Atacama: An Open FPGA-based Platform for Mixed-Criticality Communication in Multi-Segmented Ethernet Networks

Gonzalo Carvajal, Miguel Figueroa
Dept. of Electrical Eng. and
Center for Optics and Photonics
Universidad de Concepcion
Concepcion, Chile
{*gcarvaja, mifiguer*}*@udec.cl*

Robert Trausmuth
Dept. of Embedded Systems
UAS Technikum Wien
Vienna, Austria
robert.trausmuth@fhwn.ac.at

Sebastian Fischmeister
Dept. of Electrical and Computer Eng.
University of Waterloo
Waterloo, Canada
sfischme@uwaterloo.ca

***Abstract*—Ethernet is widely recognized as an attractive networking technology for modern distributed real-time systems. However, standard Ethernet components require specific modifications and hardware support to provide strict latency guarantees necessary for safety-critical applications. Although this is a well-stated fact, the design of hardware components for real-time communication remains mostly unexplored. This becomes evident from the few solutions reporting prototypes and experimental validation, which hinders the consolidation of Ethernet in real-world distributed applications.**

This paper presents *Atacama*, the first open-source framework based on reconfigurable hardware for mixed-criticality communication in multi-segmented Ethernet networks. *Atacama* uses specialized modules for time-triggered communication of real-time data, which seamlessly integrate with a standard infrastructure using regular best-effort traffic. *Atacama* enables low and highly predictable communication latency on multi-segmented 1Gbps networks, easy optimization of devices for specific application scenarios, and rapid prototyping of new protocol characteristics. Researchers can use the open-source design to verify our results and build upon the framework, which aims to accelerate the development, validation, and adoption of Ethernet-based solutions in real-time applications.

***Keywords*-Real-Time Ethernet; safety-critical systems; open-source hardware design; multi-segmented networks;**

I. Introduction

For more than a decade, researchers have targeted Ethernet as the natural replacement of legacy fieldbuses [1] in modern distributed applications. Low-cost, high-speed, and easy integration with existing networking infrastructure, are appealing characteristics of this technology. However, standard Ethernet is unable to provide hard latency guarantees required for distributed safety-critical applications found in avionics, automobiles, industrial control, etc. [2]. Thus, industry and academy started to explore mechanisms to provide latency guarantees on top of Ethernet, commonly referred as Real-Time Ethernet (RTE). The literature and recent developments in the area reveal two key points [2]–[5]: (1) Ethernet is by definition a best-effort protocol with a competitive approach, and then even high-end Commercial-Off-The-Shelf (COTS) devices are intrinsically unable to provide strict latency guarantees, and (2) Ethernet devices require specific modifications and hardware support to fit distributed safety-critical applications.

Within the real-time systems community, the perceived high-cost of deploying hardware components hinders the experimental validation of proposed enhancements for RTE. At the same time, due to the general misconception that high-speed and high-throughput devices (network processors, gigabit switching fabrics, etc.) make timming analysis unnecessary [6], hardware designers tend to overlook strict real-time requirements. The few available commercial solutions for RTE [7]–[9] are tailored for specific application domains, offer limited integration with COTS devices, or are closed to the research community. These characteristics make them hard to extend and adapt to the always evolving requirements of modern distributed applications.

This work introduces *Atacama*, the first framework that uses reconfigurable hardware and a fully open design to encourage and accelerate the development, validation, extension, and adoption of RTE technology in safety-critical applications. *Atacama* uses a time-triggered approach [2], integrating an Application-Specific Instruction-set Processor (ASIP) that coordinates the exchange of time-sensitive data on each station executing real-time tasks. In addition, a custom forwarding path provides low and predictable propagation latency across multiple switches. The modules can be seamlessly integrated and co-exist with COTS devices using best-effort traffic, enabling mixed-criticality communication.

The main contributions of this paper are two-fold: (1) it consolidates and expands early concepts [10], [11] into the first fully-functional open-source RTE framework based on reconfigurable hardware supporting dynamic segmentation of real-time domains, with an experimentally validated latency model for real-time frames; and (2) it discusses the design choices and implementation details to enable easy validation, optimization, and rapid prototyping of the framework for specific applications. The rest of the document is organized as follows: Section II details a novel architecture for the original ASIP described in [10]. Section III introduces the custom

switching for real-time frames, which includes mechanisms for topology discovery and dynamic segmentation of large real-time domains, which is an unexplored characteristic among available RTE solutions. Section IV presents an experimental characterization of the implemented prototypes, including robustness and timing analysis for accurate modeling of the communication latency in different topologies. Finally, Section V summarizes and concludes the paper.

II. Traffic Coordination in Real-time Stations

Atacama uses the Network Code framework [12] to coordinate the communication between distributed tasks. Network Code introduces three components:

- A domain-specific instruction set to describe dynamic Time-Division Multiple Access (TDMA) arbitration
- A compiler with a verification engine that translates the programs into conflict-free executable schedules
- The entity that executes the schedules at runtime

This section describes the architecture of the execution entity, which uses an adapted subset of the Network Code instruction-set to coordinate the exchange of time-sensitive data using Ethernet infrastructure [10].

A. The Network Code ASIP

Fig. 1 shows the general architecture of the Network Code ASIP (NC-ASIP), which provides a programmable communication layer to coordinate the exchange of real-time data between tasks running on distributed stations. Tasks access real-time data by periodically accessing specific locations of the dual-port DATA-MEM block. The NC-ASIP provides a dedicated path to move data between the DATA-MEM and the Ethernet interface at specific points in time, enabling predictable processing times and preventing competition between frames containing real-time data (real-time frames). The Ethernet interface includes an arbiter to separate real-time from best-effort frames, enabling integration with traditional communication layers operating with standard best-effort traffic, which offer no delivery guarantees.

In the NC-ASIP, each word on the CFG-MEM block defines a data buffer through an initial address and length mapped to locations on the DATA-MEM. The PROG memory contains a schedule describing anisochronous communication rounds that move data between the DATA-MEM and the medium. Safety-critical systems typically define all their communication requirements and data structures in advance, and the designer must fill and verify both PROG and CFG-MEM blocks before runtime. All memory blocks have 32-bits data buses. The instructions use a 32-bit representation, and the data uses 32-bit words.

The NC-ASIP implements each instruction as a finite state machine (instruction block). The controller fetches and decodes the instructions from the PROG memory, triggering execution of the corresponding block, which enable control of data flow, timing, and execution flow.

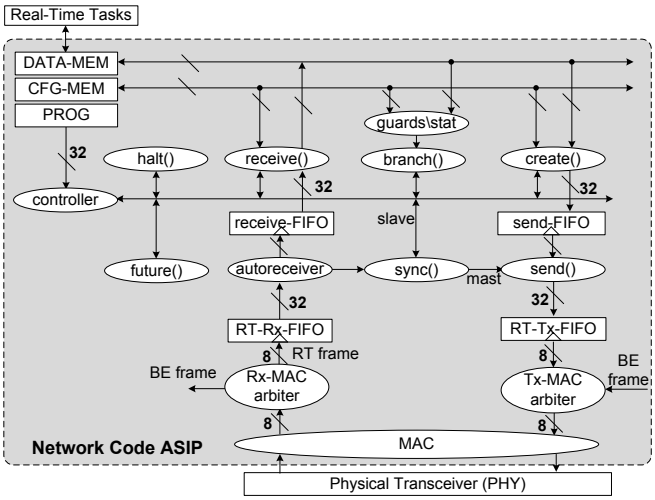


Figure 1: Architecture of the Network Code ASIP

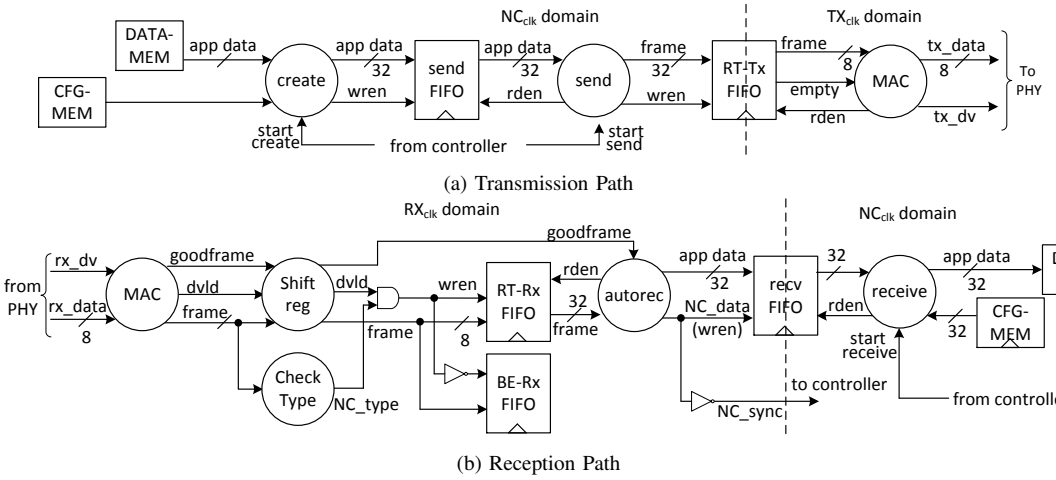
B. Data Flow Control

Data flow instructions move data between the DATA-MEM and the physical medium. The following paragraphs describe the transmission and reception paths as shown in Fig. 2.

1) Transmission path: Fig. 2a shows the path that generates real-time frames. When reading the instruction create(A), the controller triggers the create block, which reads address A of the CFG-MEM, maps the read location to the DATA-MEM, and moves the corresponding data to the send-FIFO. When reading the instruction send(channel, TTL), the controller triggers the send block, which takes the data stored in the send-FIFO, encapsulates it into an Ethernet frame, and stores it in the RT-Tx FIFO. Fig. 3 shows the format of a real-time frame, which have a special value in EthType field (NC-DATA for data frames), and the payload includes information such as the TTL (maximum number of switches in the path), the data length v_l (in 32-bit words), the logical channel (used to separate different data flows in the receivers), the ID of the origin buffer, and a sequential counter. The send block adds a pad in case the data is not large enough to comply with the minimum frame size of 64 bytes. The MAC starts transmitting the frame as soon as the RT-Tx FIFO asserts its non-empty flag, adding the Frame Check Sequence (FCS) at the end. The send block can also generate minimum sized frames tagged as NC-SYNC type, which are used for synchronization purposes.

The transmission path operates in two clock domains: $\mathsf{NC}_{\mathrm{clk}}$ driving the logic for instructions, and $\mathsf{Tx}_{\mathrm{clk}}$ driving the MAC and PHY logic. Because of the wider data buses, $\mathsf{NC}_{\mathrm{clk}}$ can run at a fraction of the frequency of the Ethernet logic and still achieve line speed on the physical links.

2) Reception path: Fig. 2b shows the path to retrieve data from the medium. The input logic uses a shift register to delay the incoming frame until checking its EthType field to separate real-time from best-effort frames. The RT-Rx



(a) Transmission Path

(b) Reception Path

Figure 2: Block Diagram for Transmission and Reception Path in the NC-ASIP

FIFO retains real-time frames until the goodframe signal from the MAC triggers the autoreceive block, which parses the frame fields to execute predefined actions. For frames tagged as NC-DATA, the block extracts the application data and stores it in the corresponding receive-FIFO (there is one FIFO per channel). When reading a receive(channel, A) instruction, the controller triggers the receive block, which reads the specified channel and stores the data on the location of DATA-MEM specified for variable A. If the schedule does not execute a receive() instruction, the data stays in the FIFO until a new frame arrives. For frames tagged as NC-SYNC, the autoreceive block advertises the event to the controller without further processing.

C. Timing Control

Timing control instructions enable the definition of periodical communication rounds to coordinate the exchange of real-time data using TDMA arbitration. Network Code defines a Reference Broadcast Synchronization (RBS) [13] approach to maintain a common sense of time across distributed stations. The network contains a single master station that generates a synchronization frame to signal the start of communication rounds. All other stations are slaves that start executing their local actions for the round (transmitting or retrieving data at specific times) only after receiving a synchronization frame. For conflict-free operation, the schedules must consider a worst-case value for the time span since the transmitter emits a frame until all the receivers processes it.

The instruction sync(mode, dl) implements the RBS scheme using two operation modes: (1) in master mode, the controller triggers the send block to emit a synchronization frame (NC-SYNC type without application data), and (2) in slave mode, the controller will stall the execution of the next instructions until either the arrival of a synchronization frame, or the expiration of a countdown timer with initial value dl.

The instruction future(L, dl) starts a countdown timer with initial value dl. The halt() instruction stalls the program waiting for the expiration of this timer, and then resumes execution at label L. These instructions enable the assignment of time slots to coordinate the communication tasks.

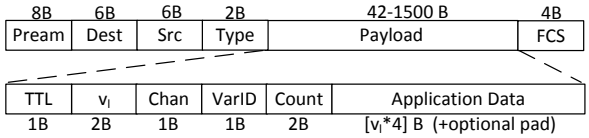


Figure 3: Real-time Frame Format

D. Execution Flow Control

The branch(guard, L) instruction enables the implementation of dynamic schedules that make on-the-fly decisions based on guards. Guards check for particular conditions based on values of buffers, execution history, flags, etc. If the evaluated guard returns TRUE, the schedule will continue execution at the specified label L; otherwise, the schedule will continue with the next instruction. For example, a station can stop transmitting data when a certain variable lies below a predefined threshold.

E. Hardware implementation

We accelerate the execution time of schedules by mapping each instruction to independent hardware resources and enabling concurrent execution. Table I summarizes the dependencies between instructions in the NC-ASIP. For two sequential instructions x()-y(), x() specifies the column, y() the row, and the characters have the following meaning:

- c : instructions can run concurrently
- b : instructions share a resource, and then the second one waits until the first one releases the shared resource (see Fig. 1)
- w : control dependency. These are related to branch, halt, and sync(slave) instructions, since they must wait

Table I: Summary of instruction dependencies

	create	send	receive	sync(m)	sync(s)	halt	future	branch
create	w	c	b	c	w	w	c	w
send	c	w	c	w	w	w	c	w
receive	b	c	w	c	w	w	c	w
sync(m)	c	w	c	w	w	w	c	w
sync(s)	c	c	c	c	w	w	c	w
halt	c	c	c	c	w	w	c	w
future	c	c	c	c	w	w	w	w
branch	b	c	b	c	w	w	c	w

Sensor

```
1 L0: future (10,L0)
      create (A)
3     send   (ch0)
      halt   ()
```

Logger

```
  L1: future  (L1, 10)
2     receive (A, ch0)
      halt    ()
```

Figure 4: Example schedules

for specific conditions to decide when and/or where to continue the program execution.

The controller checks the running state (idle/busy) of each block to calculate the locking conditions during the decoding phase of the instructions. After triggering a block, the controller fetches and decodes the next instruction. If Table I permits concurrent execution, the controller will trigger the second block. Otherwise, it will stop fetching new instructions until the lock is resolved.

The modular design also allow the designer to take advantage of static reconfiguration to optimize the use of resources and increase the reliability of the network. The designer can optimize the architecture according to the particular schedule on each station. For instance, Fig. 4 shows an example schedule for periodical exchange of data between a sensor and a data logger. For simplicity, we assume that the devices are synchronized. Since the traffic is unidirectional, we can simply remove the reception path from the sensor and the transmission path from the data logger. This optimizes the use of resources and power consumption in the network interfaces, which is beneficial for embedded systems. This also helps on keeping the integrity of the real-time network, since it prevents the interference of unplanned frames from faulty units or design glitches in receiver-only units.

III. Dedicated Real-time Forwarding Path

This section introduces a custom forwarding path for real-time frames in the switches. Fig. 5 shows a general diagram of an enhanced switch. The modules in gray implement a *cut-through* broadcast path for real-time frames generated from NC-ASIPs, which resembles the classical bus topology used in most legacy fieldbuses [1]. This path naturally spans across multiple switches, and provides functionalities such as discovery of real-time capable devices in the path

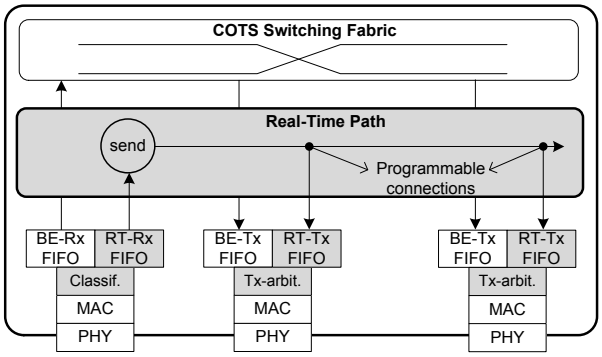


Figure 5: Enhanced Ethernet Switch

(NC-ASIPs and switches), and logical segmentation of real-time domains in large networks. Unlike other switching architectures tailored for time-triggered communication [11], [14], the proposed path omits previous configuration steps.

A. Classifying and Forwarding Real-time frames

Fig. 6 shows a block diagram of the dedicated path for real-time frames. A classifier on each input port checks the EthType and TTL of arriving frames. The classifier stores real-time frames (tagged as NC-DATA or NC-SYNC) with TTL≥ 1 in the dedicated RT-Rx-FIFO, and automatically discards frames with a TTL$= 0$. As soon as the RT-Rx-FIFO asserts its non-empty flag, the send block will forward the frame to the dedicated RT-Tx buffer in all the other ports (*cut-through* forwarding) decrementing the TTL by one. Considering the broadcast nature and that the schedules running in the NC-ASIPs prevent competition between real-time frames, the dedicated path omits any address processing and queuing mechanisms to handle contention. This is an essential characteristic to reduce the latency and jitter.

Best-effort frames follow the traditional path through the COTS switching fabric. Real-time frames receive strict priority access to the output ports. Whenever the RT-Tx-FIFO has data to transmit, the Tx-arbiter automatically blocks access to the output port from the COTS fabric. To account for potentially interrupted best-effort transmissions, the arbiter waits the time required to transmit a minimum sized frame and the Inter-Frame Gap (IFG), and then starts reading the data from the RT-Tx-FIFO. Once emptying the real-time FIFO, the arbiter waits again for the IFG before giving access to the output port back to the best-effort FIFO. As a result, stations using COTS components communicate transparently, but their available bandwidth will be reduced according to the amount of real-time traffic flowing through the network. We assume that higher layers deal with corrupted best-effort frames due to interrupted transmissions.

B. Cyclic Topology Discovery

Stations using COTS Network Interface Card (NIC)s are unable to interpret the format of real-time frames generated

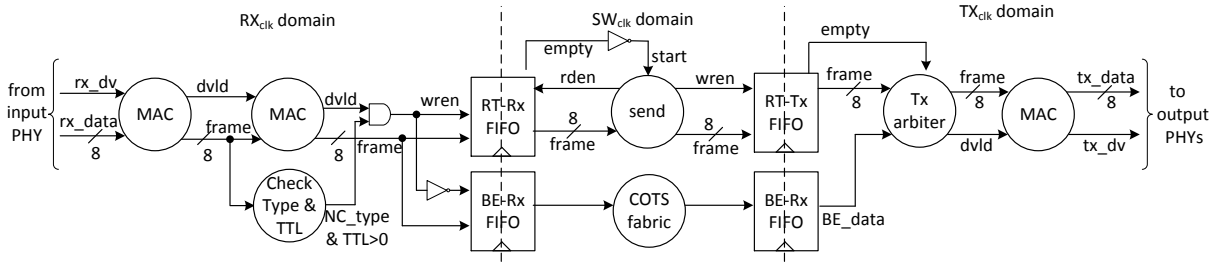


Figure 6: Real-time Path Inside the Switch

from NC-ASIPs, and thus it is necessary to prevent the propagation of real-time frames to these stations.

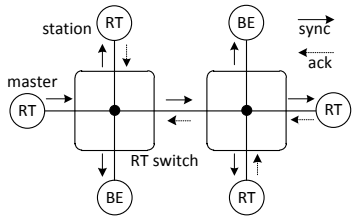
Fig. 7 illustrates a mechanism for discovering real-time capable devices during the propagation of synchronization frames at the start of communication rounds. Switches always broadcast synchronization frames to all the output ports. Real-time capable devices (either NC-ASIPs or switches) automatically respond to synchronization frames using acknowledge frames tagged as NC-ACK type (Fig. 7a). The switch will assert the programmable connection between the real-time path and the ports that received either a synchronization or an acknowledge frame from external sources. Input ports use dedicated logic to locally process acknowledge frames on-the-fly while the synchronization frame propagates, without storing or propagating them. After the synchronization slot, the switch will forward real-time data frames only to the ports attached to real-time capable devices (Fig. 7b). Consequently stations using COTS NICs will only suffer from blocking and interruptions during the synchronization slots, and remain unaffected during the exchange of real-time data.

The switch updates the connections on each synchronization slot. Providing an adequate upper bound for the latency along the entire network, the designer can plan the schedules without knowing the specific topology, and location of real-time stations can change at runtime.

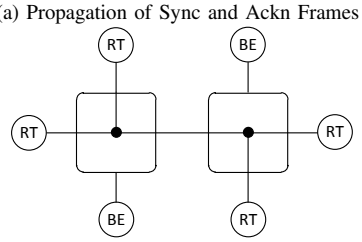
C. Real-time Ethernet Segments

A real-time segment contains stations that synchronize to a single master to exchange real-time data. A data subsegment is a subset of devices within a real-time segment exchanging data between them, without propagating frames to devices outside the subsegment. Fig. 8 shows two examples of segmentation of real-time stations in a large network.

Fig. 8a splits the network into two independent real-time segments, each one broadcasting data within its own domain. The designer can specify real-time domains by simply tapping a link with a filter/gateway to block real-time frames and prevent their propagation from one segment to another. It is also possible to accomplish the same behavior using a COTS managed switch for blocking specific frames. This configuration enable easy segmentation of real-time stations, without affecting the propagation of best-effort frames.



(a) Propagation of Sync and Ackn Frames



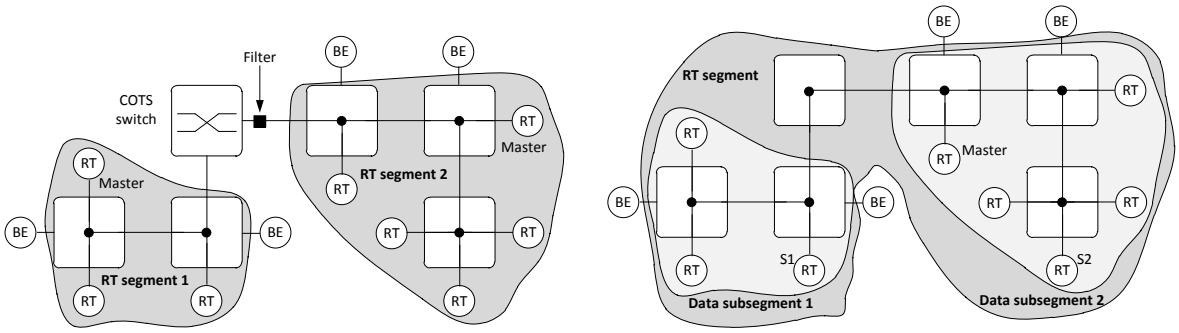
(b) Real-time Path After Sync Slot

Figure 7: Discovery of Real-time Capable Devices

Fig. 8b shows a logical division of a real-time segment into multiple data subsegments. In this case, all switches are real-time capable, and the logical separation is based on the TTL value of real-time frames. Frames from stations S_1 and S_2 are only useful within their corresponding subsegments. We can implement logical isolation of data frames between subsegments by using a TTL=2 and TTL=3 for frames generated from S_1 and S_2, respectively. Synchronization frames from the master must have a TTL≥ 4 to reach all the stations in the segment. This approach enables concurrent transmission of data in different subsegments during the same time slot, increasing the effective network utilization. However, it requires careful planning of the schedules considering the specific topology of real-time stations.

IV. Experimental Results

This section summarizes the experimental characterization of prototypes implemented on the NetFPGA platform [15]. The prototypes include an evaluation platform that holds up to four independent instances of the NC-ASIP, and a four-port enhanced switch that integrates the dedicated real-time path to the COTS switch design available for the NetFPGA. Both devices can operate at line rate over 1Gbps links.



(a) Independent Real-time Segments

(b) Synchronized Data Dubsegments

Figure 8: Logical Real-time Segments Inside a Larger Ethernet Network

Table II: Device Utilization Summary on Virtex2 Chip

Device	FFs	4-in LUTS	BRAMs
NC-ASIP (full)	4.3%	7.8%	7.8%
NC-ASIP (no-tx)	3.4%	6.4%	7.3%
NC-ASIP (no-rx)	2.9%	6.0%	6.0%
Reference COTS Switch	25%	40.5%	48.3%
Custom switch	26%	40.7%	59.3%

A. Device Utilization

Table II summarizes the device utilization for each prototype. The percentages relate to a total of 47 232 registers, 47 232 four-input LUTs, and 232 16Kb Block RAMs available in the Virtex2 chip included in the NetFPGA.

The table shows the utilization of a full instance of the NC-ASIP as described in Section II-A, and alternative implementations that remove the instruction blocks for transmission and reception. As stated in Section II-E, reconfigurable hardware allows the designer to fit the architecture according to the particular schedules. For instance, removing the reception instructions for a simple sensor that only needs to transmit data saves around 20% of logical resources in relation to the full implementation. We aim to develope an integrated compilation tool to generate the schedules and the matching architecture for each station from high-level specifications.

The custom switch retains all the functionality of the COTS switch, with the additional modules for the real-time path in all the ports. Since the MAC interface is already available in the COTS device, the logic for the dedicated path has a negligible impact in the total utilization. The higher cost relates to the memory required for the additional FIFOs. This shows that the we can integrate the real-time path as a default characteristic to COTS switches at a very low cost, and only real-time stations must assume the cost of specialized NICs.

B. Execution Time of Schedules

Fig. 9 shows the measured execution time for the create, send, receive, and branch instructions as a function of v_l. The graph reports the number of clock cycles since the controller

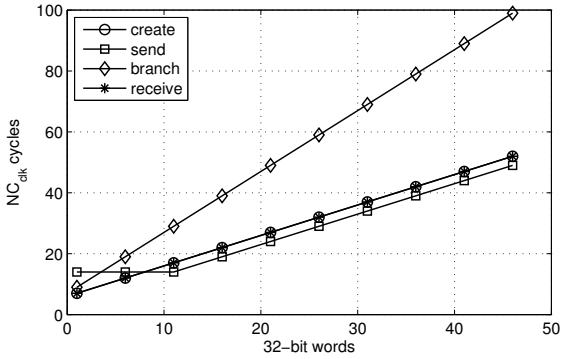


Figure 9: Execution Time for NC Instructions

triggers the corresponding instruction block until it returns to the idle state. For the branch instruction, the experiment considers the longest execution time corresponding to the comparison of two variables of length v_l. As we see, the execution time grows linearly as a function of the variable length. Because the minimum size of the frames is 64 B, send has a constant execution time for variables shorter than 11 words. The measured time for each case remained constant through 50 samples, indicating that the time for moving data between the MAC and the data buffers is predictable, removing the variability of processing time in the end-stations. This is a relevant characteristic for efficient TDMA arbitration. Equivalent implementations using software stacks lead to pessimistic worst-case scenarios that limit the efficient utilization of Ethernet infrastructure [12], [16].

C. End-to-end Latency

We define the end-to-end latency (EEL) for a real-time frame as the time required to move a byte from a data buffer in the origin to the receive-FIFO in the destination. The EEL between two stations with N_S switches in the path is:

$$\text{EEL} = \text{Tx}_{\text{ASIP}} + \text{Rx}_{\text{ASIP}} + N_S\text{Sw} + (N_S + 1)\text{L} \quad (1)$$

where

Table III: Maximum Observed Latency

	Max. latency [125MHz clock pulses]
Tx_{ASIP}	60
Sw	148
Rx_{ASIP}	94, for $v_l \leq 11$ 94 + 4(v_l-11), for $v_l > 11$
L	48

- Tx_{ASIP}: defined for a create-send sequence, without further instructions in the middle. Time span since triggering the create block until the first byte of application data leaves the MAC.
- Sw: switch forwarding time since the first byte of application data arrives to the MAC in the input port until it leaves the MAC in the output port.
- Rx_{ASIP}: time span since the first byte of application data arrives to the MAC in the destination until it reaches the receive-FIFO for the corresponding channel.
- L: effects of physical links. Time span since the first byte of data leaves the MAC in the origin port until it reaches the MAC in the other end.

We characterized these parameters on the implemented prototypes. The Rx_{clk} and Tx_{clk} driving the Ethernet interfaces run at 125MHz for 1Gbps links. The SW_{clk} for the forwarding path in the switch also runs at 125MHz, and the NC_{clk} in the NC-ASIP runs at 62.5MHz (see Figs. 2 and 5). Table III shows the maximum value for each term in (1) as a function of v_l, measured on each device using the Chipscope tool with a sampling clock of 125MHz. The reported value corresponds to the highest value (worst-case) observed in a set of 100 samples for each case. For all cases, the difference between the highest and lowest value in the sample was never higher than two clock cycles, which we attribute to the drift between the clock domains inside the devices and the sampling clock.

Both NC-ASIPs and switches start transmission as soon as they detect available data in the FIFOs, and then the latency is independent of v_l. The total latency in the switch includes a delay of 32 cycles related to the input classifier, and 76 cycles before the RT-Tx-FIFO gets access to the MAC in the output port. As discussed in Section III-A, this waiting time accounts for the IFG (12 cycles) and the transmission time of a minimum sized frame (64 cycles). The rest accounts for internal processing. In the case of the links, the latency must be characterized for each specific configuration. For this experiment, we considered two Broadcom BCM5464SR PHYs linked with 7.62m CAT6 cables.

Replacing the observed values in (1), we model the worst-case EEL as:

$$\text{EEL}(N_S) = 1.624 + 1.576N_S[\mu s] \quad (2)$$

for $v_l \leq 11$, and

Table IV: Worst-case End-to-end Latency [μs]

v_l [words]	**model** [μs]	**observed** [μs]	**error**
5	6.352	6.32	0.51 %
20	6.64	6.608	0.48 %
35	7.12	7.088	0.45 %
70	8.24	8.224	0.20 %

$$\text{EEL}(N_S, v_l) = 1.272 + 1.576N_S + 0.032v_l[\mu s] \quad (3)$$

for $vl > 11$.

To verify the model, we directly measured the EEL between two instances of the NC-ASIP implemented on a single NetFPGA using a reference global clock, which exchange real-time frames across three enhanced switches. We additionally connected three ports of a traffic generator broadcasting best-effort frames at 800Mbps each. Table IV compares the highest observed latency from a set of 100 samples for each case, to the worst-case value obtained from (2) and (3). As we see, even under the highly saturated scenario for best-effort traffic, the models provide a tight and effective upper-bound for the latency of real-time frames.

D. Robustness Against Co-existing Best-effort Traffic

In this experiment we used an IXIA traffic generator to collect long-term statistics over controlled streams. One port emits two periodical reference streams of timestamped frames. One stream generates frames of configurable length tagged as NC-SYNC, with a period of 100µs. The other stream is similar, but generates regular best-effort frames. These streams propagate through three switches to a second port in the generator which collects the instantaneous propagation latency for each type. Two additional ports broadcast raw best-effort frames of 1,000B at a rate of 470Mbps each. This configuration generates an aggregated bandwidth close to over-utilization, and allows us to stress the effects of jitter in the latency of the different classes.

Fig. 10 shows the observed latency versus the transmission rate on the reference streams. Latency measurements are only valid when there are no dropped frames. The markers show the average latency over a sample set of 10^7 frames for each type. The ends of the bars represent the minimum and maximum value on each set. On the one hand, we see that the latency and jitter of the reference best-effort stream increases with the transmission rate. When the rate of timestamped streams raises over 50Mbps, the total traffic exceeds the capacity of the links, and the reference best-effort stream starts reporting dropped frames. On the other hand, the real-time path provides a bounded latency and no losses for real-time frames, independent of the rate of best-effort traffic. The *cut-through* forwarding in the real-time path also reduces the propagation latency in relation to the traditional *store-and-forward* used in most COTS switches.

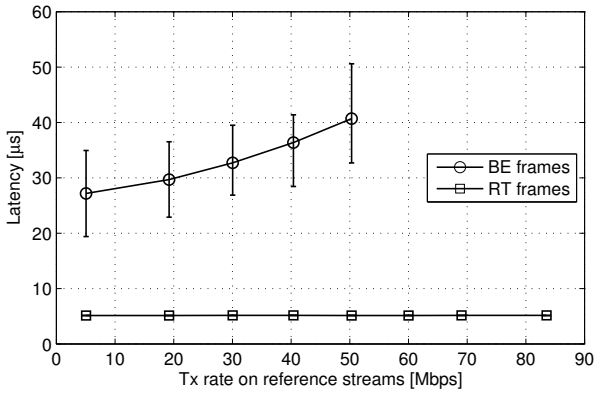


Figure 10: Robustness of Real-time Latency

E. Application Example: Real-time Video Streaming

To visualize the effect of the timing properties, we prepared a setup for real-time video streaming. In the setup, a source station emits frames with video lines every 31.68µs (the required period to display a 640x480 video at 60fps). The frames propagate through three switches to a sink station. The task in the sink reads the DATA-MEM at the same period to display the received pixels. The DATA-MEM only holds one video line (no video buffering), and then any variation in the EEL will degrade the quality of the displayed video.

The experiment shows two properties of the framework: (1) the NC-ASIPs can cope with the strict periodicity requirement for the fluid displaying of the video in the sink station, and (2) the timing properties of real-time frames are resilient to injected best-effort traffic. A detailed description of the setup and results are available on-line [17].

V. Conclusions

This work introduces our open-source RTE framework called *Atacama* for multi-segmented networks with mixed-criticality traffic. The framework introduces an ASIP to coordinate the exchange of time-sensitive data between real-time stations. In addition, a custom switching path provides low and predictable forwarding latency of real-time frames, automatic discovery of real-time stations, and dynamic segmentation of real-time domains in large networks. The modules feature a dual-pathway to provide separate resources for real-time and best-effort traffic, enabling mixed criticality communication. Using experimental data from implemented prototypes, we derived a model for the communication latency of real-time frames. The model provides an accurate upper-bound for the end-to-end latency between distributed real-time tasks, only limited by physical characteristics such as the drift between clock domains and uncertainty in the physical links. The experiments also showed that latency guarantees are robust to the injection of best-effort traffic. *Atacama* is the first fully-functional RTE framework available as an open-source project (available under request), enabling researcher to validate the resources, build upon, and test the devices on their own applications.

On-going and future work includes porting the designs to fit different FPGA chips, development of design tools to generate the schedules and matching architectures for the distributed ASIPs from high-level specifications, and expanding the framework to 10Gbps Ethernet devices.

Acknowledgments

This research was supported in part by grants MECESUP #FSM0601 and Fondecyt #1121010 from the Chilean government, and projects NSERC DG 357121-2008, ORF-RE03-045, ORF-RE04-036, ORF-RE04-039, APCPJ 386797-09, CFI 20314 and CMC, and the industrial partners associated.

References

[1] J.-P. Thomesse, "Fieldbus Technology in Industrial Automation," *Proc. IEEE*, vol. 93, no. 6, pp. 1073 –1101, Jun 2005.

[2] H. Kopetz, "The Rationale for Time-Triggered Ethernet," in *Real-Time Systems Symposium*, Dec. 2008, pp. 3 –11.

[3] M. Felser, "Real-Time Ethernet - Industry Prospective," *Proc. IEEE*, vol. 93, no. 6, pp. 1118–1129, Jun. 2005.

[4] J.-D. Decotignie, "The Many Faces of Industrial Ethernet," *IEEE Ind. Electron. Mag.*, vol. 3, no. 1, pp. 8 –19, Mar. 2009.

[5] J. Jasperneite, J. Imtiaz, M. Schumacher, and K. Weber, "A Proposal for a Generic Real-Time Ethernet System," *IEEE Trans. Ind. Informat.*, vol. 5, no. 2, pp. 75 –85, May. 2009.

[6] J. A. Stankovic, "Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems," *Computer*, vol. 21, no. 10, pp. 10–19, 1988.

[7] "Profinet," http://www.profibus.com.

[8] "Ethercat Technology Group," http://www.ethercat.org/.

[9] "Time-Triggered Ethernet," http://www.tttech.com.

[10] S. Fischmeister, R. Trausmuth, and I. Lee, "Hardware Acceleration for Conditional State-Based Communication Scheduling on Real-Time Ethernet," *IEEE Trans. Ind. Informat.*, vol. 5, no. 3, pp. 325–337, Aug 2009.

[11] G. Carvajal and S. Fischmeister, "A TDMA Ethernet Switch for Dynamic Real-Time Communication," in *Proc. of the 18th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2010, pp. 119–126.

[12] S. Fischmeister, O. Sokolsky, and I. Lee, "A Verifiable Language for Programming Communication Schedules," *IEEE Trans. Comput.*, vol. 56, no. 11, pp. 1505–1519, Nov 2007.

[13] J. Elson, L. Girod, and D. Estrin, "Fine-grained Network Time Synchronization Using Reference Broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, Dec. 2002.

[14] K. Steinhammer, P. Grillinger, A. Ademaj, and H. Kopetz, "A Time-Triggered Ethernet (TTE) Switch," in *Design, Automation and Test in Europe (DATE)*, Mar. 2006, pp. 794–799.

[15] "NetFPGA project webpage," http://www.netfpga.org.

[16] P. Grillinger, A. Ademaj *et al.*, "Software Implementation of Time-Triggered Ethernet Controller," in *Workshop on Factory Communication Systems*, Jun 2006, pp. 145–150.

[17] "Embedded Software Group at University of Waterloo," http://esg.uwaterloo.ca.