Evaluation of Communication Architectures for Switched Real-Time Ethernet

Gonzalo Carvajal, Chun Wah Wu, and Sebastian Fischmeister, *Member*, *IEEE*

Abstract—Safety-critical distributed real-time applications operating with strict temporal constraints rely on deterministic networks with low latency and jitter. Traditional fieldbus systems deliver these guarantees, but they have limited compatibility with open infrastructures and limited support for high transmission rates. Ethernet technology rises as a low-cost, high-speed, and ubiquitous alternative to fieldbus systems; however, standard Ethernet requires special arbitration mechanisms to support real-time traffic because of the standard's inherent nondeterministic behavior. This work explores the associated tradeoffs for three different solutions for real-time communication over switched Ethernet. The paper presents and discusses three architectures that modify different network components, enhancing them with additional customized modules to support time-triggered communication based on Network Code. Using the NetFPGA platform as the unified prototyping technology for all the components, we developed an open-source framework to characterize each solution using experimental data for the latency, jitter, throughput, robustness, and cost in logical resources. The results provide insights to help future developers of real-time communication technology decide which components to modify according to the requirements of their applications.

Index Terms—Real-time Ethernet, switched Ethernet, Network Code, dynamic TDMA, NetFPGA

1 Introduction

Safety-critical distributed real-time systems, such as avionics, plant process control, and networked medical devices, among others, require reliable networks capable of providing strict delay and bandwidth guarantees for message exchange between spatially dispersed stations. Typically, these systems relate to control tasks at the field-level automation requiring timely exchange of small amounts of data between embedded devices (e.g., sensors, controllers, and actuators). Traditional fieldbus systems provide optimized communication networks to fulfill the specific requirements of different application domains [1]. However, due to their limited speed and compatibility, these solutions are unfit for modern real-time applications requiring higher bandwidth, and seamless integration of field devices with traditional computer systems within a single network [2], [3], [4].

Nowadays, switched Ethernet is the most popular technology used in traditional wired Local Area Networks (LANs) for office and home environments. Advantages of switched Ethernet over traditional fieldbus systems include reduced costs, increased bandwidth, and support for open infrastructures. However, the standard Ethernet protocol permits connected stations to exchange data of any size at any time, so they must compete for the shared communication resources, which lead to nondeterministic behavior. Hence, standard Ethernet is intrinsically unfit for hard real-time applications requiring predictable delays and guaranteed bandwidth [5], [6].

To provide real-time guarantees, Ethernet networks must implement additional arbitration mechanisms to coordinated access to the communication medium [2], [3]. We can classify these solutions in terms of functional principle (e.g., time-triggered, master-slave, token passing), timing guarantees (e.g., hard or soft real time), compatibility with standard Ethernet (e.g., homogeneous, heterogeneous), and supported topologies (e.g., star, ring, etc.). In general, the literature shows that communication of hard real-time traffic over Ethernet is only possible by modifying at least one Ethernet hardware component. Based on this premise, we identify three principal architectures that enable real-time communication over switched Ethernet: modified Network Interface Card (NIC) with Commercial-Off-The-Shelf (COTS) switches, modified NICs and modified switches, and COTS NICs with modified switches.

Because of the different options, it is important to understand the tradeoffs when implementing real-time-capable Ethernet networks. In this work, we use a common framework to design and implement hardware prototypes for the components of each architecture, and collect experimental data to provide evidence of the associated tradeoffs for each solution. The presented results are relevant for future developers of real-time communication technology because they must decide upon the architecture they will use for their applications, and be aware of the consequences of their design choices.

The proposed solutions rely on time-triggered communication based on the Network Code framework [7]. Using adapted versions of the original Network Code Processor (NCP) [8], we designed custom modules to enhance COTS Ethernet devices that enable support for real-time

- *G. Carvajal is with the Departamento de Ingenieria Electrica, Universidad de Concepcion, Barrio Universitario S/N, Concepcion, Chile. E-mail: gcarvaja@udec.cl.*
- *C.W. Wu and S. Fischmeister are with the Department of Electrical and Computer Engineering, University of Waterloo, Engineering 5, E5-4112, Waterloo, ON N2L 3G1, Canada. E-mail: {cwwu, sfischme}@uwaterloo.ca.*

Manuscript received 18 Oct. 2011; revised 4 June 2012; accepted 3 July 2012; published online 11 July 2012.
Recommended for acceptance by K. Ghose.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-10-0742.
Digital Object Identifier no. 10.1109/TC.2012.177.

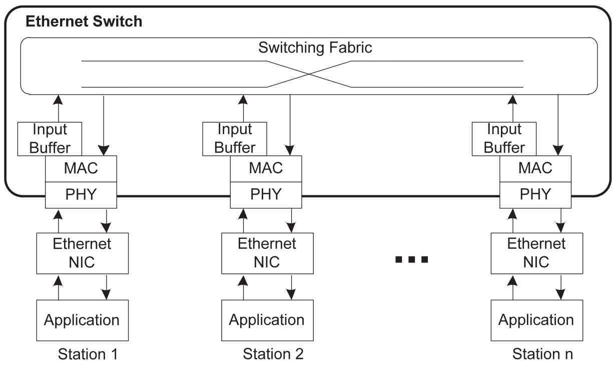


Fig. 1. A microsegmented Ethernet network.

communication. The *NIC-only architecture* uses custom NICs with embedded NCP units that connect to a COTS switch. The *NIC/Switch architecture* relies on the same custom NICs, but uses a custom switch that classifies incoming frames and provides independent forwarding paths for real-time and best-effort traffic. Finally, the *Switch-only architecture* uses COTS NICs and a custom switch with NCP units embedded into each input port.

Using the NetFPGA platform [9] as the unified prototyping technology for all the components, we collect experimental data to characterize and evaluate each solution in terms of latency, jitter, throughput, robustness of real-time guarantees in the presence of coexisting best-effort traffic, and cost in logical resources. The evaluation considers a microsegmented topology, which is supported in all the architectures, and we provide some discussion on how to extend the work to multiswitch topologies. The entire implementation framework, including hardware designs and software tools, are available as an open-source project for the academic and research community [10]. This open approach allows the users to validate our results and get insight about real-time Ethernet solutions by using actual prototypes rather than theoretical models. We also expect to contribute on the development of a low cost and integral solution to fit the communication requirements for different application domains.

The remainder of the paper is structured as follows: Section 2 introduces general concepts and definitions for the different architectures. Section 3 presents an overview of the Network Code framework. Section 4 describes the implementation details for the components of each architecture. Section 5 shows the experimental characterization obtained from the implemented prototypes. Section 6 discusses the results and lessons learned, and provide insights for future work. Finally, Section 7 summarizes and concludes the paper.

2 Architectures for Real-Time Ethernet

This section presents a brief overview of general Ethernet concepts, and provides details on the three principal architectures considered for evaluation.

2.1 Network Model and Definitions

Fig. 1 depicts a microsegmented network model. In this configuration, there is a single switch and each station connects to one of the switch ports using an Ethernet NIC. Each NIC contains a Medium Access Control (MAC) module with a unique identifier (MAC address), and the physical layer transceiver (PHY). Each switch port also includes its own MAC and PHY, and communicates with the connected NIC through a full-duplex connection. Most COTS switches operate with the *store-and-forward* paradigm, which retain the complete incoming frame in the input buffer waiting for the Frame Check Sequence to verify the validity of the received data. The switch automatically discards malformed frames and forwards valid frames to the switching fabric. The fabric analyzes the destination MAC address encapsulated in the frame headers and forwards the frame only to the port on which the addressed station is connected, enabling simultaneous forwarding of frames addressed to different stations. Alternatively, the *cut-through* paradigm sends the frame to the switching fabric just after receiving the destination MAC address, without performing an integrity check. This operation mode reduces the memory requirements and forwarding latency, but it is prone to waste bandwidth by propagating erroneous frames; this approach is not commonly used by COTS devices.

The switching fabric also includes buffers for each output port. If multiple frames addressed to the same port arrive within a short period of time, the fabric will retain them in the corresponding buffer and transmits them sequentially according to some queuing algorithm. This approach maximizes the throughput because it retains a continuous flow of data through the ports. However, even with advanced queuing mechanisms and the use of priorities for different classes of traffic in high-end switches, unconstrained traffic bursts still lead to nondeterministic forwarding latencies and possibly dropped frames due to buffer exhaustion under high traffic loads [5], [6]. This inherent best-effort approach of standard Ethernet is unfit for safety-critical application requiring hard communication guarantees.

In our model, stations support two traffic classes: real-time traffic reserved for time-critical communication, and best-effort traffic used for non-time-critical communication. We assume that real-time traffic is periodic and transmitted in a broadcast fashion, which is the typical case for time-triggered control applications [11]. Best-effort traffic is arbitrary and follows the standard addressing mechanisms of switched Ethernet without delivery guarantees.

2.2 NIC-Only Architecture

The NIC-only architecture uses COTS switches and custom NICs to provide controlled access to the shared communication resources. The underlying idea is to implement communication schedules that permit stations to transmit data only during specified time slots. The distributed NICs must share a global sense of time to guarantee that at most one station transmits a frame at any given time, thereby preventing competition. Existing solutions using this approach include Ethernet Powerlink (EPL) [12], Ethernet for Plant Automation (EPA) [3], RETHER [13], and the original versions of Time Triggered Ethernet (TTE) [14] and Flexible Time Triggered Ethernet (FTT-E) [15], among others.

By relying on COTS switches, this architecture supports any network topology; however, the propagation delay and jitter will increase with as the number of switches in the path between two stations increases. The designer must consider these parameters when designing the schedules to prevent all potential conflicts at runtime, which usually requires the designer to assume the worst-case delay. Some disadvantages of this architecture include the potentially large communication overhead incurred from the propagation of tokens or synchronization messages to coordinate the stations, the need to manipulate and configure multiple hardware units separately, and the requirement to use only custom NICs in the network, as uncoordinated traffic emitted by COTS NICs may interfere with the real-time traffic.

We previously evaluated this architecture using custom NICs featuring-embedded programmable modules, or *scheduling units*, which execute predefined Network Code schedules for dynamic Time Division Multiple Access (TDMA) arbitration [8].

2.3 NIC/Switch Architecture

The NIC/Switch architecture uses custom NICs to coordinate real-time traffic in conjunction with a modified switch that isolates the real-time frames from the best-effort frames, enabling the coexistence of both traffic classes without affecting the required guarantees for time-critical data. The first proposed solution using this approach was Profinet IRT [16]. Later versions of TTE [17] and FTT-E [18] also introduced custom switches that complement the functionality of their modified NICs. In all cases, the custom switches intentionally break the standard forwarding rules to isolate the two traffic classes, allowing standard Ethernet devices to connect to the switch and communicate without interfering with real-time traffic. This is possible by assigning either dedicated paths (TTE) or specific time slots (FTT and Profinet IRT) for each class. The coexistence of both traffic classes in the same network enable vertical integration of time-critical systems with traditional workstations, which is a common requirement of modern control systems [4].

FTTE and Profinet IRT are time-triggered protocols and rely on programming the custom NICs with static schedules, which must be carefully designed and verified offline to prevent all potential conflicts at runtime. This is a common approach for safety-critical systems subject to certification; however, it generally leads to inefficient resource utilization because the static schedules assign communication slots to all stations whether they actually need them or not. On the other hand, the switched version of FTT employs a master/slave approach with a centralized coordination unit (the master) controlling the forwarding tasks in the switch. The master polls the stations to retrieve their data and then forwards it according to the master schedule, which supports online modifications to respond to variable operational conditions. This approach provides greater flexibility and more efficient resource usage. As we will see in Section 3, Network Code implements conditional state-based communication schedules for dynamic TDMA, adapts characteristics of these two approaches. For a discussion about the advantages and disadvantages of each method, see [19], [20].

Our proposed solution for this architecture uses the custom NICs described in Section 2.2, and an enhanced switch with independent forwarding paths for each traffic class (best-effort and real time), giving strict forwarding priority to real-time frames on the output ports.

2.4 Switch-Only Architecture

The third architecture relies on custom switches, while stations can use COTS NICs to transmit both best-effort and real-time frames. Reported solutions of this kind are based on using frame priorities (IEEE 802.1p standard) [5], enhanced scheduling algorithms [21], or connection oriented channels for real-time traffic on networks-on-chip [22]. However, all of them are unable or limited in providing hard real-time guarantees. In fact, some of them rely on higher communication layers of the OSI model, which are usually absent in embedded devices with limited computational resources.

For our implementation, we propose a switch that retains the independent paths approach for each traffic class, but also includes an embedded scheduling unit for each input port. The ports retain incoming real-time frames in a dedicated input buffer, and forward them to the other output ports at the time specified in the programmed schedule. Similar to the previous architecture, the switch enforces strict priority of real-time traffic over best-effort traffic when accessing the output ports. Because the coordination tasks are centralized in the switch, stations can connect using COTS NICs. This approach is particularly suitable for embedded field-level devices, such as sensors and actuators, which typically operate with well-defined periodic traffic patterns.

In this architecture, the switch provides a central point for configuring and controlling the scheduling policies. This characteristic allows us to expand the operational flexibility of the system through an enumerative approach [20]. For example, the switch may store multiple schedules representing different modes of operation, and use a mode-switch protocol to switch between these modes depending on the current environment [23].

3 The Network Code Framework

The NC framework consists of three elements:

- An expressive domain-specific programming language providing a basic instruction set to describe conditional state-based communication schedules.
- A compiler with a verification engine that translates the programs into checked executable schedules.
- The entity that executes the schedules at runtime.

We extensively described the individual components in previous works [7], [8]. In this section, we give a brief overview of the NC communication model and the hardware implementation of the execution entity.

3.1 System Model and Definitions

The NC framework was originally designed to coordinate the traffic generated from distributed stations used for run real-time applications that share a communication medium. Fig. 2 shows the layered model employed by each station in the distributed system. Real-time tasks run in the upper

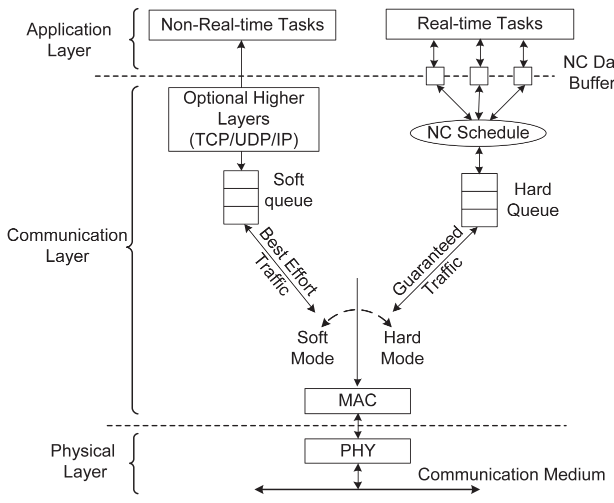


Fig. 2. Layered model for the NC framework.

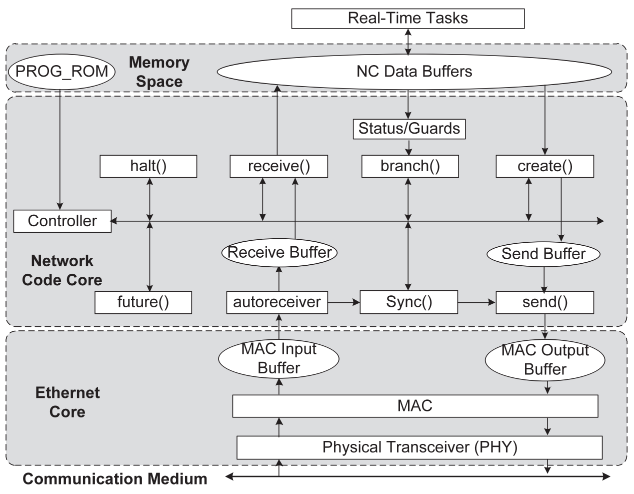


Fig. 3. Block diagram of the NCP.

application layer, and they use the services provided by the programmable communication layer to transmit and receive data within strict timing constraints. Both layers run independently, and they only interact by exchanging data through shared buffers.

Real-time tasks produce and consume hard-values and soft-values. Tasks access hard-values by reading/writing from/to predefined data buffers at specific points in time. The system assumes that accesses to hard-values follow repetitive and well-known temporal patterns. The delivery of hard-values is guaranteed to happen within bounded time. The management of soft-values is beyond the scope of this paper, and is not further addressed in this document (for more details, see [7]).

Within the communication layer, verified NC schedules define an isochronous communication rounds that grant each station specific time slots in each round to transmit and receive hard-values. For transmission, the scheduler reads the application data from a specific NC data buffer, encapsulates it in an Ethernet frame, and sends it to the physical layer which broadcasts the frame. Receiver stations synchronize with the transmitter, perform the matching operations to retrieve the frames from the medium, and store the real-time data into a specific NC data buffer, making it locally available to the receivers' real-time tasks.

To achieve coordination, all stations must share a global view of time. NC follows a Reference-Broadcast Synchronization (RBS) scheme [24], which broadcasts periodical synchronization beacons across the network. A synchronization beacon does not contain an explicit time stamp, but receivers use the arrival time of these beacons as a point of reference to coordinate their actions. This approach is a simple and low-cost alternative to distributed clock mechanisms, commonly used in traditional fieldbus systems [2], [24]. Even though it is also possible to use a distributed clock mechanism, such as IEEE 1588 [25], these solutions come at an additional cost in special hardware (high-precision reference clocks and time-stamping units), communication overhead (additional message exchange), and computation overhead (clock correction mechanisms in stations).

In the practical implementation, a designated master broadcasts synchronization frames, often to signal the start of a communication round. Slaves wait for the arrival of this message to start executing their local actions defined for the corresponding round. The synchronization accuracy relates to the variations in the propagation delay and processing time of the frame on each slave station. The schedules must consider the worst-case delay for these parameters to guarantee conflict-free communication at runtime. NC assumes that actions within each round follow well-defined temporal patterns, and all necessary message exchanges and data structures are known in advance and computed offline. Under these conditions, we can apply static verification [7] and analysis [26] to evaluate system correctness and detect potential conflicts before they occur at runtime. This is an important property for safety-critical systems requiring evidence-based certification.

3.2 The Network Code Processor

The NCP is a hardware implementation of the programmable communication layer. As Fig. 3 shows, it consists of three main sections: the Network Code Core (NCC) that implements the main functionality, and the memory space, and Ethernet core, which provides the interfaces to the application and physical layer, respectively.

The NCC block features a superscalar architecture; it implements each instruction of the NC language as an independent hardware block. The controller interprets the instructions fetched from the program ROM, PROG_ROM, and triggers the execution of the corresponding blocks. The controller also calculates dependencies between consecutive instructions, enabling concurrent execution of multiple blocks whenever it is possible [8].

The instruction blocks provide control of data flow, timing, and conditional execution. In the following paragraphs, we present a brief functional description of the instruction blocks in the NCC. For further details about the formal semantics and parameters of Network Code, see [7].

3.2.1 Data Flow Control

Transmission of real-time data is driven by create-send sequences. The create block copies application data from a specific NC data buffer to the send buffer. The send block reads this buffer, encapsulates the data into an Ethernet frame using a special identifier (NC-DATA Type) in the

EthType field, and sends it to the MAC layer for transmission. For reception, the Ethernet core automatically detects incoming valid frames and triggers the asynchronous autoreceiver block, which checks the *EthType* field, unpacks the application data from frames tagged as NC-DATA Type, and stores it in the receive buffer. The receive block can then move this data to a local NC buffer. If the controller does not trigger a receive instruction, the received data will stay in the buffer until the next arriving frame replaces it.

3.2.2 Timing Control

The future, halt, and sync blocks implement temporal coordination of actions between multiple stations. The future block starts a countdown timer with some initial value and generates an interrupt when it expires. The halt block stalls the program, waits for a generated interrupt from the future block, and then resumes execution at a particular program label. The sync block operates in two different modes. In master mode, the block generates the broadcast frame that signals the start of a communication round. In slave mode, the block stalls the execution of the schedule until the arrival of a master sync frame. Synchronization frames use a special value (SYNC Type) in the *EthType* field and do not contain application data.

3.2.3 Conditional Execution

The branch instruction enables the implementation of dynamic schedules that make on-the-fly decisions based on guards. These guards continually check for specific conditions on values of the NC buffers, execution history, or the current status of the system. The branch block forces a jump to a predefined label in the program based on the guard condition. For example, the communication system can stop transmitting data when a certain variable lies below a predefined threshold.

4 IMPLEMENTATION FRAMEWORK

This section shows how the NC framework serves as the basis for implementing the hardware components for each architecture. It first introduces the NetFPGA platform, and then describes the most relevant characteristics of the custom NICs and switches.

4.1 The NetFPGA Platform

The NetFPGA is a low-cost and open platform targeted for the implementation of reusable hardware code for high-performance networking applications. The platform package consists of three main elements: 1) an FPGA-based hardware board, 2) software tools to integrate the board with a host workstation, and 3) fully functional open-source application projects.

The hardware board attaches to the Peripheral Communication Interconnect (PCI) bus of any standard PC. A Xilinx Virtex-II Pro 50 FPGA that holds and executes custom logic, and four separate 1-Gbps Ethernet ports are key components built into version 2.1 of the board [27].

The software tools include a Linux device driver and a set of basic utilities that allow the user to access the internal memory, and reconfigure the FPGA functionality directly from the workstation, without requiring additional cables or even physical access to the hardware.

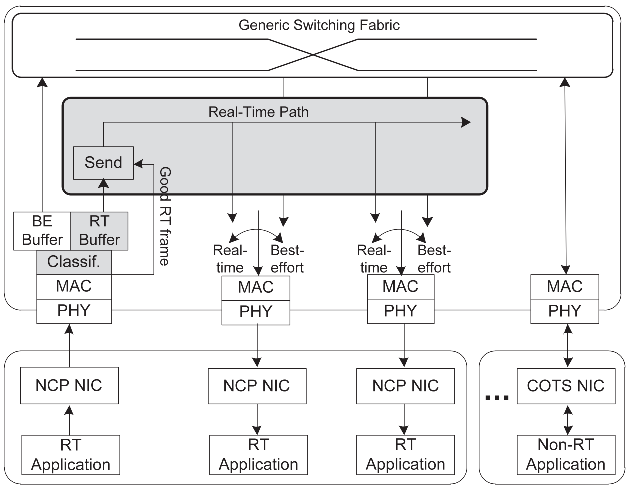


Fig. 4. Block diagram of the NIC/Switch architecture.

4.2 NIC-Only Architecture

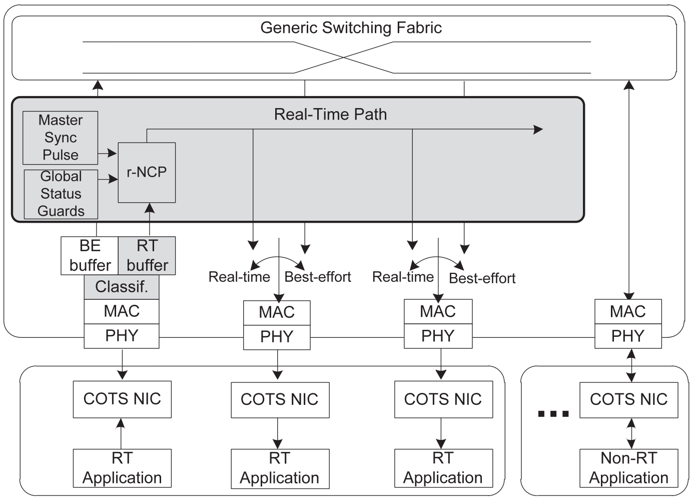
We previously implemented and evaluated an FPGA-based version of the NCP architecture described in Section 3.2 for an Ethernet NIC. The original custom NICs operate at line rate on 100-Mbps links with high timing accuracy [8]. For the current work, we adapted and enhanced the existing modules to prepare an Intellectual Property (IP) core for the NCP that fits onto the Virtex2 chip in the NetFPGA platform and operates over 1-Gbps links. The core includes some additional logic to interface the PCI port and internal debugging registers that keep track of runtime statuses and statistics (Rx/Tx frames, synchronization time-outs, etc.).

Using the new IP core, we implemented an evaluation platform containing four independent NCP-based NICs for time triggered-communication on the NetFPGA board. The user can attach the prototype to a workstation, and use the provided software tools to configure, execute, and evaluate specific scenarios for distributed communication by filling the PROG_ROMs with verified NC schedules, emulating real-time tasks by reading/writing the data buffers, and setting/checking internal system statuses at runtime.

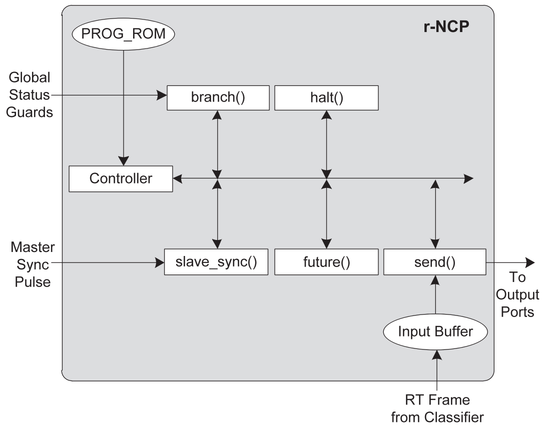
The COTS switch prototype for this architecture corresponds to a direct implementation of the open-source reference switch design available for the NetFPGA [10].

4.3 NIC/Switch Architecture

Fig. 4 illustrates the proposed implementation of the NIC/Switch architecture introduced in Section 2.3. Stations that send and receive real-time traffic connect to the switch using NCP-based NICs, while stations that only send and receive best-effort traffic can connect directly using COTS NICs. We modified the original data path of the reference COTS switch by adding the modules depicted in gray in Fig. 4. The modified switch includes an input classification module on each port. This module checks the *EthType* field of input frames to separate best-effort from real-time frames, storing each type of frame in independent input buffers. Best-effort frames follow the traditional path through the switching fabric as in any COTS switch. Real-time frames (tagged as NC-DATA or SYNC type) route through a dedicated path from the corresponding input buffer to all other output ports. Because of the broadcast nature of real-time frames and the coordination of the



(a) Block diagram of the Switch-only Architecture



(b) Block diagram of the embedded scheduling unit

Fig. 5. Switch-only architecture.

custom NICs, the real-time path omits any address processing and queuing mechanisms that would otherwise handle runtime conflicts. The resulting path resembles a classical shared-bus connection; we can then expect a reduction in the latency and jitter for real-time traffic with respect to the NIC-only architecture. For evaluation purposes, the real-time path adopts the store-and-forward approach, which is the most commonly employed approach in COTS switches. However, in practice, the cut-through paradigm fits better to the needs of high-performance real-time systems, and its implementation in the proposed architecture is straightforward.

The custom switch strictly prioritizes real-time frames over best-effort class. Upon detecting a valid real-time frame in some input port, the port's corresponding MAC generates a signal to trigger the send block, which takes immediate control of all the other output ports and interrupts any ongoing best-effort transmission in the switching fabric. Before forwarding the real-time frame, the send block waits for the Interframe Gap (IFG) to elapse to ensure that the receiver detects the potentially interrupted best-effort transmission as an error, which can be handled at the higher communication layers in the stations (if applicable). After forwarding the frame, the real-time path waits again for the IFG to elapse and then returns the access of the output port back to the generic switching fabric. To keep the design simple, the current implementation does not incorporate any mechanisms that handle and compensate for interrupted best-effort frames. In future versions, we plan to evaluate and implement some kind of retransmission mechanism like the one used in TTE [17], or the traffic light principle described in [28] (in fact, our design implements a simplified traffic light control without the amber phase). We also plan to include some kind of negotiation mechanism between the switch ports and the stations to prevent interruptions of best-effort frames on stations running exclusively non-real-time applications.

4.4 Switch-Only Architecture

Fig. 5a shows a diagram of the Switch-only architecture. The modified switch follows the concept of using dedicated paths for each traffic class described in the previous architecture, but it relocates the scheduling units from the NICs to the real-time path within the switch. Instead of automatically forwarding the real-time traffic, the input ports retain the frame in a buffer, and the embedded scheduling units forward them to the output ports at the time specified in a preprogrammed NC schedule. Fig. 5b shows the internal architecture of the scheduling units, which we refer to as *reduced-NCPs* (r-NCPs), because they are based on a simplified version of the original NCP architecture described in Section 3.2. The r-NCP retains the instruction blocks for timing control and conditional execution of the original architecture. The major simplifications come from the data flow operations, because the new scheduling unit only needs a simplified send instruction to forward the frames stored in the input buffer to the output ports without any further processing. An internal module generates synchronization pulses to signal the start of communication rounds and coordinate the scheduling units. The user configures the module to set the pattern of these pulses, and all r-NCP units work exclusively in the slave mode.

Stations must tag the real-time traffic with the corresponding value in the *EthType* field, and transmit data at the same or shorter period than the specified in the schedules inside the switch, ensuring that there will be new data available for each scheduled frame transmission. The real-time buffer on each input port will hold only one frame; thus, the most recently received frame will overwrite the previous one. This approach is well suited for periodical time-triggered control applications, where the controller is more interested in the most recent sensor reading rather than the sequence of values. Because synchronization is now internal to the switch, stations no longer need to share a global view of time. Receivers must be always ready to receive and process frames that are locally consumed. Since the architecture uses broadcast for real-time traffic, we assume that stations will filter the packets locally with an ID field encapsulated in the frames, for example. Unlike the previous two systems, this architecture only guarantees bounded switching latency, as the latency from the NIC to the application depends on the particular implementation

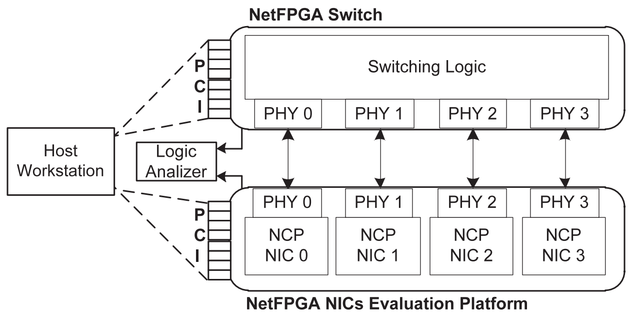


Fig. 6. Experimental setup.

(e.g., stack and hardware). We can also configure the system to prevent interruption of best-effort traffic in stations that do not run real-time applications.

Centralizing the scheduling units brings important practical advantages in relation to the distributed approach:

- The internal synchronization pulse is subject to less latency and jitter than the broadcast master sync frame, as it does not need to traverse the medium to reach the scheduling units. Avoiding the synchronization frames also reduces the communication overhead and increases the available bandwidth for application data.
- Guards representing the global system status for all the scheduling units enable efficient implementation of "enumerative reconfiguration" [20]. The scheduling units can check these guards and efficiently switch between different predefined configurations at the beginning of each communication round without compromising system correctness or interrupting the regular exchange of application data to transmit configuration or status messages.
- The user only needs to manipulate a single device instead of multiple distributed scheduling units.

5 Experimental Results

This section presents the experimental characterization of the architectures using the implemented prototypes for each device. We characterize the different systems in terms of throughput, latency, jitter, robustness of real-time guarantees against injected best-effort traffic, and the evaluation of a practical case study.

5.1 Experimental Setup

Fig. 6 shows the general setup used for the experiments. The system consists of two NetFPGA boards attached to the PCI ports of a host workstation. One board implements the evaluation platform for the custom NICs described in Section 4.2. The NICs can be individually configured with NC schedules to emit different patterns of either real-time or best-effort traffic. The units also hold a set of counters that maintain relevant statistics, such as the number of transmitted and received frames for each traffic class, internal buffer overflows, running time, and so on. The other board can be reprogrammed to implement the different switches. The ports of each board are directly connected to the other board through network cables, representing a microsegmented network like the one shown in Fig. 1.

Additionally, we used Xilinx's Chipscope tool as a logic analyzer for the FPGA's internal signals. We configured the logic analyzer with a sampling frequency of 125 MHz (8 ns clock resolution), which is the clock frequency for the PHY in 1-Gbps connections, and also the maximum operation frequency for all the systems under examination.

5.2 Throughput

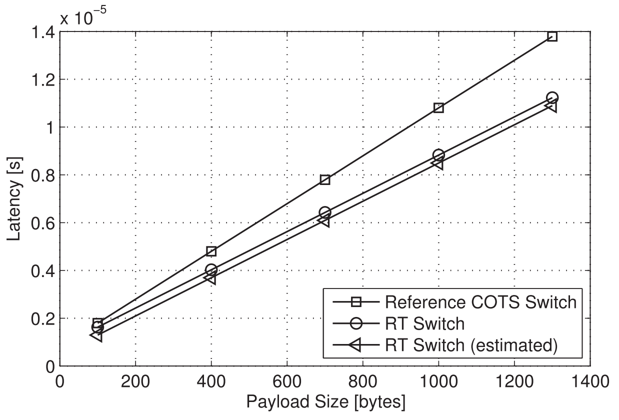
The real-time throughput is the maximum number of bytes of valid real-time application data that a station can receive per second. To measure this value, we configured one instance of the evaluation platform for NICs in the setup to emit real-time frames at a fixed period, while varying the payload size, to represent different rates. We checked the counters and the internal signals on both NICs and switches to detect errors and buffer overflows.

The custom NICs can emit data at around 931 Mbps, the COTS switching fabric can operate at line rate, and the real-time path in the custom switch without scheduling units can forward around 800 Mbps of guaranteed traffic. However, due to a bottleneck in the autoreceiver block that retrieves the application data from the MAC input buffer (see Fig. 3), the NICs can only receive at a maximum rate of 468 Mbps without overflowing the buffer. This is an inherited limitation from the original NCP architecture, which was designed to operate with 100-Mbps connections. This bottleneck does not affect the generality of the results and the conclusions of this work, and we plan to address it for future revisions.

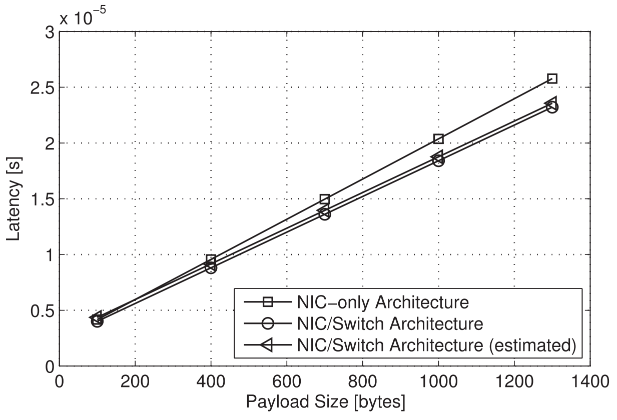
Summarizing, the reception mechanisms in the current version of the custom NICs limit the real-time throughput for the NIC-only and Switch/NIC architectures to 468 Mbps. As for the Switch-only architecture, the maximum throughput is 800 Mbps, but the actual throughput will depend either on the schedules programmed in the scheduling units inside the switch or the particular COTS NICs used in the implementation.

5.3 Latency

The switching latency is the time required to forward a byte through the switch; it is measured as the time elapsed from the arrival of the first byte of the payload at the input port's MAC to the departure of the same byte at an output port's MAC. The end-to-end latency is the time required for moving application data from one station to another, which is measured as the time elapsed from the triggering of a create instruction in the transmitter station to the moment that the first byte of the payload corresponding to that create instruction is written to the *receive buffer* in the other end (see Fig. 3). At this point, the receiver station can execute a receive instruction to move the data to the appropriate application buffer(s). We define the nominal latencies as the measured values when all internal buffers are empty and the observed frame has exclusive access to the switching resources (no simultaneous transmissions occurring from multiple stations). Fig. 7 shows the mean values of a set of samples of the nominal switching and end-to-end latencies versus the payload size for the evaluated architectures.



(a) Switching latency for real-time traffic



(b) End-to-end latency for real-time traffic

Fig. 7. Nominal latency on real-time frames versus injection rate of best-effort traffic.

Fig. 7a shows the nominal switching latency. The prototype of the custom switch processes both real-time and best-effort frames using the store-and-forward paradigm. Therefore, the higher latency in the COTS switch relates exclusively to the queuing and address processing mechanisms in the switching fabric, which is bypassed in the custom switches. In our setup, the real-time switch is around 16 percent faster on average, but the absolute speed up will depend on the particular implementation of the switching fabric. The processing units in the real-time path (gray blocks in Fig. 4) perform a predefined number of steps, which we can use to estimate the real-time path's latency for forwarding a variable of a particular size. For the current implementation, we characterize this latency as

$$\mathrm{Sw_{est}} = (44 + p_s) \times 10^{-9}[s], \tag{1}$$

which accounts for the fixed delays in the classification stage and the IFG before transmitting the frame, and the delay incurred from the store-and-forward approach, which depends on the payload size, p_s, measured in bytes. As the graph shows the estimated latency from (1) is lower than the measured value because it omits the delays related to the signaling mechanisms between the MAC and other logic in the path. Even though this additional latency will depend on the particular implementation, bounded latency is expected and the latency will correspond to a parameter of the device. The reported latencies for the real-time switch accounts for both the NIC/Switch architecture and Switch-only architecture. In the latter case, these values represent the minimum switching latency, if we consider the ideal case where the internal scheduling unit starts transmission as soon as the frame arrives. However, this latency, in practice, will depend on the waiting times specified in the schedules programmed on each input port.

Fig. 7b shows the nominal end-to-end latency for the NIC-only and NIC/Switch architectures. In the NIC/Switch architecture, the complete path from the source to the destination performs a predefined number of steps. We can model the associated latency as

$$\mathrm{EEL_{est}} = \mathrm{NCP_{Tx}} + \mathrm{Sw_{RT}} + \mathrm{NCP_{Rx}} + 2L, \tag{2}$$

where:

- NCP_{Tx}. Time elapsed during create-send sequence in the transmitting station; measured as the time elapsed from the triggering of the create instruction to the departure of the first byte of the corresponding application data from Ethernet core's MAC.
- Sw_{RT}. The measured switching latency shown in Fig. 7a (accounting for the frame processing tasks and signaling mechanisms)
- NCP_{Rx}. Time elapsed from the arrival of the first byte of application data at the destination's MAC until it reaches the receive buffer
- L. Effects of PHYs and cables linking two devices. L corresponds to the time elapsed from the first byte of data that leaves the MAC of the transmitting Ethernet core until it reaches the MAC receiving the data at the other end.

Table 1 summarizes the estimated values of the terms in (2) based on our experimental setup. In the case of the links, this value depends on the specific configuration and must be characterized for each case. In our setup, the links consider two Broadcom BCM5464SR PHYs linked with 1-meter CAT5 cables.

We are unable to obtain the end-to-end latency for the Switch-only architecture because the latency depends on the particular COTS NICs that are used in the system.

5.4 Jitter

Hard real-time systems define their communication tasks based on worst-case scenarios rather than statistical parameters. Therefore, instead of using the standard deviation, we define the latency jitter as the difference between the minimum and maximum value observed in a set of latency

TABLE 1
Estimated Latency Parameters

	Estimated latency [μs]
$\mathrm{NCP_{Tx}}$	0.48
Sw	$0.8 + 8p_s \times 10^{-3}$
$\mathrm{NCP_{Rx}}$	$0.392 + 8p_s \times 10^{-3}$
L	0.352

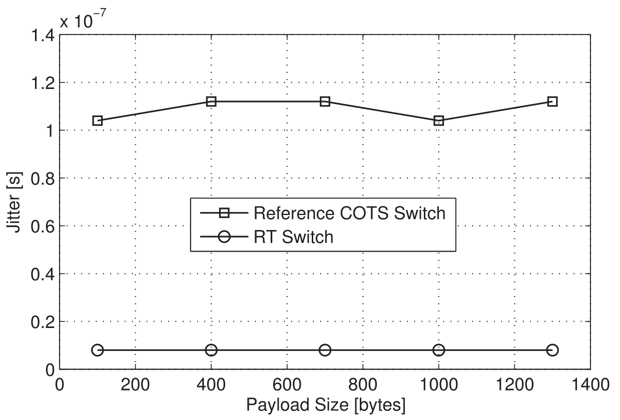


Fig. 8. Jitter on nominal switching latency.

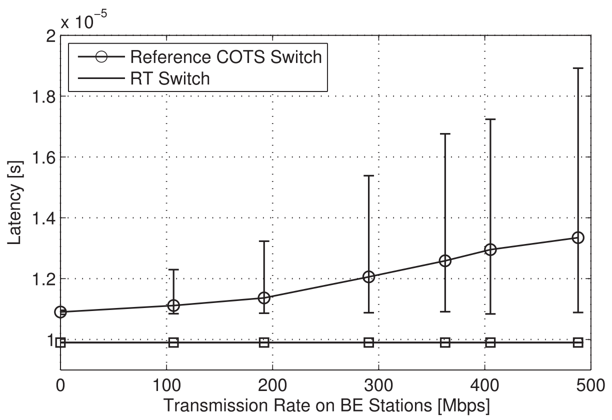


Fig. 9. Measured latency on real-time frames versus injection rate of best-effort traffic.

measurements. Fig. 8 shows the jitter for the nominal switching latencies of the previous experiment. As stated before, the results for the real-time switch account for both NIC/Switch and Switch-only architectures. In the custom switch, the only source of uncertainty in the latency relates to the drift in the clocks used for the Ethernet interfaces and the custom logic employed in the real-time path, which is an inherent characteristic of hardware implementations with multiple clock domains. This factor also affects the COTS switch. We considered the propagation of a single frame without additional traffic in our measurements, but even without competition, COTS switches have some inherent uncertainty due to the processing tasks that occur in the switching fabric. In our implementation, the jitter in the COTS switch is around 13.5 times higher than the reference COTS switch. Again, the absolute value depends on the particular implementation. In all cases, the jitter for the nominal latency is independent of the frame size.

5.5 Robustness against Coexisting BE Traffic

The goal of this experiment is to evaluate the delivery guarantees for real-time traffic in the presence of possible congestion due to the flow of unconstrained best-effort traffic. We consider two different metrics: variation in the latency of real-time frames with respect to its nominal value, and number of lost real-time frames between the end stations. These metrics are mutually exclusive: if there are no lost frames between the stations, then the latency will become the representative metric; otherwise, the percentage of lost frames will represent the performance of the real-time communication.

The setup consists of one NIC emitting real-time traffic and two NICs emitting best-effort traffic. The schedule of the real-time NIC emits frames periodically with a fixed payload of 550 bytes at a rate of 156 Mbps. The two best-effort NICs emit broadcast frames with a fixed period, but we varied the payload size to achieve different transmission rates. The difference between the number of emitted frames from one NIC emitting real-time traffic and the number of correctly received frames at the other NICs indicate the number of lost frames. We used a logic analyzer to measure the switching latency when there were no lost frames.

Fig. 9 shows the measured latency for real-time frames versus the transmission rate on both best-effort NICs. The error bars represent the minimum and maximum observed values for each best-effort rate. As we can see, both mean latency and jitter increase with the transmission rate of best-effort traffic in the NIC-only architecture. When the best-effort injection rate is 488 Mbps, the mean latency is around 1.35 times higher than its nominal value. The custom real-time switches retain the nominal latency for any rate of injected best-effort traffic.

Fig. 10 shows the percentage of lost frames for transmission rates over 500 Mbps on each best-effort NIC. Fig. 10a shows that the number of lost real-time frames in the NIC-only architecture grows as the injection rate of

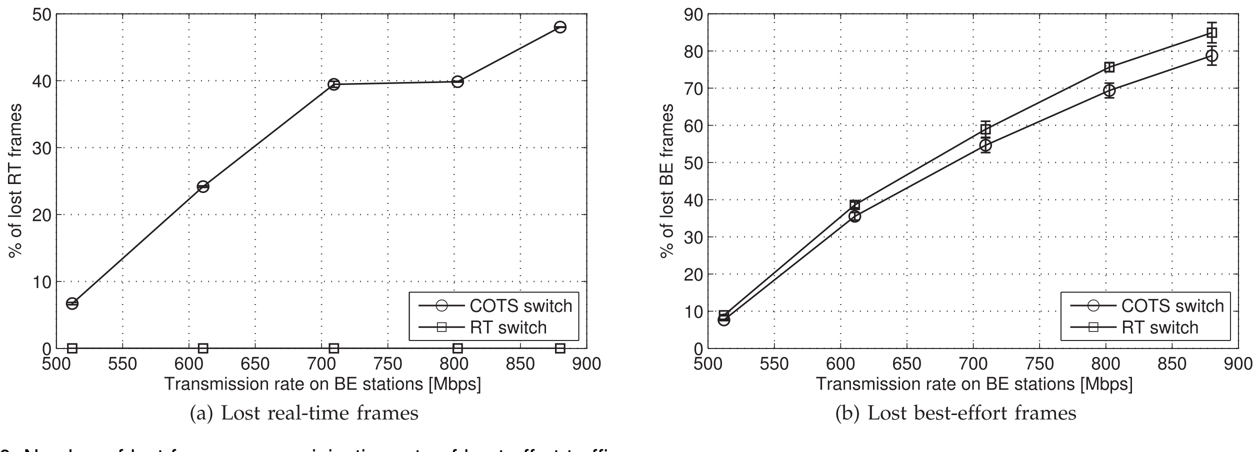


Fig. 10. Number of lost frames versus injection rate of best-effort traffic.

TABLE 2
Communication Requirements for UAV

Data Type	Frequency [Hz]	Size of Payload [B]
GPS	1	512
Log	5	512
Left Servo 1	60	16
Left Servo 2	60	16
Right Servo 1	60	16
Right Servo 2	60	16
Gear Servo	60	16
Tail Servo	60	16
Telemetry	30	512
Feedback Control	30	512
Video Camera	30	6825

best-effort traffic increases, while the architectures using the custom switches remain unaffected for any best-effort traffic injection rate. Fig. 10b shows that the number of lost best-effort frames is higher in the architectures that employ custom switches than in the one using the COTS switch. This higher number is a consequence of the restricted access to the output ports from the switching fabric in the custom switches, which increases the probability of both buffer overflows inside the switch and corrupted frames due to interrupted transmissions. In general, as the custom switch only forwards best-effort frames when there are no real-time frames, best-effort traffic may be subject to bandwidth starvation as the load of real-time traffic increases.

5.5.1 Case Study

To evaluate the practical effects of using the different architectures, we consider a case study based on an Unmanned Aerial Vehicle (UAV). We aggregate the communication requirements from a real UAV at the University of Waterloo, and from the UAVs described in [29], [30]. Table 2 states the required communication frequency and payload size of each device and service within the UAV that requires a time-bounded communication guarantee. Violating the strict timing requirements on the message delivery could cause the UAV to lose flight control and become unstable.

The video camera is effectively streaming video at 200 kbps based on the specified transmission frequency and payload size. Since the required payload for streaming video exceeds the maximum capacity that one Ethernet frame can encapsulate, the payload is delivered in multiple Ethernet frames.

To evaluate the system, we defined periodic schedules that meet the requirements specified in Table 2. We assume a microsegmented model, where all devices are connected directly to the same switch. Since the NetFPGA has only four ports, we emulate the distributed system using only two stations: the *source station* executes a schedule that models all real-time transmissions, and the *sink station* executes a reception schedule that matches the transmissions from the source. We can safely model the real-time communication using one transmitter and one receiver because at most one device is transmitting at any given time in the absence of best-effort traffic.

For the experiments, we configured two NICs representing the sink and source stations, and used the remaining two NICs to inject best-effort traffic at different rates. Fig. 11 summarizes the measured performance of the scheduled communication tasks on the sink station when using the NIC-only architecture. The figure shows the total number of received real-time frames and the number of correctly executed receive instructions, both as a percentage of the total number of frames emitted from the source station. We can clearly see that the real-time performance quickly degrades with the amount of injected best-effort traffic. When operating a system that relies on time-critical data, a frame arriving late is as bad as a corrupted or dropped frame. From the results, we clearly see that even though the COTS switch does not drop real-time frames with small amounts of best-effort traffic, the latency jitter still generates errors in the scheduled reception tasks due to delayed frames. Therefore, using the NIC-only architecture is a valid option only when all the stations operate with coordinated traffic.

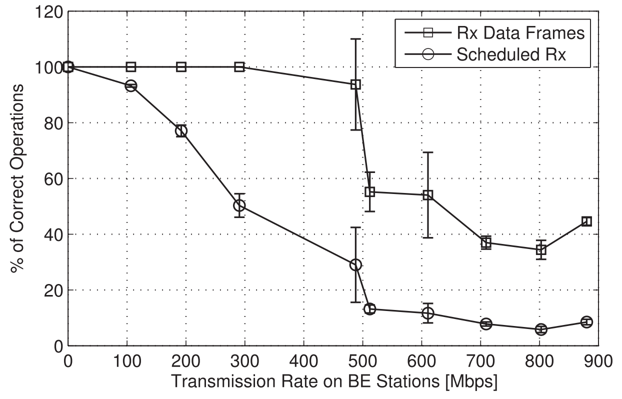


Fig. 11. Measured reception performance on the sink station of the case study.

TABLE 3
Device Utilization on Virtex2 FPGA Chip

Component	Flip-flops	4-in LUTS	BRAMs
NIC platform	25.6%	43.2%	34.5%
COTS Switch	19.5%	31%	46.1%
Switch (NIC/Switch)	23.9%	37.4%	60%
Switch (Switch-only)	27.3%	44.4%	62%

We also verified that, as expected, both the NIC/Switch architecture and Switch-only architecture deliver a 100 percent of correct scheduled receptions for any rate of injected best-effort traffic.

5.6 Device Utilization

Table 3 shows the cost in logical resources for each component. The percentages are related to a total of 47,232 flip-flops, 47,232 four-input LUTs, and 232 Block RAMS available in the Virtex2 chip embedded in the NetFPGA platform.

6 Discussion

As expected, the results on latency and jitter clearly show the advantages of using a dedicated path for real-time traffic inside the switch. Even under coordinated communication to avoid competition for the output ports, the processing of the destination addresses and the queuing mechanisms in the COTS switching fabric introduce a higher delay and uncertainty for real-time frames in relation to the custom switch. In addition, the latency and jitter of COTS switches depend on the particular implementation of the switching fabric and there are significant variations between different brands and models [6], [8]. Consequently, using the NIC-only architecture will always require the designer(s) to assume the worst-case delay when designing the schedules to assure conflict-free communication at runtime. This requirement leads to an inefficient utilization of the communication medium, especially when low-end COTS switches with high latency variations are used.

The reception logic in the current version of the custom NICs limit the throughput to a fraction of the line rate. We do not consider this characteristic as a major drawback, as safety-critical system are more interested in predictable latency than high throughput. However, we are currently addressing this bottleneck and will enable line speed operation in future revisions of the hardware designs.

While the NIC-only architecture is the simplest solution for systems operating exclusively with coordinated traffic, it should not be adopted for applications requiring the integration of real-time and best-effort traffic in the same network. The results showed that such an architecture is highly susceptible to dropped and/or delayed real-time frames even with small loads of coexisting best-effort traffic. The NIC/Switch and Switch-only architectures are both viable choices for real-time networks requiring support for both traffic classes, because they retain delivery guarantees for real-time frames regardless of the amount of coexisting best-effort traffic. However, real-time guarantees come at the expense of reduced bandwidth for best-effort stations, and best-effort traffic can even be starved under high loads of real-time traffic.

From the device utilization report, we see that the inclusion of the real-time path module in the COTS switch has a minor impact on the resource usage. The switch for the Switch-only architecture reports the highest cost as it must integrate the scheduling units and the programming interfaces. The reduced cost in resources and the easy integration of the real-time path as additional modules in the reference COTS architecture, make this solution a more effective, simpler, and cheaper alternative to high-end COTS switches that support traffic priorities and/or advanced scheduling algorithms [6], [31]. The early prototypes of the custom devices are still subject to further optimizations to reduce the footprint and add new functionalities, which we actually encourage with its release as an open-source framework.

Even though we focused our evaluation on a microsegmented topology to provide an homogeneous evaluation scenario, it is important to consider the potential application of each architecture to multiswitch topologies commonly used in modern Ethernet networks. The NIC-only architecture naturally supports multiple switches between stations and any topology because the architecture is based on COTS switches. However, the accumulated latency and jitter in a path will increase with the increase in the number of switches, making worst-case scenarios very pessimistic and reducing the efficiency of the real-time communication. The custom switches introduced for the NIC/Switch architecture are an interesting option to address the previous limitation. The configuration-free real-time path naturally expands across multiple switches, forming a logical bus topology for real-time frames within a larger Ethernet network. The resulting path would offer the determinism of legacy fieldbuses while exploiting the high-speed and integration of Ethernet networks. On the contrary, a major limitation of the Switch-only architecture is that real-time networks only support simple microsegmented topologies. This is a common limitation for solutions that centralize the synchronization and coordination in the switch [18]. Support for hard real-time communication on multiswitch topologies is still an open-problem in the area, and we are currently working on a detailed evaluation of these properties using our prototypes.

7 Conclusion

Building real-time capable Ethernet systems requires modifications on the network components to provide coordinated access to the communication resources. Modifications on different components bring different benefits and costs. Because of this, it is important for designers of real-time distributed applications to understand the associated tradeoffs of each solution, so they can select the solution that is best suited for their particular application.

In this work, we presented and evaluated three different architectures that enable real-time communication on switched Ethernet networks. Each architecture introduces specific modifications to the network components (NICs and/or switches), enabling support for time-triggered communication based on the Network Code framework.

The NIC-only architecture uses COTS switches and custom NICs with embedded programmable scheduling units. This architecture provides hard communication guarantees only when all stations use the same custom NICs, and synchronize for temporal isolation of transmitted traffic. The other two architectures use custom switches that enable coexistence of real-time and uncoordinated best-effort traffic in the same network. The switch of the NIC/Switch architecture retains the forwarding mechanisms of a COTS switch, but includes a dedicated path for coordinated traffic coming from the custom NICs. The Switch-only architecture includes a scheduling mechanism on each input port of the switch to coordinate the traffic coming from COTS NICs.

Using the NetFPGA platform as the unified prototyping technology, we implemented and evaluated hardware prototypes of the components for each architecture. We collected experimental data and characterized each solution in terms of throughput, latency, jitter, robustness of real-time guarantees against injected best-effort traffic, and cost in logical resources.

The resulting analysis is relevant for future developers of real-time communication systems, as it provides insights on determining which components to modify according to the specific requirements of a particular application. Additionally, we leveraged the advantages of the NetFPGA platform to prepare an easy-to-use evaluation platform for each component. The resulting prototypes, including hardware modules and software tools, are available as an open-source project for the academic and research community. This open approach allows the users to validate our results, and provides them with a powerful and low-cost framework to experiment with real-time Ethernet systems using the actual hardware. We expect this will help in the development of an integral solution to fit the communication requirements for different application domains.

Acknowledgments

This research was supported in part by NSERC DG 357121-2008, ORF-RE03-045, ORF-RE04-036, ORF-RE04-039, APCPJ 386797-09, CFI 20314 and CMC, ISOP IS09-06-037, and the industrial partners associated with these projects. Exchange Program (GSEP) funded by Foreign Affairs and International Trade Canada (DFAIT).

References

[1] M. Felser and T. Sauter, "The Fieldbus War: History or Short Break between Battles?" *Proc. IEEE Fourth Int'l Workshop Factory Comm. Systems*, pp. 73-80, 2002.

[2] J.-D. Decotignie, "Ethernet-Based Real-Time and Industrial Communications," *Proc. IEEE,* vol. 93, no. 6, pp. 1102-1117, June 2005.

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

[4] T. Sauter, "Integration Aspects in Automation - A Technology Survey," *Proc. IEEE Conf. Emerging Technologies and Factory Automation (ETFA '05),* vol. 2, pp. 255-263, Sept. 2005.

[5] E. Jasperneite and P. Neumann, "Switched Ethernet for Factory Communication," *Proc. IEEE Eighth Conf. Emerging Technologies Factory Automation (ETFA),* vol. 1, pp. 205-212, Oct. 2001.

[6] A. Jacobs, J. Wernicke, S. Oral, B. Gordon, and A. George, "Experimental Characterization of QoS in Commercial Ethernet Switches for Statistically Bounded Latency in Aircraft Networks," *Proc. IEEE 29th Int'l Conf. Local Computer Networks (LCN '04),* pp. 190-197, Nov. 2004.

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

[8] S. Fischmeister, R. Trausmuth, and I. Lee, "A Verifiable Language for Programming Communication Schedules," *IEEE Trans. Industrial Informatics,* vol. 5, no. 3, pp. 325-337, Aug. 2009.

[9] Official NetFPGA Project Webpage, http://www.netfpga.org, 2013.

[10] NetFPGA Open-Source Application Projects, http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/ProjectTable, 2013.

[11] R. Obermaisser, *Event-Triggered and Time-Triggered Control Paradigms - An Integrated Architecture.* Springer-Verlag, 2005.

[12] Ethernet Powerlink Standardization Group, http://www.ethernet-powerlink.org, 2013.

[13] C. Venkatramani and T. Cker Chiueh, "Supporting Real-Time Traffic on Ethernet," *Proc. Real-Time Systems Symp.,* pp. 282-286, Dec. 1994.

[14] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The Time-Triggered Ethernet (TTE) Design," *Proc. IEEE Eighth Int'l Symp. Object-Oriented Real-Time Distributed Computing (ISORC '05),* pp. 22-33, May 2005.

[15] P. Pedreiras, P. Gai, L. Almeida, and G. Buttazzo, "FTT-Ethernet: A Flexible Real-Time Communication Protocol that Supports Dynamic QoS Management on Ethernet-Based Systems," *IEEE Trans. Industrial Informatics,* vol. 1, no. 3, pp. 162-172, Aug. 2005.

[16] Z. Hanzalek, P. Burget, and P. Sucha, "Profinet IO IRT Message Scheduling with Temporal Constraints," *IEEE Trans. Industrial Informatics,* vol. 6, no. 3, pp. 369-380, Aug. 2010.

[17] K. Steinhammer, P. Grillinger, A. Ademaj, and H. Kopetz, "A Time-Triggered Ethernet (TTE) Switch," *Proc. Conf. Design, Automation and Test Europe (DATE),* vol. 1, pp. 794-799, Mar. 2006.

[18] R. Santos, R. Marau, P. Vieira, P. Pedreiras, A. Oliveira, and L. Almeida, "A Synthesizable Ethernet Switch with Enhanced Real-Time Features," *Proc. IEEE 35th Ann. Conf. Industrial Electronics,* pp. 2817-2824, Nov. 2009.

[19] L. Almeida, S. Fischmeister, M. Anand, and I. Lee, "A Dynamic Scheduling Approach to Designing Flexible Safety-Critical Systems," *Proc. Seventh ACM and IEEE Int'l Conf. Embedded Software (EmSoft),* pp. 67-74, 2007.

[20] S. Fischmeister and A. Azim, "Design Choices for High-Confidence Distributed Real-Time Software," *Proc. Int'l Symp. Leveraging Applications Formal Methods, Verification and Validation (ISoLA),* pp. 327-342, Oct. 2010.

[21] Q. Wang, S. Gopalakrishnan, X. Liu, and L. Sha, "A Switch Design for Real-Time Industrial Networks," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp.,* pp. 367-376, Apr. 2008.

[22] S. Varadarajan and T. Chiueh, "Ethereal: A Host-Transparent Real-Time Fast Ethernet Switch," *Proc. Sixth Int'l Conf. Network Protocols,* pp. 12-21, Oct. 1998.

[23] J. Real and A. Crespo, "Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal," *Real-Time Systems,* vol. 26, no. 2, pp. 161-197, 2004.

[24] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts," *Operating Systems Rev.,* vol. 36, no. SI, pp. 147-163, http://doi.acm.org/10.1145/844128.844143, Dec. 2002.

[25] *Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems,* IEEE Standard 1588, 2008.

[26] M. Anand, S. Fischmeister, and I. Lee, "An Analysis Framework for Network Code Programs," *Proc. IEEE Sixth Ann. ACM Conf. Embedded Software (EmSoft),* pp. 122-131, Oct. 2006,

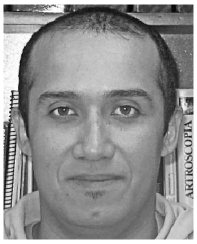
[27] G. Gibb, J. Lockwood, J. Naous, P. Hartke, and N. McKeown, "NetFPGA: An Open Platform for Teaching How to Build Gigabit-Rate Network Switches and Routers," *IEEE Trans. Education,* vol. 51, no. 3, pp. 364-369, Aug. 2008.

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

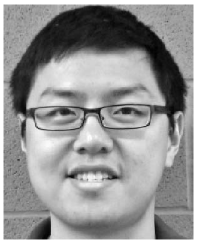
[29] A. Symington, S. Waharte, S. Julier, and N. Trigoni, "Probabilistic Target Detection by Camera-Equipped UAVs," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '10),* pp. 4076-4081, May 2010.

[30] M. Zhu, M. Liu, W. Chen, and F. He, "Autonomous Helicopter Navigating and Tracking with Stereo-Vision in IIM-USTC," *Proc. First Int'l Symp. Systems and Control Aerospace and Astronautics (ISSCA '06),* pp. 1388-1391, Jan. 2006.

[31] A. Martinez, G. Apostolopoulos, F. Alfaro, J. Sanchez, and J. Duato, "Efficient Deadline-Based QoS Algorithms for High-Performance Networks," *IEEE Trans. Computers,* vol. 57, no. 7, pp. 928-939, July 2008.



Gonzalo Carvajal received the MASc and PhD degrees in electrical engineering from the University of Concepcion, Chile, in 2009 and 2013, respectively. He is currently a research associate at the University of Waterloo, Canada. His research interests include embedded systems, hardware accelerated computation, real-time communication networks, and adaptive signal processing.



Chun Wah Wu received the BASc degree from the University of Waterloo in computer engineering in 2010. He is currently working toward the MASc degree at the University of Waterloo, Canada. He has also had industrial experience as an embedded software developer in wireless and Bluetooth technologies at Qualcomm and Research in Motion. In addition to real-time distributed systems, he is performing research in runtime verification of embedded systems.

Sebastian Fischmeister received the MASc degree in computer science from the Vienna University of Technology, Austria, and the PhD degree from the University of Salzburg, Austria. He is an associate professor in the Department of Electrical and Computer Engineering at the University of Waterloo, Canada. He received the APART stipend in 2005 and was a research associate at the University of Pennsylvania, until 2008. He performs systems research at the intersection of software technology, distributed systems, and formal methods. He is a member of the IEEE.