Guaranteeing the End-to-End Latency of an IMA System with an Increasing Workload

Min-Young Nam, Jaemyoun Lee, *Student Member*, *IEEE*, Kyung-Joon Park, *Member*, *IEEE*, Lui Sha, *Fellow*, *IEEE*, and Kyungtae Kang, *Member*, *IEEE*

Abstract—New features are often added incrementally to avionics systems to minimize the need for redesign and recertification. However, it then becomes necessary to check that the timing constraints of existing as well as new applications are met. We facilitate these checks by introducing a new data switch that bounds the latency of end-to-end communications across a network. This switch runs a clock-driven switching algorithm that is throughput-optimal with a bounded worst-case delay for all feasible traffic. We propose associated heuristics that determine whether the timing constraints of an integrated modular avionics (IMA) system network that uses this switch are met, even if new features have caused traffic to increase, and then search for alternative network configurations if necessary. Virtual integration is used to make a combined analysis of the worst-case delay in the network and the local buses of individual computing modules. This analysis considers the shared network topology, local hardware architectures, and specified IMA configurations. Our approach can be used by a system architect as an effective method for quickly determining which possible system architectures should be pursued to meet timing constraints, and it allows the cascading effects of changes to be tracked and managed. We demonstrate how these heuristics work through an example in which changes are made to an environmental monitoring facility within an avionics system that uses our switch.

Index Terms—Virtual integration, end-to-end latency, real-time switch, integrated modular avionics (IMA)

1 Introduction

A fast and predictable response is an essential characteristic of avionics and other real-time systems. Therefore it is critical to bound the end-to-end latency of data transmission across such systems. Avionics systems contain multiple computing modules running applications that receive sensor data and send commands to actuators. In modern avionics systems, these modules are usually connected through a high-speed switched network [1], which is shared by many applications so as to reduce weight, power consumption, and maintenance cost. Moreover, industry constantly seeks to increase the number of applications that can run in an avionics system, so as to meet requirements for new features, as well as to make the most of improvements in the hardware infrastructure. However, increasing the number of applications complicates system design since each application has to be assigned to a computing module, and as dictated by computational efficiency, additional modules may be needed to handle the increased workload.

The Integrated Modular Avionics (IMA) design concept has been proposed to simplify the development of airborne real-time networked systems and applications [1]. The IMA concept consists of an integrated architecture with application software that is portable across an assembly of common hardware modules. Multiple applications can be assigned to temporally scheduled IMA partitions to isolate different processor activities from each other.

An IMA network can connect computing modules running many diverse applications with different levels of timing criticalities. Assigning an application to a particular computing module or partition will influence many subsequent network configuration decisions. The ARINC 664 [2] standard, which describes a star-topology switched Ethernet avionics network based on IEEE 802.3, is designed to ensure that multiple data paths sharing network connections are scheduled in a way that guarantees the bandwidth of each path. However, even when the entire set of applications and the network infrastructure are known, determining a configuration that satisfies the worst-case latency requirements of each transaction is difficult and costly. In practice, this requires repeated tests, so that 'design' time can end up being predominantly spent on refining the partial solution that is already in place.

Because certification is mandatory in the avionics industry, it is highly desirable to employ established types of computing modules. Even if a newly designed module is more efficient than that which it is intended to replace, its introduction may require many existing applications to be reassigned to different modules: the reprogramming, testing, documentation, certification, and maintenance required to integrate a new module can incur a total cost that outweighs any gains in efficiency. Even if the logical behavior of a new computing module has been verified, the attempted

- *M.-Y. Nam and L. Sha are with the Department of Computer Science, University of Illinois at Urbana-Champaign, IL 61801. E-mail: {mnam, lrs}@illinois.edu.*
- *J. Lee and K. Kang are with the Department of Computer Science and Engineering, Hanyang University, Ansan 426-791, Korea. E-mail: {poken01, ktkang}@hanyang.ac.kr.*
- *K.-J. Park is with the Department of Information and Communication Engineering, Daegu Gyeongbuk Institute of Science and Technology, Daegu 711-873, Korea. E-mail: kjp@dgist.ac.kr.*

Manuscript received 12 Feb. 2012; revised 17 Sep. 2012; accepted 12 Dec. 2012. Date of publication 30 Dec. 2012; date of current version 09 June 2014. Recommended for acceptance by K. Li. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2012.300

integration of that module may result in cascading failures in related modules. In particular, the addition of new computing modules or the new constraints placed on network paths resulting from the introduction of additional applications can increase network traffic and significantly delay existing applications and modules. In fact, even quite limited changes to existing avionics systems can be challenging. For example, reconfiguring an existing module may not affect existing applications directly, but their performance may be impaired by the reallocation of system resources. These pressing issues motivate the search for faster and more cost-effective ways of introducing new computing modules and applications into avionics networks. This search would be easier if the architectures of networked systems were more readily extensible, so that networks could be updated more easily, and hence more frequently over their lifetimes.

One promising method of building extensible systems is virtual integration [3], in which an abstract model of a system is constructed and used for early analysis to improve the prognosis for high-level design decisions without an actual hardware or software implementation. Virtual integration is currently being adopted by the avionics industry as well as by other manufacturers of cyber-physical systems [4], [5], including the automotive and medical industries. This method is of particular relevance to these sectors because it allows designers to consider safety requirements at an early stage in the building or extension of networked systems, as well as making them more predictable and easier to upgrade.

In this paper, we propose a *real-time switching algorithm* that is specifically designed to facilitate the virtual integration of an IMA system because the switching algorithm that is used is a fundamental determinant of the manageability of a networked IMA system. Our switching algorithm is an implementation of an *optimal clearance-time* policy [6], which performs clock-driven scheduling [7] with guaranteed optimal throughput, so as to ensure that any feasible traffic is switched in two clock periods. This bound on the switching period makes it much easier to determine the worst-case delay in the switched network. And when we have found the worst-case network delay, we can model local data traffic using I/O configuration semantics to support the assignment of applications and estimate the worst-case local bus delays in the computing modules. We do this modeling by virtual integration, using our ASIIST [8] tool, which can make an accurate assessment of the worst-case end-to-end latency of an IMA system.

We also present a heuristic method for reconfiguring a switched network so as to re-satisfy the worst-case end-to-end latencies of all the applications in a system after the network traffic has increased as a result of adding applications, partitioning existing applications, or adding computing modules. We use examples to explore the implementation of this heuristic method in the context of our overall objective of achieving incremental design of real-time environmental monitoring avionics systems to satisfy timing constraints.

Our main contributions to the evolution of the avionics design process are the combination of our switch with heuristic network reconfiguration methods, and the use of virtual integration to obtain a more accurate evaluation of the end-to-end latency of IMA systems. Our approach allows us to model the sharing of a network among multiple applications running on different computing modules and even the sharing of local bus components among applications assigned to different partitions within a single computing module. Thus, we can determine whether almost any type of new feature will compromise the timing constraints of existing applications in a system. We also provide a heuristic that makes it easier to reconfigure a data network to cope with increased traffic, before a new configuration is implemented, by finding a network topology and configuration with a high likelihood of running both new and existing applications in a timely manner.

The remainder of this paper is organized as follows: In Section 2, we introduce the timing constraints required for real-time avionics systems and outline the issues that these raise. In Section 3, we show how to estimate the worst-case local bus delays in individual computing modules. In Section 4, we describe our real-time switching algorithm, which facilitates the analysis of the delay in a switched avionics network. In Section 5 we present a heuristic method of network management that ensures that the timing constraints of all applications continue to be met when an avionics system is modified. In Section 6 we assess the effectiveness of our heuristic method on some examples. In Section 7 we review related work. Finally, conclusions are offered in Section 8.

2 Problem Statement

2.1 Timing Constraints in Avionics Systems

An avionics system must be designed to handle external events predictably within appropriate timing constraints, even when several simultaneous events compete for the same service and resources. As new modules are added to an existing avionics system, the performance of other applications in the system should be degraded as little as possible; any unavoidable degradation should occur gracefully, predictably and in a localized way. To achieve these goals, all the computing modules in an avionics system must exhibit deterministic behavior; and the entire system must meet the time-constrained functional requirements of the processes that the system is designed to control before it can be adopted.

2.2 Research Motivations and Objectives

Avionics systems should require as little hardware as possible, so as to reduce aircraft weight and maintenance costs. An obvious example of this policy is the use of IMA for computation and the introduction of switches to simplify network cables. Avionics companies are also trying to use more commercial off-the-shelf (COTS) hardware components to reduce costs and accelerate system integration. However, because COTS devices are not purpose-built, their use can reduce the predictability of avionics system behavior.

IMA has been successfully applied to commercial airliners, such as the Boeing 777; but this required a specialized backplane data bus called the SAFEBus [9], designed to handle unpredictable I/O traffic [10]. Recent avionics applications that provide crew with visual assistance require more data throughput; this has resulted in the introduction of many hardware components with commodity internal bus architectures, such as those in the PCI family. All the known side-effects of introducing COTS components need to be accounted for when estimating the delay within a module. Since avionics systems are safety-critical, this involves a worst-case analysis.

There has been considerable research on the use of high-speed COTS switches and routers in a network infrastructure, but this has largely focused on handling traffic in a best-effort manner. We need to ensure that data will still be transferred in an accurate and timely manner, even after new modules or applications are introduced. Methods of finding worst-case delay bounds for real-time traffic have been proposed, but they do not isolate streams of traffic, and thus a change to just one stream requires the validity of all delay bounds to be reconfirmed. ARINC 664 [2] networks are able to achieve a degree of isolation through traffic prioritization. This guarantees that a lower-priority message does not preempt a higher-priority message, which in turn ensures that a lower-priority application will not delay a higher-priority application. This attribute contributes to system safety but not to system flexibility. To achieve both safety and flexibility, we need to identify the extent to which other network connections will be affected by an increase in the traffic through a switch. ARINC 664 requires that a message which is not delivered within its allowable maximum delay is discarded. If the high-priority traffic between some modules increases, applications running on other modules and transmitting lower-priority messages need to be re-evaluated because of the overall change in network latency, which is a considerable task in large networks. The present 'check and drop' policy needs to be replaced with something more accurate: we propose an incremental procedure based on predicted transmission delay.

3 Model-Based Design for Local Delay Analysis

To examine the end-to-end latency of a data connection across a network, it is necessary to consider the local delays in the source and destination modules. Even if a processor is schedulable, and the latency in the network is sufficiently small, a bottleneck in a local bus can intermittently cause applications to experience extended wait times to make do with old sensor data, and to respond late. Even though IMA permits the safe execution of multiple applications on a single processor, the I/O traffic generated by each application can still mutually interfere on the local buses, making it difficult to analyze the worst-case local delay.

3.1 Side-Effects of I/O In IMA Systems

In an IMA system, applications are assigned to time-slots, called IMA partitions, which are executed on a cyclic schedule, following a predefined timetable. The ARINC 653 standard [11] defines the APEX (Application Executive) interface between the applications and the operating system (OS) that provides this partitioned service. Only one application from a single partition can be executed at any given time. However, I/O data can flow between partitions. In other words, I/O does not have to be partitioned to be ARINC 653 compliant. This is beneficial to performance because large data can arrive for an application while a second application in a different partition is being run by the same processor. Data of this sort has to be saved in local memory until it can be processed, which occurs when the partition containing the receiving application is activated. While this permits reasonable performance, the interaction of I/O traffic created by applications in different partitions can cause side-effects that are not restricted to a single partition.

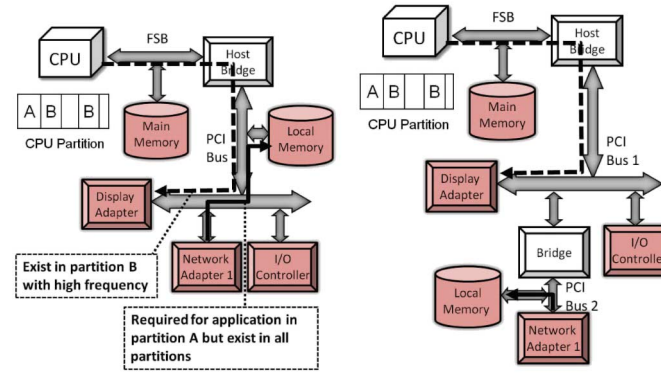


Fig. 1. Examples of different bus architectures.

When the aggregate amount of I/O data required by all of the applications that share a processor is large, the bus delay that results from this I/O traffic side-effect can become sufficiently significant to affect the design of a system.

The way in which this I/O side-effect interacts with different bus topologies is illustrated in Fig. 1. In both of the example topologies shown in this figure, an application is assigned to partition A and another to B. The application in partition A is constantly receiving a large amount of data through the network adapter, which has to be saved in local memory. The application in partition B must run frequently, and when it does, it sends a small amount of data to the display adapter. Fig. 1(a) is a simple architecture with a common PCI bus. The data that is received through the network adapter will delay data that is sent to the display adapter when the application in partition B is running. If this application has to wait for the other application's data transfer to complete, that wait time must be considered in analyzing schedulability. Since the application in partition B runs frequently, processor utilization (computation time divided by period) will increase greatly even with a small I/O wait time increase. The bus topology in Fig. 1(b) eliminates the contention between the two bus transactions. The drawback is an increased delay in reading from local memory, as data must now cross two bridges to get to the CPU. This example shows that a full evaluation of local I/O delay and its effect on the schedulability of an IMA system requires simultaneous consideration of many factors: bus architecture, I/O configuration, and IMA partition scheduling. This evaluation forms the basis for performance estimation and feasibility checking when a new application is added to an avionics system.

3.2 Local Delay Analysis Using ASIIST

Model-based engineering (MBE) is aimed at reducing system design errors by using high-level models that can be automatically analyzed before integration of the actual system. Thus the designer is guided towards an architecture which is more likely to be successful. We apply MBE by using the Architecture Analysis and Design Languages (AADL) [12] and the Application-Specific I/O Integration Support Tool (ASIIST) [8]) to analyze local bus delays. Given an initial AADL model of the system, including the hardware and software architectures, ASIIST can determine the worst-case local delays for a set of applications that are assigned to the partitions of an IMA system.

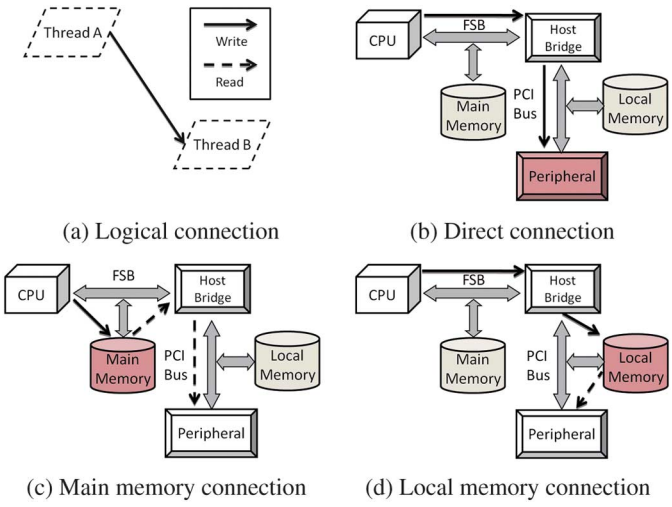


Fig. 2. Example I/O configurations.

The evaluation of different IMA partition assignments is facilitated by an extension to ASIIST which can process I/O configuration semantics and automatically update all I/O-related properties in an AADL model of a partition assignment. We have defined a number of I/O configurations that can be saved in the AADL model of an application. ASIIST can interpret these I/O configurations using pre-defined semantics to create and display physical data paths. Fig. 2 shows an example set of I/O configurations for a simple bus architecture. The logical connection between the two threads shown in Fig. 2(a) are realized into physical data-flows by creating a set of read and write bus transactions for each possible configuration. Thread A is an application that is running on the local CPU, whereas Thread B may be an application running on another network-connected CPU.

The following is a list of AADL properties that represent I/O configurations. These properties are associated with sending and receiving computing modules by attaching the postfixex `Src` and `Dst`.

- IO_Configuration (required): represents the type of I/O configuration and can take one of the values Direct, Main_Memory, or Local_Memory.
- IO_Controller (optional): refers to a device that is capable of initiating a read or write bus transaction to move copies of the I/O data.
- IO_Network_Adapter (optional): specifies the arrival and departure points of the I/O data on a hardware platform.
- IO_Local_Memory (optional): refers to a local storage memory component that is used to store data temporarily.

Table 1 shows the set of eight valid I/O configurations. More complicated configurations may exist, but we expect them to be very rare, and any unusual path that is required can be described by listing the hardware components in the path of a bus transaction. Fig. 3 shows the set of physical data paths that is generated for Case 5 of Table 1. The value of `IO_Configuration` specifies a type of configuration in which data is immediately read or written by the CPU. In Fig. 3(a), data is first written to the main memory and then copied by the I/O controller to the local memory, after which the network adapter reads the data from the local memory. This arrangement can be specified by assigning properties as follows:

`Src_IO_Configuration` → `Main_Memory`
`Src_IO_Controller` → I/O Controller
`Src_IO_Network_Adapter` → Network Adapter 1
`Src_IO_Local_Memory` → Local Memory

The properties of the destination module in Fig. 3(b) are assigned by giving values to `Dst_IO_Configuration`, `Dst_IO_Controller`, `Dst_IO_Network_Adapter`, and `Dst_IO_Local_Memory`. In this case, the values are identical because the same component names are used in the destination module, but the data paths that are derived are opposite in direction, and the read and write bus transactions are interchanged in the destination module. Other I/O configurations can be similarly interpreted. Some types of configuration, such as Cases 1 and 6, are used for local communication where the sending and receiving applications are assigned to the same CPU but different IMA partitions. These I/O configuration descriptors allow us to capture existing data-flows efficiently, so as to analyze the local bus delay and support the assignment of applications to processors.

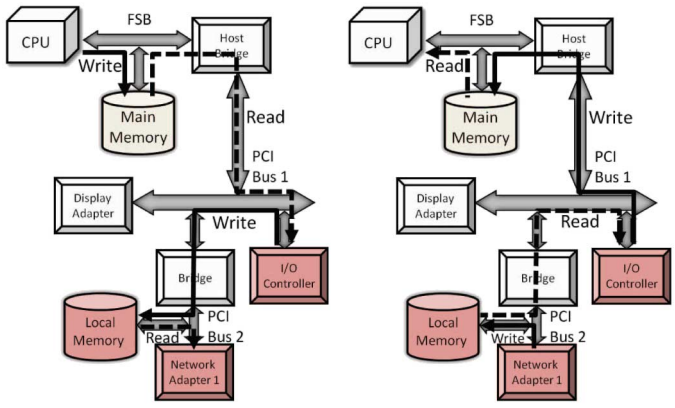
The user assigns applications to partitions by drag-and-drop in the GUI provided by ASIIST. If the application is moved to a processor that is connected to a different bus architecture, ASIIST would search for matching hardware components, based on their names and types, which would allow I/O configuration properties to be applied correctly. Fig. 1 has already shown how the same I/O configuration can produce different sets of physical data-flows.

When the data-flows have been generated, ASIIST finds all the data-flows which interfere. This involves knowing which

TABLE 1
Valid I/O Configurations

IO_Configuration	IO_Controller	IO_Network_Adapter	IO_Local_Memory	Comments
Direct	×	×	√	Case 1, Used for local communication
	×	√	×	Case 2
Main_Memory	√	√	×	Case 3
	×	√	×	Case 4
	√	√	√	Case 5
	√	×	√	Case 6, Used for local communication
Local_Memory	√	√	√	Case 7
	√	√	×	Case 8

(Same for source and destination modules)



(a) Source computing module (b) Destination computing module

Fig. 3. Physical data paths for Case 5.

component initiates the bus transaction, comparing bus protocols and examining partition assignments, within the context of IMA. For example, a bus transaction initiated by an application only exists during the partition to which the application is assigned. ASIIST assumes that bus transactions initiated by an I/O controller can exist during any partition as long as it is not configured differently. Depending on bus protocols, bus transaction types also affect how interfering and aggregated flows are determined for subsequent analysis. After the worst-case local delay has been calculated, memory access wait times (for reading or writing to or from memory) can be used to modify estimated execution times of applications. The schedulability analysis is then repeated to check whether any increase in partition sizes are necessary. If all the new partitions can still fit into the processor, the new application is considered to be schedulable.

If it is possible to define the operational modes (e.g. take-off, flying, landing) of an aircraft, they can be modeled by assigning AADL `modes` to applications and their connections to separate the corresponding data-flows. Hardware-based isolation can be achieved by using certifiable hardware components that regulate data traffic using pre-configured schedules. Our model does not require such AADL modes or hardware devices; but ASIIST supports model-based specification of both. When multiple modes are used, the user can select any combination and analyze the corresponding latency values. To facilitate hardware-based isolation, we have defined AADL property types that can be used to model real-time bridges which store incoming and outgoing network data and forward it during designated IMA partitions at configurable data-rates. We have extended the functionality of ASIIST to support these considerations for IMA systems and to provide a designer with analytical information based on IMA properties when alternative designs are being explored.

4 Design of a Real-time Switch for Avionics Networks

It is easier to design a practical real-time avionics system in which modules are connected by a packet-switched network using a switch with a known latency; and this also favors incremental development. We have designed a real-time switching algorithm that can guarantee a bounded switching delay with any feasible traffic. The underlying hardware fabric that we adopt is $N \times N$ virtual-output queue (VOQ) crossbar [13]. The data bus from each input intersects with the data bus of each output. The intersections can be turned on or off during runtime by the switch scheduling logic. To facilitate the design of the switching logic, crossbar switches transfer packets in fixed-size fragments called time-slots. Therefore, the scheduling logic works periodically: it determines a matching between inputs and outputs at the beginning of each time-slot; then all scheduled packets are transferred synchronously across the crossbar fabric, taking one time-slot; and then the next period starts, so on and so forth. We will also use the widely-adopted VOQ architecture, where each input maintains a virtual output queue for each output. VOQs eliminate head of line blocking [14], but packets from different inputs' VOQs still contend for the same output. Thus it is required to reduce this contention, so as to improve the hardware utilization.

4.1 Clock-Driven Scheduling as a Virtual-machine Task

Clock-driven scheduling [7] is widely applied to real-time virtual-machine tasks (VM-tasks) [15]–[17]. Suppose a VM-task $(\mathcal{P}, C)$ is served C times during each clock period $\mathcal{P}$. Let f_k^{ij} denote the kth real-time flow from input port I_i to output port O_j, where $k = 1, 2, \ldots, K_{ij}$, so that K_{ij} is the total number of flows from I_i to O_j. The flow f_k^{ij} is additionally associated with a VM-task $(\mathcal{P}, C_{ijk})$, because f_k^{ij} has C_{ijk} packets to be served. The delay to f_k^{ij} will be bounded as long as the switching algorithm can guarantee a worst-case bound on the time required to forward all C_{ijk} packets of this flow.

Let C_{ij} denote the total number of packets to be forwarded from I_i to O_j during one clock period, so that $C_{ij} = \sum_{k=1}^{K_{ij}} C_{ijk}$; and in this present analysis we will assume that each packet is served within a single time-slot. For the set of VM-tasks $\{(\mathcal{P}, C_{ijk})\}$, $i = 1, \ldots, N$, $j = 1, \ldots, N$, $k = 1, \ldots, K_{ij}$, we introduce a feasibility condition for the real-time traffic generated during each clock period $\mathcal{P}$, which contains L time-slots, as follows:

$$\sum_{j=1}^{N} C_{ij} \le L, i = 1, 2, \ldots, N, \quad (1)$$

$$\sum_{i=1}^{N} C_{ij} \le L, j = 1, 2, \ldots, N. \quad (2)$$

Infeasible traffic, which does not meet these conditions, is naturally unschedulable.

4.2 Clearance-Time-Optimal Switching Policies

In order to design a switching algorithm with a guaranteed delay, we introduce a clearance-time-optimal policy. Let $Q_{ij}(t)$ denote the number of packets in queue (i, j) at time-slot t. If every queue in the switch initially contains $Q_{ij}(0)$ packets, and there are no further arrivals, then there is *one-shot traffic*; and the clearance time is that required to serve every packet in the switch. We will express a clearance time as the number of time-slots required to serve every packet in the switch; it can easily be converted to a delay measured in

seconds by multiplying by the length of a time-slot. A switching policy that minimizes the clearance time is called a clearance-time-optimal policy.

Because no more than one packet can be switched at any port of a switch in one time-slot, due to the crossbar constraint, the minimum clearance time T^*_{clear} is obviously the maximum number of packets waiting at any input port, so that:

$$T^*_{\text{clear}} \geq \max\left(\max_i \sum_{j=1}^{N} Q_{ij}(0), \max_j \sum_{i=1}^{N} Q_{ij}(0)\right). \quad (3)$$

This minimum clearance time T^*_{clear} can be achieved by a *critical-port scheduling policy*. Port i is *critical* if its *weight* [18], that is the length of its queue, is no smaller than that of any other port; and a critical-port matching matrix M serves every critical port. A critical-port scheduling policy is one that generates a critical-port matching at every time-slot, which guarantees a minimal clearance time and optimal throughput for one-shot traffic. See the proof in Appendix.

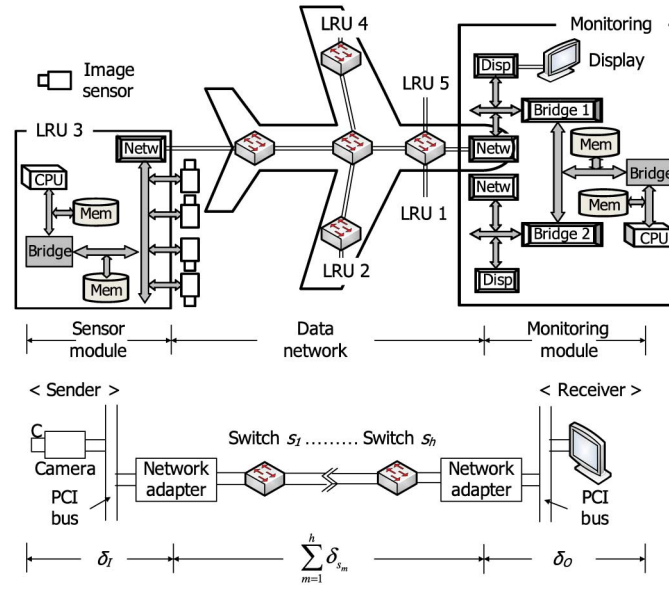


Fig. 4. Example architecture of an avionics system for environmental monitoring.

4.3 Design of a Real-Time Switch with Clock-Driven Scheduling

We can now combine the concepts of clearance-time optimality and a critical-port policy to design a switching algorithm with a VM-task architecture that can guarantee a bounded switching delay for any feasible traffic, as defined in (1) and (2).

The traffic arriving during each clock period is buffered and served in the next clock period. This is one-shot traffic, and permits clearance-time-optimal scheduling; and that in turn guarantees that any feasible traffic is served in two clock periods. In effect, we are accepting an additional switching delay of one clock period to ensure a deterministic delay bound of $2\mathcal{P}$.

Most switching algorithms do not provide a tight upper bound on worst-case delay, and their performance must therefore be assessed in terms of average delay. But the timing constraints on real-time traffic make a bound on the delay more important than a low average delay. Also the cost of an extra clock period which we propose to pay for this bound is not in any case excessive, because the clock period $\mathcal{P}$ is usually much shorter than the typical delay constraint for real-time applications. Also, existing network traffic remains bounded as new traffic is added, right up until the total traffic going through the switch becomes infeasible. This is a significant contribution to incremental development, as we will demonstrate on an example system architecture in the following section.

5 Ensuring Limited Worst-Case End-to-End Latency during Upgrades of Networked Avionics Systems

5.1 Target System Architecture

Fig. 4 is a diagrammatic representation of an aircraft's environmental monitoring system, which consists of several line-replaceable units (LRUs); each unit potentially is a complicated device such as a COTS computing module. The LRUs are sealed units that can be replaced quickly, reducing both system development cost and maintenance time. In the example system, sensor modules, which are LRUs containing cameras, allow a pilot to look in different directions without having to turn the aircraft; this feature can be found on military aircraft. These sensor modules are connected to a switch in the data network, which relays image data from the cameras to a designated LRU for monitoring, which we call a monitoring module.

In order to design such a system, we first need to ask many questions, such as: how many sensors can the network support in real time without damaging existing data-flows? and what is a feasible network topology? We need answers to these questions before beginning physical integration, when the discovery of scheduling problems can lead to expensive changes.

The environmental monitoring system shown in Fig. 4 can be partitioned into modules that perform computing tasks, and those that comprise the data network. Each sensor module, which sends data, consists of a network adapter and the circuits connecting it to cameras. The monitoring module, which receives data, consists of a network adapter and the circuits connecting it to monitoring devices. The image data has to be preprocessed so that real-time annotations and alerts can be provided to the pilot. The sensor and monitoring modules should also be able to act as common computing modules which can perform general tasks, so as to maximize hardware utilization.

The cabling that connects the network adapter on the sensor module to the network adapter on the monitoring module includes several switches that form the data network. If environmental monitoring is a new feature, then a system designer needs to know whether it can be supported by the existing data network, taking into account the data-flows generated by the modules already installed. The timing constraints of the combined operation of the existing and the new applications must also be verified.

5.2 Assessment of the Worst-Case End-to-End Latency

Fig. 4 shows potential sources of delay in the example architecture: let δ_I denote the worst-case delay introduced between a camera and its network adapter in the sensor modules; and let δ_O denote the worst-case delay introduced by internal bus communication between the network adapter and the pilot's monitor in the monitoring module. These modules contain processors, storage elements, and peripheral I/O devices, and use PCI-based COTS components for communication. We used ASIIST to analyze the delay incurred in bus communications between these complicated modules.

Each sensor module has a CPU, a main memory, a local memory and a network adapter, which are interconnected through a PCI bus to receive visual data from cameras and forward it to a monitoring module. The monitoring module has a general-purpose CPU which is shared with other applications, and a display adapter that is used to show the processed visual data to the pilot. We estimated the local worst-case bus delay δ_I in the sensor module to be 0.156 ms when four cameras are connected to a single PCI bus, and 0.313 ms for eight cameras; this assumes that each camera acquires 625 bytes of data in each successive period of 10 ms. In the airplane there are a total of five LRUs which are continuously sending their data to the monitoring module in the cockpit, as depicted in Fig. 4. The bandwidth of the front-side bus and the PCI bus bandwidth were assumed to be 19.2 Gb/s and 1,064 Gb/s respectively, and the maximum amount of data that can be retained in a bridge within any sensor or monitoring module is assumed to be 1 kb.

The local worst-case bus delay δ_O in the monitoring module incurred from storing the visual data from all the LRUs and forwarding them to the display adapter depends on the number of cameras that sends data through the network as well as the set of applications, bus architecture, and I/O configurations of the monitoring module. As for the data that arrives from the LRUs, they are initially saved in a local memory because of their large size. The data is then fetched from the local memory every 10 ms and sent for display every 100 ms. Any I/O interference from other existing applications are modeled and analyzed altogether to estimate a worst-case delay for δ_O.

Because the propagation delays between the switches and the modules are negligible, the total delay experienced by a packet traversing the data network between the network adapters of the sensor and those of the monitoring modules can be simplified to $\sum_{m=1}^{h} \delta_{s_m}$, where δ_{s_m} is the switching delay introduced by a switch s_m, and h is the hop count, which denotes the number of switches in the data network traversed by each packet.

The switching algorithm proposed in Section 4 can be used to ensure that the end-to-end network latency $\delta^{xx'}_{\text{network}}$ between the sensor module x and monitoring module x' has the following bound:

$$\delta^{xx'}_{\text{network}} = \sum_{m=1}^{h} \delta_{s_m} \leq \sum_{m=1}^{h} 2\mathcal{P}_m = \bar{\delta}^{xx'}_{\text{network}}, \tag{4}$$

where $\mathcal{P}_m$ is the clock period of the switching algorithm running in the mth switch in the route from x to x'. This formulation relies on the assumption that the traffic input to all the switches is feasible, implying that it must satisfy the following conditions:

$$\forall m \in \{1, 2, \ldots, h\}, \sum_{j=1}^{N} C^{xx'}_{m,ij} \leq L_m, i = 1, 2, \ldots, N, \tag{5}$$

$$\forall m \in \{1, 2, \ldots, h\}, \sum_{i=1}^{N} C^{xx'}_{m,ij} \leq L_m, j = 1, 2, \ldots, N, \tag{6}$$

where $C^{xx'}_{m,ij}$ denotes the total number of packets to be switched from I_i to O_j during one clock period at the mth switch from the sensor module x, which is running the set of VM-tasks $\{(\mathcal{P}_m, C^{x \to x'}_{m,ij})\}$, $i = 1, \ldots, N$, $j = 1, \ldots, N$. The length of the clock period $\mathcal{P}_m$ of the mth switch, in time-slots, is L_m, and it can be calculated as follows:

$$L_m = \left\lfloor \frac{\mathcal{C}^{xx'}_{p,m} \times \mathcal{P}^{xx'}_{m}}{\tau} \right\rfloor, \tag{7}$$

where $\mathcal{C}^{xx'}_{p,m}$ is the port capacity of the th switch from the sending module x, and τ is the size of a packet. Note that all packets are of the same fixed size and that the transmission of each packet takes exactly one time-slot. Finally, the worst-case end-to-end latency between x and x' can be calculated as follows:

$$\bar{\delta}^{xx'}_{\text{e2e}} = \delta_I + \bar{\delta}^{xx'}_{\text{network}} + \delta_O. \tag{8}$$

The system will meet the timing constrains if feasibility conditions (5) and (6) are satisfied for all switches, and the worst-case end-to-end latency on each connection between all pairs of modules connected to the data network does not exceed $\gamma^{xx'}_L$, which is the maximum latency allowable between x and x', so that

$$\forall (x \to x'), \bar{\delta}^{xx'}_{\text{e2e}} \leq \gamma^{xx'}_L. \tag{9}$$

5.3 Algorithms to Ensure the Achievement of Timing Constraints in an Environmental Monitoring System

When a new traffic flow occurs, because of the addition of a new sensor or monitoring module, or both, these new modules can be virtually integrated into an existing model of an avionics platform, which is then tested using the procedure shown in Fig. 5. To ensure that the avionics system will meet its timing constraints, we first check that the overall feasibility of the network traffic is not affected by the new flow. If the feasibility test fails, then the network topology has to be modified until it satisfies the feasibility condition. Secondly, we check the maximum latency of all flows. If the network fails this test, it is again necessary to modify the network topology to satisfy the latency requirement without compromising feasibility.

5.3.1 Traffic Feasibility

The additional flow which is created when a sensor module X and a monitoring module X' are installed in an existing avionics system, in which all the existing connections meet their timing constraints, only affects the feasibility of the

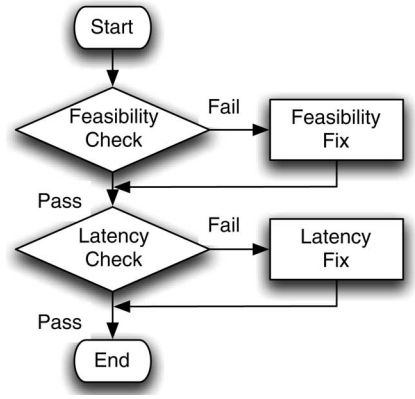


Fig. 5. Overview of the algorithms for virtual integration.

traffic through the switches on the path between X and X'. Thus feasibility only needs to be verified for these switches.

Algorithm 1 Feasibility check at a switch component

function NODEFEASIBILITYCHECK(G, F_e, S)
  P: clock period of S
  $I_1, \cdots, I_N$: input ports of S
  $O_1, \cdots, O_N$: output ports of S
  for $1 \leq i, j \leq N$ **do**
    $f_1, \cdots, f_l$: flows in F_e that pass through I_i and O_j
    $C_{ij} \leftarrow \sum_{k=1}^{l}$ (maximum number of packets of f_k in P)
  end for
  $L \leftarrow$ number of time-slots in P
  for $1 \leq i \leq N$ **do**
    if $\sum_{j=1}^{N} C_{ij} > L$ **then**
      return Failure
    end if
  end for
  for $1 \leq j \leq N$ **do**
    if $\sum_{i=1}^{N} C_{ij} > L$ **then**
      return Failure
    end if
  end for
  return Success
end function

We first connect the two modules X and X' to the switches that are physically closest to them. We then check the feasibility of traffic through a *single switch* using Algorithm 1, which adds the new flow to the virtually integrated avionics platform. The parameter G contains the network topology (including the link bandwidth information), F_e contains information about all the flows in the network, and S identifies the switch component at which feasibility is to be verified. We run Algorithm 1 for all the switches along the path, as shown in Algorithm 2.

Algorithm 2 Feasibility check for a path

function FEASIBILITYCHECK(G, F_e, X, X')
  Let $(S_1, S_2, \cdots, S_h)$ be the network path from X to X' in G
  for $1 \leq i \leq h$ **do**
    if NODEFEASIBILITYCHECK(G, F_e, S_i) $=$ Failure **then**
      return Failure
    end if
  end for
  return Success
end function

If this test fails, we have to modify the network topology. One simple modification is to reconnect the new modules to different switches in the network. This can be done using Algorithm 3. The parameters of this algorithm are the existing network topology (G) without the new switch components, the existing flows (F_e), and the new flow (X, X'). If Algorithm 3 succeeds, then the modified topology G' will satisfy the feasibility condition. This is the easiest way of achieving feasibility after the addition of traffic, and the cost of the resulting modifications is low. However these modifications may affect the feasibility of other existing flows. If X and X' do not have nearby switches that provide a feasible path, then we need to modify the topology of the existing network in another way.

Algorithm 3 Check nearby switches for a feasible path

function FINDFEASIBLEPATH(G, F_e, X, X')
  $F'_e \leftarrow F_e$ and a new flow (X, X')
  $C \leftarrow \{U_1, U_2, \cdots, U_m\}$ where U_i is a switch that can be directly connected to X
  $C' \leftarrow \{U'_1, U'_2, \cdots, U'_n\}$ where U'_i is a switch that can be directly connected to X'
  for all (i, j) s.t. $1 \leq i \leq m$ and $1 \leq j \leq n$ **do**
    $G' \leftarrow$ new topology obtained by adding switches between X and X', and connections (X, U_i) and (U'_j, X')
    if FeasibilityCheck$(G', F'_E, X, X') =$ Success **then**
      return G'
    end if
  end for
  return Failure
end function

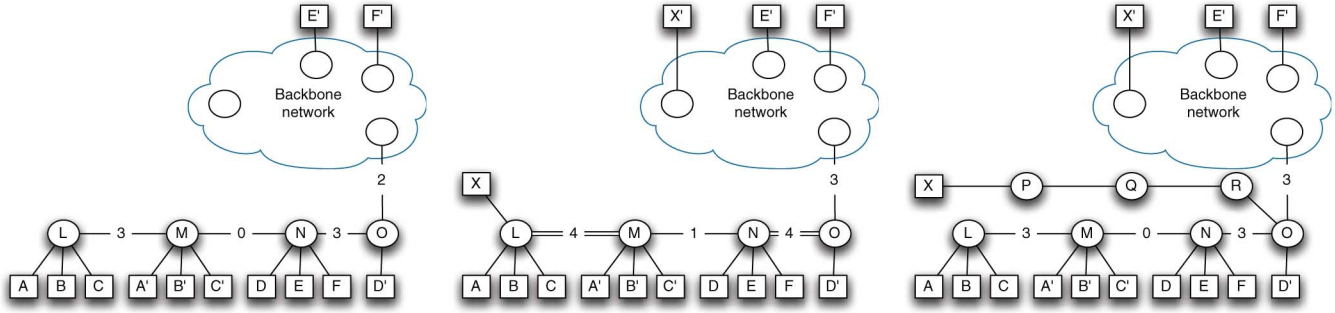


Fig. 6. Achieving feasibility by creating an alternative path. Each line is labeled with the number of flows that it carries. Unlabeled links have only one flow.

Fig. 6(a) shows an example of an avionics system which is stable in terms of timing constraints. It supports six flows, $A \to A'$, $B \to B'$, $C \to C'$, $D \to D'$, $E \to E'$, and $F \to F'$. We assume that the backbone network has sufficient bandwidth to handle all the traffic in the system. We will now analyze the configuration in terms of numbers of flows, referring to Algorithm 4; note that this is a simplified procedure and a practical feasibility test needs to check rates of packet transmission.

When the new flow $X \to X'$ is added, as shown in Fig. 6(b), the links $L \to M$ and $N \to O$ become overloaded, because they do not have sufficient capacity to handle four flows. To solve this problem, we can install three new switches, P, Q, and R, to create an alternative path $P \to Q \to R$ for traffic from X to X'. Algorithm 4 creates this new path, starting from the switch closest to the backbone network, by duplicating the switches that are carrying too much traffic, and thus creating a complete parallel path to the end-module. The number of new switches that are required is equal to the hop count, on the old path, from the switch closest to the backbone network to the end-module. The network topology produced by Algorithm 4 is shown in Fig. 6(c); and now the path between X and X' passes the feasibility test without making the worst-case latencies of the existing flows exceed their limits; thus all the flows in Fig. 6(c) are feasible. Although this method of restoring feasibility has no influence on other existing flows, it carries the additional cost of installing new switch components in the avionics system.

Algorithm 4 Install new switches to create a new feasible path

function FIXFEASIBILITY(G, F, X, X')
  $P \leftarrow$ the path from X to X'
  if P passes through the backbone network **then**
    $P_1 \leftarrow$ the path from X to the backbone network
    $P_2 \leftarrow$ the path from X' to the backbone network
  else
    $S' \leftarrow$ the path through the backbone network of the switches along P with the shortest hop count
    $P_1 \leftarrow$ the path from X to S'
    $P_2 \leftarrow$ the path from X' to S'
  end if
  FixFeasibilityPath(G, F, P_1)
  FixFeasibilityPath(G, F, P_2)
end function
function FIXFEASIBILITYPATH(G, F, P)
  $P = S_1 S_2 S_3 \cdots S_n$
  $l \leftarrow \arg\max_i$(S_i-S_{i+1} has overflowed)
  if l exists **then**
    Disconnect X from S_1
    Add new switches $S'_1, S'_2, \cdots, S'_l$
    Connect X-S'_1-S'_2-$\cdots$-S'_l-S_{l+1}
  end if
end function

Algorithm 5 Fix the latency

function LATENCYFIX G, F, X, X'
  $\gamma_L \leftarrow$ Required latency of the flow $X \to X'$
  While MaxLatency(G, F, X, X') $> \gamma_L$ **do**
    $XS_1S_2 \cdots S_hX'$ is the path from X to X'
    $C_1 \leftarrow$ the cost of connecting X to S_1
    $C_2 \leftarrow$ the cost of connecting X' to S_h
    If $C_1 < C_2$ **then**
      Disconnect X and S_1 in G
      Connect X and S_2 in G
    else
      Disconnect X' and S_h in G
      Connect X' and S_{h-1} in G
    end if
  end while
  return G
end function

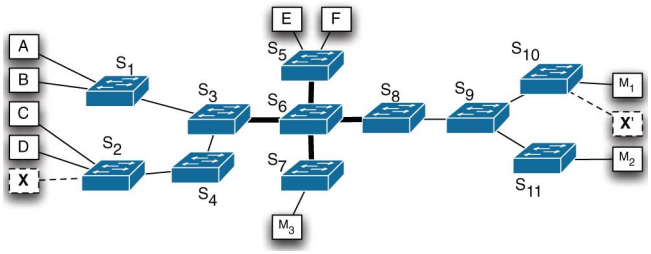


Fig. 7. Existing environmental monitoring architecture. Ordinary links are shown as thin lines, and backbone links as thick lines.

TABLE 2
Flows in the System of Fig. 7. Each Camera Supplies Data at 5 Mb/s

Sensor module	No. of cameras	Monitoring module	Traffic (pkts/$\mathcal{P}$)	Rate (Mb/s)
A	4	M_1	20	20
B	5	M_2	25	25
C	6	M_1	30	30
D	4	M_2	20	20
E	7	M_3	35	35
F	8	M_3	40	40
X	4	X'	20	20

5.3.2 End-to-End Latency

Once the traffic in the system is feasible, the next step is to check the end-to-end latency of each connection. Because the method of establishing feasibility described in the previous section does not affect any of the paths used by existing flows, the end-to-end latencies of these flows are not affected. Therefore it is only necessary to check the maximum latency of the new flow, which can be obtained from (8).

If the flow between X and X' fails this latency check, then the network topology needs to be modified, without violating the feasibility condition. This can be done using Algorithm 5, which reduces the number of hops between X and X' by removing a sufficient number of intermediate switches from the path to meet the maximum latency requirement γ_{L}. this can be done by omitting either the first or the last switch in the path. We select whichever is more effective in terms of an appropriate cost, such as cable length. Whichever switch is omitted, this method of restoring an acceptable end-to-end latency is simple and requires little modification of the system architecture.

6 Examples

We now present some examples based on an environmental monitoring application (EMA) running in a networked avionics system. When a new sensor module is added and the traffic flowing through the data network increases, we need to ensure timely delivery of the new flow while assuring that the worst-case end-to-end latencies of existing flows remain within their limits.

Fig. 7 illustrates a simple network architecture for environmental monitoring. The sensor modules that collect environmental images are attached to switches on the edge of the network through a normal link. These switches are also connected to the switched backbone network, in which the switches are connected by high-bandwidth backbone links. The switches at the edge of the network are assumed to be 4×4 crossbar switches with a port capacity of 100 Mb/s. The backbone switches are 16×16 with a port capacity of 100 Gb/s. The size of a packet is fixed at 10 kbits. If we also assume that all the switches have the same clock period $\mathcal{P}$, of 10 ms, the switches and the links at the edge of the network can handle 100 packets during each clock period, whereas the backbone switches and links can handle 100,000 packets.

In our example system, there are six sensor modules A, B, C, D, E, and F, each of which has a matching monitoring module M_1, M_2, M_1, M_2, M_3, and M_3. Thus there are six flows $A \rightarrow M_1$, $B \rightarrow M_2$, $C \rightarrow M_1$, $D \rightarrow M_2$, $E \rightarrow M_3$, and $F \rightarrow M_3$ in the network, before we add two new modules X and X' and the flow from X to X'. Switches S_2 and S_{10} are closest to X and X' respectively, and are therefore connected appropriately. Details of the flows are shown in Table 2.

6.1 Example 1: Feasibility Check

When modules X and X' are added to the network, it fails the feasibility check at switches S_8 and S_9, because S_8 has an output traffic of $20 + 25 + 30 + 20 + 20 = 115$ packets per clock period, which exceeds its link (or port) capacity of 100 packets in a clock period. The input traffic to S_9 also exceeds its link capacity.

Fig. 8 shows the solution to this problem which is found by Algorithm 4. Because S_8 and S_9 have insufficient link capacity, X' must be connected to a different switch. Therefore, we add two new switches, S_{12} and S_{13}, to the network and use them as relays connecting X' to S_8. The resulting topology passes the latency check because the traffic which pass through $S_8 \rightarrow S_9$ in Fig. 7 is now distributed between $S_8 \rightarrow S_9$ and $S_8 \rightarrow S_{12}$.

6.2 Example 2: Internal Bus Delay

After having derived an initial network topology, we need to estimate the worst-case internal bus delay for the sensor module as well as the monitoring module. The internal bus delay is then used in evaluating the end-to-end latency requirement and should be updated whenever any changes are made to the internal architecture of the modules. This includes, but are not limited to, adding additional sensors to the sensor modules, adding new applications to share the CPUs, or using alternative hardware architectures.

We assume that the CPUs of the sensor modules are not shared with any other application. Thus, the local delay (δ_I) depends on the number of sensors that are directly connected. As more sensors are added, the contention in the PCI bus that is directly connected to the network adapter increases. For the

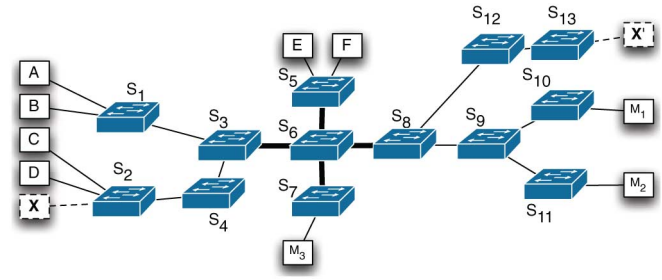


Fig. 8. Network topology modified by Algorithm 4 has feasible traffic.

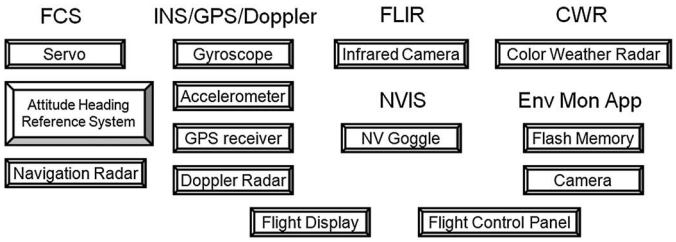


Fig. 9. Hardware components.

pre-existing monitoring modules (M_1, M_2, M_3), we also assume that it has a single EMA that processes the sensor data to forward it to display. These modules have the hardware architecture, as described in Fig. 4. Delay depends on the total amount of sensor data that is received through the network from multiple LRUs.

For the newly added monitoring module (X'), we will assume that its CPU is shared with a set of applications. An advanced Search And Rescue (SAR) system is considered to coexist with the EMA through IMA partitioning. The additional set of tasks and features that the SAR system supports includes the following:

- Flight Control System (FCS).
- Inertial Navigation System (INS)/Global Positioning System (GPS)/Doppler navigation system that uses motion sensor (accelerometers, gyroscopes) and satellite data.
- Forward looking infrared radar (FLIR) for human detection and imaging in sea water.
- Night vision imaging system (NVIS).
- Color Weather Radar (CWR) system for weather visualization via satellite.

Safety-critical hard real-time tasks, such as the FCS and INS/GPS/Doppler navigation system, which are directly related to the aircraft's safety are included. Other tasks are focused on visual display data because they constitute most of the heavy data traffic in modern avionics systems. In a IMA system, these tasks should safely share the resources of a computer system. With multi-functional systems and advanced hardware, more tasks are included to be analyzed for schedulability in modern avionics systems.

Special hardware components which are required for the new monitoring module to handle the aforementioned features are shown in Fig. 9. Some components such as the flight display or the flight control panel are used by multiple applications. These components must be connected through predefined buses and bridges to build a well connected hardware platform for the system.

Multiple data from external devices can come through a common network adapter depending on network protocols

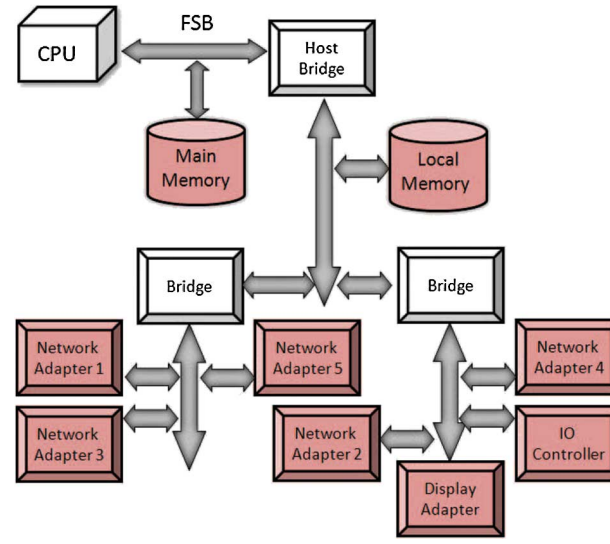


Fig. 10. Hardware model of monitoring module.

the device supports. We have assigned all the hardware components in Fig. 9 to 5 different network adapter devices. Table 3 shows each assignment. The hardware architecture is described in Fig. 10 which also includes a display adapter. EMA receives sensor data through Net_adapter_5 and sends its output through the display adapter. The local memory is used to save arriving sensor data during any IMA partition. This is initiated by an I/O controller. Each application is already assigned to a partition depending on other criteria such as fault containment and safety. Some applications will be sharing partitions as shown in Table 4. Any connection with large visual data traffics is configured to make use of the I/O controller and the local memory. Control related connections which have relatively small data sizes would directly store the data in the main memory to minimize latency for control. With the addition of the new application EMA in a shared partition (vm3), all partition sizes are update based on the local delay analysis and the memory access wait time that is applied to the each applications. The new partition size and schedule is proven to be valid with the given values shown in Table 4. Fig. 11 shows a screen shot of ASIIST after analyzing the local bus delay for the newly added application EMA. Beneath each incoming and outgoing connection, it shows all bus transactions that exist in each IMA partition due to the logical connection. For incoming data, which is sent from the Camera and received by the EMAThread has a worst-case of 27.39 ms delay. Data that is sent to the display adapter has a 19.64 ms delay. δ_O is the total value of 47.03 ms. Note that it is not recommended to change the bus architecture when adding a new application to an existing certified system. Generally, the cost of re-certifying the physical platform is very high

TABLE 3
Hardware Component to Peripheral Assignment

Hardware device	Network adapter
Servo, AHRS	Net_adapter_1
Nav Radar, Gyroscope, Infrared Cam	Net_adapter_2
Accelerometer, Color Weather Radar	Net_adapter_3
GPS Receiver, NVGoggle, FD	Net_adapter_4
Doppler Radar, Camera, FCP	Net_adapter_5

TABLE 4
Partition Assignment of Applications for X'

Application	Partitions	size (%)
Flight Control System (FCS)	vm0	5
INS/GPS/Doppler (Nav System)	vm1	1
FLIR, CWR	vm2	20
NVIS, EMA	vm3	28

Description	ID	Existing Partition	Source	Src Port	Bus	Destination	Dst Port	Period(sec)	Delay(sec)	Data Size(Byte)	PCI Transaction
Processor		Platform.MissionProcessor	MissionProcessor			MissionProcessor					
Partition		Platform.MissionProcessor.vm0	vm0			vm0					
Partition		Platform.MissionProcessor.vm1	vm1			vm1					
Partition		Platform.MissionProcessor.vm2	vm2			vm2					
Partition		Platform.MissionProcessor.vm3	vm3			vm3					
process		Platform.MissionProcessor.vm3	EMA			EMA					
thread			EMAThread			EMAThread		0.1			
Connection	17		Camera	outport2		EMAThread	oeConverterIn	0.1	0.027394690...	25000	
PCI PostedWrite	319	Platform.MissionProcessor.vm0	IOcontrol			localMemory		0.01	0.000399351...	25000	postedWrite
PCI PostedWrite	320	Platform.MissionProcessor.vm1	IOcontrol			localMemory		0.01	0.000399351...	25000	postedWrite
PCI PostedWrite	321	Platform.MissionProcessor.vm2	IOcontrol			localMemory		0.01	0.001158699...	25000	postedWrite
PCI PostedWrite	322	Platform.MissionProcessor.vm3	IOcontrol			localMemory		0.01	0.002319154...	25000	postedWrite
PCI DelayedRead	323	Platform.MissionProcessor.vm0	networkAdapter5			IOcontrol		0.01	0.011464045...	25000	delayedRead
PCI DelayedRead	324	Platform.MissionProcessor.vm1	networkAdapter5			IOcontrol		0.01	0.011464045...	25000	delayedRead
PCI DelayedRead	325	Platform.MissionProcessor.vm2	networkAdapter5			IOcontrol		0.01	0.013116459...	25000	delayedRead
PCI DelayedRead	326	Platform.MissionProcessor.vm3	networkAdapter5			IOcontrol		0.01	0.017802422...	25000	delayedRead
PCI DelayedRead	327	Platform.MissionProcessor.vm3	localMemory			vm3		0.1	0.007273113...	250000	delayedRead
Connection	6		EMAThread	lcdOut		Display_Adapter	inportLCD	0.1	0.019639421...	250000	
process		Platform.MissionProcessor.vm3	NVIS			NVIS					

Fig. 11. ASIIST local bus delay result for EMA in X' (4 Cameras).

and thus it should be considered as one of the last changes to try depending on cost comparison to other possible solutions such as adding a new module to the network.

6.3 Example 3: Latency Check

We now perform a latency check on the network topology shown in Fig. 8. Table 5 shows the worst-case latencies between sensor and monitoring modules measured by ASIIST, for the architecture and configuration of the PCI bus described in Section 5.2 and Fig. 10, the resulting bound on the end-to-end latency of each flow, and the maximum allowable latency of each flow. The parameters of each flow are given in Table 2 and the network topology is that shown in Fig. 8.

In this example, all the switches have the same clock period, so the upper bound on each latency is proportional to the path length. Two flows, $E \rightarrow M_3$ and $F \rightarrow M_3$, have three intermediate switches; two flows, $A \rightarrow M_1$ and $B \rightarrow M_2$, have six intermediate switches; and three flows, $C \rightarrow M_1$, $D \rightarrow M_2$, and $X \rightarrow X'$, have seven intermediate switches. Therefore, the upper bounds on the latencies of these group of flows are 60 ms, 120 ms, and 140 ms respectively. Summing the latencies in the sensor and monitoring modules, the maximum latency of the new flow $X \rightarrow X'$ is 128.8 ms if the new sensor module X contains four cameras, which is less than the maximum permitted latency of 180 ms. But if the number of cameras is increased to eight, the worst-case latency of the new flow is too high.

This problem can be addressed by reducing the hop count between X and X'. Fig. 12 shows one possible topology, in which X is moved from switch S_2 to S_4. Now, the flow from X to X' has only six intermediate switches between X and X' and so the maximum latency in the data network becomes 120 ms. This satisfies the latency requirement for this flow. A possible drawback of this approach is that it may increase the connection cost, because the newer is no longer connected to the physically closest switch.

TABLE 5
Latency in the System of Fig. 8. The Numbers Inside the Parenthesis Represent the Number of Cameras in the Sensor Module, Each of Which Supplies Data at 5 Mb/s

Sensor module	Monitoring module	δ_I	δ_O	$\bar{\delta}_{e2e}$	γ_L
A (4)	M_1	1.61	17.14	138.75	160
B (5)	M_2	2.01	15.29	137.30	160
C (6)	M_1	2.45	17.14	159.59	160
D (4)	M_2	1.61	15.29	156.90	160
E (7)	M_3	2.88	27.52	90.40	100
F (8)	M_3	3.32	27.52	90.84	100
X (4)	X'	1.61	47.03	168.64	**180**
X (8)	X'	3.32	90.20	213.52	**180**

7 Related Work

The most widely used standard for aircraft data networks has been ARINC 429 [19], which supports transmission speeds of up to 100 kb/s. The more recent ARINC 629 [20] was successfully applied to the Boeing 777, with a data-rate of 2 Mb/s, but this required customized hardware. Previous work on the analysis of IMA systems [21], [22] has relied on ARINC 629, sometimes in the form of its implementation, SAFEBUS, to protect against the side-effects caused by using COTS components. SAFEBUS is designed to be fault tolerant and deterministic, but it is an expensive proprietary bus.

As COTS modules become more advanced, and are more widely used by avionics companies to make cost reductions [23], [24], new standards, such as ARINC 664, are being developed to address the problems inherent in using these modules more directly [25]. Legacy ARINC 429 network modules can now be connected to ARINC 664 networks through appropriately specified gateways and routers, and methods of estimating end-to-end latencies in ARINC 664 networks are receiving increasing interest [26].

With the Ethernet-based speed of ARINC 664, customized internal hardware architectures using COTS components, such as PCI bridges and buses, are being developed and used

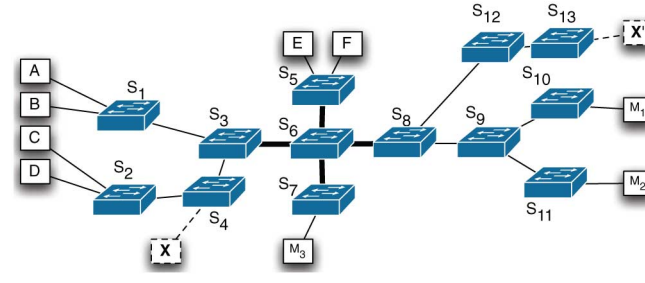


Fig. 12. Reducing the number of hops using Algorithm 5 to reduce end-to-end latency.

to replace SAFEBUS. While these architectures provide higher average performance, they require comprehensive analysis to guarantee latency. Model-based engineering (MBE) provides the designing environment for implementing such analysis that identifies sets of data flows that interferes with each other for every bus segment in a computing module.

Previous work that use MBE for latency analysis either consider local delay as homogenous [27], or simplify the bus architecture [28] for static analysis, or rely on testing and simulation to measure delay [29], [30]. Such approaches do not require a detailed hardware architecture model but also removes the potential of identifying problems and solutions from a hardware design perspective. They also rely on special hardware that guarantees such constant performance for analysis results to be accurate which is generally not the case with COTS components.

Considerable effort has been devoted to analyzing the performance of high-speed switches and routers and to obtaining their delay bounds [31], [32]. The scheduling of crossbar switches can be reduced to a search for a matching array within a graph, for which fast matching algorithms have already been developed [33]. However, because these algorithms are based on a stochastic model of traffic patterns, they provide asymptotic performance bounds that are not sufficient for real-time avionics systems in which predictability is critical. There have been some recent studies on the design of real-time switches [14], [34]. In particular, an efficient switch design for real-time applications was proposed by Qixin et al. [14], who provide a mechanism for guaranteeing that a certain number of communication slots are allocated to a task over a fixed time interval, under the assumption of deterministic and periodic real-time traffic.

To the best of our knowledge, none of the work reviewed above employs the virtual integration of a real-time switch to create support for the creation of feasible network paths; neither does it consider the internal I/O architecture within computing modules, which is a necessary step in determining the true end-to-end latency of a networked COTS-based avionics system. The key difference in our approach is its ability to evaluate the contribution of each system component to latency and to determine whether added workload will affect the delay bounds on existing network connections.

8 Conclusions

In hard real-time systems with tight timing constraints, a small change to one component can cause a sequence of adverse consequences: in this paper we have shown how to track and manage these cascading effects. We believe that this is the first example of support for system architecture decisions being provided so early in the design of networked avionics systems. We showed how AADL and ASIIST models can be used to analyze the local bus delay in a computing module of an integrated modular avionics system. We went on to propose a real-time switching algorithm based on clock-driven scheduling, and established that it can provide a bounded switching delay for any feasible traffic. Then we demonstrated how model-based engineering enables us to accurately assess the end-to-end latency of applications running on integrated modular avionics systems. In addition, we have proposed a heuristic algorithm to verify the real-time constraints of an avionics system in terms of traffic feasibility and end-to-end latency, and another heuristic to search, when necessary, for a new network topology that satisfies timing constraints. Finally, we constructed an example avionics platform containing instances of our real-time switches, analyzed a test case in which network traffic increases due to the installation of new computing modules, and presented results that demonstrate the effectiveness of our approach.

Acknowledgments

A preliminary version of this paper appeared in the Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'11). The authors would like to thank R. Bradford (Rockwell Collins) for his suggestions on improving our communication to the audience of the avionics community. This work was supported in part by ONR N00014-12-1-0046, by NSF CNS 13-02563, by Lockheed Martin 2009-00524 and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2013R1A1A1059188). K. Kang is the corresponding author of this paper.

References

[1] Aeronautical Radio Inc., "Design guidance for integrated modular avionics," SAE Int., ARINC Report, Tech. Rep. 651–1, Nov. 1997.
[2] Aeronautical Radio Inc., "Aircraft data network, part 7, avionics full-duplex switched ethernet network," SAE Int., ARINC Specification, Tech. Rep. 664P7–1, Sep. 2009.
[3] S. Mohan, M.-Y. Nam, R. Pellizoni, L. Sha, R. Bradford, and S. Fliginger, "Rapid early-phase virtual integration," in *Proc. IEEE Real-Time Syst. Symp. (RTSS'09)*, Dec. 2009, pp. 33–44.
[4] J. A. Stankovic, I. Lee, A. Mok, and R. Rajkumar, "Opportunities and obligations for physical computing systems," *Computer*, vol. 38, no. 11, pp. 23–31, Nov. 2005.
[5] L. Sha, S. Gopalakrishnan, X. Liu, and Q. Wang, "Cyber-physical systems: A new frontier," in *Proc. IEEE Int. Conf. Sens. Netw., Ubiquitous, Trustworthy Comput. (SUTC'08)*, Jun. 2008, pp. 1–9.
[6] G. R. Gupta, S. Sanghavi, and N. B. Shroff, "Node weighted scheduling," in *Proc. Int. Joint Conf. Meas. Modeling Comput. Syst. (Sigmetrics'09)*, Jun. 2009, pp. 97–108.
[7] J. W.-S. Liu, *Real-Time Systems*, Englewood Cliffs, NJ: Prentice Hall, 2000.
[8] M.-Y. Nam, R. Pellizzoni, L. Sha, and R. M. Bradford, "ASIIST: Application specific I/O integration support tool for real-time bus architecture designs," in *Proc. IEEE Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, June 2009, pp. 11–22.
[9] K. Hoyme and K. Driscoll, "SAFEbus," in *Proc. IEEE/AIAA Digital Avionics Syst. Conf. (DASC'92)*, Oct. 1992, pp. 68–73.
[10] K. Hoyme and K. Driscoll, "The airplane information management system: An integrated real-time flight-deck control system," in *Proc. IEEE Real-Time Syst. Symp. (RTSS'92)*, Dec. 1992, pp. 267–270.
[11] Aeronautical Radio Inc., "Avionics application software standard interface, part 1, required services," SAE Int., ARINC Specification, Tech. Rep. 653P1–3, Nov. 2010.
[12] P. Feiler, B. Lewis, and S. Vestal, "The SAE architecture analysis & design language (AADL): A standard for engineering performance critical systems," in *Proc. IEEE Conf. Comput.-Aided Control Syst. Des.*, Oct. 2006, pp. 1206–1211.
[13] M. J. Neely, E. Modiano, and Y.-S. Cheng, "Logarithmic delay for $N \times N$ packet switches under the crossbar constraint," *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 657–668, Jun. 2007.
[14] Q. Wang, S. Gopalakrishnan, X. Liu, and L. Sha, "A switch design for real-time industrial networks," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS'08)*, Apr. 2008, pp. 367–376.
[15] R. Davis and A. Burns, "Hierarchical fixed priority preemptive scheduling," in *Proc. IEEE Real-Time Syst. Symp. (RTSS'05)*, Dec. 2005, pp. 389–398.

[16] R. Davis and A. Burns, "Resource sharing in hierarchical fixed priority preemptive systems," in *Proc. IEEE Real-Time Syst. Symp. (RTSS'06)*, Dec. 2006, pp. 257–270.
[17] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proc. Euromicro Conf. Real-Time Syst. (ECRTS'03)*, Jul. 2003, pp. 151–158.
[18] K. Kang, K.-J. Park, L. Sha, and Q. Wang, "Design of a crossbar VOQ real-time switch with clock-driven scheduling for a guaranteed delay bound," *Real-Time Syst.*, vol. 49, no. 1, pp. 117–135, Jan. 2013.
[19] Aeronautical Radio Inc., "Digital information transfer system (DITS), part 1, functional description, electrical interface, label assignments and word formats," ARINC Specification, Tech. Rep. 429P1–17, May 2004.
[20] Aeronautical Radio Inc., "Multi-transmitter data bus, part 1, technical description," SAE Int., ARINC Specification, Tech. Rep. 629P1–5, Mar. 1999.
[21] N. Audsley and A. Wellings, "Analysing apex applications," in *Proc. IEEE Real-Time Syst.Symp. (RTSS'96)*, Dec. 1996, pp. 39–39.
[22] Y.-H. Lee, D. Kim, M. Younis, and J. Zhou, "Scheduling tool and algorithm for integrated modular avionics systems," in *Proc. IEEE/AIAA Digital Avionics Syst. Conf. (DASC'00)*, 2000, vol. 1, pp. 1C2/1–1C2/8.
[23] T. G. Baker, "Lessons Learned Integrating COTS into Systems," *Lect. Notes Comput. Sci.*, vol. 2255, pp. 21–30, 2002.
[24] B. Reynolds, "An application of COTS ethernet for avionics," in *Proc. IEEE/AIAA Digital Avionics Syst. Conf.*, vol. 2, pp. J13/1–J13/8, Nov. 1998.
[25] T. Schuster and D. Verma, "Networking concepts comparison for avionics architecture," *Proc. IEEE/AIAA Digital Avionics Syst. Conf.*, Oct. 2008, pp. 1.D.1-1–1.D.1–11.
[26] J.-L. Scharbarg, F. Ridouard, and C. Fraboul, "A probabilistic analysis of end-to-end delays on an AFDX avionic network," *IEEE Trans. Ind. Informat.*, vol. 5, no. 1, pp. 38–49, Feb. 2009.
[27] Z. Gu, S. Kodase, S. Wang, and K.G. Shin, "A model-based approach to system-level dependency and real-time analysis of embedded software," in *Proc. Real-Time Embedded Technol. Appl. Symp. (RTAS'03)*, May 2003, pp. 78–85.
[28] R. Varona-Gomez and E. Villar, "AADL simulation and performance analysis in SystemC," in *Proc. IEEE Int. Conf. Eng. Complex Comput. Syst. (ICECCS'11)*, Jun. 2009, pp. 323–328.
[29] Y. Zhu, Y. Dong, C. Ma, and F. Zhang, "A methodology of model-based testing for AADL flow latency in CPS," in *Proc. Int.Conf. Secure Softw. Integr. Rel. Improvement Companion (SSIRI-C'11)*, Jun. 2011, pp. 99–105.
[30] M. Lafaye, M. Gatti, D. Faura, and L. Pautet, "Model driven early exploration of IMA execution platform," in *Proc. IEEE/AIAA Digital Avionics Syst. Conf. (DASC'11)*, Oct. 2011, pp. 7A2-1–7A2-11.
[31] D. Shah, P. Giaccone, E. Leonardi, and B. Prabhakar, "Delay bounds for combined input and output switches with low speedups," *Perform. Eval.*, vol. 55, no. 1/2, pp. 113–128, Jan. 2004.
[32] D. Shah, P. Giaccone, and E. Leonardi, "Throughput region of finite-buffered networks," *IEEE Trans. Parallel Distrib. Syst.*, vol 18, no. 2, pp. 251–263, Feb. 2007.
[33] S. Deb, D. Shah, and S. Shakkottai, "Fast matching algorithms for repetitive optimization: An application to switch scheduling," in *Proc. Conf. Inform., Sci. Syst. (CISS'06)*, Mar. 2006, pp. 1266–1271.
[34] S. Gopalakrishnan, M. Caccamo, and L. Sha, "Switch scheduling and network design for real-time systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS'06)*, Apr. 2006, pp. 289–300.

Min-Young Nam received the BS degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2002. He received his MS degree in electrical and computer engineering from the Ohio State University, Columbus, in 2006, and the PhD degree from the Department of Computer Science, University of Illinois at Urbana-Champaign, in 2012 and is continuing research as a postdoctoral research associate. His current research interests include computer-aided system architecture design, architecture description languages, tool development for safety critical systems in avionics, and medical systems. He is a student member of the IEEE.

Jaemyoun Lee is a PhD student in the Department of Computer Science and Engineering, Hanyang University, Seoul, Korea. His research interests include modeling and analysis of cyber-physical systems and design of medical-grade protocols for wireless healthcare systems. He is a student member of the IEEE.

Kyung-Joon Park received the BS and MS degrees from the School of Electrical Engineering and PhD degree from the School of Electrical Engineering and Computer Science, Seoul National University, Korea. He was a postdoctoral research associate in the Department of Computer Science, University of Illinois at Urbana-Champaign, from 2006 to 2010. He was with Samsung Electronics, Suwon, Korea as a senior engineer, from 2005 to 2006. He is currently an assistant professor in the Department of Information and Communication Engineering, Daegu Gyeongbuk Institute of Science and Technology, Korea. His current research interests include modeling and analysis of cyber-physical systems and design of medical-grade protocols for wireless healthcare systems.

Lui Sha recieved the PhD degree from Carnegie Mellon University, Pittsburgh, PA, USA, in 1985. He is Donald B Gillies chair professor of Computer Science in University of Illinois at Urbana Champaign. He was elected as an IEEE Fellow "for technical leadership and research contributions, which enabled the transformation of real-time computing practice." He was elected as ACM Fellow "for contributions to real time systems." He is active in dependable real time computing systems research, and served on the National Academy of Science's study committee on certifiably dependable software.

Kyungtae Kang received the BS degree in computer science and engineering and the MS and PhD degrees in electrical engineering and computer science, from Seoul National University, Korea, in 1999, 2001, and 2007, respectively. From 2008 to 2010, he was a postdoctoral research associate with the University of Illinois at Urbana-Champaign. In 2011, he joined the Department of Computer Science and Engineering, Hanyang University, Korea, where he is currently an assistant professor. His research interests include primarily in systems, such as operating systems, wireless systems, distributed systems, real-time embedded systems, and interdisciplinary area of cyber-physical systems. He is a member of the ACM.