IAC-06- B5.7.2

ETHERNET OVER SPACEWIRE – SOFTWARE ISSUES

Dr Barry M Cook
4Links Limited, UK
Barry@4Links.co.uk
Paul Walker
4Links Limited, UK
Paul@4Links.co.uk

ABSTRACT

We consider the software issues involved in combining the best of SpaceWire, such as modularity, high speed, low latency, fault-tolerance, and ease of implementation, with the vast experience of protocol design that has been implemented on Ethernet. We consider how existing Ethernet-based designs can be implemented on SpaceWire networks.
Both technologies can be used to create networks that route packets from source to destination. SpaceWire, however, has a physical layer that has proven easier to build into a Radiation-Hard environment. Ethernet is based on the legacy of a bus and so relies on broadcast with packets visible to all nodes, whereas SpaceWire is entirely point-to-point and allows multiple connections for redundancy which raises issues for broadcast.
Issues that will be addressed here, to allow Ethernet to work over SpaceWire, include: converting Ethernet addressing into SpaceWire routing; behavior in the event of faults (which may create a need to re-route); handling broadcast; and considering whether the network topology is static (as in conventional large spacecraft) or dynamic with plug and play (as suggested for responsive space on small satellites, or for the Shuttle/CEV).

FULL TEXT

INTRODUCTION

Ethernet is a long established technology which has progressed through several generations and implementations. It is the basis for a very significant proportion of data transfers within 'local' area networks – where local can easily encompass thousands of users over campus- or small-town-sized facilities. Take-up is extensive and it is probably the best-known networking technology in the world.
Success for Ethernet is, at least in part, due to the very wide range of protocols that are supported, and have been developed by this ubiquitous data transfer technology. There is a protocol for high-speed transfers, guaranteed delivery, real-time data such as streaming audio and video and dozens more. Protocol development continues as new applications make their demands known and the underlying implementation improves. Networks with data rates of 10Mb/s, 100Mb/s and 1Gb/s are now commonly deployed with higher rates appearing [1].
SpaceWire is a relative newcomer. Its roots go back to a 1995 standard [2] but recent (relatively minor) changes resulted in SpaceWire being standardized by the European Space Agency in 2003 [3].

Although new, world take-up has been extensive in the Space industry with many missions committed to its use.

One reason for SpaceWire being adopted is its demonstrated ability to be used to build highly fault-tolerant networks and systems.

As yet, SpaceWire protocols are very thin on the ground and there is no legacy protocol code for anything like the range of applications that Ethernet offers.

Naturally, the question that has occurred to some is – What if SpaceWire networks could offer Ethernet services as well as supporting the new Space-related SpaceWire, and other, protocols? Such a combination would provide a rich set of protocols for a wide variety of applications – and allow re-use of existing, proven, software.

We have established that this is indeed possible by implementing a proof-of-concept SpaceWire network running native Ethernet protocols. It was achieved by writing a Linux network driver for a SpaceWire interface – no other change being required in the operating system.

A companion paper [4] compares Ethernet and SpaceWire at the hardware level to reveal why SpaceWire is attractive and discusses the, few, issues that have to be considered in implementing the network. This paper concerns the software required to complement SpaceWire hardware to provide an Ethernet network.

The software requirements described here can easily be implemented in a SpaceWire device driver which appears, to the operating system, to be a standard Ethernet driver. All the interfacing can be hidden so that the SpaceWire network appears to the operating system, and hence the user, as a standard Ethernet network.

The first requirement is to route Ethernet data from source to destination, using information normally used for an Ethernet implementation. This can be within a network that is assumed never to change or in a dynamic network. We will see that the latter is a more useful model since faults do change the network and this approach can recover from a wider range of faults than a static configuration will allow.

Data multicast and broadcast is not directly supported by SpaceWire but is a basic assumption for Ethernet. Software must be used to emulate Ethernet's assumptions in a SpaceWire network.

A NOTE FOR THE PURISTS

The formal definition of Ethernet, as contained in IEEE802.3, uses precise terms for aspects of the standard. These terms are not always used by the world at large and other, not formally specified, words commonly substituted. SpaceWire does not share exactly the same terms of Ethernet's formal definition – but has much in common with Ethernet's informally used terminology. In order to avoid much tedious explanation, we have used the common set of terms in this paper and hereby apologize to the Ethernet community for our informality.

ETHERNET PACKET STRUCTURE AND ROUTING

Each Ethernet packet is defined to consist of

- A destination address – the address of the node to which the packet must be directed;
- A source address – where the packet came from;
- An indicator of either the length of the packet or of the protocol used;
- Data;
- A Cyclic Redundancy Check over the packet.

Only the destination address is important to us for the purpose of routing. A unique 48-bit value is used so that no two devices (in the whole world) have the same address. It is possible to attach any two or more devices to a network and be sure they do not have the same address.

One bit of the address is used to indicate whether the packet is intended to be received by a single destination (unicast) or by many destinations (multicast and broadcast). A unicast packet may be broadcast to all destinations – the decision as to whether to accept a packet rests with the receiver. It is not possible to rely on the routing network only to deliver packets addressed to a receiver.

The routing network starts by not knowing where any packet is to be directed. A packet with an unknown destination address is broadcast to all nodes. As nodes respond, their source addresses are noted by the network and routes to them become known. This knowledge soon results in a packet being sent only along the required path to its destination.

Dynamic networks, where nodes move between ports, are often not well supported as the mechanism for changing internal tables of locations can take too long to drop a route and re-discover the new location when a node moves.

SPACEWIRE PACKET STRUCTURE

Each SpaceWire packet is defined to consist of

- Data.

SpaceWire does not specify any structure on the data in a packet but leaves it to the node receiving the data to interpret it.

There are extensions to the standard that suggest some structure. One example is the ‘Protocol Identification’ [5] which corresponds to Ethernet’s Type/Length field in indicating how the rest of the data in the packet should be interpreted.

SPACEWIRE ROUTING

There is no specified addressing in the packet. Instead, each switch interprets the first byte of the packet for routing purposes.

The first byte is interpreted in different ways, depending on its value

- 0: direct the packet to the switch
- 1-31: direct the packet to the physical port indicated
- 32-255: use the value as an index into a routing table that indicates what to do with the packet.

Address 0 is used to control / communicate with the switch, for example to set the routing table entries.

Addresses 1 to 31 constitute ‘physical routing’ and the packet is forwarded after deleting the first byte. This exposes the second byte of the original packet to the next router. A complete path through the network may be explicitly specified by the packet sender – ‘source routing’.

Addresses 32 to 255 constitute ‘logical routing’ where a table in the switch is used to determine how the packet is to be forwarded. This allows the output port or group of ports to be specified and also determines whether the first byte is to be deleted or retained.

Logical routing retaining the first byte allows a packet to be directed through several switches to the destination with only a single addressing byte.

A mix of physical and logical routing may be used.

ETHERNET OVER SPACEWIRE

Two situations may be identified, routing a unicast packet to its destination and routing multicast / broadcast packets to all destinations. We will identify here the principles of what we need to achieve in the hardware and leave details such as precise values to use to the companion software issues paper.

Unicast Packets

Each Ethernet packet to be routed will begin with a 48-bit destination address. Somewhere in the network will be a destination node with that address.

If we can satisfy the following constraints

- The network topology will not change;
- Each device is always connected to the same port on the network;
- The address of each device is known.

Then there is a simple mechanism to achieve routing.

1. Allocate a logical address for each device;
2. Statically configure each routing table to forward packets the correct device;
3. Translate each 48-bit destination address to its corresponding 8-bit logical address.

Run-time effort is restricted to address translation which need be little more than a table look-up (albeit a sparse table of 48-bit addresses).

A degree of fault tolerance can be achieved by using group adaptive routing over alternate paths – provided some care is taken over topology, see below.

It may be argued that minor changes, such as a link failure, can still be considered to be a static network in that a static configuration may not need to change. More significant changes, such as multiple link failures, routing switch failures, or powering-up a cold-redundant unit are very likely to be beyond a static configuration.

Changing topology or device location (including adding spare units) requires the ability to re-configure routing switches and, possibly, translation tables.

Several years ago, 4Links demonstrated a plug-and-play network where any topology could be used and devices connected anywhere, and the topology changed and the devices moved. The routing tables and address translations automatically changed to match the network. At least some of the techniques used there could be employed in this application. The companion paper describes the basic operation of that dynamic system.

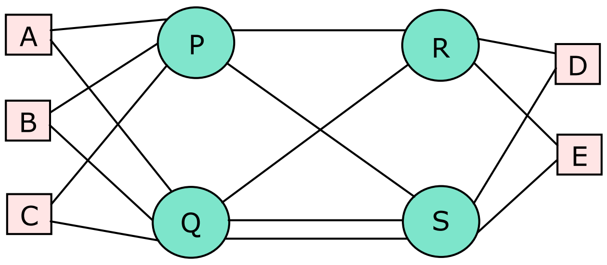

Fig. 1: Example Network

Multicast / Broadcast Packets

We have already discussed the problems with multicast and broadcast and their implementation by multiple unicast messages. The impact on the system is that translation to a SpaceWire logical address must be extended to replicate packets to all known logical addresses.

SpaceWire switches are permitted to 'distribute' packets in a limited way and this may be used to reduce the number of packet copies transmitted.

EXAMPLE NETWORK

In order to give concrete examples on configuration and operation of a network without and with faults we will use the example network shown in Fig. 1.

Devices A to E are connected to routing switches P, Q, R and S. Lines represent SpaceWire links over which data may flow in both directions (full-duplex). Each device has two SpaceWire links for redundancy and the whole is configured into a classic cross-strapped redundant network. There are at least two routes between any two devices. Switches Q and S are connected by two links to provide more bandwidth and/or more redundancy.

STATIC CONFIGURATION

Static configuration relies solely on group adaptive routing for fault tolerance. This is a powerful technique as fault detection and fail-over to an alternate path is very fast – of the order of micro-seconds.

It is easy to see how to configure this network to forward packets. Each device is allocated a logical address and each switch is configured so that packets are directed toward the destination using groups of as many alternate paths are possible.

Using logical address 200, say, for device D we configure switches to forward packets with header value 200 as follows …

P forwards to R or S
Q forwards to R or S
R forwards to D
S forwards to D

Packets from device A, for example directed to device D can take one of many routes:

A – P – R – D
A – P – S – D
A – Q – R – D
A – Q – S – D

This allows transmission even when failures occur either in links or switches. A link failure between switch P and switch R, Fig. 2a, still leaves three working routes

A – P – S – D
A – Q – R – D
A – Q – S – D

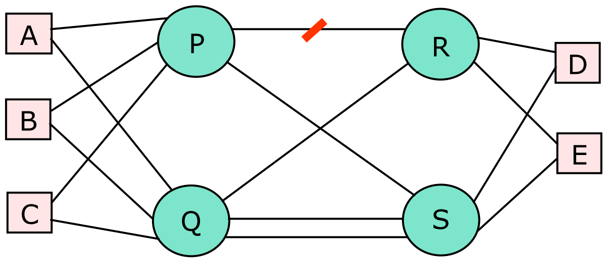

Fig. 2a Example Network with failed link

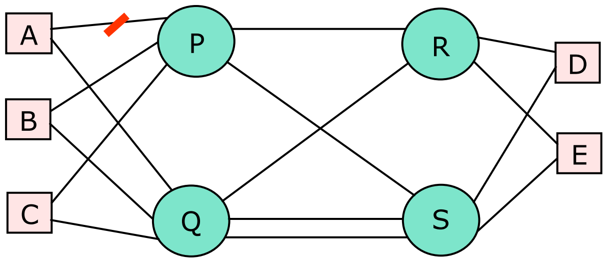

Fig. 2b Example Network with failed link

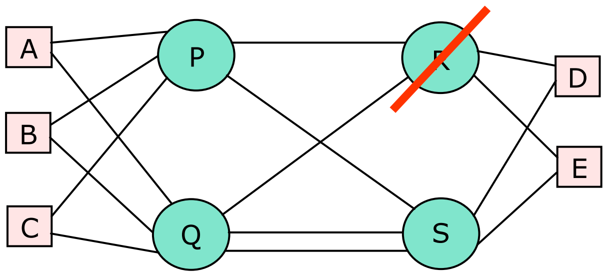

Fig. 2c Example Network with failed switch

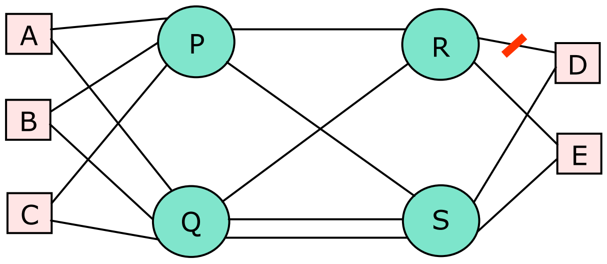

Fig. 2d Example Network with failed link

A link failure between device A and switch P, Fig. 2b, still leaves two working routes

A – Q – R – D
A – Q – S – D

A failure of switch R, Fig. 2c, still leaves two working routes

A – P – S – D
A – Q – S – D

This network appears to provide a good level of fault tolerance where we still have two or more usable routes. Indeed, it appears that multiple faults can be corrected.

Fig. 2d, however shows a more difficult fault. A link immediately connected to device D has failed. Routes through switch S are operational but switches P and Q, using grouping, may send packets to either switch S or R. Packets sent to switch R cannot be delivered to device D as the link between them has failed – and there is no alternative path. The obvious solution, to provide an alternate path from switch R to switch S, Fig 3, must be carefully considered. For greatest effect we would appear to have to set switch R to forward packets destined for device D to use a group of links to D or S and similarly set a group on switch S to forward to D or R. But this gives rise to the risk that a packet will be sent from R to S to R one or more times before the correct link to D is selected. If both links to D, or the device itself, fails then we are guaranteed an infinite loop between R and S. We really need the table entry for a logical address to depend in some way on where the packet came from – a feature not envisaged by the authors of the SpaceWire standard (although supported by 4Links Flexible SpaceWire Router products).

LIMITATIONS OF STATIC CONFIGURATION

As we have seen, such static configuration may not be able to deliver the desired level of fault tolerance.

Perhaps the underlying difficulty is that a fault changes the network and it is only static until an event it is intended to handle occurs. Faults make the network dynamic. Recovery

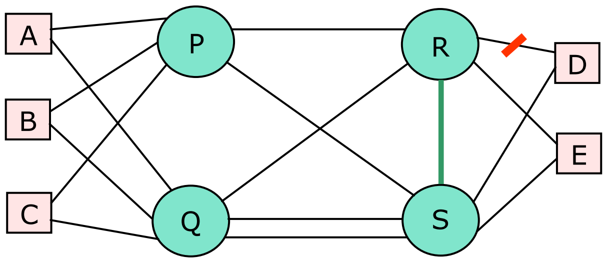

Fig. 3 Solution to problem of Fig 2d?

from faults can also be dynamic – powering-up a cold-redundant switch, for instance. Dynamic configuration would have been able to reconfigure around the fault in the network of Fig 2d.

DYNAMIC CONFIGURATION

Dynamic configuration acknowledges that the network can change, possibly as the result of faults or recovery from them. This is not because there is expected to be active re-arrangement in a plug-and-play situation, although that is also supported.

Two stages are involved:

1. Discover the current Network to identify the switches, connections and devices (at least those devices offering an Ethernet interface);
2. Select routes and configure devices and switches.

These are repeated either at regular (not necessarily frequent) intervals or whenever a change is detected (by, for example, unexpected behavior of a protocol).

NETWORK DISCOVERY

Each active component must be able to be probed and respond with information to assist the discovery process. Exactly what information is required is a matter of some judgment – a minimal set reduces initial traffic and is most easily implemented but a more complete reply may reduce the total number of packets required.

We have found that a (not quite minimal) response should contain

- Whether the component is a switch or a device;
- How many ports are present;
- Which port received the request;
- A unique identifier both to detect loops (redundant paths) in the network and to contain an Ethernet address.

The first two of these may be a single value encoding both pieces of information. Our first implementation, for example, recognized that a component with a single port could not be a switch but had to be a device. The question of whether a 2-port component is a (trivial) switch or a device with two ports was avoided by a different view of two-port devices.

Unique identifiers do not require strictly unique values – a switch and a device could share a value since the combination of type of component and value *is* unique. Ethernet routing is well supported if the unique identifier of a device is its 48-bit Ethernet

MAC address. Switches can be differently numbered (and do not even need the same length identifier).

Discovery proceeds from some point in the network – *any point*, there is no need for dedicated locations performing discovery. Probe requests are sent from each port on this initial location to discover what is immediately connected. For each connected component that is a switch, recursively probe each of its ports to see what is connected. Probing stops with devices or previously probed switches.

This is most easily demonstrated with an example. Fig 4 adds marks indicating port number 1 on each component. Ports are numbered clockwise from port 1.

Starting to discover the network from Device A:

1. Probe the component on port 1 to discover it is an 8-port switch and the probe request came in on port 5, thus A.1 = P.5
2. Probe on port 2, thus A.2 = Q.6

We have completed all ports on A but now know about two switches, P and Q so we proceed to probe what is connected to their ports:

3. P.1 = R.4
4. P.2 = S.5
5. P.3 = C.1
6. P.4 = B.1
7. P.5 = A.1 (already known)
8. P.6 unconnected
9. P.7 unconnected
10. P.8 unconnected
11. Q.1 = R.3
12. Q.2 = S.4
13. Q.3 = S.3
14. Q.4 = C.2
15. Q.5 = B.2
16. Q.6 = A.2 (already known)
17. Q.7 unconnected
18. Q.8 unconnected

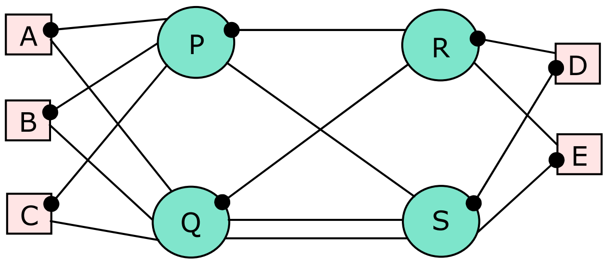

Fig. 4 Example network with marked ports – marks indicate port number 1, numbers ascend in clockwise order

Revealing devices B and C which cannot route data, and switches R and S which can:

19. R.1 = D.2
20. R.2 = E.2
21. R.3 = Q.1 (already known)
22. R.4 = P.1 (already known)
23. R.5 unconnected
24. R.6 unconnected
25. R.7 unconnected
26. R.8 unconnected
27. S.1 = D.1
28. S.2 = E.1
29. S.3 = Q.3 (already known)
30. S.4 = Q.2 (already known)
31. S.5 = P.2 (already known)
32. S.6 unconnected
33. S.7 unconnected
34. S.8 unconnected

And we have finished.

Seven of the 34 probes do not need to take place as the information is already known from earlier probes. Twelve probes are to unconnected ports – adding information to the probe reply about active ports could have avoided these messages.

The network is more clearly described by placing these results in a table – see table 1.

ROUTING

At this point we have received, from each device, a probe response containing the device identifier which is also its Ethernet address. We can select the device to receive a packet by comparing its address with the destination address of the Ethernet packet to be transmitted.

Knowing the destination device and the network topology, we can select the route(s) through the network for the packet.

As an example, assume we want to send a packet (from device A) to device E. We can see that several routes are possible:

A.1 – P.1 – R.2 – E.2
A.1 – P.2 – S.2 – E.1

Port:	1	2	3	4	5	6	7	8
A	P5	Q6						
B	P5	Q5						
C	P4	Q4						
D	S1	R1						
E	S2	R2						
P	R4	S5	C1	B1	A1	-	-	-
Q	R3	S4	S3	C2	B2	A2	-	-
R	D2	E2	Q1	P1	-	-	-	-
S	D1	E1	Q3	Q2	P2	-	-	-

Table 1 Map of the example network

A.2 – Q.1 – R.2 – E.2
A.2 – Q.2 – S.2 – E.1
A.2 – Q.3 – S.2 – E.1
A.1 – P.1 – R.3 – Q.2 – S.2 – E.1
A.1 – P.1 – R.3 – Q.3 – S.2 – E.1
A.2 – Q.2 – S.5 – P.1 – R.2 – E.2
A.2 – Q.3 – S.5 – P.1 – R.2 – E.2

We must select one of these, perhaps by shortest route, perhaps by some criterion that includes bandwidth allocation and prefix the Ethernet packet with suitable routing bytes. The physical path is immediately available from the chosen route. If we use the first route above as an example, we must send the packet from Port 1 of device A with header bytes 1 and 2. The first byte is interpreted by switch P (connected to A's port 1) and the value 1 will send the packet (minus that byte) to switch R. Switch R will see the new first byte – 2 – and use this to direct the packet (minus that byte) to the device connected to port 2 – E, our required destination. At this point the packet will be reduced to the original Ethernet packet.

If we chose the last route above, the packet would be sent from port 2 with header values 3, 5, 1, 2 resulting in routing through Q, S, P and R.

There is no need to re-create the network map for each packet sent and discovered routes could be cached to avoid searching for and selecting routes for each packet.

Re-discovery and re-routing is required only when the network changes. A periodic check may be used to detect this.

MULTICAST / BROADCAST

Each device that runs a network discovery procedure and creates a network map knows how to route packets to all the devices on the network. Multicast by repeated transmission is thus straightforward.

ROBUSTNESS

All operations required to route a packet from a device take place on that device. There is no dedicated controller and no central table to be accessed. The result is a very robust system.

CONCLUSIONS

SpaceWire is easy to implement and very well suited to the construction of highly fault tolerant networks. Ethernet offers a rich set of tried-and-tested protocols – and software. Ethernet and SpaceWire deliver largely similar low-level services, except multicast and broadcast. Translation of Ethernet addressing to SpaceWire addressing can be achieved in a distributed and very robust manner – with very modest processing requirements.

Ethernet over SpaceWire can not only be delivered but be delivered with a very good degree of fault tolerance.

REFERENCES

[1] IEEE Std 802.3 – 2002: IEEE Standard for Information Technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.

[2] IEEE Std 1355-1995: Heterogeneous InterConnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction).

[3] ECSS-E-50-12A 24 January 2003 Space engineering: SpaceWire – Links, nodes, routers and networks.

[4] "Ethernet Over SpaceWire- Hardware Issues", B M Cook & P Walker, in Proceedings of 57th International Astronautical Congress, Valencia, 2006.

[5] ECSS-E-50-12 Part 2 Draft B SpaceWire Protocol ID Jan 2005