scenario). Each empirical measure (point in graphics) is the result of the analysis of all the packets that were generated during the current instance of measurement. We took the average of these delays in order to determine the empirical measure of delay for the current instance and we plot the corresponding point. This procedure is repeated fifteen times for each scenario. Blue-dashed line represents the empirical average delay (average of fifteen points) for each scenario, while the green line shows the average delay obtained by [4] mathematical framework.

Scenario	Theoretical Av. Delay (sec)	Empirical Av. Delay (sec)
Scenario 1	0.00356	0.028
Scenario 2	0.0036	0.030
Scenario 3	0.0036	0.031
Scenario 4	0.0038	0.040
Scenario 5	0.0043	0.042
Scenario 6	0.005	0.044

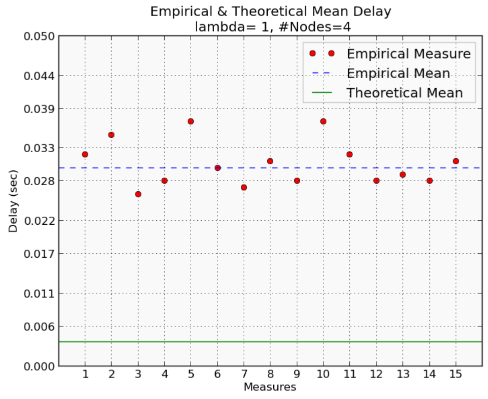
Table III: Theoretical VS Empirical Delay.

VI. Discussion

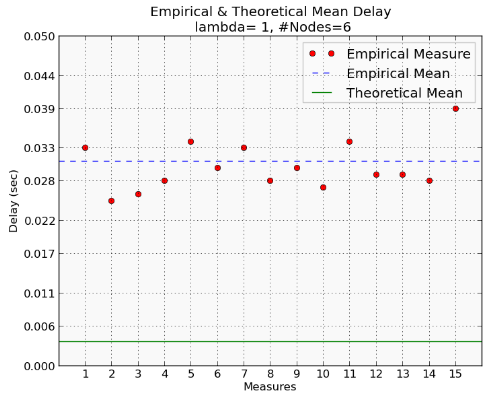
As we can see from Figure 3, 4 and Table III it is evident that there is an important gap between empirical and theoretical results. We see from results that when the number of nodes increases the delay also increases. This is normal since as the number of nodes grows, the number of times the channel is found busy also increments. This behavior is expected as well for the number of collisions. We also see that delay when considering a mean arrival rate of $\lambda = 10$ is greater than the one found for scenarios having $\lambda = 1$. This is due to the fact that for scenarios having mean arrival rate of $\lambda = 10$, the queue is expected to be busy during a non negligible time. That means that delay for those packets in the queue will increase since it would have to wait until the acknowledgement from the previous sent packet arrives. An important issue we found is regarding the first scenario. In this case we have two nodes and mean arrival rate $\lambda = 1$ so we do not expect to have too much collisions, channel is expected to be idle most of the time and in theory, the queue should be in idle state most of the time. However, the gap between theoretical and empirical delays is almost 24ms. How can we explain this behavior ?. By analysing the TinyOS traces during the experiment we found that a number of deferred packets skips the immediate superframe delaying the transmission to the next one. For these cases, the node's queue would not be necessary idle most of the time as in theory we would expect for this low-traffic scenario. This happens because deferred packets would have an extra-delay and then packets arriving would find the node busy and they will have to wait until the deferred packet is acknowledged. Then, contrarily to the expected behavior, the queue won't be idle most of the time and this fact will impact in the average packet delay. [4] Markov chain model is a very good approach for modeling the IEEE 802.15.4 MAC protocol considering and covering the main aspects of the protocol behavior (packet collisions, packet deferring, random backoffs, etc). Simulation results shown in [4] validate the mathematical approach. However, as we discussed in section



(a) Scenario 1.

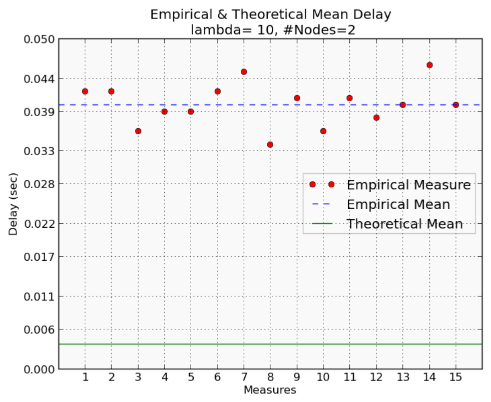


(b) Scenario 2.

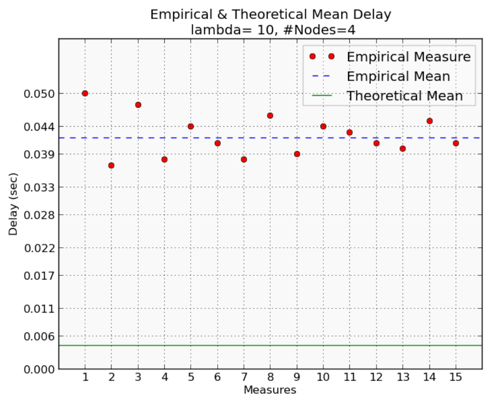


(c) Scenario 3.

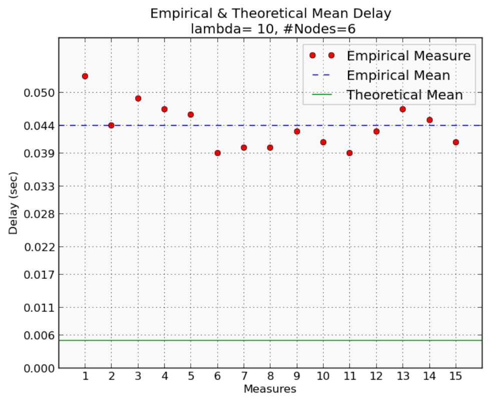
Figure 3: Traffic Load $\lambda = 1$ packets per seconds.



(a) Scenario 4.



(b) Scenario 5.



(c) Scenario 6.

Figure 4: Traffic Load $\lambda = 10$ packets per second.

II.B, non negligible delays arise when consider a realistic scenario with real motes due to operations of the underlying operating system. From the practical point of view, omitting the operating system features and behavior would leads to a useless model for analysing the protocol performance, in this case, the delay. A question that emerges immediately: is it possible to reduce the gap between theoretical and empirical results?. The first thing that we can do in order to reduce the gap is to focus our attention in the protocol implementation. In our analysis in section II.B, we detected an anomaly in the TKN154 implementation concerning the deferred packets. As we saw, some deferred packets skip the incoming superframe delaying the transmission until the reception of the next one (which amounts 0.5 seconds for the set of parameters defined here). So an important point to do is to detect anomalies in the implementation that could introduce extra-delays in the protocol and fix them. However, as shown in Figure 2, even considering an implementation exempt from anomalies and errors, delays are present due to function calls, hardware operations such as resources request, radio switch to reception/transmission mode, etc. Considering a free-error implementation, one approach then to reduce the gap is to quantify the existing delays in the implementation as done in section II.B and add them to the theoretical model in such a way that extra-delays are also considered when computing the average delay, for instance by changing (3.36) formule in [4].

VII. Conclusion

In this paper we have presented an analysis of the average delay in slotted IEEE 802.15.4 protocol in real scenarios considering the TKN154 implementation over TinyOS. Our objective was to determine the gap between a Markov chain approach for estimating the average delay presented in [4] and a real implementation in TinyOS. By analysing the execution of the protocol in real nodes we were able to determine constant and random delays inherent to the operating system operation such as hardware event handlers threw due to hardware interrupts whose execution can preempt a particular task and thus delaying its completion. Also we have noticed that some deferred packets skips the immediate superframe delaying the transmission to the next one. Even though we consider that [4] model is a good approach for estimating the average delay we conclude that it is incomplete since it does not consider the main aspects of real scenarios with an underlying operating system and real motes. One way to reduce this gap is to detect anomalies in the implementation side as the one we found in section II.B and fix them to have a free-error protocol implementation. Then, by considering this free-error implementation the next step should be to quantify the delays due to function calls, hardware operations, etc. and add them to the theoretical framework in order to have a realistic way for estimating packets delay within the protocol.

VIII. Acknowledgements

This work was partially supported by the ANR Quasimodo project under n° ANR 2010 INTB 0206 01.

References

[1] IEEE Standard for Information Technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - specific requirement Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Technical report, 2007.

[2] G. Bianchi. Performance analysis of the ieee 802.11 distributed coordination function. *IEEE J.Sel. A. Commun.*, 18(3):535–547, September 2006.

[3] Jan-Hinrich Hauer. Tkn15.4: An ieee 802.15.4 mac implementation for tinyos 2. TKN Technical Report Series TKN-08-003, Telecommunication Networks Group, Technical University Berlin, March 2009.

[4] Jelena Misic and Vojislav Misic. *Wireless Personal Area Networks: Performance, Interconnection, and Security with IEEE 802.15.4*. Wiley Publishing, 2008.

[5] P. Park, P. Di Marco, C. Fischione, and K. H. Johansson. Delay analysis of slotted IEEE 802.15. 4 with a finite retry limit and unsaturated traffic. In *IEEE Global Communications Conference*, page 18, 2009.

[6] Petcharat Suriyachai, Utz Roedig, and Andrew Scott. Implementation of a MAC protocol for QoS support in wireless sensor networks. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, page 16, 2009.

[7] M. Zuniga and B. Krishnamachari. Analyzing the transitional region in low power wireless links. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, page 517 526, 2004.

Session 3

Calculation of Worst Case Backlog for AFDX Buffers with Two Priority Levels using Trajectory Approach

Naga Rajesh Garikiparthi, Rodrigo Coelho, Gerhard Fohler
Technische Universität Kaiserslautern, Germany
{rajesh,coelho,fohler}@eit.uni-kl.de

***Abstract*—AFDX (Avionics Full Duplex Switched Ethernet) is a network standard chosen to replace point to point connections in avionics systems. AFDX guarantees bandwidth reservation by means of virtual links, which can, according to the standard (ARINC 664 Part-7) be classified with two priority levels: high and low. AFDX switches must have buffers at their output ports and in order to not incur in data loss, buffer overflow must be avoided. AFDX standard determines the minimum amount of buffer dedicated to an output port, however it is up to the designer to select the actual buffer size and how to allocate it for the two priority levels of virtual links.**

Previous works make use of trajectory approach (TA) to analyze AFDX networks and compute the worst case backlog of output buffers for single priority flows. A method to compute the worst case backlog for two priorities buffers is the issue addressed in this paper.
We make use of the TA to determine the frames competing for the output port. Further, we analyze the worst case scheduling scenario of frames competing for output ports and show how the arrival of frames in the input links impacts the computation of the backlog for two priority level buffers.

I. Introduction

AFDX is the network chosen to interconnect the nodes in avionics systems [1]. AFDX offers a high network bandwidth (typically 100Mbps) and allows for bandwidth isolation among the network traffic by defining the concept of virtual links (VL). Virtual links additionally define the logical path from one source end-system (ES) to one or more destination ES. The physical route of each VL is statically defined at design time and therefore the switches traversed by each VL is known before run-time.
AFDX networks make use of store and forward switches. In order to cope with contention for the switches' output ports, each output port offers FIFO buffers.

Besides bandwidth isolation, AFDX further allows for the classification of VLs into two priority levels: high and low. Switches have to send all high priority frames before the low priority ones. Considering the non-preemptive property of frame scheduling, the switch cannot abort the transmission of a low priority frame in favor of a high priority one.

The AFDX standard [2] (ARINC 664 Part-7) specifies the minimum number of frames that must be buffered on the switches output ports. The actual output port buffer size for each priority level is, however, left as a design decision and is used in the configuration phase of the network ([2] section 4.7.3.2). Thus, in order to avoid buffer overflow in the output ports and consequently packet loss, the designer must compute the upper bound backlog for both high and low priority buffers.

In our work we present a method to compute the worst case backlog for each priority output port buffer. We make use of the TA [3] to compute the number of frames competing for the same output port. We then analyze the frame arrival scenarios (frame scheduling) and identify those that lead to the worst case backlog for each priority level. We present distinct scenarios in increasing order of complexity to facilitate the analysis. Finally, we show how to compute an upper bound for the backlog for the general case.
Previous works address the computation of the maximum backlog for AFDX switches using TA considering single priority VLs. In this paper we consider two priorities VLs and present the computation of maximum backlog for the two priority buffers.

This paper proceeds as follows: we present the related work in section II; section III contains a short summary of basic TA concepts required in this paper; in sections IV and V we show how to compute the worst case backlog for the low and high priority buffers, respectively; and in section VI we present our conclusion and future work.

II. Related Work

A number of works address the analysis of AFDX networks in distinct aspects. Early studies on the computation of an upper bound for the end-to-end (e2e) delay of packets transmitted on AFDX networks applied network calculus (NC). Despite the simple graph analysis used in NC, both the combination of flows into arrival curves and computation of the combined service curves is not trivial [4] and leads to pessimistic results [5]. [4] makes use of NC to compute the upper bounds for e2e delay in AFDX networks for non-preemptive priority flows. [6] and [7] respectively present how to compute the probabilistic bounds on e2e delays and backlogs, based on stochastic NC.

A series of more recent works apply the trajectory approach for the computation of AFDX e2e delays and backlogs. In contrast to NC, which considers each VL as a flow, the TA analyzes the VL traffic at a finer granularity, accounting for the individual frames of the VLs. Therefore, TA leads to tighter bounds than NC for most investigated cases [5].
In [8] and [9] the authors make use of TA to compute the *e2e*

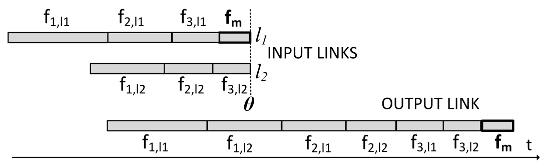


Fig. 1: Competing frames sharing two input links

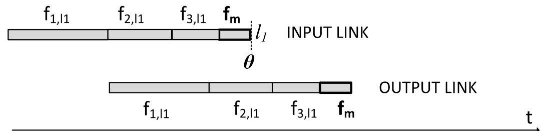


Fig. 2: Considering competing frames only from l_1

delay for FIFO output buffers with *single priority* and *distinct static priority flows* respectively. [5] computes the worst case *backlog* for FIFO output buffers with *single priority flows*. [5] further extends the previously mentioned works and presents an analysis of the pessimism intrinsic to TA.
Our paper tackles the issue of analyzing the worst case *backlog* for AFDX networks with *two priority flows*, based on TA.

III. Worst Case Backlog Computation for Single Priority flows using Trajectory Approach

This section describes the main idea behind the computation of worst case backlog using TA as presented in [5]. The concepts presented in this section will be used in section IV.

For flows with single priority, an AFDX switch provides a single FIFO buffer. In order to determine the upper bound for the backlog in the buffer, we need to compute the maximum backlog encountered by a frame sharing the buffer. Therefore, we need to establish the worst case arrival sequence of frames competing for an output port.

Figure 1 presents the arrival of frames from two input links of an AFDX switch. The leftmost frame is the first arriving one and the rightmost frames are those that arrive at last.
The scenario depicted in Figure 1 represents the worst case scenario of frames competing with f_m arriving at the switch from two input links l_1 and l_2, without any idle times, in decreasing order of sizes. The first frames are received at different times, while the last frames are received at the same time θ. According to TA, f_m has to be the last frame to be received in the sequence. To proceed with the computation of the worst case backlog faced by f_m, the earliest starting sequence among the two links is determined. The sequence whose first frame is received earlier, is the earliest starting sequence. This is achieved by comparing lengths of both sequences without their first frames.

In our example of Figure 1, the earliest starting sequence is the sequence on link l_1. Suppose, it is the only sequence present as seen in Figure 2, from Lemma 1 of [5], the backlog at the reception of f_m is the size of the largest frame on l_1, $s(f_{1,l_1})$. The maximum backlog is less if idle times are present between frame arrivals on l_1. Therefore, to produce the worst case, idle times are avoided on the input links.

When frames arrive on a single input link in decreasing order of sizes without idle times, the quantity of data entering the switch is equal to the quantity of data leaving the switch. Hence, at the reception of each frame, starting from f_{2,l_1} on l_1, a residual backlog belonging to the previously received frame is always present in the buffer. Consequently, no idle times are created on the output link as seen in Figure 2.

Now consider the sequence on l_2. The arrival of frames from l_2 increases the quantity of incoming data between the first and last frame of l_1. However, an increase in the quantity of data being forwarded by the switch is inhibited by the absence of idle times on the output link. Therefore, the backlog *bl* at the reception of f_m is increased by the frames on l_2 i.e. f_{1,l_2}, f_{2,l_2} and f_{3,l_2} and is given by the the following equation:

$$bl = s(f_{1,l_1}) + \sum_{i=1}^{3} s(f_{i,l_2}) \tag{1}$$

If n_{l_j} represents the number of frames on link l_j, above equation can be written as:

$$bl = \sum_{j=1}^{2} \sum_{i=1}^{n_{l_j}} s(f_{i,l_j}) - \max_{j=1,2} \sum_{i=2}^{n_{l_j}} s(f_{i,l_j}). \tag{2}$$

Hence, in the worst case, frames arrive continuously in decreasing order of sizes. The above scenario is presented in detail in Lemma 2 and Lemma 3 of [5].

The above example can be generalized from two to arbitrarily many input links. The worst case backlog faced by f_m when competing frames share N input links is the sum of the largest frame from the earliest starting sequence and all frames from the other sequences. Accordingly, the backlog at time θ is:

$$bl = \sum_{j=1}^{N} \sum_{i=1}^{n_{l_j}} s(f_{i,l_j}) - \max_{1 \le j \le N} \sum_{i=2}^{n_{l_j}} s(f_{i,l_j}). \tag{3}$$

IV. Worst Case Backlog Computation in Low Priority Buffer using Trajectory Approach

As we consider static priority queuing with two priority levels, two buffers with high and low priority are present at the output port of an AFDX switch. We determine guaranteed backlog bounds in these two buffers using the TA. Accordingly, the following points have to be addressed to compute the backlog encountered by a frame f_m:

- Identify flows that compete with f_m for an output port of a switch.
- Compute the number of frames of each competing flow which lead to the worst case backlog encountered by f_m at the output port.
- Determine the scheduling of these competing frames which lead to the worst case backlog for f_m at the output port.

A. Notations

- p – priority of a flow and a buffer, where $p \in \{H, L\}$ and *H*, *L* represent high and low priority respectively.
- ω^H – total number of competing high priority flows.
- $Flows_H$ – set of all high priority flows that compete for an output port of the switch.

- v – index of a high priority flow and frame, $v \in \{1..\omega^H\}$.
- τ_v^H – a competing high priority flow of index v, $Flows_H = \{\tau_v^H \forall v \in \{1..\omega^H\}\}$.
- ω^L – total number of competing low priority flows.
- $Flows_L$ – set of all low priority flows that compete for an output port of the switch.
- u – index of a low priority flow and frame, $u \in \{1..\omega^L\}$.
- τ_u^L – a competing low priority flow of index u, $Flows_L = \{\tau_u^L \forall u \in \{1..\omega^L\}\}$.
- f_k^p – frame of flow $\tau_k^p \in \{Flows_H, Flows_L\}$, where $k \in \{v, u\}$.
- f_{k,l_j}^p – frame of flow $\tau_k^p \in \{Flows_H, Flows_L\}$ on link l_j, where $1 \leq j \leq N$, N is the total number of input links.
- $n_{l_j}^p$ – number of frames of priority p on link l_j.
- $s(f_k^p)$ – size of frame f_k^p.
- f_m – frame under study, $\{f_m \in \tau_i^p \mid i \in \{v, u\}\}$.
- τ_{k,l_j}^p – flow τ_i^p on link l_j.
- $\eta_{i,v,t}^h$ – number of frames of a high priority flow τ_v^H at switch h that impact $f_m \in \tau_i^p$ generated at time t at its source end system.
- $\kappa_{i,u,t}^h$ – number of frames of a low priority flow τ_u^L at switch h that impact $f_m \in \tau_i^p$ generated at time t at its source end system.
- $Buffer^H$ – buffer for high priority frames.
- $Buffer^L$ – buffer for low priority frames.
- bl_{max}^L – worst case backlog faced by a low priority frame.
- bl_{max}^H – worst case backlog faced by a high priority frame.

B. Identification of the Competing Flows

Our method considers two priorities, therefore a flow belongs to either high (H) or low (L) priority. To compute the worst case backlog in $Buffer^L$, we consider a frame $f_m \in \tau_i^L$. Then, we determine $Flows_H$ and $Flows_L$, which compete with f_m at the desired output port, with the help of the network description which defines the routing of flows.

C. Computation of Number of Frames of Each Competing Flow

We use TA to calculate the upper bound for the number of frames of each flow in $Flows_H$ and $Flows_L$ competing with f_m in its busy period (busy period as defined in [10]).

According to TA, if any frame with low priority arrives after f_m, it cannot contribute to the backlog of f_m [5]. However, in contrast, if a frame with high priority arrives after f_m, it may contribute to the backlog of f_m. The number of high and low priority frames determines the length of the busy period and therefore contributes to the worst case backlog faced by f_m.

Consider that at time t, f_m is generated at its source node. The number of frames of a competing low priority flow, $\kappa_{i,u,t}^h$, and the number of frames of a competing high priority flow, $\eta_{i,v,t}^h$, that impact the backlog encountered by f_m can be derived from equations (5) and (6) presented in [9].

D. Worst Case Scheduling of Competing Frames

In order to compute the maximum backlog faced by f_m, we need to establish a worst case arrival sequence of the competing frames in the busy period of f_m. According to TA, f_m has to be the last frame to be received in that sequence. We consider the following three cases to determine the worst case arrival sequence and compute the worst case backlog encountered by f_m:

- Section IV-D1 explains the simple case of frames competing with f_m arriving from the same input link as f_m.
- Section IV-D2 explains the case of frames competing with f_m arriving from two input links.
- Section IV-D3 explains the general case of frames competing with f_m arriving from arbitrary number of input links.

For the sake of simplicity and without loss of generality, we assume that one unit of time is the time required for the transmission and reception of one unit of data.

1) Case 1: Competing Flows Sharing Single Input Link: The top of Figure 3 shows the worst case arrival sequence of frames competing with f_m sharing the same input link as f_m. The low priority frames in the sequence are preceded by the high priority frames. The following Lemma 1 computes maximum backlog faced by f_m in $Buffer^L$.

Lemma 1: *Given the frames $f_1^p, f_2^p, ...f_k^p$ competing with f_m for an output port of a switch, grouped in high priority followed by low priority $f_1^H, f_2^H, ...f_{\omega'}^H, f_1^L, f_2^L ...f_\omega^L, f_m$ with no idle times on the input link, the maximum backlog faced by f_m is bl_{max}^L. Introducing idle times in the input link never leads to a backlog larger than bl_{max}^L. Consequently, the worst case backlog faced by f_m is given by:*

$$bl_{max}^L = \begin{cases} \max\limits_{\substack{1 \leq v \leq \omega^H \\ 1 \leq u \leq \omega^L}} \left(s(f_v^H), s(f_u^L)\right) & \text{if } bl_{max}^H < \sum\limits_{u=1}^{\omega} s(f_u^L) \\ \sum\limits_{u=1}^{\omega} s(f_u^L) & \text{otherwise} \end{cases} \tag{4}$$

The following properties are considered to prove the lemma.

1) The backlog encountered by f_m does not increase when an idle time is present on the input link.
2) A grouped order of arrival of frames with high priority followed by low priority before f_m, leads to worst case backlog at the reception of f_m.

We define *grouped arrival order of frames* the sequence of frames in which all frames arrive back-to-back in two groups, and each group contains only frames of same priority.

Property 1: *Given a sequence of frames $f_1^p, f_2^p, ...f_k^p$ arriving in any order on a single input link and sharing an output port of a switch, the maximum backlog faced by f_m, is obtained when there is no idle time during the reception of the input sequence.*

Proof: As proved in Property 1 of Lemma 1 in [5], if z units of idle time is present on the input link and the output port is accessible for any buffer, a maximum of z units of data can be forwarded from the output port within this idle time. Therefore, the backlog does not increase in either buffer as

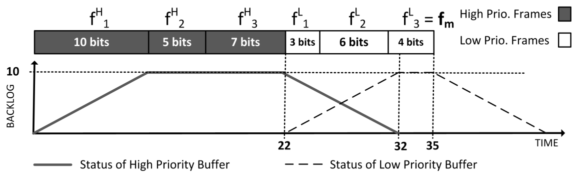


Fig. 3: Grouped arrival order of frames

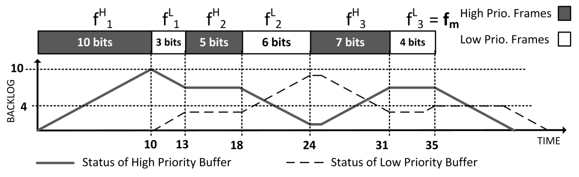


Fig. 4: Random arrival order of frames

there are no incoming frames during the idle times. Hence, we conclude that the presence of idle times on the input link never increases the worst case backlog for f_m. ■

PROPERTY 2: *Given a sequence of frames $f_1^p, f_2^p, ...f_k^p$ arriving in grouped order with all high priority frames followed by low priority frames, on a single input link without idle times, the backlog encountered at the reception of the last frame f_m is always maximum.*

Proof: Considering that both input and output links have the same bandwidth, as in AFDX networks, the backlog in $Buffer^L$ only increases during the time when the following condition holds: $1)$ low priority frames arrive from the input link *and* $2)$ no low priority frame leaves from the output link. Therefore, to exploit the worst case scenario for the computation of bl_{max}^L, we analyze the moments in time when low priority frames arrive from the input link.

[5] proved that idle times in the input links do not lead to worst case scenarios, therefore for $1)$, we analyze the cases when all low priority frames arrive with no idle time.

$2)$ occurs if no complete frame is available in $Buffer^L$. An incomplete frame in $Buffer^L$, i.e. a low priority frame arriving from the input link, will increase the backlog in $Buffer^L$ by at most the size of the largest low priority frame. [5] proved that $\max\limits_{1 \le u \le \omega^L}(s(f_u^L))$ is the maximum backlog for single priority flows in a single input link, independent of the arrival order of frames. Thus the worst case scenario does not restrict the sequence of arrival within the group of low priority frames. $2)$ may also occur if a high priority frame is being transmitted. Then, the worst case scenario occurs when $Buffer^H$ has the largest backlog, i.e. bl_{max}^H. In this case, the backlog in $Buffer^L$ will increase as long as low priority frames arrive from the input link and bl_{max}^H units of data is being transmitted. The worst case scenario that leads to this condition occurs when all low priority frames arrive after all high priority frames.

The next examples depict two scenarios to highlight the worst case presented in the previous paragraph. Consider an example of a sequence of 6 frames with different sizes arriving without idle times as depicted in Figures 3 and 4. The graphs represent the backlogs in $Buffer^H$ and $Buffer^L$ during the storing and forwarding of the given sequence according to their priorities. Our aim is to compute the worst case backlog faced by f_m - f_3^L in this example. Consider Figure 4, where f_m is preceded by a non-grouped arrival of frames on the input link. At *t=10*, f_1^H is completely stored and f_1^L starts to buffer until *t=13*. From *t=10* to *t=13*, the backlog in $Buffer^H$ decreases, because the output port is accessible to forward data and there are no incoming high priority frames. Traversing the graphs along the sequence for both the buffers, we observe that the backlog in $Buffer^L$ is 4 at the reception of f_3^L at *t=35*.

Now consider Figure 3, where frames are grouped and the high priority group arrives first. The maximum backlog in $Buffer^H$ is 10 which is same as the scenario in Figure 4. The low priority frames arriving after f_3^H are accumulated in $Buffer^L$ from *t=22* until the output port is free at *t=32*. We observe that the backlog in $Buffer^L$ at the reception of f_3^L is 10. In the grouped order of arrival, independent of how frames are ordered within the groups, except for f_3^L, the backlog encountered by f_3^L is always 10. We can see that when $Buffer^H$ contains at least one completely stored frame, it delays the forwarding of low priority frames from $Buffer^L$. In the worst case, $Buffer^L$ stores bl_{max}^H units of data. In a non-grouped arrival order, all low priority frames might be forwarded before the reception of f_3^L, as shown in Figure 4.

For a grouped order of arrival with high priority followed by low priority frames, it is certain that in $Buffer^L$ maximum accumulation of low priority data occurs. Therefore, maximum backlog at the reception of f_m is attained in a grouped order of arrival of frames.

When all high priority frames arrive consecutively, the maximum backlog in $Buffer^H$ at the reception of the last high priority frame is:

$$bl_{max}^H = \max_{1 \le v \le \omega^H}\left(s(f_v^H)\right). \tag{5}$$

For example in Figure 3, $bl_{max}^H = 10$. At *t=22* $Buffer^L$ starts to accumulate the low priority data until $Buffer^H$ is empty at *t=32*. The amount of data accumulated bl_{acc}^L in $Buffer^L$, is defined as:

bl_{acc}^L: Backlog accumulated from the point in time when the first bit of the first low priority frame gets buffered in $Buffer^L$ to a point in time when $Buffer^H$ is empty.

We achieve the following cases to calculate bl_{acc}^L:

- Case A: If $bl_{max}^H < \sum\limits_{u=1}^{\omega} s(f_u^L)$, as seen in Figure 3 between *t=22* and *t=32*,

$$bl_{acc}^L = bl_{max}^H = \max_{1 \le v \le \omega^H}(s(f_v^H)). \tag{6}$$

- Case B: If $bl_{max}^H \ge \sum\limits_{u=1}^{\omega} s(f_u^L)$.

$$bl_{acc}^L = \sum_{u=1}^{\omega} s(f_u^L). \tag{7}$$

Accordingly, for the above two cases, the maximum backlog in $Buffer^L$ is:

$$bl_{max}^L = bl_{acc}^L. \tag{8}$$

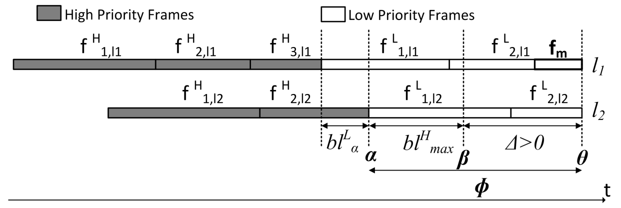


Fig. 5: Scheduling of frames on two input links when $\Delta > 0$ and ${}_{int}bl^L_{l_2} > s(f^L_{1,l_2})$.

- Case C: However, if the size of the incoming low priority frame is greater than bl^H_{max}, the maximum backlog in $Buffer^L$ is the low priority frame with maximum size.

$$bl^L_{max} = \max_{1 \le u \le \omega^L}(s(f^L_u)) \tag{9}$$

■

2) Case 2: Competing Flows Sharing Two Input Links: Figure 5 depicts the worst case arrival sequence to compute the maximum backlog at the reception of f_m in $Buffer^L$, when competing frames share two input links. The frame under study, f_m, shares the first input link l_1. The sequence on each link is grouped into high priority followed by low priority frames, as in Case 1. Further, frames within grouped sequences are arranged in decreasing order of sizes. The following points describe Figure 5 and will be used in Lemma 2:

- Last frames on l_1 and l_2 are received at the same time θ.
- α is the point in time when the last bit of the last high priority frame is received by the switch.
- β is the point in time when $Buffer^H$ is empty.
- Φ: Sum of sizes of all low priority frames in the shortest low priority sequence between α and θ.
- Δ: Data arriving on a single input link between β and θ. For negative values, $|\Delta|$ represents the backlog in $Buffer^H$ at t=θ.
- bl^L_α: Backlog in $Buffer^L$ at α.

Lemma 2: *Given two continuous sequences of frames on two input links l_1 and l_2, each grouped into high priority followed by low priority frames and all frames within the grouped sequence arranged in decreasing order of sizes, $s(f^H_{1,l_j}) > s(f^H_{2,l_j}) > ... > s(f^H_{n^H_{l_j},l_j}), s(f^L_{1,l_j}) > s(f^L_{2,l_j}) > ... > s(f^L_{n^L_{l_j},l_j}), f_m$ where j = {1,2}. If the last frames of both sequences on l_1 and l_2 are received at the same time θ, the backlog at time θ is:*

$$bl^L_{max} = \begin{cases} \sum_{j=1}^{2}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) - \min_{j=1,2}\sum_{u=1}^{n^L_{l_j}}(s(f^L_{u,l_j}) + bl^H_{max} \\ \textbf{\textit{if}}\ {}_{\mathbf{int}}\mathbf{bl^L_{l_1}} > \mathbf{s(f^L_{1,l_1})}\ \textbf{\textit{and}}\ \mathbf{\Delta > 0} \\ \\ \sum_{j=1}^{2}\sum_{u=1}^{n^L_{l_j}}(s(f^L_{u,l_j})) \\ \textbf{\textit{if}}\ {}_{\mathbf{int}}\mathbf{bl^L_{l_1}} > \mathbf{s(f^L_{1,l_1})}\ \textbf{\textit{and}}\ \mathbf{\Delta \le 0} \\ \\ \sum_{j=1}^{2}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) - \max_{j=1,2}\sum_{u=2}^{n^L_{l_j}} s(f^L_{u,l_j}) \\ \textbf{\textit{if}}\ {}_{\mathbf{int}}\mathbf{bl^L_{l_1}} < \mathbf{s(f^L_{1,l_1})} \end{cases} \tag{10}$$

Proof: As in case 1, in the worst case, all low priority frames arrive consecutively (in case 2, simultaneously from the two input links) when the backlog in $Buffer^H$ is the maximum. In this scenario, no frame leaves $Buffer^L$ while the largest possible number of low priority frames enters $Buffer^L$. From section III we know that bl^H_{max} occurs when all high priority frames arrive in decreasing order of sizes. To ensure this worst case condition, we construct the scenario in which all high priority frames arrive in decreasing order of sizes before all low priority frames. Next we will present how to compute bl^H_{max} using the example presented in Figure 5.

First, we determine the earliest starting high priority sequence. In the scenario of Figure 5, l_1 is this sequence. If l_1 were to be the only sequence, the maximum backlog in $Buffer^H$ is $s(f^H_{1,l_1})$ as shown in Case 1.

Let us consider now that the high priority frames arrive also from l_2. Assuming that the high priority sequences on l_1 and l_2 end at the same time and due to the non-preemptiveness:

$$bl^H_{max} = \sum_{j=1}^{2}\sum_{v=1}^{n^H_{l_j}} s(f^H_{v,l_j}) - \max_{j=1,2}\sum_{v=2}^{n^H_{l_j}} s(f^H_{v,l_j}) + \max_{1 \le u \le \omega^L}\big(s(f^L_u)\big). \tag{11}$$

The above bound may be pessimistic because the high priority sequences on both the links do not end at the same time. Further, when $Buffer^H$ reaches bl^H_{max}, $Buffer^L$ achieves a backlog of bl^L_α as shown in Figure 5. To calculate bl^L_α, the point in time α has to be determined. Due to the grouped order of arrival, the shortest low priority sequence on l_2 determines α. Hence,

$$\Phi = \sum_{u=1}^{n^L_{l_2}}(s(f^L_{u,l_2})). \tag{12}$$

Consequently the backlog in $Buffer^L$ at α is:

$$bl^L_\alpha = \sum_{u=1}^{n^L_{l_1}}(s(f^L_{u,l_1})) - \Phi. \tag{13}$$

From point α to β, $Buffer^H$ imposes a delay of bl^H_{max} units of time on the low priority frames in $Buffer^L$. We compute bl^L_{acc}, the backlog accumulated in $Buffer^L$ at β:

$$bl^L_{acc} = bl^L_\alpha + (2 \times bl^H_{max}). \tag{14}$$

The factor 2 in the above equation accounts for bl^H_{max} amount of data from both the links.

To compute the worst case backlog in $Buffer^L$, as seen in section III, all low priority frames on each link have to arrive in decreasing order of sizes. Next, we have to determine the earliest starting low priority sequence. In our scenario, it is the low priority sequence on l_1.

At β, f^L_{1,l_1} is forwarded from $Buffer^L$ as l_1 is the earliest starting sequence for low priority frames. Hence, the maximum backlog in $Buffer^L$ is computed by adding Δ to bl^L_{acc}:

$$bl^L_{max} = bl^L_{acc} + \Delta \tag{15}$$

where, $\Delta = \Phi - bl^H_{max}$.

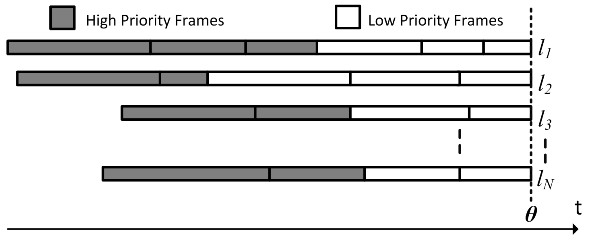


Fig. 6: Worst case arrival scenario to compute maximum backlog in Low Priority Buffer

(15) is expanded as follows:

$$bl^L_{max} = \sum_{j=1}^{2}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) - \min_{j=1,2}\sum_{u=1}^{n^L_{l_j}}(s(f^L_{u,l_j}) + bl^H_{max}. \quad (16)$$

The above equation is derived when $\Delta > 0$ for the scenario in Figure 5. Consider another case where $Buffer^H$ is empty after θ, i.e. $\Delta \leq 0$. Then, $Buffer^L$ stores all low priority frames. Thus, when $\beta > \theta$:

$$bl^L_{acc} = bl^L_{\alpha} + (2 \times \Phi) = \sum_{j=1}^{2}\sum_{u=1}^{n^L_{l_j}}(s(f^L_{u,l_j})). \quad (17)$$

Consequently,

$$bl^L_{max} = \sum_{j=1}^{2}\sum_{u=1}^{n^L_{l_j}}(s(f^L_{u,l_j})). \quad (18)$$

(16) and (18) are derived if f^L_{1,l_1} is completely stored at time β. In case, f^L_{1,l_1} is not completely stored it cannot be forwarded from $Buffer^L$. Therefore, we need to check if the backlog contributed by the earliest starting sequence of low priority at β contains a completely stored frame. Let ${}_{int}bl^L_{l_1}$ be the backlog contributed to bl^L_{acc} by the earliest starting sequence of low priority l_1:

$${}_{int}bl^L_{l_1} = bl^L_{\alpha} + bl^H_{max} \quad (19)$$

If $s(f^L_{1,l_1})\ >_{int}\ bl^L_{l_1}$ and $\Delta > 0$, then:

$$bl^L_{max} = \sum_{j=1}^{2}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) - \max_{j=1,2}\sum_{u=2}^{n^L_{l_j}} s(f^L_{u,l_j}). \quad (20)$$

■

3) Case 3: Competing Flows Sharing Multiple Input Links: This case generalizes Lemma 2 from two to multiple input links. Figure 6 shows the worst case arrival sequence when competing flows share N input links. The following lemma gives the maximum backlog at time θ in $Buffer^L$.

LEMMA 3: *Consider N continuous sequences of frames on N input links each grouped into high priority followed by low priority and all frames ordered in decreasing size in their respective group $s(f^H_{1,l_j}) > s(f^H_{2,l_j}) > ... > s(f^H_{n^H_{l_j},l_j}), s(f^L_{1,l_j}) > s(f^L_{2,l_j}) > ... > s(f^L_{n^L_{l_j},l_j}), f_m$ where $1 \leq j \leq N$. If the last frames on all links arrive at the same time θ, the backlog at time θ is:*

$$bl^L_{max} = \begin{cases} \sum_{j=1}^{N}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) - \min_{1\leq j\leq N}\sum_{u=1}^{n^L_{l_j}}(s(f^L_{u,l_j}) + bl^H_{max} \\ \quad \textbf{\textit{if}}\ {}_{\mathbf{int}}\mathbf{bl^L_{l_r}} > \mathbf{s(f^L_{1,l_r})}\ \textbf{\textit{and}}\ \mathbf{\Delta > 0} \\ \sum_{j=1}^{N}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) \\ \quad \textbf{\textit{if}}\ {}_{\mathbf{int}}\mathbf{bl^L_{l_r}} > \mathbf{s(f^L_{1,l_r})}\ \textbf{\textit{and}}\ \mathbf{\Delta \leq 0} \\ \sum_{j=1}^{N}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) - \max_{1\leq j\leq N}\sum_{u=2}^{n^L_{l_j}} s(f^L_{u,l_j}) \\ \quad \textbf{\textit{if}}\ {}_{\mathbf{int}}\mathbf{bl^L_{l_r}} < \mathbf{s(f^L_{1,l_r})} \end{cases} \quad (21)$$

Proof: The procedure followed is similar to the proof of Lemma 2.

To compute the worst case backlog in $Buffer^H$, assume that all high priority sequences end at the same time. bl^H_{max} is given by:

$$bl^H_{max} = \sum_{j=1}^{N}\sum_{v=1}^{n^H_{l_j}} s(f^H_{v,l_j}) - \max_{1\leq j\leq N}\sum_{v=2}^{n^H_{l_j}} s(f^H_{v,l_j}) + \max_{1\leq u\leq \omega^L}\left(s(f^L_u)\right) \quad (22)$$

The backlog in $Buffer^L$ at α when $Buffer^H$ reaches bl^H_{max} is given by:

$$bl^L_{\alpha} = \sum_{j=1}^{N}\left(\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) - \Phi\right) \quad (23)$$

where,

$$\Phi = \min_{1\leq j\leq N}\left(\sum_{u=1}^{n^L_{l_j}}(s(f^L_{u,l_j}))\right). \quad (24)$$

The accumulated backlog in $Buffer^L$ at β due to the delay imposed by $Buffer^H$ is:

$$bl^L_{acc} = bl^L_{\alpha} + (N \times bl^H_{max}). \quad (25)$$

If $\beta > \theta$, i.e. $\Delta \leq 0$, then all low priority frames are accumulated:

$$bl^L_{acc} = bl^L_{\alpha} + (N \times \Phi) = \sum_{j=1}^{N}\sum_{u=1}^{n^L_{l_j}}(s(f^L_{u,l_j})). \quad (26)$$

At β, we need to check if the backlog contributed by the earliest starting sequence of low priority includes a completely stored frame. Let the earliest starting starting sequence be l_r. Then ${}_{int}bl^L_{l_r}$, the backlog contributed by l_r until time β is given by:

$${}_{int}bl^L_{l_r} = bl^L_{\alpha,l_r} + bl^H_{max}, \text{ where } bl^L_{\alpha,l_r} = \sum_{u=1}^{n^L_{l_r}}(s(f^L_{u,l_r})) - \Phi. \quad (27)$$

The worst case backlog in $Buffer^L$ at θ when ${}_{int}bl^L_{l_r}$ includes $s(f^L_{1,l_r})$ is:

$$bl^L_{max} = bl^L_{acc} + (N-1) \times \Delta. \tag{28}$$

(28) is expanded as:

$$bl^L_{max} = \sum_{j=1}^{N}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) - \min_{1\le j\le N}\sum_{u=1}^{n^L_{l_j}}(s(f^L_{u,l_j}) + bl^H_{max}. \tag{29}$$

If $\Delta \le 0$, then all low priority frames are included in bl^L_{acc}. Therefore,

$$bl^L_{max} = \sum_{j=1}^{N}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}). \tag{30}$$

If $s(f^L_{1,l_r}) >_{int} bl^L_{l_r}$ and $\Delta > 0$, then

$$bl^L_{max} = \sum_{j=1}^{N}\sum_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j}) - \max_{1\le j\le N}\sum_{u=2}^{n^L_{l_j}} s(f^L_{u,l_j}). \tag{31}$$

■

E. Worst Case Backlog Computation

To compute the worst case backlog encountered by f_m, we have to include the number of frames of each competing high and low priority flow that are present in the busy period of f_m, as presented in IV-C.

PROPERTY 3: *For a frame f_m of a low priority flow τ^L_i, generated at time t at its source node. Let τ^H_{v,l_j} and τ^L_{u,l_j} be the high and low priority flows from link l_j competing with f_m. The worst case backlog encountered by f_m at the output port of switch h with N input links is:*

$$bl^L_{max_{i,t}} \le \begin{cases} \sum\limits_{j=1}^{N}\sum\limits_{f^L_u\in\tau^L_{u,l_j}} \left(\kappa^h_{i,u,t}\times s(f^L_u)\right) \\ \quad - \min\limits_{1\le j\le N}\left(\sum\limits_{f^L_u\in\tau^L_{u,l_j}}\left(\kappa^h_{i,u,t}\times s(f^L_u)\right)\right) + bl^H_{max_{i,t}} \\ \quad \textit{if } {}_{\mathbf{int}}\mathbf{bl^L_{l_r}} > \mathbf{s(f^L_{1,l_r})} \textit{ and } \mathbf{\Delta > 0} \\ \sum\limits_{j=1}^{N}\sum\limits_{f^L_u\in\tau^L_{u,l_j}} \left(\kappa^h_{i,u,t}\times s(f^L_u)\right) \\ \quad \textit{if } {}_{\mathbf{int}}\mathbf{bl^L_{l_r}} > \mathbf{s(f^L_{1,l_r})} \textit{ and } \mathbf{\Delta \le 0} \\ \sum\limits_{j=1}^{N}\sum\limits_{f^L_u\in\tau^L_{u,l_j}} \left(\kappa^h_{i,u,t}\times s(f^L_u)\right) - \Xi^h_{i,t} \\ \quad \textit{if } {}_{\mathbf{int}}\mathbf{bl^L_{l_r}} < \mathbf{s(f^L_{1,l_r})} \end{cases} \tag{32}$$

with:

$$\Xi^h_{i,t} = \max_{1\le j\le N}\left(\sum_{f^L_u\in\tau^L_{u,l_j}}\left(\kappa^h_{i,u,t}\times s(f^L_u)\right) - \max_{f^L_u\in\tau^L_{u,l_j}} s(f^L_u)\right) \tag{33}$$

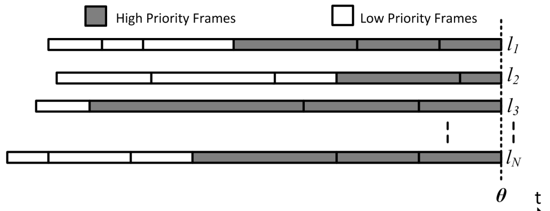


Fig. 7: Worst case arrival scenario to compute maximum backlog in High Priority Buffer

and

$$bl^H_{max_{i,t}} \le \sum_{j=1}^{N}\sum_{f^H_v\in\tau^H_{v,l_j}}\left(\eta^h_{i,v,t}\times s(f^H_v)\right) - \Lambda^h_{i,t} \tag{34}$$

with

$$\Lambda^h_{i,t} = \max_{1\le j\le N}\left(\sum_{f^H_v\in\tau^H_{v,l_j}}\left(\eta^h_{i,v,t}\times s(f^H_v)\right) - \max_{f^H_v\in\tau^H_{v,l_j}} s(f^H_v)\right) \tag{35}$$

Proof: The above equations for $bl^L_{max_{i,t}}$ complements (21). The term $\sum\limits_{j=1}^{N}\sum\limits_{f^L_u\in\tau^L_{u,l_j}}\left(\kappa^h_{i,u,t}\times s(f^L_u)\right)$ computes the sum of all low priority frames to be transmitted from all links at the switch and corresponds to the term $\sum\limits_{j=1}^{N}\sum\limits_{u=1}^{n^L_{l_j}} s(f^L_{u,l_j})$.

$\Xi^h_{i,t}$ determines the longest low priority sequence among the input links at switch *h* without including the largest frame and corresponds to the term $\max\limits_{1\le j\le N}\sum\limits_{u=2}^{n^L_{l_j}} s(f^L_{u,l_j})$.
Further, $bl^H_{max_{i,t}}$ which corresponds to the term bl^H_{max}, represents data to be buffered due to the delay imposed by $Buffer^H$ on $Buffer^L$.
Similarly, the terms in $bl^H_{max_{i,t}}$ correspond to the terms in (22).

For each low priority flow that shares $Buffer^L$, the worst case backlog for one of its frames has to be computed. The maximum value among these individual low priority flow worst case backlogs results in the worst case backlog in $Buffer^L$. ■

V. Worst Case Backlog Computation in a High Priority Buffer using Trajectory Approach

The frame under study f_m belongs to a high priority flow i.e. $f_m \in \tau^H_i$. Figure 7 represents the worst case arrival scenario to compute maximum backlog faced by f_m. All the high priority frames arrive continuously in decreasing order of sizes, with the last frames ending at the same time θ. The order of arrival of low priority frames is neglected. Lemma 4 gives the worst case backlog in $Buffer^H$.

LEMMA 4: *Consider N continuous sequences of frames arriving from N input links, each grouped into low priority frames followed by high priority frames, and only the high priority frames ordered in decreasing order of sizes $s(f^H_{1,l_j}) > s(f^H_{2,l_j}) > \dots > s(f^H_{n^H_{l_j},l_j}), f_m$. If the last frames on all the* *links arrive at the same time θ, the backlog at time θ is:*

$$bl^H_{max} = \begin{cases} \sum_{j=1}^{N}\sum_{v=1}^{n^H_{l_j}} s(f^H_{v,l_j}) - \max_{1\le j\le N}\sum_{v=2}^{n^H_{l_j}} s(f^H_{v,l_j}) + \max_{1\le u\le \omega^L}\left(s(f^L_u)\right) \\ \textit{if } {}_{\mathbf{int}}\mathbf{bl}^{\mathbf{H}}_{\mathbf{l_y}} > \mathbf{s}(\mathbf{f}^{\mathbf{H}}_{\mathbf{1,l_y}}) \textit{ and } \boldsymbol{\Delta} > \mathbf{0} \\ \\ \sum_{j=1}^{N}\sum_{v=1}^{n^H_{l_j}} (s(f^H_{v,l_j})) \\ \textit{if } {}_{\mathbf{int}}\mathbf{bl}^{\mathbf{H}}_{\mathbf{l_y}} > \mathbf{s}(\mathbf{f}^{\mathbf{H}}_{\mathbf{1,l_y}}) \textit{ and } \boldsymbol{\Delta} \le \mathbf{0} \\ \\ \sum_{j=1}^{N}\sum_{v=1}^{n^H_{l_j}} s(f^H_{v,l_j}) - \max_{1\le j\le N}\sum_{v=2}^{n^H_{l_j}} s(f^H_{v,l_j}) \\ \textit{if } {}_{\mathbf{int}}\mathbf{bl}^{\mathbf{H}}_{\mathbf{l_y}} < \mathbf{s}(\mathbf{f}^{\mathbf{H}}_{\mathbf{1,l_y}}) \end{cases} \tag{36}$$

with

$$\begin{aligned} \Delta &= \Phi - \max_{1\le u\le \omega^L}\left(s(f^L_u)\right) \\ \Phi &= \min_{1\le j\le N}\left(\textstyle\sum_{v=1}^{n^H_{l_j}}(s(f^H_{v,l_j}))\right) \\ {}_{int}bl^H_{l_y} &= bl^H_{\alpha,l_y} + \max_{1\le u\le \omega^L}\left(s(f^L_u)\right) \\ bl^L_{\alpha,l_y} &= \textstyle\sum_{v=1}^{n^L_{l_y}}(s(f^H_{v,l_y})) - \Phi \end{aligned} \tag{37}$$

Proof: In the worst case, due to non-preemption, a high priority frame is delayed by the largest low priority frame. Therefore, $Buffer^L$ imposes a delay of only $\max_{1\le u\le \omega^L}(s(f^L_u))$ and not bl^L_{max} on $Buffer^H$. As we are not concerned with the maximum backlog possible in $Buffer^L$, the order of arrangement of the low priority frames is neglected. Thus, the worst case sequence to compute maximum backlog for a high priority frame occurs, when all the high priority frames arrive continuously in decreasing order of sizes.

Therefore, the backlog contributed due to contention among only high priority frames is, sum of the largest high priority frame from the earliest starting sequence and all the other high priority frames from other sequences. Further data to be added due to the delay imposed by $Buffer^L$ is given by $\max_{1\le u\le \omega^L}(s(f^L_u))$. If l_y is the earliest starting high priority sequence, we obtain the equations presented in the Lemma 4 to compute bl^H_{max}.

The worst case backlog is computed by including the number of frames of each competing high priority flow, $\eta^h_{i,v,t}$ from section IV-C, in (36),

$$bl^H_{max_{i,t}} \le \begin{cases} \sum_{j=1}^{N}\sum_{f^H_v\in\tau^H_{v,l_j}}\left(\eta^h_{i,v,t}\times s(f^H_v)\right) - \Lambda^h_{i,t} + \max_{1\le u\le \omega^L}(s(f^L_u)) \\ \textbf{if } {}_{\mathbf{int}}\mathbf{bl}^{\mathbf{H}}_{\mathbf{l_y}} > \mathbf{s}(\mathbf{f}^{\mathbf{H}}_{\mathbf{1,l_y}}) \textbf{ and } \boldsymbol{\Delta} > \mathbf{0} \\ \\ \sum_{j=1}^{N}\sum_{f^H_v\in\tau^H_{v,l_j}}\left(\eta^h_{i,v,t}\times s(f^H_v)\right) \\ \textbf{if } {}_{\mathbf{int}}\mathbf{bl}^{\mathbf{H}}_{\mathbf{l_y}} > \mathbf{s}(\mathbf{f}^{\mathbf{H}}_{\mathbf{1,l_y}}) \textbf{ and } \boldsymbol{\Delta} \le \mathbf{0} \\ \\ \sum_{j=1}^{N}\sum_{f^H_v\in\tau^H_{v,l_j}}\left(\eta^h_{i,v,t}\times s(f^H_v)\right) - \Lambda^h_{i,t} \\ \textbf{if } {}_{\mathbf{int}}\mathbf{bl}^{\mathbf{H}}_{\mathbf{l_y}} < \mathbf{s}(\mathbf{f}^{\mathbf{H}}_{\mathbf{1,l_y}}) \end{cases} \tag{38}$$

with

$$\Lambda^h_{i,t} = \max_{1\le j\le N}\left(\sum_{f^H_v\in\tau^H_{v,l_j}}\left(\eta^h_{i,v,t}\times s(f^H_v)\right) - \max_{f^H_v\in\tau^H_{v,l_j}} s(f^H_v)\right) \tag{39}$$

■

VI. Conclusion

AFDX is a network widely used in distributed avionics systems. The AFDX standard describes the characteristics of the switches and states that "data contention at the output ports is resolved by buffering". The standard further defines the minimum size for the output buffer. However, in order to avoid buffer overflow, and consequently data loss, the actual total size for each priority level buffer (high and low) is left as a design decision [2].
Previous works presented methods to compute upper bound backlog for AFDX switches considering single priority VLs. In this paper we presented a method to compute an upper bound for the backlog of both low and high priority buffers. We showed how to identify and compute the frames competing for a buffer output port and further presented the arrival sequence of frames (scheduling of frames) that lead to the worst case backlog for both priority levels.

Future work includes a detailed comparison on the results achieved by our paper with those achieved using network calculus. Additionally, we will investigate the impact of off-line scheduled messages into the computation of the buffer backlog and extend the analysis from two to arbitrarily many priority flows.

References

[1] R. Alena, J. Ossenfort, K. Laws, A. Goforth, and F. Figueroa, "Communications for integrated modular avionics," in *IEEE Aerospace Conference*, 2007, pp. 1–18.

[2] "ARINC specification 664 P7-1. Aircraft Data Network Part-7 Avionics Full-Duplex Switched Ethernet Network," September 2009.

[3] S. Martin and P. Minet, "Schedulability analysis of flows scheduled with fifo: application to the expedited forwarding class," in *IPDPS*, 2006.

[4] M. Boyer and C. Fraboul, "Tightening end to end delay upper bound for AFDX network calculus with rate latency fifo servers using network calculus," in *WFCS*, 2008, pp. 11–20.

[5] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Worst-case backlog evaluation of avionics switched ethernet networks with the trajectory approach," in *ECRTS*, July 2012, pp. 78 –87.

[6] F. Ridouard, J.-L. Scharbarg, and C. Fraboul, "Probabilistic upper bounds for heterogeneous flows using a static priority queueing on an AFDX network," in *ETFA*, September 2008, pp. 1220 –1227.

[7] ——, "Stochastic network calculus for buffer overflow evaluation in an avionics switched ethernet," in *Junior Researcher Workshop on Real-Time Computing*, March 2007, pp. 55–58.

[8] H. Bauer, J. Scharbarg, and C. Fraboul, "Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network," in *ETFA*, 2009, pp. 1–8.

[9] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Applying trajectory approach with static priority queueing for improving the use of available AFDX resources," *Real-Time Systems*, vol. 48, no. 1, pp. 101–133, January 2012.

[10] S. Martin and P. Minet, "Worst case end-to-end response times of flows scheduled with FP/FIFO," in *ICN/ICONS/MCL*, April 2006, p. 54.

MTU Assignment in a Master-Slave Switched Ethernet Network

Mohammad Ashjaei, Moris Behnam, Thomas Nolte
Mälardalen University, Västerås, Sweden
{mohammad.ashjaei, moris.behnam, thomas.nolte}@mdh.se

Luis Almeida
IT / DEEC, University of Porto, Portugal
lda@fe.up.pt

Abstract—**In this paper, we investigate the problem of selecting the Maximal Transmission Unit (MTU) size that maximizes the schedulability of real-time messages. We focus on a bandwidth-efficient master-slave switched Ethernet protocol, namely the FTT-SE protocol. We propose an algorithm to find the MTU for each message in order to maximize the schedulability of the messages. Moreover, we evaluate our proposed algorithm and we show that setting the MTU for messages using the algorithm increases the schedulability of messages compared with assigning the MTU to the maximum value that the protocol can support.**

I. Introduction

Nowadays, there is an increasing demand towards using high performance network solutions for real-time Networked Embedded Systems (NES) due to the growth of the number of nodes in such systems, their increased amount of functionalities and the high amount of information being transmitted between the nodes. Ethernet has been proposed as an interesting technology for such systems as it provides high throughput, low cost, wide availability and general maturity. To overcome the limitation of Ethernet with respect to real-time guarantees, it has been complemented with suitable transmission control mechanisms, being the base for several real-time communication protocols currently used in NES, such as PROFINET, Ethernet POWERLINK, TTEthernet and FTT-SE. To keep the high performance of the Ethernet based protocols, the network should be configured properly and one of the configuration parameters that has a significant effect on the performance of the protocols is the selection of the maximum packet size of messages [1].

In the area of Ethernet protocols, a packet is defined to hold up to 1500 data bytes which is relatively high compared with other communication technologies. As a configuration parameter, the maximum data size that a packet can hold in Ethernet is called the Maximal Transmission Unit (MTU), which has a big impact on the performance of the network and the bandwidth utilization. For instance, the MTU size affects the minimum slot time in TTEthernet [2], while in cyclic base protocols, such as the FTT-SE protocol [3], it affects the size of the idle time, which is considered to prevent overruns between cycles. In this paper, we mainly focus on the FTT-SE protocol which is based on a master-slave switched Ethernet technology.

Considering industrial real-time applications, the amount of data to be transmitted can vary from small to very large, for instance, the flow size might be rather large for automation applications based on video streams or machine vision. Therefore, the data of such applications should be fragmented to several packets to be transmitted sequentially. However, selecting the best MTU that guarantees the real-time requirements for all messages is challenging. On one hand, a larger MTU will reduce the number of packets needed to transmit messages which in turn reduces the total transmission time of messages due to a lower amount of overhead associated with Ethernet packets. On the other hand, the large MTU increases the idle time used in every cycle to prevent overruns which in turn decreases the efficiency of the protocol. This contradicting effect has been discussed in [1] for the FTT-SE protocol and two algorithms (optimal and simplified) have been proposed to find the optimum MTU for all messages. The algorithms are based on the utilization bound schedulability test and they only consider the effect of messages that share the same destination node, while the impact of other messages that might delay the transmission of messages has not been included in the analysis.

In this paper, we generalize the solution presented in [1] by including the effect of all messages that can delay the transmission of messages. In addition, we propose an algorithm based on the response-time schedulability analysis to find a proper MTU for each message to increase the schedulability of systems. We show that the proposed algorithm increases the schedulability compared with the case when the MTU is set (configured) to the maximum Ethernet packet in the network.

The rest of the paper is structured in the following way. Section II presents related work. Section III outlines the basics of the FTT-SE protocol. Section IV presents the system model and Section V sketches the schedulability analysis. Moreover, Section VI proposes the heuristic algorithm, while Section VII shows the evaluation of the algorithm. Finally, Section VIII concludes the paper and presents the future work.

II. Related Work

The problem of fragmenting messages into smaller packets transmitted over large heterogeneous networks has been discussed in [4]. In such networks, some routes can carry limited packet size and large messages should be fragmented leading to a higher protocol overhead and a lower throughput. For such a problem, optimal routing techniques are developed to avoid message fragmentation as much as possible.

In the context of wireless networks, having several small packets degrade the throughput of the network due to the protocol overhead inherent to the transmission of each packet.

On the other hand, using a large packet size may also affect the efficiency due to retransmission of faulty packets. Therefore, optimal solutions have been proposed in [5] and [6] in the area of wireless networks. Moreover, in [7] an algorithm to dynamically adjust the packet size in multi-level security wireless networks is proposed in which the goal is to minimize the overhead in each packet. The same goal as finding the optimized packet length for wireless sensor networks is presented in [8], where the criterion for optimization is energy efficiency rather than the bandwidth efficiency.

However, the above proposed solutions are not applicable in this paper as the source of the problem and the goals are different where we focus mostly on real time guarantees.

The work presented in [9] proposed an algorithm to select optimal preemption points in order to increase the schedulability of real-time tasks. The criteria to select the optimum preemption points is based on decreasing the overhead of task preemption and the blocking from lower priority tasks on the higher priority tasks. Adding preemption points inside the execution of tasks can be modeled as fragmenting the tasks into a set of subtasks which is similar to fragment messages into a set of packets. Nevertheless, the proposed algorithm is not suitable for our case as it tries to optimize the non-preemptive regions (between two preemption points) of lower priority tasks that block the execution of higher priority tasks. While in our case selecting the MTU of higher priority message can contribute to their transmission time and also the idle time included on every scheduling cycle for each message affecting the schedulability of all messages.

The work presented in [1] is the most related work, where two algorithms were proposed to find one optimum MTU for all messages in the scope of the FTT-SE protocol. However, the presented algorithms are based on the utilization bound schedulability and do not consider all messages that can delay the considered messages. In this paper, we use a tighter schedulability test based on response time analysis and we consider all messages that can interfere with the messages under consideration. In addition and to improve the efficiency, we assign an individual MTU for each message unlike the previous work where only one MTU is assigned for all messages. Note that the algorithms presented in [1] can only give optimal results for very simple cases assuming that all messages are forwarded to the same destination. Otherwise, a very high computational complexity algorithm is required to find the optimal solution. In this paper, we propose a heuristic algorithm based on the response time analysis to find MTUs for all messages that keep the system schedulable, i.e., all messages meet their deadlines.

III. The FTT-SE Basics

The FTT-SE protocol [3] is an Ethernet real-time communication protocol that uses a master-slave technique to coordinate all traffic in the network. This protocol supports both synchronous and asynchronous traffic. The former is time-triggered and activated by the scheduler according to its period, whereas the latter traffic is issued by applications in the nodes.

The master node organizes the traffic in fixed time slots called Elementary Cycles (EC) and broadcasts a specific message, which is called the Trigger Message (TM), at the beginning of the EC. The scheduling of messages is carried out on-line according to some suitable scheduling policy, and the scheduled messages are encoded into the TM. The network nodes receive the TM, decode it and initiate the transmission of messages.

As depicted in Figure 1, the data communication in each EC is divided into two specific windows to handle synchronous and asynchronous traffic, which is called the synchronous window and asynchronous window respectively. Once the nodes in the system receive the TM, the time they need to decode it and initiate the transmissions is called turn around time (TRD).

The asynchronous messages make use of a signaling mechanism that allows the master to become aware of them and consider them in its internal traffic scheduling [10]. The signaling mechanism is based on so-called signaling messages (SIG) sent by the nodes to the master node, informing it of the status of the nodes queues. Whenever an asynchronous message becomes active in one node, this node informs the master in the next SIG message that it sends to schedule the asynchronous messages in the upcoming ECs, e.g., the messages A and B in Figure 1.

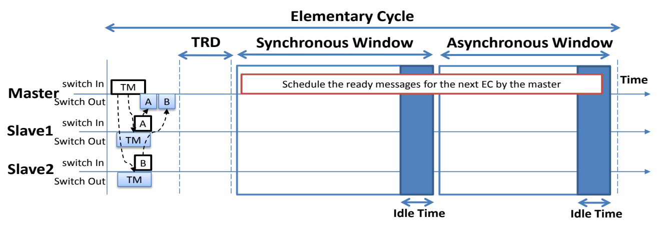


Fig. 1. The FTT-SE Elementary Cycle

The FTT-SE protocol automatically fragments large messages into several packets that are scheduled sequentially by the master node.

IV. System Model

In this paper, we consider the real-time periodic model to describe both synchronous and asynchronous messages as presented in the following set:

$$\Gamma = \{m_i(C_i, D_i, T_i, S_i, Ds_i, MTU_i, nP_i), i = 1..N\} \qquad (1)$$

In this set, C_i is the total transmission time of the message including all physical layer overheads such as inter-frame gap, D_i and T_i are the relative deadline and period of messages respectively, which are presented as integer number of ECs. Moreover, S_i is the source node and Ds_i is the destination node of the message (we assume unicast streams in this paper).

Also, MTU_i is the maximum packet size among the packets that compose m_i and nP_i is the number of packets. We model both, synchronous and asynchronous messages, with the same set in which T_i presents the minimum inter-arrival time for asynchronous messages. In this paper we assume that Γ is sorted by the descending priority of the messages. Finally, the fixed priority scheduling algorithm is used to schedule messages and the priorities of messages are assigned according to the Rate-Monotonic (RM) algorithm.

The switches are assumed to be Commercial Off-The-Shelf (COTS) and cut-through switching and ready messages in switches are scheduled using the First In First Out (FIFO) approach. We also consider the switch relaying latency (Δ) in the schedulability analysis.

According to the FTT-SE protocol, all messages which are scheduled to be transmitted in one EC should be received by the end of the EC. In order to prevent any overrun of the traffic, a message that cannot be fully transmitted within the transmission window is suspended for the next EC, e.g., m_1 in Figure 2.

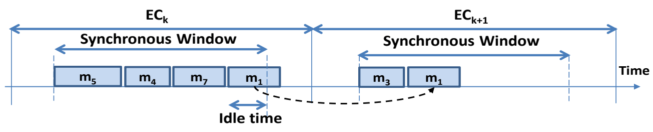


Fig. 2. The Idle Time Presentation

This property introduces an idle time in each transmission window that should be taken into account in the schedulability analysis.

V. SCHEDULABILITY ANALYSIS

In the FTT-SE protocol, the system is schedulable when all messages meet their deadlines. The schedulability is investigated by computing the response time (RT) for all messages. The system is guaranteed to be schedulable if $\forall m_i : RT(m_i) \leq D_i$. In addition, the FTT-SE scheduling is based on reserving a bandwidth every EC for each type of messages, both synchronous and asynchronous, which is similar to the periodic resource model presented in [11]. Therefore, in order to calculate the response time analysis, we perform the analysis based on a *request bound function (rbf)* and a *supply bound function (sbf)*.

The $rbf_i(t)$ represents the maximum load generated by m_i and all higher priority messages that can delay m_i within the time interval $[0,t]$. Therefore, the $rbf_i(t)$ is calculated by summing the total transmission time of the message itself, the interfering messages and the remote load interference denoted by $W_i(t)$ which will be discussed later in this section. The $rbf_i(t)$ computation is presented in (2), where $hp(m_i)$ is the set of messages with priority higher than that of m_i.

$$rbf_i(t) = C_i + \Delta + \sum_{\substack{\forall m_j \in hp(m_i) \wedge \\ (S_j = S_i \vee Ds_j = Ds_i)}} \lceil \frac{t}{T_j} \rceil C_j + W_i(t) \qquad (2)$$

Besides the interference of the messages that share links with the message under analysis m_i (i.e., $\forall m_j \in hp(m_i) \wedge (S_j = S_i \vee Ds_j = Ds_i)$), the message m_i may still be delayed indirectly through other messages. To show this effect let us consider the example illustrated in Figure 3.

In Figure 3, m_1 is transmitted from Node A to Node B, m_4 is sent from Node A to Node C and m_3 is transmitted from Node B to Node C and m_2 is sent from Node B to Node A. In this example we focus on m_4 and we assume that it is the lowest priority among the other messages. In this scenario, m_1 is delaying m_4 which cause delay in m_3 reception. If m_1 was not scheduled for this EC, it would be possible for m_3 to be transmitted in the EC. Therefore, m_1 delays m_3 even though they do not share links. We can call this effect remote load delay. Since m_4 has the lowest priority, the scheduler in the master node will suspend the transmission of m_4 to the later EC.

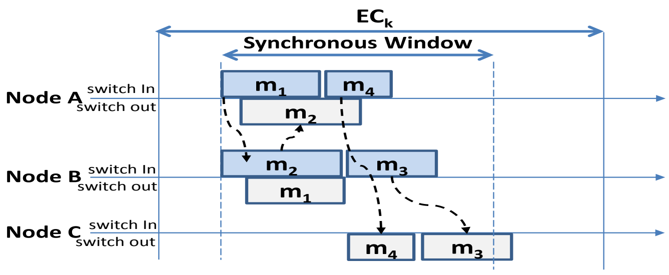


Fig. 3. The Remote Load Interference

To consider this delay in the analysis, all higher priority messages that share source node with the interfering messages are considered as higher priority interfering messages in the response time analysis as shown in (3).

$$W_i(t) = \sum_{\substack{\forall m_k \in hp(m_j) \wedge S_k = S_j \\ \wedge \forall m_j \in hp(m_i)}} \lceil \frac{t}{T_k} \rceil C_k \qquad (3)$$

The sbf(t) is the minimum effective communication capacity that the network supplies within the time interval $[0,t]$. Note that in each EC, a particular bandwidth is provided for transmitting each type of message which is imposed by $LSW - I$, where LSW is the length of the synchronous window and I is the idle time in that window. The idle time is upper bounded by the maximum packet size among the higher priority messages and the message under analysis. Thus, for the message m_i the supply bound function $sbf_i(t)$ is computed in (4).

$$\begin{aligned} sbf_i(t) &= (\tfrac{LSW - I_i}{EC}) \times t \\ I_i &= \max_{\forall m_j \in hp(m_i)} (MTU_i, MTU_j) \end{aligned} \qquad (4)$$

The response time of m_i is computed based on (5).

$$t^* = min(t > 0) : sbf_i(t) \geq rbf_i(t) \tag{5}$$

In order to determine t^*, the inequality should be checked in all instants that $rbf_i(t)$ changes due to interference of other messages up to D_i. Therefore, a set of check points is given by (6).

$$\begin{aligned} &CP_{rbf_i} = [\cup cp_{m_a}, \forall_{m_a \in hp(m_i)}] \cup D_i \\ &where, cp_{m_a} = T_a, 2T_a, ..., nT_a, n = \lfloor \tfrac{D_i}{T_a} \rfloor \end{aligned} \tag{6}$$

Finally, we compute the response time in number of ECs which is given by (7).

$$RT(m_i) = \lceil \frac{t^*}{EC} \rceil \tag{7}$$

The analyses explained above are suitable for the synchronous messages and the asynchronous messages. However, for asynchronous messages additional 2 EC delay should be added to the RT. The reason for this is that the request for asynchronous messages may have to wait 1 EC before the node signals it in the next SIG and the master then executes the scheduling one EC before the respective dispatching.

VI. MTU Assignment Algorithm

The maximum and minimum possible packet transmission times are limited to MTU_{max} and MTU_{min} which are defined according to the protocol specification. In this section we present an algorithm to find the MTU_i within $[MTU_{min}, MTU_{max}]$ such that the system becomes schedulable.

According to the schedulability analysis presented in Section V, the selection of MTU_i affects the response time analysis as it influences the bandwidth utilization in $sbf_i(t)$ through the idle time. Increasing MTU_i might increase I_i in (4) for message m_i and the other lower priority messages that share the same destination node. As a result, increasing I_i will decrease the $sbf_i(t)$ and hence decreasing the schedulability of the message. Moreover, increasing MTU_i will require less packets to transmit the data which in turn will decrease the total transmission time of the message and it will decrease $rbf_i(t)$ and as a result it will increase the schedulability of the system. Considering these two contradicting effects in the schedulability analysis, we may conclude that there is a tradeoff between decreasing the effect of idle time and the protocol overhead when changing the MTUs of messages.

Looking at (4, 2), we can conclude that selecting the MTU for a message not only affects the response time of that message itself but it affects the schedulability of the lower priority messages through the higher priority interference and remote load interference delay. In order to find the optimum solution a combination of all possible MTU ranges for all messages should be checked which requires an algorithm with an exponential computational complexity. Thus, in this section we present a heuristic algorithm to find the MTU_i.

In order to present the effect of MTU_i in the request bound function, the total transmission time of the message is formulated based on the MTU_i. The total message transmission time C_i includes the actual data transmission time C_i^* and the protocol overhead O. The protocol overhead O is a constant value which is added to each packet separately and includes Ethernet overhead, the FTT-SE protocol overhead and the inter-frame gap between the packets. Therefore, the total message transmission time equals to $C_i^* + nP_i \times O$. Note that, we consider the actual transmission time of a message equally split among its packets. This helps avoiding residual short packets and leads to increase the schedulability as described before. Thus, the MTU_i is evaluated in (8).

$$MTU_i = \lceil \frac{C_i^*}{nP_i} \rceil \tag{8}$$

The number of packets for each message can be expressed as in (9).

$$nP_i = \lceil \frac{C_i^*}{MTU_i} \rceil \tag{9}$$

We can reformulate the total message transmission time C_i to be a function of MTU_i by considering nP_i from (9). Moreover, we can approximate the equation by removing the ceiling in the equation which is presented in (10).

$$C_i = C_i^* + (\frac{C_i^*}{MTU_i} + 1) \times O \tag{10}$$

We expand the inequality (5) by substituting the $sbf_i(t)$ from (4) and the $rbf_i(t)$ from (2). Also, the transmission time C_i in $rbf_i(t)$ can be replaced with (10). Due to the max function in the $sbf_i(t)$, we need to evaluate the inequality in two different conditions.

In the first condition, we assume that the MTU_i is greater or equal to the maximum MTU of all higher priority messages than that of m_i. Therefore, a quadratic equation is derived as a function of MTU_i as it is presented in (11).

$$\tfrac{t}{EC} MTU_i^2 + (C_i^* + M(t)) \times MTU_i + (C_i^* \times O) \leq 0 \tag{11}$$

$$\begin{aligned} M(t) = \quad & O + \Delta - \tfrac{LSW \times t}{EC} + \sum_{\substack{\forall m_j \in hp(m_i) \wedge \\ (S_j = S_i \vee Ds_j = Ds_i)}} \lceil \frac{t}{T_j} \rceil C_j \\ & + \sum_{\substack{\forall m_k \in hp(m_j) \wedge S_k = S_j \\ \wedge \forall m_j \in hp(m_i)}} \lceil \frac{t}{T_k} \rceil C_k \end{aligned} \tag{12}$$

Note that, the quadratic equation (11) has two solutions which shows a range of solutions that satisfies the inequality. Moreover, the coefficient of MTU_i^2 is always positive as t and EC are always positive integers. Therefore, the parabola opens upwards in this case, i.e., the quadratic has a minimum value. As a result, given $MTU_i[lo]$ and $MTU_i[hi]$ as lower value and higher value of the solutions respectively, that make the left side of the equation equal to zero, all the values within the range $[MTU_i[lo], MTU_i[hi]]$ guarantee the schedulability of that message.

If the primitive condition is not satisfied, i.e., MTU_i is less than the maximum MTU of all higher priority messages, we need to evaluate MTU_i using (13) and (14).

$$MTU_i \leq \frac{-C_i^* \times O}{C_i^* + L(t)} \tag{13}$$

$$L(t) = M(t) + \frac{t}{EC} \times \max_{\forall m_j \in hp(m_i)} (MTU_j) \tag{14}$$

If the evaluated MTU_i from (13) satisfies the assumed condition, i.e., $\forall m_j \in hp(m_i) : MTU_i < max(MTU_j)$, then the solution is accepted, otherwise there is no solution to make the message schedulable.

Algorithm 1 shows an algorithm to find the MTU for all messages in order to make the system schedulable. As we have seen above, selecting MTU for a message always affects itself and the lower priority messages. Therefore, the algorithm starts from the highest priority message and it continues to the lowest priority messages. Algorithm 1 starts with calculating the MTU range for m_1 based on (11). As m_1 is assumed to be the highest priority message in the set, the set of MTUs (MTU_{hp}) from messages with priority higher than m_1 is set to zero. Moreover, for all messages the MTU_i is calculated when t is assigned to the deadline of the message, i.e., $t = D_i$.

Algorithm 1 MTU Assignment Algorithm

```
1: //Find the range of $MTU_1$ according to (11)
2: $t = D_1, MTU_{hp} = 0$
3: $sched = -1$
4: $MTU_1[hi,lo] = MTUcalc(t, MTU_{hp})$
5: $MTU_1[hi,lo] = CheckRange(MTU_1[hi,lo])$
6: //Change MTU to nP according to (9)
7: $nP[min,max] = translate(MTU_1[hi,lo])$
8: for $i = nP[min] \rightarrow nP[max]$ do
9:     for $m_j = m_2 \rightarrow m_N$ do
10:        $t = D_j, MTU_j = 0$
11:        //Find the range of $MTU_j$ according to (11), (13)
12:        $MTU_j[hi,lo] = MTUcalc(t, MTU_{hp}[hi])$
13:        $MTU_j[hi,lo] = CheckRange(MTU_j[hi,lo])$
14:        //If there is a solution, set the flag
15:        if $MTU_j[hi] > 0$ then
16:            $sched = 1$
17:            $update(MTU_{hp})$
18:        else
19:            $sched = -1$
20:            break
21:        end if
22:    end for
23:    //If there is a solution, no need to check other nP
24:    if $sched == 1$ then
25:        break
26:    end if
27: end for
28: return $MTU_j, sched$
```

The $[MTU_1]$ range is evaluated and the algorithm checks the range with the maximum possible protocol range of MTU, i.e., the range should be within $[MTU_{min}, MTU_{max}]$ (lines 4 and the following). Afterwards, we need to find the value within the evaluated MTU range such that all other lower priority messages are schedulable. Therefore, the algorithm iterates for all possible number of packets (only for m_1) from the evaluated range of the MTU_1 according to (9) (line 7) starting from the highest down to the lowest value of MTU. Given the value of MTU_1 the algorithm checks the schedulability of other messages starting from message m_2 the second highest priority to evaluate its MTU and then continue with the other messages.

In the message iteration, the algorithm calculates the range of the MTU_j for each message according to (11, 13) considering the highest evaluated MTU values of higher priority which is denoted by $MTU_{hp}[hi]$ (lines 12 and the following). This value is already computed for m_1 and will be calculated for other messages in the loop, then it is updated to be used later in the calculations of MTU for the other lower priority messages(line 17). For instance, in the iteration of m_3, the $MTU_{hp}[hi]$ includes both $MTU_2[hi]$ and MTU_1 which are calculated in the previous iterations. The loop continues for other messages unless the solution is not found with the evaluated values of the MTU_{hp}.

Whenever the algorithm does not find a range for at least one of the messages, it breaks the inner loop and it continues for the next number of packets in the outer loop (lines 18 and the following). Otherwise if the range is evaluated for all messages, the algorithm stops the outer iteration and returns the MTU_j for all messages (lines 24). When computing MTU_j the algorithm considers the highest evaluated values of the MTUs ($MTU_{hp}[hi]$) of the higher priority messages $m_2, ..m_{j-1}$. The reason for this is that the higher value of MTU requires lower number of packets which may lead to better response times of messages.

The complexity of the algorithm is $O(N \times nP)$ where nP is a function of MTU which is between $[MTU_{min}, MTU_{max}]$. Note that the presented algorithm is not an optimal algorithm and it is sufficient but not necessary meaning that if it does not find a solution for a system it does not mean than there is no solution for that system.

VII. Evaluation

In this section, we evaluate the improvements that can be achieved by the presented algorithm in terms of increasing the schedulability of messages and we compare the results of the algorithm with the results of using the maximum protocol's MTU (MTU_{max}) for all messages. The evaluation is carried out using four different simulation studies. In each study, the algorithm is applied on a number of randomly generated message sets given the following parameters as input to the message sets generation program. The range of the message period is defined $[T_i^{min}, T_i^{max}]$, the number of messages is denoted by N and the transmission time of the messages is selected within $[C_i^{min}, C_i^{max}]$.

For each study, 100000 message sets are randomly generated given the range of the above mentioned input parameters.

In all studies the following assumptions are made: the network capacity is set to $100 Mbps$, number of slave nodes in the network is 5, the elementary cycle duration is $EC = 1.5ms$, the protocol overhead is 44 bytes including Ethernet overhead and the FTT-SE overhead, and the Ethernet inter-frame gap is $96 bits$ which makes $O = 3.96\mu s$, and the switch latency is considered $\Delta = 5\mu s$. Finally, the minimum and maximum packet size is $[100, 1500]$ bytes. Moreover, in all studies only synchronous messages are considered and for each study 20 different synchronous window durations LSW are selected from; $LSW = [100, 1000]\mu s$ in steps of $50\mu s$, where the 100000 sets are tested for every value of LSW.

The different settings in each study are:

- **Study 1** is specified to have $N = 10$, $[T_i^{min}, T_i^{max}] = [2 \times EC, 50 \times EC]$ and $[C_i^{min}, C_i^{max}] = [150, 200]\mu s$.
- **Study 2** is done with the same parameters as Study 1, except the number of messages which is $N = 30$.
- **Study 3** is specified having similar parameters in Study 1, except that the message transmission times are increased within $[C_i^{min}, C_i^{max}] = [200, 500]\mu s$.
- **Study 4** is performed having the same range for transmission time of the messages in Study 3, but changing the range of the periods within $[T_i^{min}, T_i^{max}] = [5 \times EC, 80 \times EC]$ and $N = 30$.

In all studies we count the number of scheduled sets out of the 100000 randomly generated and using the MTU values evaluated from the proposed algorithm and the same study is repeated assuming the MTU_{max} for all messages.

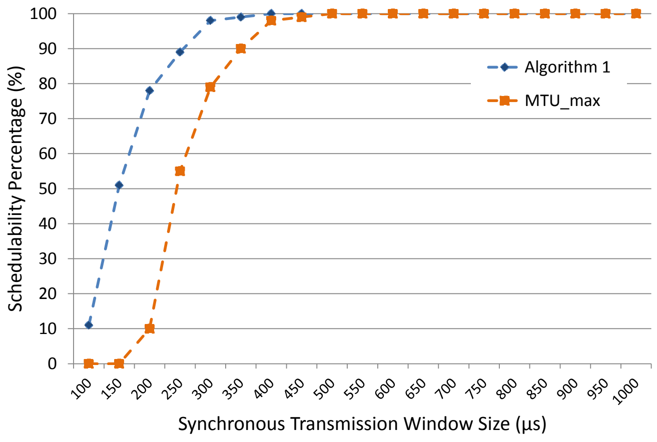


Fig. 4. The Result of Study 1

Figure 4 shows the percentage of schedulable systems in Study 1 as a function of LSW. It is clear from the figure that the results of using the proposed algorithm increase the schedulability of sets significantly compared with the case of assigning the maximum MTU. To guarantee the schedulability of all sets we need $LSW \geq 500\mu s$ for the case of maximum MTU while using the MTU from the proposed algorithm reduces the required length to $LSW \geq 400\mu s$.

In the second study, we changed the number of messages to 30 in each set to investigate the effect of the number of messages on the results. The results of Study 2 are depicted in Figure 5. The result illustrates that using the maximum MTU for the messages cannot reach to 100% meaning that the dedicated synchronous window is not enough. However, assigning MTUs according to the proposed algorithm, makes all sets schedulable even with $LSW \geq 850\mu s$. Note that increasing the number of messages in general requires more bandwidth to be dedicated to the messages to guarantee their schedulability since it increases the number of interfering messages for lower priority messages which is clear when comparing the results of Study 1 and Study 2.

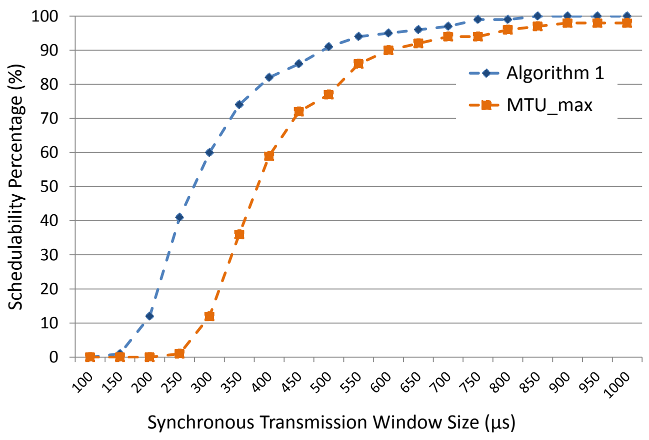


Fig. 5. The Result of Study 2

The results of Study 3 are presented in Figure 6 and it shows that increasing the transmission time of the messages highly affects the schedulability of the sets, yet setting the MTUs based on the presented algorithm makes 100% of the generated sets schedulable by using a synchronous window duration larger than $750\mu s$.

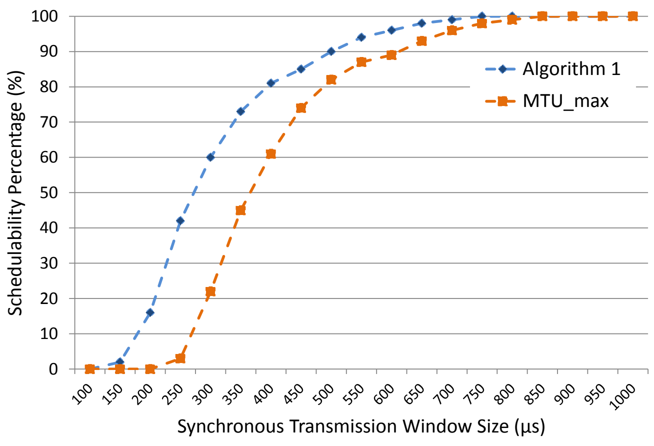


Fig. 6. The Result of Study 3

In the last study, we increase both the number of messages and the period of the generated messages. Figure 7 shows the result of Study 4. As explained previously, increasing the number of messages affects the percentage of schedulable sets. Also increasing the difference between the minimum and maximum periods of messages have the same negative effect on the schedulability since the shorter period messages may activate several times while scheduling the lower priority larger periods which increases the interference from higher priority messages. Applying the algorithm on the message sets makes 100% of the message sets schedulable when the dedicated synchronous window is more than $550\mu s$, whereas using MTU_{max} requires the synchronous window more than $750\mu s$.

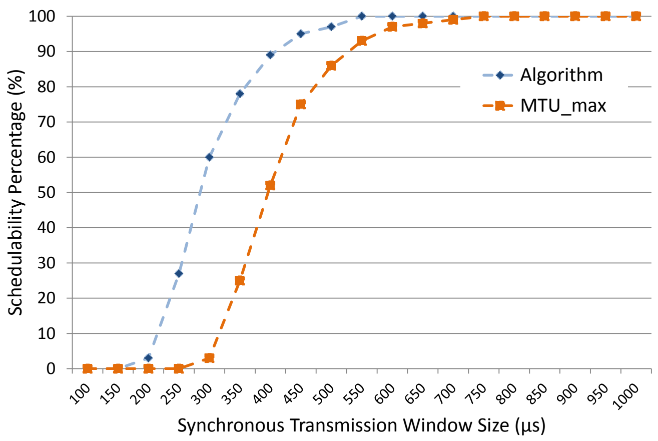


Fig. 7. The Result of Study 5

VIII. Conclusion and Future Work

In this paper, we proposed a heuristic algorithm to find a MTU for each message in order to increase the schedulability of the messages in the FTT-SE protocol. We evaluated the proposed algorithm using 4 different studies and we showed that assigning the MTU of the messages according to the proposed algorithm increases the percentage of schedulable sets compared with using the protocol maximum MTU for all messages. However, this algorithm does not evaluate the optimum MTU for the messages as the complexity is high. The future work aims at finding the MTU in the multi-hop FTT-SE protocol.

Acknowledgments

This work is supported by the Swedish Foundation for Strategic Research, via Mälardalen Real-time Research Center (MRTC) at Mälardalen University. Also, it is partially supported by FEDER through the COMPETE program, and by the Portuguese Government through FCT grant Serv-CPS PTDC/EEA-AUT/122362/2010.

References

[1] M. Behnam, R. Marau, and P. Pedreiras, "Analysis and optimization of the mtu in real-time communications over switched ethernet," in *16th IEEE International Conference on Emerging Technologies Factory Automation (ETFA'11)*, sept. 2011.

[2] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered ethernet (tte) design," in *8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, may 2005.

[3] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over cots ethernet switches," in *6th IEEE International Workshop on Factory Communication Systems (WFCS'06)*, June 2006.

[4] C. A. Kent and J. C. Mogul, "Fragmentation considered harmful," *SIGCOMM Comput. Commun. Rev.*, Jan. 1987.

[5] J. Chen, L. Gong, Y. Yang, and P. Zeng, "Average performance of packet network," in *6th International Conference on ITS Telecommunications Proceedings*, 2006.

[6] C. K. Kodikara, S. Worrall, and A. Kondoz, "Optimal settings of maximum transfer unit (mtu) for efficient wireless video communications," *IEE Proceedings of Communications*, 2005.

[7] M. Younis, O. Farrag, and W. D'Amico, "Packet size optimization for increased throughput in multi-level security wireless networks," in *IEEE Military Communications Conference (MILCOM'09)*, 2009.

[8] Y. Sankarasubramaniam, I. Akyildiz, and S. McLaughlin, "Energy efficiency based packet size optimization in wireless sensor networks," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, 2003.

[9] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo, "Preemption points placement for sporadic task sets," in *22nd Euromicro Conference on Real-Time Systems (ECRTS'10)*, 2010.

[10] R. Marau, P. Pedreiras, and L. Almeida, "Asynchronous traffic signaling over master-slave switched ethernet protocols," in *6th International Workshop on Real Time Networks (RTN'07)*, July 2007.

[11] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *24th IEEE International Real-Time Systems Symposium (RTSS'03)*, 2003.

Implementing Virtual Channels in Ethernet using Hierarchical Sporadic Servers

Zahid Iqbal, Luis Almeida
Instituto de Telecomunicações
University of Porto, Faculty of Engineering
Porto, Portugal
{dee11021,lda}@fe.up.pt

Moris Behnam
Mälardalen Real-Time Research Center
Mälardalen University
Västerås, Sweden
{moris.behnam}@mdh.se

***Abstract*—Composability is an important property to build complex applications. One important technique to achieve composability, particularly in the time domain, is multi-level hierarchical server-based design. However, composability must not only be supported among components sharing computing resources but also in their interactions. In this paper we implement hierarchical server-based traffic scheduling within the Flexible Time-Triggered Switched Ethernet (FTT-SE) protocol. The novelty in this work is the use of hierarchical sporadic servers in such setting. We report an efficient usage of the network bandwidth, short reponse times and the temporal isolation across servers.**

I. Introduction and Related Work

Limited forms of multilevel server-based scheduling, typically with two levels, have existed in the network domain for many years, to provide virtual channels and bound the impact of burstiness of certain types of traffic. Traffic shapers are one form of servers as they limit the amount of traffic that a node can submit to the network in a given time window. The shaper has one scheduling discipline associated to its arrival queue, typically FCFS, and another scheduling discipline manages its output at a global level. The leaky bucket is a common traffic server found frequently in networks that belongs to the category of shapers [1]. Network servers use techniques similar to CPU servers, based on capacity that is consumed and then eventually replenished. Different server policies exist that differ in the way the replenishment is done. For example, Polling Server, Periodic Server and the Deferrable Server offer periodic replenishments. However, network servers unlike CPU servers often lack clear or fixed dynamic priority management schemes due to a wide variety of Medium Access Control (MAC) protocols, e.g., mix of round-robin and first-come first-served scheduling, and multiple priority levels.

Considering Real-Time Ethernet (RTE) protocols, we can find some limited forms of reservations for specific traffic types. One class of RTE includes time-triggered frameworks that allow reserving fixed windows or slots for the transmission of different kinds of traffic, typically aperiodic traffic. Industrial examples of this kind include TTEthernet [2], Ethernet PowerLink [3] and PROFINET [4]. Such slots or windows are in fact polling or periodic servers on which the channel bandwidth is available on a periodic basis. The bandwidth allocated to these servers is exclusive, which means that if no traffic of the respective type is pending, the bandwidth cannot be used by other types of traffic and is, thus, wasted. Moreover, these servers use fixed slots within rigid cyclic frameworks, which impose a compromise between bandwidth and response time, i.e., low response times can only be achieved with high bandwidth requirements. Finally, these protocols do not allow either arbitrary server policies with hierarchical composition to support complex applications or the servers dynamic management.

Another class of RTE protocols follows the event-triggered paradigm without global synchronization [1], implementing traffic shapers at the end nodes. Probably the most flexible and widely available such approach is LinuxTC [5], which already supports arbitrary server policies and their hierarchical composition. However, the dynamic management of the servers requires a global management entity that is not covered by LinuxTC and this framework intrinsically allows capacity overruns corresponding to both the on-going packet transmissions and interference from the operating system in implementing certain servers policies, particularly aperiodic ones such as the Sporadic Server [6], degrading temporal isolation.

Finally, other RTE frameworks have explored including hierarchical aperiodic servers within the network switches. The work in [7] integrates a full hierarchical scheduling framework while that in [8] integrates two levels, for time and event-triggered traffic, respectively. These frameworks, however, require specific switches being, thus, less general solutions.

In this paper we follow the time-triggered approach using FTT-SE [9] which combines the flexibility of on-line traffic scheduling with the time-triggered model. A proof-of-concept implementation of servers within FTT-SE was carried out in [10], called Server-SE. However, it considered rather limited servers, particularly concerning their hierarchical composition. A full hierarchical server scheduling framework within FTT-SE was reported in [11] but using polling servers, which suffer from the referred coupling between bandwidth and response time.

In this paper we report the successful implementation of hierarchical server-based traffic scheduling in FTT-SE using sporadic servers, which decouple response times from the allocated bandwidth. Our specific contributions are:

- adaptation of the hierarchical server-based architecture to use the sporadic server model.
- reference implementation of sporadic servers in FTT-SE
- experimental verification of low response time and temporal isolation between contending applications.

The paper is organized as follows: the next section presents some background on FTT-SE and sporadic servers. Section III describes the implementation aspects of the hierarchical server-based scheduling in FTT-SE using sporadic servers. Section IV presents the experimental evaluation from our implementation showing its practical feasibility. Finally in Section V we conclude the paper and present some future work directions.

II. Background

This section briefly describes the real-time Ethernet protocol FTT-SE [9] as well as the operation of a typical sporadic server.

A. *FTT-SE Brief Overview*

FTT-SE [9] is a master/slave protocol for real-time communication on Ethernet that exploits the advantages brought by micro-segmentation on typical star-topologies, namely parallel forwarding paths and absence of collisions. The communication is organized in fixed duration slots called Elementary Cycles (ECs). The EC duration is a design-time parameter, tunable to best suit the application dynamics. Typical values range from 1 ms to tens of ms. Each EC starts with a *Trigger Message* (TM), issued by the master, which contains the schedule for that interval. The remaining nodes in the system receive the TM, decode it and transmit the messages indicated therein. Each EC is further divided in two windows, for synchronous and asynchronous traffic classes, respectively (Figure 1). The share reserved for the synchronous traffic is also a design-time fixed parameter. Typically, the master first schedules the synchronous traffic up to the synchronous window and only then schedules asynchronous one, using the remaining time in the EC. Windows overruns are not allowed by schedule construction.

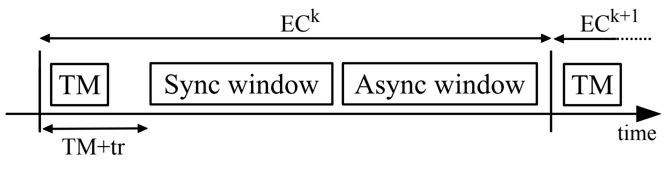


Fig. 1: The FTT-SE EC structure.

When scheduling, the master uses a special transmission time accountancy per link to make sure that the traffic indicated in each TM can be fully transmitted within the respective EC (see [12] for details). This feature makes sure that all switch queues are empty by the end of each EC and the traffic pattern at an EC scale follows the scheduling performed by the master.

For the asynchronous traffic, FTT-SE provides a signalling mechanism that aggregates transmission requests in each node during each EC and conveys them to the master in a specific minimum sized packet (see [13] for details). Once the asynchronous requests arrive at the master, from all nodes in parallel, they can be scheduled with any desired policy. The extra latency implied by this signalling mechanism ranges from 1 to 2 ECs.

In this work we will assume no synchronous traffic and we will use an hierarchical scheduling framework to schedule the asynchronous messages. Nevertheless, synchronous traffic can still be added without invalidating the work herein presented, simply by adding a synchronous window in each EC and reducing correspondingly the bandwidth available for the hierarchical scheduling framework.

B. *Sporadic Server*

A sporadic server can be represented with the model (C_{server}, T_{mit}) where C_{server} is the budget and T_{mit} is the minimum-inter-transmission time of the server. Consider a request for message m_1 that arrives at time $t_{arrival-m_1}$. The server sets a replenishment instant for itself that is T_{mit} after the message arrival time i.e., the next server replenishment shall take place at $t_{arrival-m_1}+T_{mit}$. Let us say that message requests k time units from the sporadic server. The server budget is decremented to $C_{server} - k$ units. At time $t_{arrival-m_1}+T_{mit}$, k time units are returned to the current server capacity. If a request for message m_2 arrives soon after, at time $t_{arrival-m_2}$ needing j units from the sporadic server, its capacity will be decremented to $(C_{server} - k) - j$ units, and the next server replenishment will take place at $t_{arrival-m_2} + T_{mit}$ for the consumed j units (Figure 2). Thus, a sporadic server enforces a bandwidth equal to C_{server}/T_{mit} and its worst-case impact on the rest of the system equals that of a periodic task with similar parameters.

III. Implementing Hierarchical Servers

We implement hierarchical sporadic servers as part of the FTT-SE scheduling algorithm. For implementation purposes, we need a mechansim that can manage the activations and capacity of servers in accordance with the particular server policy. For example, a polling server [14] must be activated and its capacity replenished periodically whereas a sporadic server follows a sporadic consumption/replenishment model; it has its successive activations at least *mit* time units apart and replenishment follows a more complicated model [15]. Next we describe, the important components of our implementation.

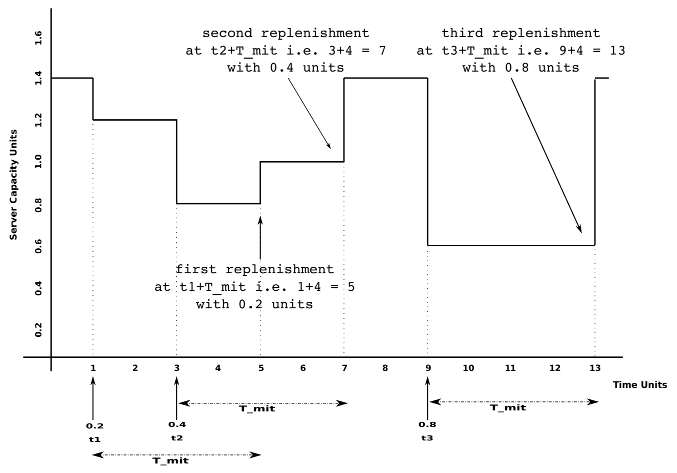


Fig. 2: The sporadic server with $C = 1.4$ time units and $T_{mit} = 4$ time units.

A. Hierarchical Scheduling Framework

With the Hierarchial Server-based Scheduling (HSS) framework, a set of servers is connected in a tree-like structure as shown in Figure 3. A server has an associated scheduler, a set of children servers and/or streams and an interface that specifies its resource requirements i.e. period and budget. A server can provide its budget to its child servers and/or streams. The streams are connected to the leaf servers of the hierarchy and represent the actual application load that will consume the network bandwidth. With sporadic servers, a message activation at the leaf server will trigger the scheduling process that begins at the root of the respective hierarchy; the root server selects one of its ready children servers. A parent server checks two conditions to select one of its child servers. For the first condition, it checks the remaining capacity and state of the server. A server that is in ACTIVE state and has non-zero remaining capacity is eligible (see section III-E). Following this, the child server with minimum period is chosen i.e we use RM scheduling policy. Other algorithms can be used such as such as Earliest Deadline First (EDF) or Fixed Priority Scheduling (FPS) [16]. Then the scheduled child server will select another child server and the same procedure will be repeated down the tree until the leaf server is reached which will finally schedule a message for transmission. The amount of bandwidth given to the scheduled stream is the minimum among the remaining capacities of all parent servers along the path from leaf to the root and the consumed bandwidth by the stream is discounted from the remaining capacity of all the servers along this path. If the remaining capacity of a server is exhausted, the server becomes suspended until its capacity is replenished.

1) Architectural Overview of Servers Integration in FTT-SE: The system consists of n nodes connected to a single switch with m full-duplex ports. The port input on the switch that receives streams from the connected station is called *uplink* and the port output that sends data to the station is called *downlink*. It is important to define the concept Independent Server Hierarchy (ISH). ISH manages all the streams that originate in the same source and are forwarded to the same destination. The master station prepares ISHs for a source station depending on the number of its destinations i.e. one ISH for each destination. Hence, there can be upto $n \times (n-1)$ ISHs in a system with n nodes. Figure 3 shows how the communcation of each station is managed with ISHs.

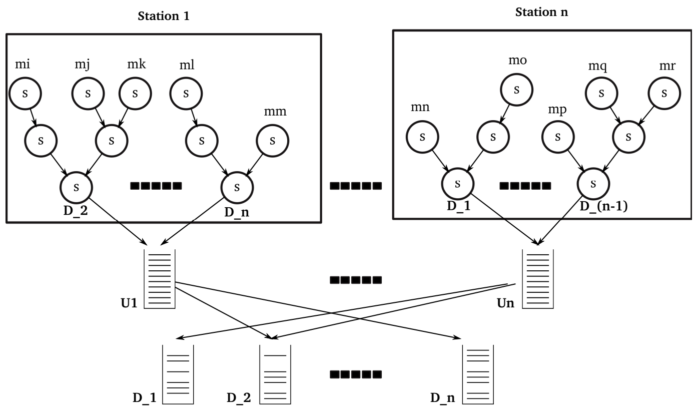


Fig. 3: Hierarchical Server Based Scheduling Architecture

B. Servers Repository

The servers repository is a database of existing servers in the system. For any server in the system, there is a corresponding entity in the repository that maintains a view of that server static and dynamic attributes. Static attributes include period, budget, id, level in the hierarchy and communication ends, whereas dynamic information includes its current capacity, finish status, state and overrun flag.

C. Servers and Streams Model

We use an HSS architecture to manage asynchronous traffic only. Consider N asynchronous streams (AS_x), modeled using sporadic real-time model (1), where C_x is the message transmission time of a stream AS_x instance, $Tmit_x$ represents the respective minimum interarrival time and D_x the deadline. A large message stream may generate several packets, which have a size between $Mmin_x$ and $Mmax_x$. P_x identifies the parent server, i.e, the server to which the stream is connected to and RT_x is its computed response time.

$$AS_x = (C_x, Tmit_x, Mmax_x, Mmin_x, P_x, RT_x, D_x) \qquad (1)$$

A server Srv_x is characterized in (2) by its capacity C_x, replenishment period T_x, deadline D_x and a few data extracted from the set of children components, either servers or streams, namely the maximum and minimum packet transmission times ($Mmax_x$ and $Mmin_x$, respectively). Moreover, the server Srv_x is associated with a parent server P_x and a corresponding computed upper bound response time RT_x. Despite the similarity between the characterization of servers and streams, there is a fundamental difference; streams would use the actual transmission time on the network, whereas servers merely characterize a reservation of the network resource.

$$Srv_x = (C_x, T_x, Mmax_x, Mmin_x, P_x, RT_x, D_x) \tag{2}$$

In the remainder of the paper we will refer to both streams and servers as components, in an integrated way.

D. Replenishment Management

An important component of the scheduling model is the management of servers replenishment that is done as follows.

We associate two queues to a server, a message queue and a replenishment queue. The message queue holds the application messages that must be handled by the server whereas the replenishment queue holds records for future replenishments of the server capacity. An entry in the replenishment queue is a pair of the type (C, t_{rep}), where C is the capacity of the associated request, and t_{rep} is the replenishment instant. t_{rep} is computed as: $t_{rep} = t + T_x$ where t is the EC in which the request arrives and T_x is the server Srv_x replenishment period. With this model in place, the scheduling algorithm for hierarchical sporadic servers works as follows.

At EC_i, we probe the replenishment queue for all servers. For each entry (C, t_{rep}) such that $t_{rep} == EC_i$ and we replenish the server with C units. At this instant, we update the state as ACTIVE for every server in the respective ISH that has non-zero capacity, and invoke the server-based scheduling from ISH root. This execution can schedule those messages that may have been left unscheduled in the previous runs on account of insufficient capacities either at the leaf server or an intermediate/root server level. With the replenished capacity it may be possible to schedule such messages.

E. Processing Message Arrivals

The FTT master uses a global queue (ART_S_QUEUE) to handle the ready asynchronous traffic and build the EC-schedules. Asynchronous messages that are not associated to servers go directly to the global queue when they become ready. In this case, FTT-SE uses a simple protection mechanism that enforces a sporadic arrival behavior [13]. When an asynchronous message associated to a server arrives, the respective ISH is processed to see if there is enough capacity. If so, the message is placed in the global queue. Else, the message is kept in the server ready message queue until enough capacity is replenished. The arrival of message stream activates the leaf server as well as the servers along the path from that leaf up to the root. The scheduling is then invoked from the root server. So the servers along the path become ready when there is a message activation at the leaf server.

Processing an ISH involves verifying all servers in the path from that leaf (note that all servers receiving messages are leafs) up to the root. The capacity requested by a message is discounted from the remaining capacities of all the servers along the path upto the root. Thus, servers act as a preliminary filter for the associated asynchronous traffic. An entry is set in the replenishment queue of all the servers along the path indicating the future replenishment instant and the capacity to be added.

F. Handling Message Packets

An important part of the sporadic server implementation deals with the transfer of message between server ready queues and (ART_S_QUEUE) global queue. Note that FTT-SE fragments large messages into small packets, which are individually scheduled and thus our servers also handle messages as bundles of packets that are then processed individually.

The following is the structure that represents a message item in the ready queues where *packets_no* represents the total number of packets in the message.

```
typedef struct ready_queue_idx{
        void *message_DB_pointer;
        void *rp;
        unsigned int packets_no;
        unsigned int to_be_served;
        struct ready_queue_idx *link;
}READY_QUEUE_IDX;
```

We say a packet needs scheduling when it is already inside the global queue. We say a packet needs to be served when it is still inside the server queue. The two fields namely *packets_no*, and *to_be_served* are used to denote the number of packets which are in one of the two states. Upon initialzation, *to_be_served* is the same as *packets_no* . But, when a packet can fit in the server capacities, we only update *to_be_served* and not the *packets_no*. When no more packets fit the server capacities, those that did (were served) are copied to the global ART_S_QUEUE.

Hence, a message maybe partially present in both the server queue and the ART_S_QUEUE at the same time. This happens when a message is partially served and is partially scheduled.

IV. Evaluation

Our experimental setup consists of one switch, a master station and three slave stations A, B, and C.

Beyond showing the feasibility of the hierarchical sporadic servers framework, we also aim at showing the superiority of these servers with respect to the polling servers used in [11] concerning their response times. Note that with sporadic servers messages can be immediately served whereas with a polling server a message may have to wait until the next period to receive capacity. Moreover, we also aim at verifying the temporal isolation capabilities of the sporadic servers. In our experiments, applications generate asynchronous traffic that is managed through sporadic servers only.

Station A contains two applications. These applications have distinct components and data to send to the other two stations. The applications are managed in the master node through an ISH where one ISH represents one application. Station B has two applications; one sending traffic to station A and the other to station C. Station C has only one application that generates data for station A. Figure 4 shows the experimental setup and ISHs that are prepared in the master station to manage the traffic generated by these applications. D_X refers to the consumer of traffic generated by the respective application; for example, in station A, D_B means that ISH will forward message $\{m_{21}, m_2, m_3\}$ to station B. The total number of messages in the system is 12 and 23 servers are used to manage these message transmissions. The parameters of servers and messages are given in Table I and Table II respectively.

We set the duration of $EC = 10ms$, and maximum packet transmission time is $Mmax_x = 88\mu s$. The length of asynchronous window is approximately $ASW = 50\%$ of the EC.

For faster reponse, we set server parameters empirically following simple rules e.g., capacity of the parent servers is more than the sum of their child servers. Capacity of each server is multiple of $88\mu s$ i.e., the packet transmission time. Normally, the capacity of each server is enough to schedule an instance of each of its children. This leads to short response times with periodic message activations. However, in the case of overload, the server capacities may not be enough for scheduling all jobs of a particular message stream. In this case, such messages remain in their queues until capacities become available. The server design problem, however, is orthogonal to the current work and it can be solved with techniques available in the literature [17].

A. Experiments

In our experiments, we measure the response time of the message set when messages are activated periodically i.e. every *mit*. We consider reponse time the duration in number of ECs measured between the EC in which a message request signal is received in the master node and the EC in which the message is dispatcehd by the master node. Also, we verify that temporal isolation is achieved among message streams that share the network bandwidth. In particular, we consider three cases a) temporal isolation between messages that belong to the same ISH b) temporal isolation between messages that belong to different applications but share the same source node and c) temporal isolation between message streams that belong to different ISHs but share the destination.

We show here the results of case "b" above and take station B that has two ISHs $U_B\_D_C : \{m_{11}, m_{12}, m_{13}\}$ and $U_B\_D_A : \{m_{14}, m_{15}\}$. We make m_{11} and m_{13} bursty and verify that $\{m_{14}, m_{15}\}$ sharing the same source node are unaffected (Figure 5). m_{13}, for example, has longer response times despite being in a separate branch of the hierarchy. The reason is that its server s_{22} has a period of 16 EC while the server s_{22} at the same level has a period of 8 EC. With RM policy s_{21} is favored over s_{22}. Also, s_{21} has a bursty stream connected to its child server s_{23}. This configuration has the effect of consuming most of the root server budget and resulting in longer response times for m_{13}. However, the periodic message m_{12} in the same ISH is unaffected. In particular, all the messages with periodic arrivals have short response time between 1 and 2 EC. This is one order of magnitude improvement with respect to the the polling server (experiments shown in [11]).

V. Conclusions and Future Work

In this paper, we have shown that multilevel hierarchial server-based architecture with different server policies can be implemented within a master-slave real-time Ethernet protocol as FTT-SE. We made an implementation of hierarchical sporadic servers to be used within the scheduling model of FTT-SE. Our results show that temporal isolation is achieved between message streams that are sharing the network bandwidth. Furthermore, we report shorter response times than with the polling server policy. As future work we plan to demonstrate the effectiveness of this approach in a complex real case study. Moreover, we plan to analyse the scalability of this framework as the number of messages/servers increases.

Acknowledgement

This work was partially supported by the Portuguese Government through FCT grant PTDC/EEI-TEL/3006/2012 - CodeStream.

References

[1] J. Löser and H. Härtig, "Low-Latency Hard Real-Time Communication over Switched Ethernet," in *Proc. of the 16th EUROMICRO Conference on Real-Time Systems (ECRTS'04)*. IEEE Computer Society, Jul. 2004, pp. 13–22.

[2] TTTech, "TTEthernet," http://www.tttech.com/technologies/ttethernet/, November 2008.

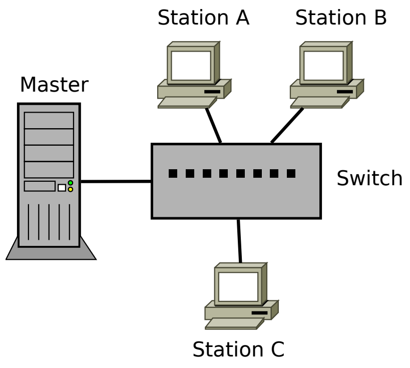
[3] "Ethernet Powerlink protocol," http://www.ethernet-powerlink.org, 2008.

[4] "Real-time PROFINET IRT," http://www.profibus.com.

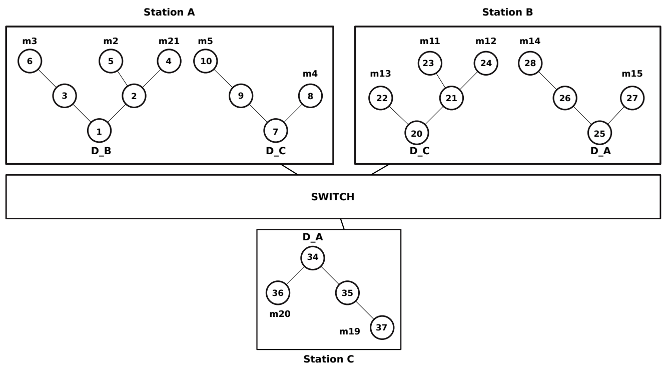
[5] M. A. Brown, "Linux Traffic Control," http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html.

[6] M. J. Stanovich, T. P. Baker, and A.-I. A. Wang, "Experience with sporadic server scheduling in linux: Theory vs. practice," in *Real-Time Linux Workshop*, vol. 2011, 2011.

[7] R. Santos, A. Vieria, R. Marau, P. Pedreiras, A. Oliveira, L. Almeida, and T. Nolte, "Implementing server-based communication within ethernet switches," in *Proceedings of the 2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'09) in conjunction with the 30th IEEE International Real-Time Systems Symposium (RTSS'09)*, December 2009.

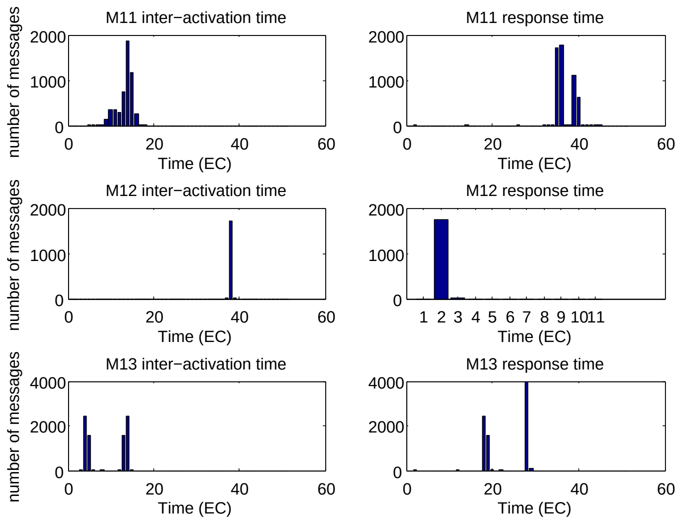


(a) The experimental setup

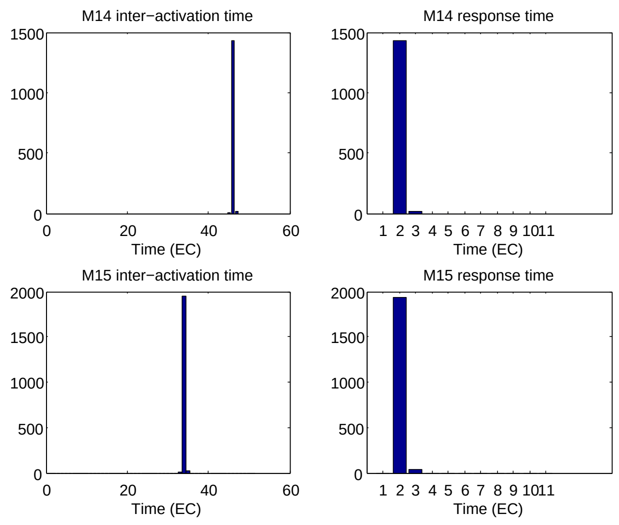


(b) Independent Server Hierarchies (ISHs)

Fig. 4: The experimental setup showing the arrangement of ISHs at the uplinks of slave stations



(a) m_{11}, m_{12} and m_{13}



(b) m_{14} and m_{15}

Fig. 5: Bursty activations in m_{11} and m_{13} do not impact the response time of other messages in the source station i.e. station B

[8] G. Carvajal, M. Figueroa, R. Trausmuth, and S. Fischmeister, "Atacama: An open fpga-based platform for mixed-criticality communication in multi-segmented ethernet networks," in *Proc. of the 21st IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2013.

[9] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over COTS Ethernet switches," in *Proc. of 6th Int. Workshop on Factory Communication Systems (WFCS'06)*. Torino, Italy: IEEE, 27 Jun. 2006, pp. 295–302.

[10] R. Marau, N. Figueiredo, R. Santos, P. Pedreiras, L. Almeida, and T. Nolte, "Server-based Real-Time Communications on Switched Ethernet," in *1st Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'08-RTSS'08)*, Nov. 2008.

[11] Z. Iqbal, L. Almeida, R. Marau, M. Behnam, and T. Nolte, "Implementing hierarchical scheduling on cots ethernet switches using a master/slave approach," in *SIES'12*, 2012, pp. 76–84.

[12] R. Marau, L. Almeida, K. Lakshmanan, and R. Rajkumar, "Utilization-based Schedulability Analysis for Switched Ethernet aiming Dynamic QoS Management," in *The 15th IEEE Conf. on Emerging Technologies and Factory Automation (ETFA'10)*, Sep. 2010.

[13] R. Marau, P. Pedreiras, and L. Almeida, "Signaling asynchronous traffic over a Master-Slave Switched Ethernet protocol," in *Proc. on the 6th Int. Workshop on Real Time Networks (RTN'07)*, Pisa, Italy, 2 Jul. 2007.

[14] J. Strosnider, J. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments," *Computers, IEEE Transactions on*, vol. 44, no. 1, pp. 73 –91, jan 1995.

[15] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Real-Time Systems*, vol. 1, no. 1, pp. 27–60, 1989. [Online]. Available: http://dx.doi.org/10.1007/BF02341920

[16] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*. IEEE, 2003, pp. 2–13.

[17] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analysis and server design," in *Proceedings of the 4th ACM international conference on Embedded software*. ACM, 2004, pp. 95–103.

Station	ISH	Id(x)	$C_x(\mu\text{s})$	T_x (EC)
Station A	U_A-D_B	1	1998	4
		2	1470	8
		3	440	10
		4	440	16
		5	778	17
		6	352	20
	U_A-D_C	7	1682	6
		8	528	15
		9	902	12
		10	714	23
Station B	U_B-D_C	20	2024	4
		21	1936	8
		22	880	16
		23	792	16
		24	704	18
	U_B-D_A	25	1410	5
		26	880	10
		27	704	16
		28	704	22
Station C	U_C-D_A	34	1848	6
		35	880	12
		36	704	13
		37	704	24

TABLE I: Server parameters for stations

ISH	Id(i)	C_i (μs)	T_i (EC)
U_A-D_C	m_{21}	528	50
	m_2	440	40
	m_3	176	45
U_A-D_D	m_4	352	35
	m_5	528	47
U_C-D_D	m_{11}	616	40
	m_{12}	440	37
	m_{13}	440	35
U_C-D_A	m_{14}	440	45
	m_{15}	528	33
U_D-D_A	m_{19}	528	28
	m_{20}	440	50

TABLE II: Message parameters