Limiting Worst-Case End-to-End Latency when Traffic Increases in a Switched Avionics Network*

Min-Young Nam, Eunsoo Seo, Lui Sha
Dept. of Computer Science
University of Illinois at Urbana-Champaign
{mnam, eseo2, lrs}@illinois.edu

Kyung-Joon Park
Dept. of ICE
DGIST
kjp@dgist.ac.kr

Kyungtae Kang†
Dept. of CSE
Hanyang University
ktkang@hanyang.ac.kr

Abstract

New features are often added incrementally to avionics systems. This avoids redesign and recertification but still requires verifying the timing constraints of both new and existing applications. We introduce a new switch that facilitates this verification by bounding the latency of end-to-end communication across a network. Our clock-driven real-time switching algorithm is throughput-optimal with a bounded worst-case delay for all feasible traffic. Associated heuristics can verify whether the timing constraints of an avionics network are met, after new features have caused traffic to increase, and then search for alternative network configurations if necessary. We show how these heuristics cope with changes to an example environmental monitoring architecture within an avionics system that incorporates our switch. Our approach to analysis can be used to determine, quickly but rigorously, which system architecture meet timing constraints; and it allows the system architect to manage the cascading effects of component changes in a comprehensive manner.

1 Introduction

Fast response is an essential characteristic of avionics and other real-time systems. Therefore we need to ensure that the end-to-end latency of data transmission across such a system is always bounded and controlled, so that the overall system behavior is predictable.

An avionics system consists of multiple computing modules connected through a switched network. Each module runs one or more applications, which can receive sensor data and send commands to actuators. There is usually pressure to increase the number of applications running in an avionics system, in order to meet requirements for new features and to make the most of the steadily improving performance of the hardware infrastructure. However, the presence of a large number of applications complicates the task of the system designer, who has to assign each new application to a computing module, and may eventually have to add extra modules to deal with the increasing workload.

Integrated modular avionics (IMA) [1] "is an" airborne real-time networked "systems". The network in an IMA connects computing modules running many diverse applications with different levels of criticality. Assigning an application to a particular module will influence many subsequent decisions on the network configuration. 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 share network connections in a scheduled manner which guarantees the bandwidth of each path. However, it is difficult and costly to find a new configuration that satisfies the worst-case latency requirements of each transaction, even if we have complete knowledge about the entire set of applications and the network infrastructure. In addition to the programming required, there is the necessity for extensive testing and certification. It is usually far more practical to modify an existing configuration, and what is called 'design time' is often spent in refining the partial solution provided by a system that is already installed in a successful aircraft.

Making limited changes to an existing avionics system is challenging: even if a reconfiguration does not directly affect existing applications, their performance may be impaired by the impact of newly added applications on system resources. Thus the search is on for more effective ways of introducing new applications or computing modules, with the aim of achieving substantial reductions in both cost and time to market. This search would be easier if the architectures of networked systems were more readily extensible, so that a network could be updated more frequently in a more manageable manner over its entire lifespan.

Future avionics systems are likely to be built on switched

*This work was supported by the research fund of Hanyang University (HY-2011-N).

†Corresponding author.

1533-2306/11 $26.00 © 2011 IEEE
DOI 10.1109/RTCSA.2011.9

networks [1] that share high-speed connections among multiple applications, in order to reduce weight and maintenance cost. In such a network, the switching algorithm that is used is a fundamental determinant of the manageability of the network. We therefore propose a *real-time switch* that is designed to make an avionics system easier to analyze. Our switching algorithm is an implementation of an *optimal clearance-time* policy [3], which guarantees maximum throughput: and it also uses clock-driven [4] scheduling, to ensure that any feasible traffic is switched within two clock periods. We then present a heuristic method for reconfiguring a switched network in a manner that satisfies the worst-case end-to-end latencies of all the applications in the system when the traffic increases, as a consequence of new applications or the addition of new computing modules. We use example studies to explore the combination of this heuristic with our switching algorithm, in the context of our overall objective of promoting the efficient incremental design of real-time avionics systems for environmental monitoring that can reliably meet timing constraints.

Our main contribution to the evolution of the IMA design process is to combine our switch design with a heuristic approach to network reconfiguration. This allows us to model the sharing of a network among multiple applications running on different computing modules within an IMA environment, and thus find out whether we can introduce a new feature without compromising the timing constraints of existing applications. We show how our heuristic method of network adjustment can be used, before any changes are implemented, to reduce the cost of reconfiguring a data network to cope with increased traffic. This is achieved by finding a network configuration that increases the likelihood that both new and existing applications will be able to meet their time constraints.

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 problem that we are going to address. In Section 3, we describe our real-time switching algorithm and its guaranteed switching period. In Section 4 we present a heuristic method of network management that ensures that the timing constraints of all applications continue to be met when a system is modified. In Section 5 we assess the effectiveness of our heuristic with some examples. In Section 6 we review related work. Finally, in Section 7 we conclude this paper.

2 Problem statements

2.1 Timing constraints in avionics systems

An avionics system must be designed to handle external events predictably within the appropriate timing constraints. The reactions of the system must be precisely planned and fully predictable, even when several simultaneous events compete for the same service and resources. Further, when new modules are added to an existing avionics system, there should be as little drop in performance possible; and any unavoidable degradation should occur gracefully, predictably and in a localized way.

Avionics system modules must exhibit deterministic and predictable behavior in order to be licensed for real-time applications; and the ability of an avionics system to meet the time-constrained functional requirements of the process that it is designed to control needs to be assessed before it is adopted. This assessment should focus on the predictability of system behavior, and can subsequently provide the basis for further evaluation of the system design and implementation, requirements correlation, and the interpretation of user feedback.

2.2 Research motivations and objectives

Avionics systems should run on the smallest possible amount of hardware, so as to reduce aircraft weight and maintenance costs. An obvious example of this policy is the use of switches to simplify cables. However this creates a switched network in which data-paths are shared, and introduce the need to ensure that data will still be transferred in an accurate and timely manner, even when new modules or applications are introduced.

The concept of IMA has been around for more than ten years, and avionics companies are trying to make it cost-effective by using a higher proportion of commercial off-the-shelf (COTS) computing modules to reduce overall costs and accelerate system integration. However, COTS modules are not designed for predictability, and their use makes it more difficult to construct an IMA system with predictable behavior. Thus we have seen quite a lot of research taking place on the use of high-speed COTS switches or routers, but this has largely been focused on handling traffic in a best-effort manner. Although ways of finding worst-case delay bounds for real-time traffic have been proposed, these methods do not isolate streams of traffic: thus a change to one stream requires the whole network to be checked to ensure that all the delay bounds are still valid.

ARINC 664 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 is a contribution to safety but not to flexibility. To improve flexibility, we need to identify the extent to which other network connections will be affected by an increase in the traffic through a switch. When a network fails to deliver a message within its allowable maximum delay, that message is discarded. Thus, if the high-priority traffic between some mod-

ules increases, applications running on other modules and transmitting lower-priority message need to be re-evaluated because of the overall change in network latency. This is a big job when the network is large. It would be much better to have an incremental development procedure that would make a switch's transmission delay predictable, replacing the present policy of 'check and drop'.

3 Design of a real-time switch for avionics networks

One way to create predictable real-time systems is to build them entirely from temporally deterministic components (i.e., computing modules with a predictable execution time for each task), and a network technology with a predictable delivery time for each message. By the same token, 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. The delay incurred by such a switch should be predictable, even in the worst case and the worst-case switching delay should not change as new network traffic is added, at least until a clearly identifiable limit is reached.

3.1 Real-time switching algorithm

To meet the considerations above, 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 [5]. 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 [6], 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.

3.1.1 Clock-driven scheduling as a virtual-machine task

Clock-driven scheduling [4] is widely applied to real-time virtual-machine tasks (VM-tasks) [7, 8, 9]. 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:

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

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

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

3.1.2 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, which is what we will call 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.

We can now show that a critical-port policy is also clearance-time optimal, as follows:

Proposition 1 *A switching policy is clearance-time optimal if and only if it is a critical-port policy.*

Proof For any clearance-time-optimal policy, the inequality $Q_i(s) \leq T^*_{\text{clear}} - s$ holds at every port i at the start of any time-slot $s < T^*_{\text{clear}}$. If it did not, the crossbar constraint would mean that the corresponding port could not be cleared by T^*_{clear}. Similarly, it is apparent that any ports at which the initial length of the queue $Q_i(0)$ was T^*_{clear} will now have a queue of length $Q_i(s)$, which equals $T^*_{\text{clear}} - s$. Consequently, each of the critical ports for which $t = s$ has a queue of length $T^*_{\text{clear}} - s$. If any critical ports are not served during time-slot s, then these ports cannot be cleared by T^*_{clear}. Hence, every clearance-time-optimal policy is a critical-port policy.

Now suppose we have a critical-port policy. Since all the ports with the highest weight, meaning the longest queues, at the start of any time-slot are critical ports, those ports will be served and the lengths of their queues will decrease by one. Hence, a critical-port policy must also be a clearance-time optimal-policy. ■

3.1.3 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. And 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.

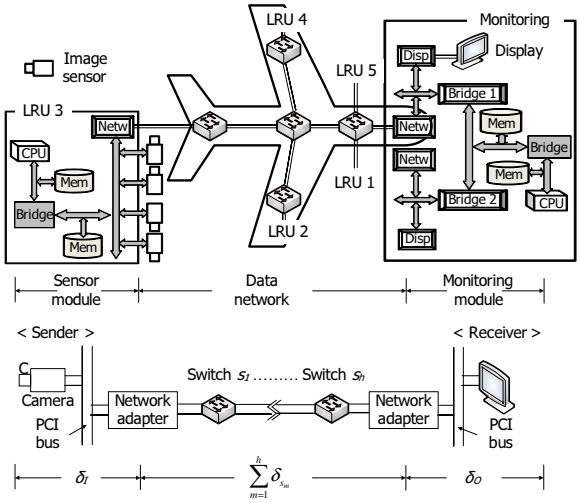


Figure 1: Example architecture of an avionics system for environmental monitoring.

4 Ensuring limited worst-case end-to-end latency during upgrades of networked IMA systems

4.1 Target system architecture

Fig. 1 is a diagrammatic representation of an aircraft's environmental monitoring system, which consists of several line-replaceable units (LRUs); each unit might potentially be 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. 1 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 a part of 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 dataflows generated by the modules already installed. The timing constraints of the combined operation of the existing and the new applications must also be verified.

4.2 Assessment of the worst-case end-to-end latency

Fig. 1 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. A software tool can be used to analyze the delay incurred in bus communications between these complicated modules; we have used one such tool, called ASIIST (Application-Specific I/O Integration Support Tool) [10].

In our previous work [11], we analyzed local latency using ASIIST for the IMA system architecture shown in Fig. 1. Each sensor modules 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 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. 1. 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. If there are four cameras in each sensor LRU, then the local worst-case bus delay δ_O in the monitoring module incurred in storing the visual data from all the LRUs and forwarding them to the display adapter will be 1.66 ms; for eight cameras it is 2.4 ms. We assumed that the set of applications [11] running together in the monitoring module generated contention in the local PCI buses. The data is fetched from the local memory every 10 ms and sent for display every 100 ms. Details about latency measurement in the LRU using ASIIST can be found elsewhere [11].

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 3 can be used to ensure that the end-to-end network latency $\delta_{\text{network}}^{xx'}$ between the sensor module x and monitoring module x' has the following bound:

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

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:

$$\forall m \in \{1, \ldots, h\}, \sum_{j=1}^{N} C_{m,ij}^{xx'} \leq L_m, i = 1, \ldots, N, \qquad (5)$$

$$\forall m \in \{1, \ldots, h\}, \sum_{i=1}^{N} C_{m,ij}^{xx'} \leq L_m, j = 1, \ldots, N, \qquad (6)$$

where $C_{m,ij}^{xx'}$ 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 $\{(\mathcal{P}_m, C_{m,ij}^{x \to x'})\}$, $i = 1, \ldots, N$, $j = 1, \ldots, N$. 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:

$$L_m = \lfloor (\mathcal{C}_{p,m}^{xx'} \times \mathcal{P}_m^{xx'})/\tau \rfloor, \qquad (7)$$

where $\mathcal{C}_{p,m}^{xx'}$ is the port capacity of the mth 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:

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

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 $\gamma_L^{xx'}$, which is the maximum latency allowable between x and x', so that

$$\forall(x \to x'),\ \bar{\delta}_{\text{e2e}}^{xx'} \leq \gamma_{\text{L}}^{xx'}. \quad (9)$$

4.3 Algorithms to guarantee the timing constraints of an environmental monitoring system

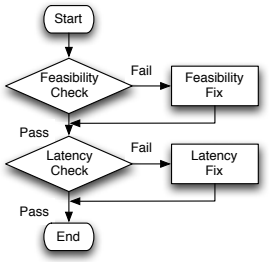


Figure 2: Overview of the algorithms for virtual integration.

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 a IMA platform, which is then tested using the procedure shown in Fig. 2. To ensure that the IMA 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.

4.3.1 Traffic feasibility

The additional flow which is created when a sensor module X and a monitoring module X' are installed in an existing IMA system, in which all the existing connections meet their timing constraints, only affects the feasibility of the traffic through the switches on the path between X and X'. Thus feasibility only needs to be verified for these switches.

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 IMA 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 to be verified. We run Algorithm 1 for all the switches along the path, as shown in Algorithm 2.

Algorithm 1 Feasibility check at a switch component

```
1: function NodeFeasibilityCheck(G, F_e, S)
2:     P: clock period of S.
3:     I_1, ..., I_N: input ports of S.
4:     O_1, ..., O_N: output ports of S.
5:     for 1 ≤ i, j ≤ N do
6:         f_1, ..., f_l: the flows in F_e that pass through I_i and O_j.
7:         C_ij ← Σ_{k=1}^{l} (Maximum number of packets of f_k in P)
8:     end for
9:     L ← the number of time-slots in P.
10:    for 1 ≤ i ≤ N do
11:        if Σ_{j=1}^{N} C_ij > L then
12:            return Failure
13:        end if
14:    end for
15:    for 1 ≤ j ≤ N do
16:        if Σ_{i=1}^{N} C_ij > L then
17:            return Failure
18:        end if
19:    end for
20:    return Success
21: end function
```

Algorithm 2 Feasibility check for a path

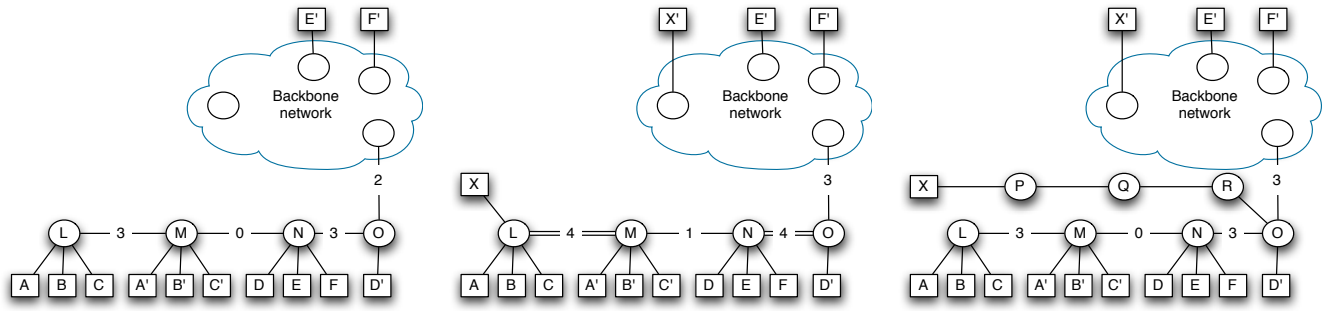
```
1: function FeasibilityCheck(G, F_e, X, X')
2:     Let (S_1, S_2, ..., S_h) be the network path from X to X' in G
3:     for 1 ≤ i ≤ h do
4:         if FeasibilityCheck(G, F_e, S_i)=Failure then
5:             return Failure
6:         end if
7:     end for
8:     return Success
9: end function
```

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. Alternatively, if X and X' do not have nearby switches that satisfy the feasibility condition, we need to modify the topology of the existing network in another way.

Fig. 3a shows an example of an IMA system which is stable in terms of timing constraints. In this configuration, there are 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.

As a result of the new flow $X \to X'$, shown in Fig. 3b, the links $L \to M$ and $N \to O$ are over loaded, because they do not have sufficient capacity to handle four flows[1]. To solve this problem, we can install three new switches, P, Q, and

[1]Our actual feasibility test checks the rate of packet transmission, but here we present a simplified argument in terms of numbers of flows.



(a) Original network (b) Infeasible path (c) Feasible path

Figure 3: 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.

Algorithm 3 Checking 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

Algorithm 4 Installing a new switch to create a 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 with the shortest hop count to the backbone network of the switches along P
    $P_1 \leftarrow$ 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)
  $P = S_1 S_2 S_3 \cdots S_n$
  $l \leftarrow \arg\max_i (S_i\text{-}S_{i+1}$ is 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

R to create an alternative path $P \rightarrow Q \rightarrow R$ for traffic from X to X'. Algorithm 4 creates this new path, starting from the switch closest to the backbone network, by duplicating switches carrying too much traffic and thus creating a 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. 3c; and now the path between X and X' passes the feasibility check without making the worst-case latencies of the existing flows exceed their limits; thus all the flows in Fig. 3c are feasible.

4.3.2 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 cause a change in 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, while continuing to satisfy the feasibility condition. This can be done using Algorithm 5, which reduces the number of hops between X and X' by removing sufficient intermediate switches from the path to meet the maximum latency requirement γ_{L}. Two possible methods of doing this are by omitting the first or last switch in the path. We select whichever option is more effective in terms of some cost, such as cable length.

5 Examples

We will now present some examples based on an environmental monitoring application running in a networked avionics system. When a new sensor module is added and the traffic flowing through the data network is increased, 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.

Algorithm 5 Latency fix

```
1: function LatencyFix(G, F, X, X′)
2:     γL ← Required latency of the flow X → X′
3:     while MaxLatency(G, F, X, X′) > γL do
4:         XS1S2 · · · ShX′ is the path from X to X′
5:         C1 ← the cost of connecting X to S1
6:         C2 ← the cost of connecting X′ to Sh
7:         if C1 < C2 then
8:             Disconnect X and S1 in G
9:             Connect X and S2 in G
10:        else
11:            Disconnect X′ and Sh in G
12:            Connect X′ and Sh−1 in G
13:        end if
14:    end while
15:    return G
16: end function
```

Fig. 4 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 assumes 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.

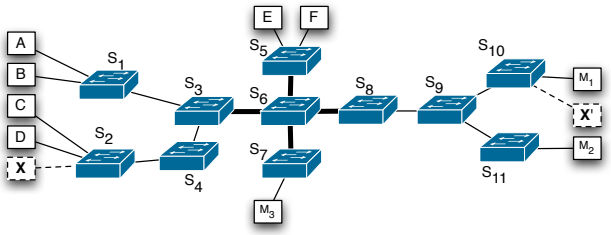


Figure 4: Existing environmental monitoring architecture. Ordinary links are shown as thin lines, and backbone links as thick lines.

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 $A \to M_1$, $B \to M_2$, $C \to M_1$, $D \to M_2$, $E \to M_3$, and $F \to M_3$ 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 1.

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

Table 1: Flows in the system of Fig. 4. Each camera supplies data at 5 Mb/s.

5.1 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. 5 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 feasibility check because the traffic which went through $S_8 \to S_9$ in Fig. 4 is now distributed between $S_8 \to S_9$ and $S_8 \to S_{12}$.

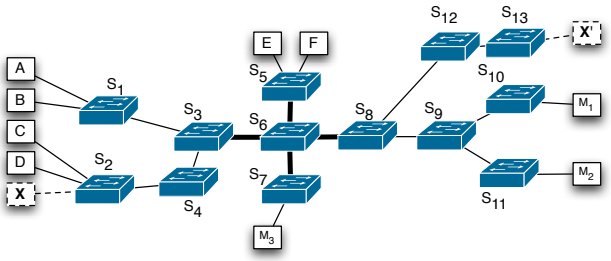


Figure 5: Network topology modified by Algorithm 4 has feasible traffic.

5.2 Example 2: latency check

We now perform a latency check on the network topology shown in Fig. 5. Table 2 shows the worst-case latencies between sensor and monitoring modules measured by ASI-IST, for the architecture and configuration of the PCI bus described in Section 4.2, 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 1 and the network topology is that shown in Fig. 5.

In this example, all the switches have the same clock period, so the upper bound on each latency is proportional to

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	7.16	148.77	**150**
X (8)	X'	3.32	14.45	157.75	**150**
					(ms)

Table 2: Latency in the system of Fig. 5. The numbers inside the parenthesis represent the number of cameras in the sensor module, each of which supplies data at 5 Mb/s.

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 148.8 ms if the new sensor module X contain four cameras, which is less than the maximum permitted latency of 150 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. 6 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.

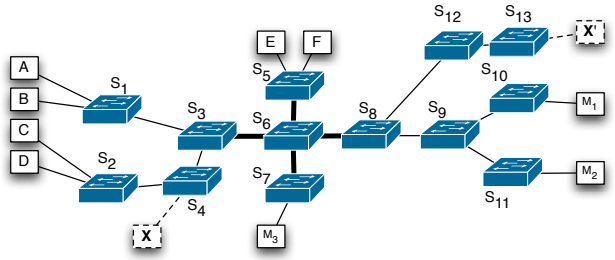


Figure 6: Reducing the number of hops using Algorithm 5 to reduce end-to-end latency.

6 Related work

The most widely used standard for aircraft data networks has been ARINC 429 [12], which supports transmission speeds of up to 100 kb/s. The more recent ARINC 629 [13] was successfully applied to the Boeing 777, with a data-rate of 2 Mb/s, but this required customized hardware. As COTS modules become more advanced, and are more widely used by avionics companies to reduce costs [14, 15], we find new standards, such as ARINC 664, referring more directly to the use of COTS modules [16]. Legacy ARINC 429 network modules can be mapped 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 [17].

Considerable effort has been devoted to analyzing the performance of high-speed switches and routers, and obtaining delay bounds [18, 19]. The scheduling of crossbar switches reduces to the search for a matching array within a graph, and fast matching algorithms have been developed for this purpose [20]. However, these are based on a stochastic model of traffic patterns, and provide asymptotic performance bounds that are not sufficient for real-time systems in which predictability is key.

There have been some recent studies on the design of real-time switches [21, 6]. In particular, an efficient switch design for real-time applications was proposed by Qixin et al. [6], 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 involves the virtual integration of real-time switch support, which is necessary to analyze the true end-to-end latency of a networked COTS-based IMA system. The key benefit of our real-time switch is our ability to configure and evaluate its contribution to latency so that added features will not effect the guaranteed delay bounds of existing network connections.

7 Conclusions

We have proposed a real-time switch which is based on the concept of clock-driven scheduling, and have shown that it can provide a bounded switching delay for any feasible traffic. This facilitates the verification of timing constraints by bounding the latency of end-to-end communication in a networked IMA systems. We constructed an example IMA platform containing instances of our real-time switches, and we explored the situation in which network traffic increases because of the installation of new computing modules. Further, we have proposed a heuristic algorithm to verify the real-time constraints of an IMA system in terms of feasibility and end-to-end latency, and another which can search for a new network topology that satisfies timing constraints if necessary.

In hard real-time systems in which there are tight timing

constraints, a small change in one component can cause a cascade of adverse consequences. Our analysis makes it possible to track and manage these cascading effects. We believe that this is the first attempt to provide support for system architecture decisions so early in the design of networked IMA systems.

Acknowledgment

We would like to thank R. Bradford of Rockwell Collins for his suggestions on improving our communication to audience in the avionics community.

References

[1] Aeronautical Radio Inc., Design guidance for Integrated Modular Avionics, ARINC Report 651-1, Nov. 1997.

[2] Aeronautical Radio Inc., Aircraft Data Network, Part 7, Avionics Full Duplex Switched Ethernet Network, ARINC Report 664P7-1, Aug. 2009.

[3] G.R. Gupta, S. Sanghavi, and N.B. Shroff, "Node weighted scheduling," *Proc. International Joint Conference on Measurement and Modeling of Computer Systems*, pp. 97-108, June 2009.

[4] J.W.-S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[5] M.J. Neely, E. Modiano, and Y.-S. Cheng, "Logarithmic delay for $N \times N$ packet switches under the crossbar constraint," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 657–668, June 2007.

[6] 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 Symposium*, pp. 367–376, Apr. 2008.

[7] R. Davis and A. Burns, "Hierarchical fixed priority preemptive scheduling," *Proc. IEEE Real-Time Systems Symposium*, pp. 389–398, Dec. 2005.

[8] R. Davis and A. Burns, "Resource sharing in hierarchical fixed priority pre-emptive systems," *Proc. IEEE Real-Time Systems Symposium*, pp. 257–270, Dec. 2006.

[9] G. Lipari and E. Bini, "Resource partitioning among real-time applications," *Proc. Euromicro Conference on Real-Time Systems*, pp. 151–158, Jul. 2003.

[10] 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," *Proc. IEEE International Conference on Engineering of Complex Computer Systems*, June 2009.

[11] M.-Y. Nam, K. Kang, and L. Sha, "Case Study for End-System Bus Delay Analysis Using Virtual Integration," University of Illinois at Urbana-Champaign, Tech. Rep. `https://netfiles.uiuc.edu/mnam/www/MinYoungNam.xapp`, Oct. 2010.

[12] Aeronautical Radio Inc., Digital Information Transfer System (DITS), Part 1, Functional Description, Electrical Interface, Label Assignments and Word Formats, ARINC Specification 429P1-17, May 2004.

[13] Aeronautical Radio Inc., Multi-Transmitter Data Bus, Part 1, Technical Description, ARINC Report 629 Part 1-5, Mar. 1999.

[14] T.G. Baker, "Lessons learned integrating COTS into systems," *Lecture Notes in Computer Science*, vol. 2255, pp. 21–30, 2002.

[15] B. Reynolds, "An application of COTS Ethernet for avionics," *Proc. IEEE/AIAA Digital Avionics Systems Conference*, vol. 2, pp. J13/1–J13/8, Nov. 1998.

[16] T. Schuster and D. Verma, "Networking concepts comparison for avionics architecture," *Proc. IEEE/AIAA Digital Avionics Systems Conference*, pp. 1.D.1-1–1.D.1-11, Oct. 2008.

[17] J.-L. Scharbarg, F. Ridouard, and C. Fraboul, "A probabilistic analysis of end-to-end delays on an AFDX avionic network," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 1, 38–49, Feb. 2009.

[18] D. Shah, P. Giaccone, E. Leonardi, and B. Prabhakar, "Delay bounds for combined input and output switches with low speedups," *Performance Evaluation*, vol. 55, no. 1/2, pp. 113–128, Jan. 2004.

[19] D. Shah, P. Giaccone, and E. Leonardi, "Throughput region of finite-buffered networks," *IEEE Transactions on Parallel and Distributed Systems*, vol 18, no. 2, pp. 251–263, Feb. 2007.

[20] S. Deb, D. Shah, and S. Shakkottai, "Fast matching algorithms for repetitive optimization: an application to switch scheduling," *Proc. Conference on Information, Sciences and Systems*, pp. 1266–1271, Mar. 2006.

[21] S. Gopalakrishnan, M. Caccamo, and L. Sha, "Switch scheduling and network design for real-time systems," *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 289–300, Apr. 2006.