URL: http://dx.doi.org/10.1109/SIES.2009.5196187

A method of computation for worst-case delay analysis on SpaceWire networks

Thomas Ferrandiz
Univ. of Toulouse, ISAE
10 avenue Edouard Belin
BP 54032
31055 Toulouse Cedex 4
France
+33 5 61 33 90 97
Email: thomas.ferrandiz@isae.fr

Fabrice Francès
Univ. of Toulouse, ISAE
10 avenue Edouard Belin
BP 54032
31055 Toulouse Cedex 4
France
Email: fabrice.frances@isae.fr

Christian Fraboul
Univ. of Toulouse, ENSEEIHT/IRIT
2, rue Charles Camichel
B.P. 7122
31071 Toulouse Cedex 7
France
Email: christian.fraboul@enseeiht.fr

***Abstract*—SpaceWire is a standard for on-board satellite networks chosen by the ESA as the basis for future data-handling architectures. However, network designers need tools to ensure that the network is able to deliver critical messages on time. Current research only seek to determine probabilistic results for end-to-end delays on Wormhole networks like SpaceWire. This does not provide sufficient guarantee for critical traffic. Thus, in this paper, we propose a method to compute an upper-bound on the worst-case end-to-end delay of a packet in a SpaceWire network.**

I. Introduction

SpaceWire ([1],[2]) is a high-speed on-board satellite network created by the ESA and the University of Dundee. It is designed to interconnect satellite equipment such as sensors, memories and processing units with standard interfaces to encourage re-use of components across several missions.

SpaceWire uses serial, bi-directional, full-duplex links, with speeds ranging from 2 Mbps to 200 Mbps, and simple routers to interconnect nodes with arbitrary topologies.

In the future, SpaceWire is scheduled to be used as the sole on-board network in satellites. As a high-speed network with low power consumption, it is well-suited to this task. It will be used to carry both payload and control traffic, multiplexed on the same links. These two types of traffic have different requirements. Control traffic generally has a low throughput but very strict time constraints. On the contrary, payload traffic does not need strict guarantees on the end-to-end delays of packets but requires the availability of a sustained, high bandwidth to be operational.

Both requirements were easy to satisfy on point-to-point SpaceWire links. But in order to connect every terminal on a single SpaceWire network, a router was designed with space constraints in mind (especially radiation constraints on memory) which led to the use of Wormhole Routing.

However, the problem with using Wormhole Routing is that it can occasionally lead to long delays if packets are blocked as shown in our SpaceWire overview in Section II. Those blocking delays make it difficult to predict end-to-end delays especially when control and payload traffic are being transmitted on the same network.

This is why previous delay evaluation methods on Wormhole networks have only focused on probabilistic results which do not provide enough guarantee for network designers, who would like to be sure that control packets are always delivered within an acceptable timeframe (simulations do not cover every possible case). To this end, a better approach would be to determine only an upper-bound on the worst-case end-to-end delay.

As a consequence, we have designed a method of computation that enables the determination of such a bound. Section III presents this method and a demonstration on a sample network is given in Section IV. Finally, we conclude and propose future topics for research in Section V.

978-1-4244-4110-5/09/$25.00 ©2009 IEEE

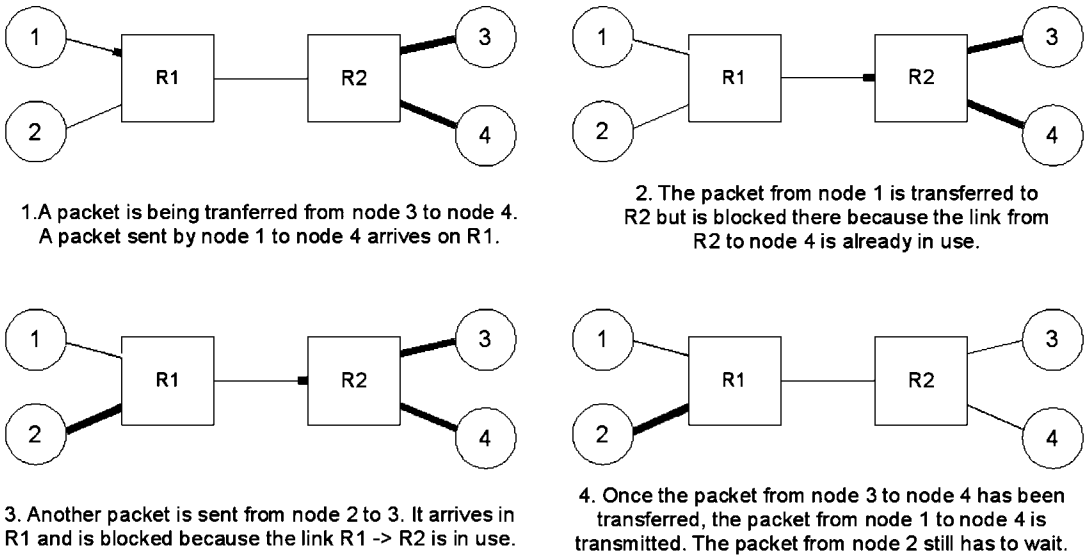

Figure 1. Packets can get blocked in a SpaceWire network

II. A BRIEF DESCRIPTION OF SPACEWIRE

A. SpaceWire as a point-to-point link

Originally, SpaceWire was designed to provide only point-to-point links between satellite equipment. It is derived from the IEEE 1355 standard [3] but uses new connectors and cables suitable for space usage.

SpaceWire uses two types of characters as transmission units: data characters and control characters. Data characters are 10-bits long and carry one byte of data along with one parity bit and one data/control flag. Control characters are 4-bits long and contain 2 bits coding a control code, one parity bit and one data/control flag. They are used to mark the (possibly erroneous) end of packets and for flow control.

SpaceWire uses a flow control mechanism to ensure that no node receives data that could cause a buffer overflow. On each link, the emitter can only send characters if the receiver allows him to by sending a control character (Flow Control Token or FCT). Each FCT allows the emitter to send 8 characters with a maximum of 56 characters authorized at any one time.

Data are organized in packets of arbitrary length with a very simple format. Each packet is composed of three fields: an address field, a cargo field and an End-Of-Packet marker. The address and cargo fields can be of any length and it is even possible to send packets of unlimited size.

SpaceWire's point-to-point links thus provide a simple and fast way to transmit data between two terminals. Furthermore, the end-to-end delay is quite simple to compute. If the source and destination do not cause delays themselves, the delay is simply $\frac{T}{C}$ where T is the packet size and C the capacity of the link.

B. SpaceWire as an interconnection network

With the increasing quantities of data exchanged by various equipment, using only point-to-point links to connect all the equipment becomes too costly. Furthermore, in order to reduce the complexity of the on-board system, it would be better to use a single network to carry both payload and command/control traffic instead of using a dedicated bus (generally a MIL-STD-1553B bus) for the control traffic and high-speed data links for the payload traffic.

These considerations have led to the addition of routers to the SpaceWire standard, allowing the construction of networks with arbitrary topologies. These routers are required to comply with two main constraints. Firstly, they must remain compatible with the point-to-point links used to interconnect equipment and secondly, they cannot use a lot of memory because radiation tolerant memory is very expensive.

As a consequence, Wormhole Routing was chosen as the way to transmit packets through the network. With this technique, instead of buffering

whole packets, a router reads a packet's header on-the-fly when it arrives. Then the router uses this information to determine the appropriate output port and checks the state of this port (see Figure I). If the output port is free (immediately after the packet arrives or after a delay), the packet header is transferred without waiting for the rest of the packet. This allows the routers to use only 8 to 64 bytes of buffer memory per input link. When the output port is allocated to the packet, it is marked as occupied by the router and no other packet can pass through as long as the current packet has not been completely transferred.

This means that, if the output port is already occupied by a packet, the second packet will have to wait until the first one has been transferred. Given that the flow control mechanism is active on each link, small parts of the second packet are blocked in each of the routers crossed by the packet as they wait for the head of the packet to get access to the occupied output port. In turn, they block access to all the output ports used by the packet in the above routers. When the occupied output port finally gets free, one of the blocked worms (packet) is elected, on a round-robin basis.

As can be seen, using Wormhole Routing satisfies both constraints cited above. Routers need little memory for each input link and no memory at all for output links. They are also backward compatible since they propagate the point-to-point SpaceWire flow control to each link along the end-to-end path, backwards from the receiver to the emitter.

However, Wormhole Routing has a major drawback which is that packets might get blocked for a long time in routers. Two mechanisms were thus added to SpaceWire routers to offset this problem. The first is a two-level static priority scheme that is used in routers so that packets sent to a high-level address are always transmitted before packets sent to low-level addresses. As SpaceWire routers are not preemptive, the priority mechanism is only applied when two packets are waiting for the same output port to be freed. The round-robin scheme is then used to arbitrate between packets of the same priority level.

The second mechanism is called Group Adaptative Routing. It works by putting two or more parallel links between two routers. In this case, they're declared as a group in the routing tables and the routers are able to automatically balance the load on the multiple links. However, as it increases the number of links instead of sharing them, this mechanism should be used only for fault tolerance or if it is really impossible to transmit all the traffic on one link.

Together, these two mechanisms can reduce the blocking problem but they cannot solve it completely. Thus, in order to evaluate the end-to-end delays in a SpaceWire network, we need to be able to determine how long packets might remain blocked.

C. The end-to-end delay determination problem: related work

Optimally, a packet is not blocked in any encountered router and endures only the switching delay of each router. The switching delay includes the time to read the header of the packet, determine the output port and copy the first character to the output link. SpaceWire router specifications require the router's switching delay to be less than $0.5\ \mu s$ when links work at 200 Mbps. This means that the delivery time of a packet across a SpaceWire network will be equal to the delivery time on a point-to-point link plus $0.5\ \mu s$ per router crossed.

However, access conflicts can occur on each link taken by a packet. Furthermore, the packet blocking a link can in turn be blocked farther on the network. This makes it very hard to determine the end-to-end delay of a single packet analytically.

A great deal of research has been published on the estimation of end-to-end delays in Wormhole Routing networks, particularly in supercomputers. However, that research only seeks to determine average latency using techniques like queuing theory (see [4] and [5] for example). Knowledge of average latency is not sufficient here since we must be certain that all messages are delivered within an acceptable timeframe.

Network Calculus [6] is another technique that has been successfully used to evaluate bounds on worst-case end-to-end delays in embedded networks. For example, it was used in [7] to analyze a Switched Ethernet network used to replace a MIL-STD-1553B bus in a military avionic

network. However, SpaceWire does not appear to be modelisable using classical network elements like multiplexers and buffers. Indeed, we could not model Wormhole routers using service curves because the service offered to a flow depends on the other flows in the network and on the network topology. To our knowledge, no one has done such a modelisation at the moment.

As a consequence, we propose a more practical approach which works by computing a bound on the worst-case end-to-end delay for each flow. This way, the individual delay of each packet is not characterized but the timely delivery of control traffic packets can still be guaranteed.

III. Computing a bound on the worst-case delay

A. Notations and definitions

We model a SpaceWire network as a connected directed graph $\mathcal{G}(N, L)$. The nodes N of the graph represent both the terminals and the SpaceWire routers. The edges L represent the set of SpaceWire links. We use two edges in opposite directions to represent a bi-directional SpaceWire link.

A communication between a source and a destination is modeled as a flow f. Each flow f has a maximum packet size, noted T_f. This includes the protocol headers. Each flow goes through a set of links that form a path in $\mathcal{G}(N, L)$. The set of flows is denoted F. Knowing the throughput of every flow is not necessary but we make the assumption that the capacity of any link is sufficient to transmit all the data using that link.

We note $links(f)$ the ordered list of the links the flow f follows. $first(f)$ returns the first link in $links(f)$. We note $next(f, l)$ the function which, given a flow f and a link l in $links(f)$ returns the next link in $links(f)$ right after l and $prev(f, l)$ the function returns the previous link in $links(f)$ just before l. If l is the last link in $links(f)$, $next(f, l)$ returns $null$. If l is the first link in $links(f)$, $prev(f, l)$ returns $null$.

We also make the following assumptions :

- the routing is static (current SpaceWire routers use static routing anyway)
- SpaceWire's Group Adaptative Routing feature is not used (thus S is not a multigraph)
- all the links have the same capacity C (this is not very restrictive since reducing link speed does not seem to reduce power consumption significantly)
- we do not take into accounts adresses priorities in the routers (i.e. all nodes have low priority adresses)
- we neglect the delays caused by Time-Codes and Flow Control Tokens (Time-Codes are special characters used by SpaceWire to synchronize a global clock on the network)
- packets are processed as soon as they arrive at their destination (i.e. the destination itself does not delay the packet)
- when a source starts to send a packet, it transmits it at the maximum possible speed until the entire packet has been transferred (i.e. the source itself does not delay the packet)

In order to remove the last two assumptions, we would have to model the behaviour of each node on a case-by-case basis since we do not have a generic model of the nodes at the moment.

B. The method of computation

The method of computation is based on the idea that, in a SpaceWire network, the delivery of a packet can be divided into two phases. During the first phase, the first character (the header) of the packet is routed to its destination through intermediary routers and creates a "virtual circuit" between the source and destination of the packet using Wormhole Routing. In the second one, the whole packet can be transferred along the path of the circuit, as if it were a point-to-point link. When the packet's transfer is completed , the links are freed and the "virtual circuit" disappears.

Let us consider a bound on the worst case delay for packet p of flow f_p. As the source and destination of the packet do not cause any delays, only the routers crossed by p can delay it during the first phase. In each router, this delay has two possible causes. The first is the switching delay of the router and includes the time needed to determine the port through which p will be transferred. This delay is constant for every packet and every SpaceWire router. We will note it d_{sw} afterwards.

The second delay occurs when the output link is already in use by another packet. With Wormhole Routing, p has to wait until the packet using the port is completely transferred. It is

also possible that one or more packets coming from other ports may already be waiting for the same output port to become free. The delays will then accumulate if those packets are transferred before p. Note that, as ports are independent in SpaceWire routers, only packets using the same output port as p can cause a delay.

SpaceWire routers use a Round Robin scheme to arbitrate between concurrent input links trying to access the same output link. Therefore, even in a worst case scenario, at most one packet from each of the other input link may be transmitted before p can be transferred.

For each of these input links, several flows can enter the router. Thus we need to compute a bound on the maximum delay that each of these flows may produce and then calculate the maximum of these potential delays.

The maximum delay that a packet q can cause p is the time to transfer q starting from the current router if q was alone in this router (i.e. q does not have to wait for the output port to become free). However, q may in turn become blocked in the next routers it crosses by other packets.

Once it can access the appropriate output link, the header of a packet is transferred to the next router and must once again wait to access the output port it needs.

As a consequence, we can now formulate a recursive definition of an upper bound on the delay $d(f,l)$ needed to deliver a packet of flow f starting from the moment when the packet tries to access link l with $l \neq null$ and $l \neq first(f)$:

$$d(f,l) = \sum_{l_{in} \in B_{f,l}} [\max_{f_{in} \in U_{l_{in}}} d(f_{in}, next(f_{in}, l)) + d_{sw}] + d(f, next(f,l)) + d_{sw} \quad (1)$$

where

$$B_{f,l} = \{l_{in} \in L,\ l_{in} \neq prev(l),\ for\ which\ \exists f_{in} \in F | next(f_{in}, l_{in}) = l\}$$

(i.e. $B_{f,l}$ is the set of links that are used immediately before $next(f,l)$ by at least one flow f_{in} and that are not $prev(l)$)

and

$$U_{l_{in}} = \{f_{in} \in F | l_{in} \in links(f_{in})\}$$

(i.e. $U_{l_{in}}$ is the set of flows that use the link l_{in}).

Thus (1) means that for every input link which a flow uses before l, we first compute the delay that the flows coming from this link may need to be transfered from this link to their destination if each of them was alone in the router. We then take the maximum for each input link and add those values to the delay for the packet from f itself.

However, (1) is valid only if l is not $null$ and if it is not the first link used by f. If l is $null$, it means that the first character of the packet has arrived at its destination. Thus there is no delay due to a router and, in our assumptions, neither does the destination itself create a delay. The only remaining delay possible is the time needed to transmit the packet on the virtual circuit created. This delay is simply

$$d(f, null) = \frac{T_f}{C} \quad (2)$$

However, when l is the first link in $links(f)$, it means that the emitter node of l is the source of the flow f. As such, there are no flows coming from other input links competing to use the link. If several flows have the same source, other packets from this source can use the output link before p can access it. If $l = first(f)$, the delay is thus

$$d(f, first(f)) = \sum_{f_{in} \in S_F} d(f_{in}, next(f_{in}, l)) + d(f, next(f,l)) \quad (3)$$

where

$$S_F = \{f_{in} \in F | f_{in} \neq f\ and\ first(f_{in}) = first(f)\}$$

(i.e. S_F is the set of flows that have the same source as f).

Taken together, these three equations give us a way to compute an upper bound on the worst case end-to-end delay of any flow f in F.

Given (1), it is easy to see that the worst-case delay is reachable when a packet has to let every possible packet pass before it. Although this scenario seems unlikely, we must take it into account in a worst-case analysis.

The method can then be used as a tool to make trade-offs on the worst-case delays for the various flows by changing the size of packets and the

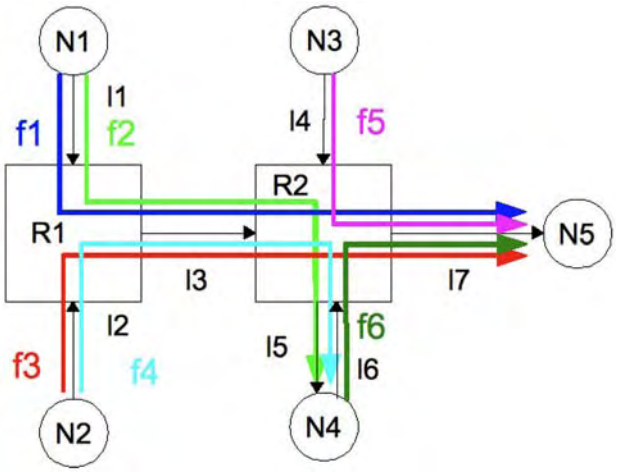

Figure 2. An example of a SpaceWire network (only edges used by at least one flow are displayed)

topology of the network as we will show in the next Section.

The proof of termination of the method and its complexity are given in the Appendix.

IV. Examples

A. Application of the method of computation

We will now study how the method works on an example of a SpaceWire network. The network is pictured on Figure 2. It contains 5 nodes and 2 SpaceWire routers. Six flows share the network and are represented. We do not need to know the throughput of each flow, only the maximum packet size, which is noted T_i, $i \in \{1, \ldots, 6\}$.

First the dependency graph (see the Appendix for the definition) is computed and it is verified that it is not cyclic. This graph is drawn in Figure 3. It is easy to see that it has no cycle so that the computation can be started safely. We will first compute a bound on the worst-case delay for flow 1 which starts in N_1 and ends in N_5.

We first have

$$\begin{aligned} d(f_1, first(f_1)) &= d(f_1, l_1) \\ &= d(f_2, next(f_2, l_1)) \\ &\quad + d(f_1, next(f_1, l_1)) \\ &= d(f_2, l_3) + d(f_1, l_3) \end{aligned} \tag{4}$$

The second half becomes

$$\begin{aligned} d(f_1, l_3) = \max(&d(f_3, l_7), d(f_4, l_5)) \\ &+ d(f_1, l_7) + 2.d_{sw} \end{aligned}$$

with

$$\begin{aligned} d(f_1, l_7) = &d(f_5, null) + d(f_6, null) \\ &+ d(f_1, null) + 3.d_{sw} \end{aligned}$$

Similarly

$$\begin{aligned} d(f_3, l_7) = &d(f_5, null) + d(f_6, null) \\ &+ d(f_3, null) + 3.d_{sw} \\ d(f_4, l_5) = &d(f_4, null) + d_{sw} \end{aligned}$$

The first part of (4) develops into

$$\begin{aligned} d(f_2, l_3) = \max(&d(f_3, l_7), d(f_4, l_5)) \\ &+ d(f_2, l_5) + 2.d_{sw} \end{aligned}$$

and

$$d(f_2, l_5) = d(f_2, null) + d_{sw}$$

Substituting the developments in (4) and using (2) we get :

$$\begin{aligned} d(f_1, l_1) = &\frac{T_1 + T_2 + T_5 + T_6}{C} \\ &+ 2.\max\left(\frac{T_3 + T_5 + T_6}{C} + 2.d_{sw}, \frac{T_4}{C}\right) \\ &+ 9.d_{sw} \end{aligned} \tag{5}$$

Similarly, for f_4 we find:

$$\begin{aligned} d(f_4, l_2) = &\frac{T_3 + T_4 + T_5 + T_6}{C} \\ &+ 2.\max\left(\frac{T_1 + T_5 + T_6}{C} + 2.d_{sw}, \frac{T_2}{C}\right) \\ &+ 9.d_{sw} \end{aligned} \tag{6}$$

For f_5 we first have:

$$\begin{aligned} d(f_5, l_4) &= d(f_5, next(f_5, l_4)) \\ &= d(f_5, l_7) \\ &= \max(d(f_1, null), d(f_3, null)) \\ &\quad + d(f_6, null) + d(f_5, null) + 3.d_{sw} \end{aligned}$$

which gives us

$$d(f_5, l_4) = \frac{T_5 + T_6}{C} + \max(\frac{T_1}{C}, \frac{T_3}{C}) + 3.d_{sw} \quad (7)$$

B. Remarks on the results

Several points can be observed concerning these results. First note that (5) and (6) are similar. (6) can be obtained from (5) by swapping T_1 and T_3 on the one hand, and T_2 and T_4 on the other hand. This is coherent with the fact that N_1 and N_2 have identical communication patterns with N_4 and N_5.

We thus see in (5) that f_5 and f_6 have a bigger impact than f_2 and f_3 on the delay endured by f_1. Therefore, unless T_4 is greater than $T_3 + T_5 + T_6$, in the worst case, a packet from f_1 will be delayed three times by packets from f_5 and f_6, twice by packets from f_3 and once by a packet from f_2.

Note that this holds true for f_4 although it does not have the same destination as f_5 and f_6. This is because a packet from f_4 can get blocked behind packets from f_3 and f_1 which may, in turn, be blocked by packets from f_5 and f_6. This example highlights the fact that the delays are not caused by an access conflict to a common destination but by the access conflicts to the shared link l_3.

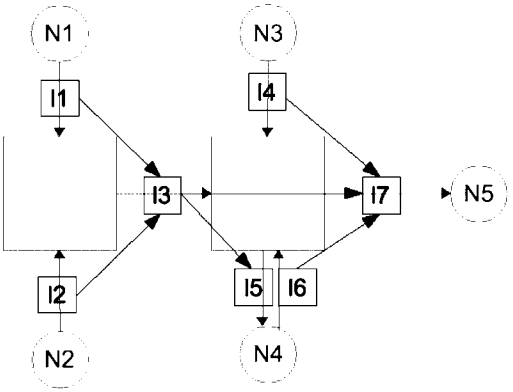

Figure 3. Dependency graph for the network example (in bold)

Besides, (7) shows that packets from f_5 are delayed only once by a packet from f_4 and once by a packet from f_1 or f_3.

These observations suggest that if several nodes send data to a common destination (a mass memory module for instance) using shared links, flows starting close to the destination have a large impact on those coming from farther on the network. On the contrary, flows starting far from the destination have a small impact on those closer from the destination.

As a consequence, putting the nodes with the largest sized packets farther from the destination is preferable. This greatly decreases the worst-case delay for the other nodes while only marginally increasing the delay for long packets because only short delays accumulate due to the closer nodes.

However, this only takes into account the worst-case delay, not average delays or throughput. Indeed, when doing a worst-case analysis, we have to assume that, at each link, packets from nodes close to the destination will be interleaved between packets coming from farther away. Of course, this may not happen every time a packet arrives at each link. In this case, sending long packets from nodes far from their destination will end up blocking several links with no benefits. Furthermore, this can potentially delay other flows not headed to the same destination but sharing links with the long packets.

C. Numerical example

To better understand the meaning of these formulas, it might be useful to consider a numerical example. We will also use this example to illustrate how our method of computation can be used as a tool to make trade-offs on the worst-case delays for the various flows by changing the size of packets.

We will use the following values for the maximum packet sizes:

$$T_1, T_3 : 5120\ B$$
$$T_2, T_4 : 50\ B$$
$$T_5, T_6 : 1000\ B$$

Those are typical values if, for instance, N_1 and N_2 are observations instruments, N_4 is a processor module, N_5 is a mass memory module and N_3 a monitoring equipment. Both N_1 and N_2 send data at a high rate with large packet sizes to N_5. They also send monitoring traffic to N_4 which synthesizes it and send it to the memory. The monitoring traffic (f_2 and f_4) is considered time-critical. N_3 also sends its results to N_5.

We use $C = 200\ Mbps$ as the capacity for every link and $d_{sw} = 0.5\ \mu s$ as the switching delay in both routers. With the same values, we compute both the best- and worst-case delays for f_1, f_4 and f_6. The best case delay is the delay when the packet suffers no delays in any router. Results are given in Table I.

Table I
DELAYS FOR THE EXAMPLE OF NETWORK (WITHOUT FRAGMENTATION)

Flow	Best Case	Worst Case
f_1	$0.257\ ms$	$1.076\ ms$
f_4	$3.5\ \mu s$	$1.076\ ms$
f_6	$50\ \mu s$	$356\ \mu s$

Concerning these results, as can be expected given their formulas, f_1 and f_4 have the same worst-case delay. However, they have very different best-case delays. As a consequence, the ratio between the best and worst case delays is far smaller for f_1 (4) than for f_4 (307). This highlights the impact of the recursive blockings that leads to long delays even for small packets. For f_6, the ratio is only 7 which is coherent with the fact that flows starting from nodes close to the destination are less influenced by other flows.

In order to reduce the worst-case delay for f_4, we can try to fragment the large packets from f_1 and f_3 in smaller chunks of 256 bytes. As can be seen in Table II, the worst-case delay for f_4 goes down to $346\ \mu s$ which is roughly one third of the delay without fragmentation. The delay for f_5 is also reduced in the same proportion.

However, the fragmentation has a cost for f_1. As we now need to send 20 fragments to transmit the same information, the worst-case delay for a whole packet becomes $6.9\ ms$ which is almost 7 times more than before.

V. CONCLUSION AND FUTURE WORK

We have presented a simple method of computation that allows us to compute an upper bound on the worst-case end-to-end delay of flows carried by a SpaceWire network. This bound is such that it can never be exceeded by a packet as long as the network does not fail. This is especially important for control traffic whose timely delivery is critical to the operation of a satellite. The method uses a recursive definition of the delay endured by each packet in each router to obtain the bound.

Table II
DELAYS FOR THE EXAMPLE OF NETWORK (PACKETS FROM f_1 AND f_3 ARE FRAGMENTED)

Flow	Best Case	Worst Case
f_1	$0.276\ ms$	$6.9\ ms$
f_4	$3.5\ \mu s$	$346\mu s$
f_6	$50\ \mu s$	$113\ \mu s$

Furthermore, the example illustrates the huge difference between the best and worst case delays for a given flow. It also illustrates the fact that the impact of each flow on the delay of other flows varies according to the distance between the source and destination of the flow. Developing the results formally reveals which flows have more impact on others. Finally, we showed on the example how our method can be used as a tool to make trade-offs on the worst-case delays for the various flows in order to better dimension the network.

We have three perspectives to extend our method of computation. The first is to complete the modelisation of SpaceWire by adding Group Adaptative Routing and packet priorities in the model. This will allow us to compute tighter bounds than at present.

At the moment, we do not make strong hypothesis on the input traffic. We only suppose that each flow has a maximum packet size. Although this increases the generality of the method, it also means that the results we get may be pessimistic. Thus we plan to use a more precise modelling of the input traffic to obtain tighter bounds. For example, we could use arrival curves to describe the traffic instead of supposing that each flow tries at the maximum speed.

Another interesting direction is to model SpaceWire-RT using the method we have applied for SpaceWire. SpaceWire-RT ([8]) is an extension of SpaceWire, designed as a Quality Of Service layer between a SpaceWire network and its applications. At the moment, it is only a draft but it should provide better temporal guarantees to critical flows. Using our method, we should be able to determine what improvements it will bring for worst-case delays.

ACKNOWLEDGMENT

This work was supported by a PhD grant from the CNES and Thales Alenia Space.

REFERENCES

[1] S. M. Parkes and P. Armbruster, "SpaceWire: A spacecraft onboard network for real-time communications," *IEEE-NPSS Real Time Conference*, no. 14, pp. 1–5, Feb 2005.

[2] ECSS, "ECSS-E-ST-50-12C, SpaceWire – links, nodes, routers and networks," pp. 1–129, Aug 2008.

[3] IEEE Computer Society", ""IEEE Standard for Heterogeneous Interconnect (hic) (lowcost, low-latency scalable serial interconnect for parallel system construction)"," *IEEE Standard 13551995*, 1996.

[4] J. T. Draper and J. Ghosh, "A comprehensive analytical model for Wormhole Routing in multicomputer systems," *Journal of Parallel and Distributed Computing*, pp. 1–34, Jan 1994.

[5] W. Dally, "Performance analysis of k-ary n-cube interconnection networks," *Computers, IEEE Transactions on*, vol. 39, no. 6, pp. 775 – 785, Jun 1990.

[6] J.-Y. Le Boudec and P. Thiran, "Network Calculus: a theory of deterministic queuing systems for the internet," *Springer Verlag*, no. LNCS 2050, pp. 1–265, Apr 2004.

[7] A. Mifdaoui, F. Frances, and C. Fraboul, "Real-Time characteristics of switched ethernet for "1553b"-embedded applications: Simulation and analysis," *Industrial Embedded Systems, 2007. SIES '07. International Symposium on*, pp. 33 – 40, Jun 2007.

[8] S. Parkes and A. F. Florit, "SpaceWire-RT initial protocol definition v2.1," pp. 1–120, Feb 2009.

[9] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *Computers, IEEE Transactions on*, vol. C-36, no. 5, pp. 547 – 553, May 1987.

APPENDIX A
PROOF OF TERMINATION OF THE METHOD

We first define the *link dependency graph* D for a network represented by the graph $\mathcal{G}(N, L)$ as the directed graph $D = \mathcal{G}(L, E)$ where the nodes L of D are the links of $\mathcal{G}(N, L)$ and the edges E are the pairs of links used successively by a flow:

$$E = \{(l_i, l_j) \in L \; for \; which \; \exists f \in F | next(f, l_i) = l_j\}$$

Theorem 1: For any flow f in F and any link l in $links(f)$, the computation of $d(f, l)$ finishes if and only if D is acyclic.

Proof: $\Rightarrow$ Suppose there's a cycle in D. Since $next(f, l)$ cannot return l for any l (a packet cannot use the same link twice in a row), this cycle's length is at least 2. Given the definition of d, this means we can find a flow f and a link l such that computing $d(f, l)$ requires calling $d(f, l)$ during the recursion. Thus the computation of $d(f, l)$ does not finish.

$\Leftarrow$ Suppose D is acyclic. We can then use topological sorting to assign a total order to the vertices of D such that if $(l_i, l_j) \in E$ then $l_i > l_j$ (if D is not connected in the graph theory sense we can order each connected component individually then arbitrarily order the component). From the definition of E and $next(f, l)$, it follows that computing $d(f, l)$ only requires calls to d with links inferior to l as parameters. Since F is a finite set and $d(f, l)$ is called only once for some given f and l, it follows that computing $d(f, l)$ always terminates.

■

Note that as shown in [9] that for an interconnection network using Wormhole Routing under assumptions similar to ours, it is equivalent to say that the link dependency graph is acyclic and that the network is deadlock-free. Since deadlock prone networks would not be valid for satellite use, Theorem 1 shows that the computation finishes in every pertinent case. This is also coherent with the fact that, when the computation does not finish, $d(f, l)$ becomes infinite which is characteristic of a deadlock.

A simple way to check beforehand if the computation will finish is to use the adjacency matrix of the dependency graph and iterate it. If it converges toward the zero matrix then there are no cycles in the graph. However, if a power of the matrix has only 1 on its diagonal, it means that the dependency graph is cyclic and the sum will diverge.

APPENDIX B
COMPLEXITY OF THE ALGORITHM

The exact number of operations required to compute $d(f, l)$ on a given network will vary with the topology and the number of conflicting flows on each link. However, we can compute an upper-bound on the number of operations without difficulties.

Let us note n the number of flows in F and m the number of links in L. If we compute naïvely the delay for each flow, the size of the computation will increase exponentially with the number of flows in conflicts on each link. However, it is easy notice that the number of possible calls to $d(f, l)$ is finite and bounded by $n.m$. Thus, if we store the results of all the calls during the computation, the complexity of the computation will be $O(n.m)$.