3 DS-Links and C104 Routers

3.1 Introduction

Millions of serial communication links have been shipped as an integral part of the transputer family of microprocessor devices. This ‘OS-Link’, as it is known, provides a physical point–to–point connection between two processes running in separate processors. It is full–duplex, and has an exceptionally low implementation cost and an excellent record for reliability. Indeed, the OS-Link has been used in almost all sectors of the computer, telecommunications and electronics markets. Many of these links have been used without transputers, or with a transputer simply serving as an intelligent DMA controller. However, they are now a mature technology, and by today’s standards their speed of 20 Mbits/s is relatively low.

Since the introduction of the OS-Link, a new type of serial interconnect has evolved, known as the DS-Link. A major feature of the DS-Link is that it provides a physical connection over which any number of software (or ‘virtual’) channels may be multiplexed; these can either be between two directly connected devices, or can be between any number of different devices, if the links are connected via (packet) routing switches. Other features include detection and location of the most likely errors, and a transmission speed of 100 Mbits/s, with 200 Mbits/s planned and further enhancement possible.

Although DS-Links have been designed for processor to processor communication, they are equally appropriate for processor to memory communication and specialized applications such as disk drives, disk arrays, or communication systems.

3.2 Using links between devices

DS-Links provide point–to–point communication between devices. Each connected pair of DS-Links implements a full–duplex, asynchronous, flow–controlled connection operating at a programmable speed of up to 100 MBits/s or more. Point to point links have many advantages over bus based communications in a system with many devices:

- There is no contention for the communication mechanism, regardless of the number of devices in the system.
- There is no capacitive load penalty as more devices are added to the system.
- The communications bandwidth does not saturate as more communicating devices are added to the system. Rather, the larger the number of devices, the greater the total communications bandwidth of the system.
- Removing the bus as a single point of failure improves the fault–tolerance of the system.

For small systems, a number of DS-Links on each device can provide complete connection between a few devices. By using additional message routing devices, networks of any size and topology can be built with complete connection between all devices.

3.3 Levels of link protocol

As with most communications systems, the links can be described at a number of levels with a hierarchy of protocols. The lowest level of electrical signals is considered in detail in chapter 4.

3.3.1 Bit level protocol

To achieve the speed required, a new, simple link standard has been produced. DS-Links have four wires for each link, a data and ‘strobe’ line for each direction. The data line carries the actual signal, and the strobe line changes state each time the next bit has the same value as the previous one[10]. By this means each DS pair carries an encoded clock, in a way which allows a full bit–time of skew–tolerance between the two wires. Figure 3.1 shows the form of signals on the data and strobe wires. All signals are TTL compatible.

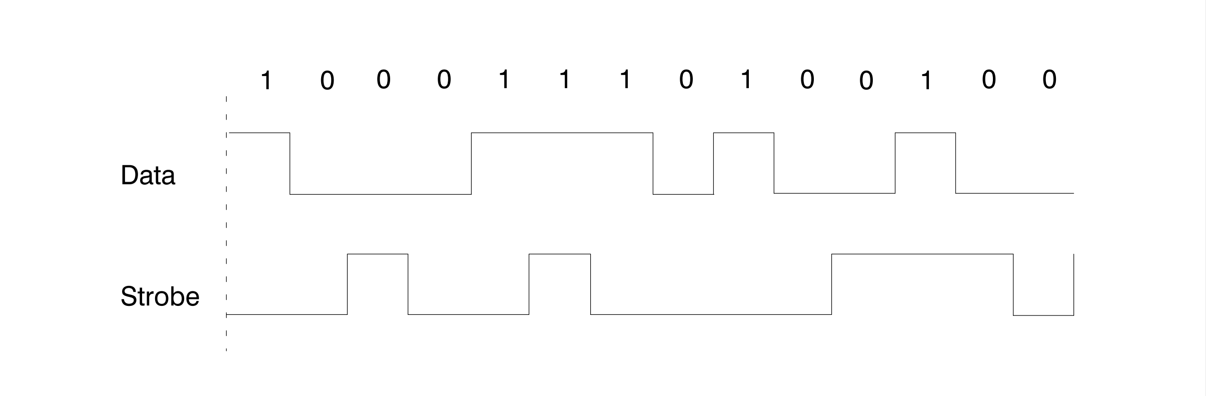


Figure 3.1 Link data format

Since the data–strobe system carries a clock, the links are asynchronous; the receiving device synchronizes to the incoming data. This means that DS-Links ‘autobaud’; the only restriction on the transmission rate is that it does not exceed the maximum speed of the receiver. It also simplifies clock distribution within a system, since the exact phase or frequency of the clock on a pair of communicating devices is not critical.

3.3.2 Token level protocol

In order to provide efficient support for higher level protocols, it is useful to be able to encode ‘‘tokens’’ which may contain a data byte or control information (in other standards these might be referred to as ‘‘characters’’ or ‘‘symbols’’ – note that they have no relation to the ‘‘token’’ of a token–ring network). Each token has a parity bit plus a control bit which is used to distinguish between data and control tokens. In addition to the parity and control bits, data tokens contain 8 bits of data and control tokens have two bits to indicate the token type (e.g. ‘end of message’). This is illustrated in figure 3.2.

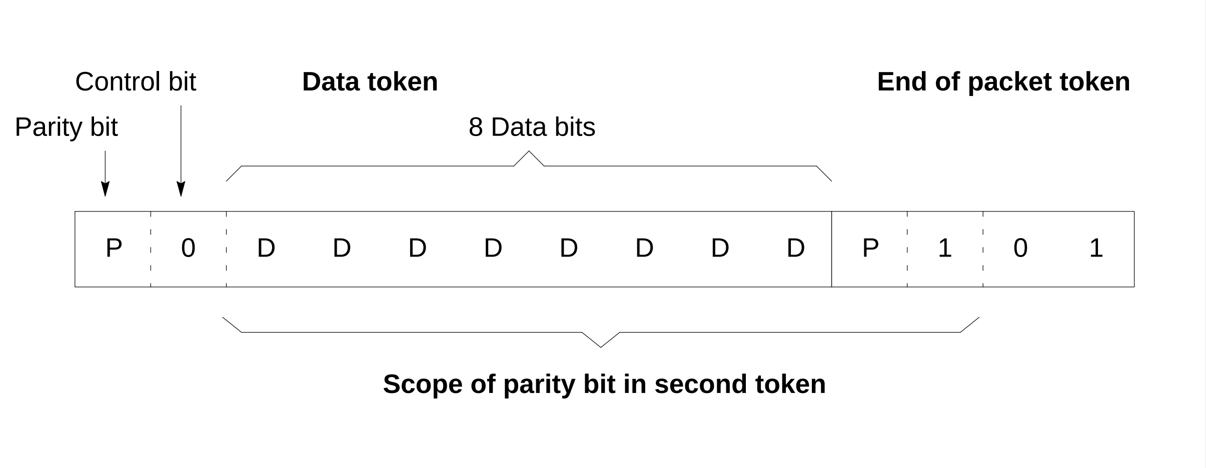


Figure 3.2 Token level protocol

[10] NB: This does not correspond with the usual meaning of ‘strobe’, which would be a signal which indicates *every* time that another signal is valid.

The parity bit in any token covers the parity of the data/control flag in the same token, and the data or control bits in the previous token, as shown in figure 3.2. This allows an error in any single bit of a token, including the token type flag, to be detected even though the tokens are not all the same length. **Odd** parity checking is used. To ensure the immediate detection of errors null tokens are sent in the absence of other tokens. The coding of the control tokens is shown in table 3.1, in which P indicates the position of the parity bit in the token.

Table 3.1 Control token codings

Flow control token	FCT	P100
End of packet	EOP	P101
End of message	EOM	P110
Escape token	ESC	P111
Null token	NUL	ESC P100

Note that the token level of the protocol is independent of details of the higher levels, such as the amount of data contained in a packet, or the particular interpretations of packets of different types.

Token level flow control

Token level flow control (i.e. control of the flow of tokens between devices) is performed in each link module, and the additional tokens used are not visible to the higher–level packet protocol. The token–level flow control mechanism prevents a sender from overrunning the input buffer of a receiving link. Each receiving link input contains a buffer for at least 8 tokens (more buffering than this is in fact provided). Whenever the link input has sufficient buffering available for a further 8 tokens, a flow control token (FCT) is transmitted on the associated link output, and this FCT gives the sender permission to transmit a further 8 tokens. Once the sender has transmitted a further 8 tokens it waits until it receives another FCT before transmitting any more tokens. The provision of more than 8 tokens of buffering on each link input ensures that in practice the next FCT is received before the previous batch of 8 tokens has been fully transmitted, so the token level flow control does not restrict the maximum bandwidth of the link. This is analyzed in detail in chapter 6.

Token level flow control greatly simplifies the higher levels of the protocol, since it prevents data from being lost due to buffer overflow and so removes the need for re–transmission unless errors occur. To the user of the system, the net result is that a connected pair of DS-Links function as a pair of fully handshaken FIFOs, one in each direction.

Note that the link module regulates the flow of data items without regard to the higher level objects that they may constitute. At any instant the data items buffered by a link module may form part or all of one or more consecutive higher–level objects. FCTs do not belong to such objects and are not buffered.

3.3.3 Packet level protocol

In order to transfer data from one device to another, it is sent as one or more packets (in some other serial standards these might be called ‘‘frames” or ‘‘cells”). This allows a number of simultaneous data transfers to be interleaved on the same link. It also allows data to be routed by packet switches such as the IMS C104 (described later).

Every packet has a header defining the destination address followed by zero or more data bytes and, finally, a ‘terminator’ token, which may be either an ‘end of packet’ or an ‘end of message’ token. See figure 3.3. This simple protocol supports data transfers of any length, even when (for reasons of smooth system performance) the maximum packet size is restricted; the receiving device knows when each packet and message ends without needing to keep track of the number of bytes received.

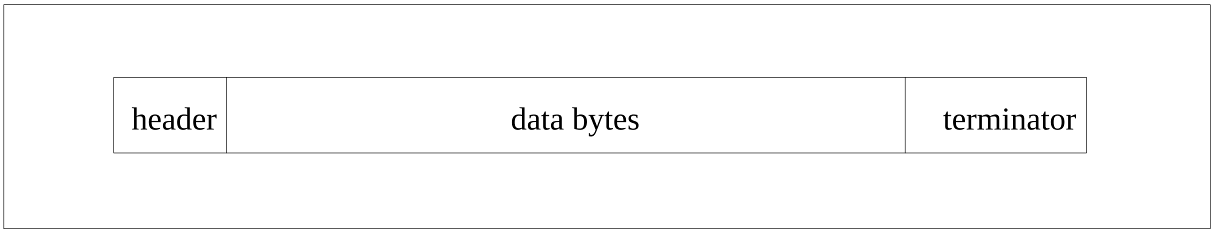


Figure 3.3 Packet format

3.3.4 Higher level protocols

A variety of higher level protocols can be layered on top of this basic system. DS-Link packets can be used as a transport mechanism for protocols defined by other standards such as ATM, SCI and FibreChannel. They also provide very efficient support for synchronised channel communication, as described below.

3.4 Channel communication

A model of communication which can be implemented very efficiently by DS-Links is based on the ideas of communicating sequential processes. The notion of ‘process’ is very general, and applies equally to pieces of hardware and pieces of software. Each process can be regarded as a ‘‘black box” with internal state, which can communicate with other processes using communication channels. Each channel is a point–to–point connection between two processes. One process always inputs from the channel and the other always outputs to it. Communication is synchronized: the first process ready to communicate waits until the second is also ready, then the data is copied from the outputting process to the inputting process and both processes continue. Because a channel is external to the processes which use it, it provides a connection between them which hides their location and internal structure from each other. This means that the interface of a process can be separated from its internal structure (which may involve sub–processes), allowing the easy application of structured engineering principles.

3.4.1 Virtual channels

Each OS-Link of the original transputers implemented only two channels, one in each direction. To map a particular piece of software onto a given hardware configuration the programmer had to map processes to processors within the constraints of available connectivity. The problem is illustrated in figure 3.4 where 3 channels are required between two processors, but only a single link connection is available.

One response to this problem is the addition of more links. However this does not really solve the problem, since the number of extra links that can be added is limited by VLSI technology. Neither does this ‘solution’ address the more general communication problems in networks, such as communication between non-adjacent processors, or combining networks in a simple and regular way.

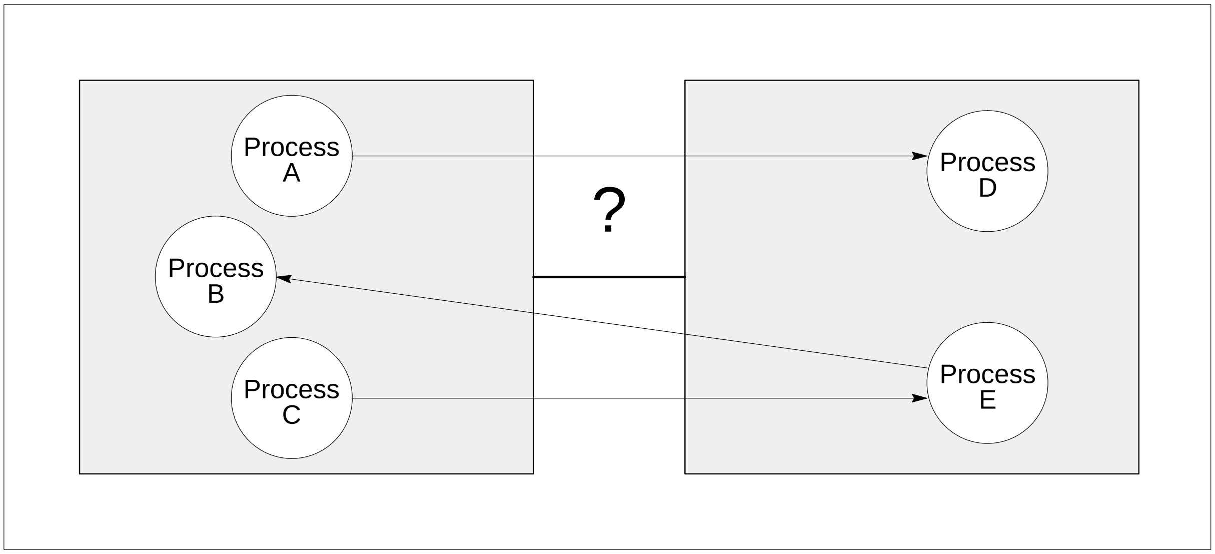


Figure 3.4 Multiple communication channels required between devices

A better solution is to add multiplexing hardware to allow any number of processes to use each link, so that physical links can be shared transparently. These channels which share a link are known as 'virtual channels'.

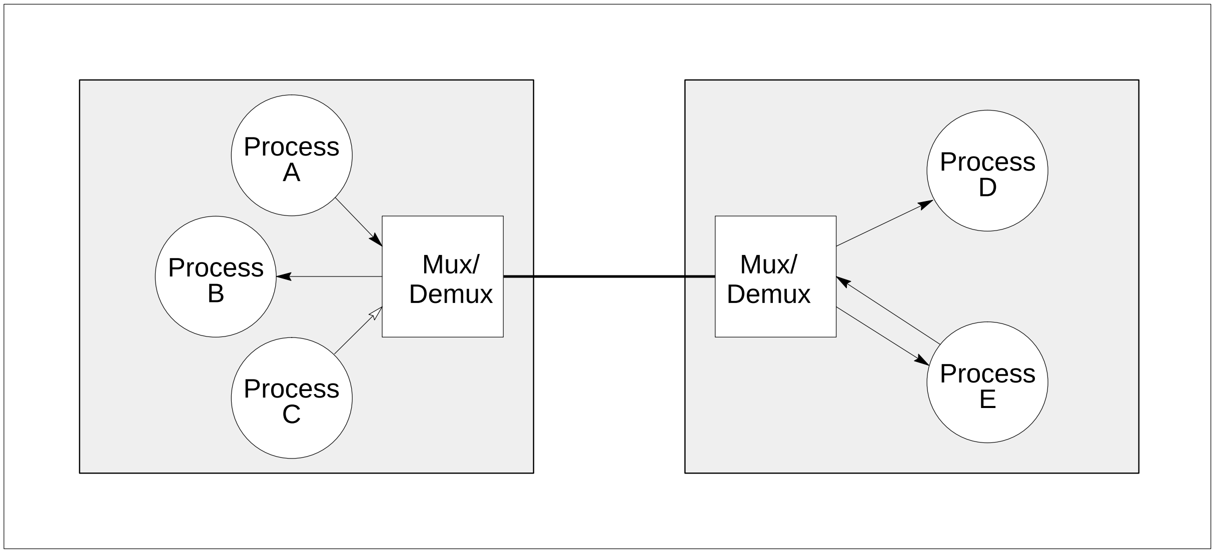


Figure 3.5 Shared DS-Links between devices

Virtual links

Each message sent across a link is divided into packets. Every packet requires a header to identify its channel. Packets from messages on different channels are interleaved on the link. There are two important advantages to this:

- Channels are, generally, not busy all the time, so the multiplexing can make better use of hardware resource by keeping the links busy with messages from different channels.

- Messages from different channels can effectively be sent concurrently – the device does not have to wait for a long message to complete before sending another.

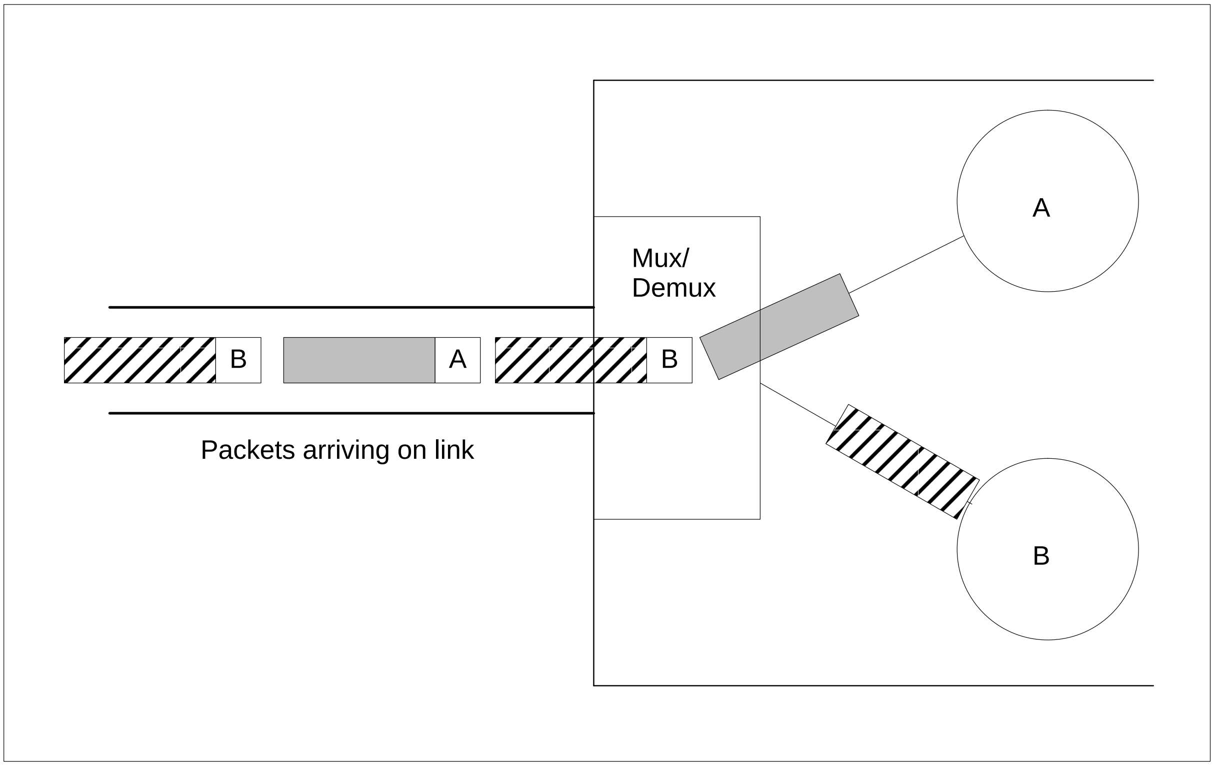


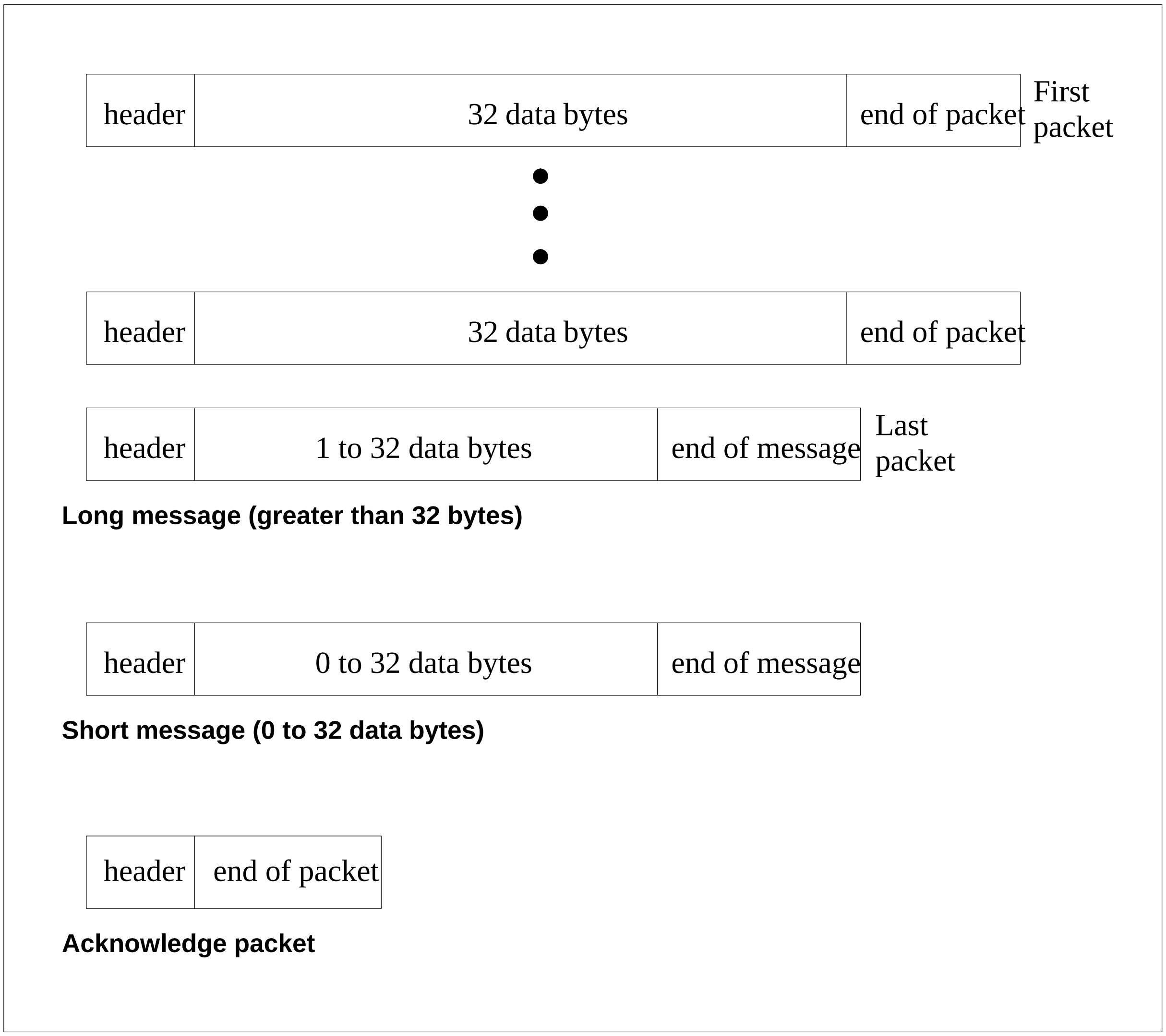
Figure 3.6   Multiple channels sharing a link

In this specific protocol, a packet can contain up to 32 data bytes. If a message is longer than 32 bytes then it is split up into a number of packets all, except the last, terminated by an ‘end of packet’ token. The last packet of the message, which may contain less than a full 32 bytes, is terminated by an ‘end of message’ token. Shorter messages can be sent in a single packet, containing 0 to 32 bytes of data, terminated by the ‘end of message’ token. Messages are always sent using the minimum possible number of packets.

Packet acknowledgements are sent as zero length packets terminated with an ‘end of packet’ token. This type of packet can never occur as part of a message because a zero length data packet must always be the last, and only, packet of a message, and will therefore be terminated by an ‘end of message’ token. Each packet of a message must be acknowledged by receipt of an acknowledge packet before the next can be sent. Process synchronization is ensured by delaying the acknowledgement of the first packet of a message until a process is ready to input from the channel, and delaying continuation of the outputting process until all the packets of the message have been sent and acknowledged.

Virtual channels are always created in pairs to form a ‘virtual link’. This means it is not necessary to include a return address in packets, since acknowledgements are simply sent back along the other channel of the virtual link. The strict acknowledgement protocol means that it is not necessary to include sequence numbers in the packets, even when the routing network is non–deterministic!

The specific formats of packets used in this system are illustrated in figure 3.7.



Long message (greater than 32 bytes)

Short message (0 to 32 data bytes)

Acknowledge packet

Figure 3.7 High Level protocol packet formats

3.5 Errors on links

The DS-Links are designed to be highly reliable within a single subsystem and can be operated in one of two environments, determined by a flag at each end of the link, called **LocalizeError**.

In applications where all connections are on a single board or within a single box, the communications system can reasonably be regarded as being totally reliable. In this environment errors are considered to be extremely rare, but are treated as being catastrophic should one occur. If an error occurs it will be detected and reported. Normal practice will then be to reset the subsystem in which the error has occurred and to restart the application. This minimizes the overheads on each communication, but if an error does occur there will be an interruption in the operation of the system.

For other applications, for instance when a disconnect or parity error may be expected during normal operation, a higher level of fault–tolerance is required. This is supported by localizing errors to the link on which they occur, by setting the **LocalizeError** bit of the link to 1. If an error occurs, packets in transit at the time of the error will be discarded or truncated, and the link will be reset automatically. This minimizes the interruption of the operation of a system, but imposes an overhead on all communications in order to deal with the possibility that data may be lost.

3.5.1 Errors detected

The DS-Link token protocol allows two common types of error to be detected. Firstly the parity system will detect all single bit errors at the DS-Link token level, and secondly, because each output link, once started, continues to transmit an uninterrupted stream of tokens, the physical disconnection of a link can be detected.

Disconnection errors

If the links are disconnected for any reason whilst they are running then flow control and token synchronization may be lost. In order to restart the link it is therefore necessary to reset both ends to a known flow control and token synchronization point.

Disconnection is detected if, after a token has been received, no tokens are seen on the input link in any 1.6 microsecond window. Once a disconnection error has been detected the link halts its output. This will subsequently be detected as a disconnect error at the other end, and will cause that link to halt its output also. It then resets itself, and waits 12.8 microseconds before allowing communication to restart. This time is sufficient to ensure that both ends of the link have observed disconnection and cycled through reset back into the waiting state. The connection may now be restarted.

Parity errors

Following a parity error, both bit–level token synchronization and flow control status are no longer valid, therefore both ends of the link must be reset. This is done autonomously by the DS-Link using an exchange–of–silence protocol.

When a DS-Link detects a parity error on its input it halts its output. This will subsequently be detected as a disconnect error at the other end, and will cause that link to halt its output also, causing a disconnect to be detected at the first end. The normal disconnect behavior described above will then ensure that both ends are reset (irrespective of line delay) before either is allowed to restart.

3.6 Network communications: the IMS C104

The use of DS-Links for directly connecting devices has already been described. The link protocol not only simplifies the use of links between devices but also provides hardware support for routing messages across a network.

The system described previously packetizes messages to be sent over a link and adds a header to each packet to identify the virtual channel. These headers can also be used for routing packets through a communication system connecting a number of devices together. This extends the idea of multiple channels on a single hardware link to multiple channels through a communications system; a communications channel can be established between any two devices even if they are not directly connected.

Because the link architecture allows all the virtual channels of a device to use a single link, complete, system-wide connectivity can be provided by connecting just one link from each device to the routing network. This can be exploited in a number of ways. For example, two or more networks can be used in parallel to increase bandwidth, to provide fault–tolerance, or as a ‘user’ network running in parallel with a physically separate ‘system’ network.

The IMS C104 is a device with 32 DS-Links which can route packets between every pair of links with low latency. An important benefit of using serial links is that it is easy to implement a full crossbar in VLSI, even with a large number of links. The use of a crossbar allows packets to be passing through all links at the same time, making the best possible use of the available bandwidth.

If the routing logic can be kept simple it can be provided for all the input links in the router. This avoids the need to share the hardware, which would cause extra delays when several packets arrive at the same time. It is also desirable to avoid the need for the large number of packet buffers commonly used in routing systems. The use of small buffers and simple routing hardware allows a single VLSI chip to provide efficient routing between a large number of links.

A single IMS C104 can be used to provide full connectivity between 32 devices. IMS C104s can also be connected together to build larger switch networks connecting any number of devices.

3.6.1 Wormhole routing

The IMS C104 includes a full 32 x 32 non-blocking crossbar switch, enabling messages to be routed from any of its links to any other link. In order to minimize latency, the switch uses 'wormhole routing', in which the connection through the crossbar is set up as soon as the header has been read. The header and the rest of the packet can start being transmitted from the output link immediately. The path through the switch disappears after the 'end of packet/message' token has passed through. This is illustrated in figure 3.8. This method is simple to implement and provides very low latency as the entire packet does not have to be read in before the connection is made.

Minimizing routing delays

The ability to start outputting a packet while it is still being input can significantly reduce delay, especially in lightly loaded networks. The delay can be further minimized by keeping the headers short and by using fast, simple hardware to determine the link to be used for output. The IMS C104 uses a simple routing algorithm based on interval labelling (described in section 3.6.3).

Because the route through each IMS C104 disappears as soon as the packet has passed through and the packets from all the channels that pass through a particular link are interleaved, no single virtual channel can monopolize a route through a network. Messages will not be blocked waiting for another message to pass through the system, they will only have to wait for one packet.

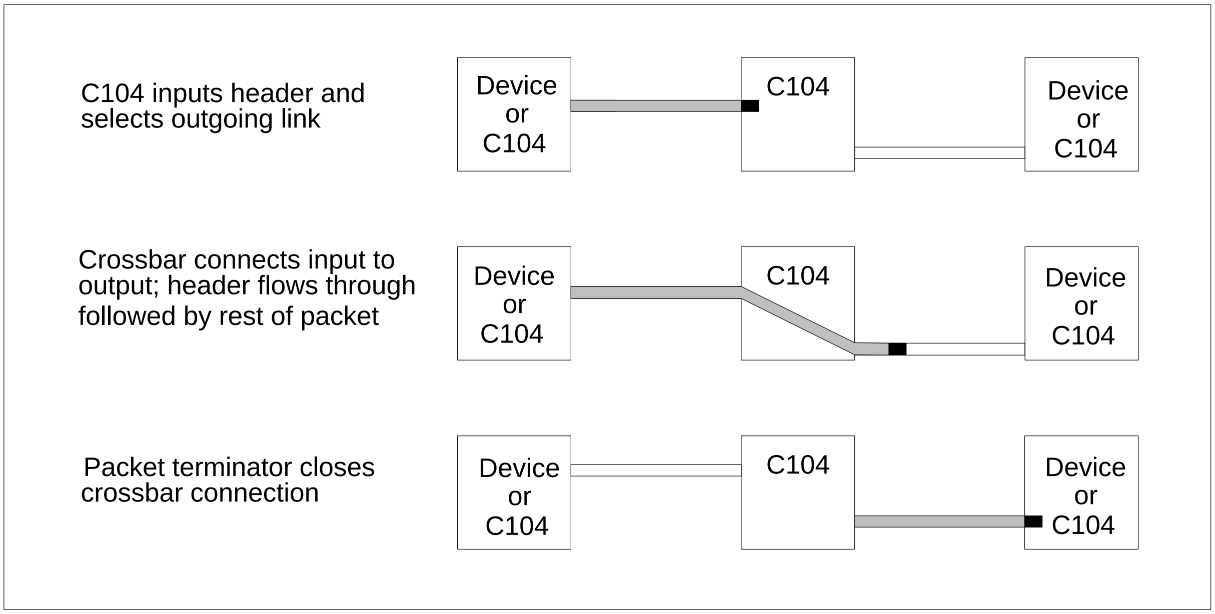

C104 inputs header and selects outgoing link
Device or C104
C104
Device or C104
Crossbar connects input to output; header flows through followed by rest of packet
Device or C104
C104
Device or C104
Packet terminator closes crossbar connection
Device or C104
C104
Device or C104

Figure 3.8 Packet passing through IMS C104

The IMS C104s that the packets pass through do not need to have information about the complete route to the destination, only which link each packet should be sent out of at each point. Each of the IMS C104s in the network is programmed with information that determines which output link should be used for each header value. In this way, each IMS C104 can route packets out of whichever link will send it towards its destination.

3.6.2 Header deletion

An approach that simplifies the construction of networks is to provide two levels of header on each packet. The first header specifies the destination device (actually, the output link from the routing network), and is removed as the packet leaves the routing system. This exposes the second header which tells the destination device which process (actually, which virtual channel) this packet is for. To support this, the IMS C104 can route packets of any length. Any information after the initial header bytes used by the IMS C104 is just treated as part of the packet, even if it is going to be interpreted as a header elsewhere in the system. Any output link of the IMS C104 can be set to do header deletion, i.e. to remove the routing header from the front of each packet after it been used to make the routing decision. The first part of the remaining data is then treated as a header by the next device that receives the packet.

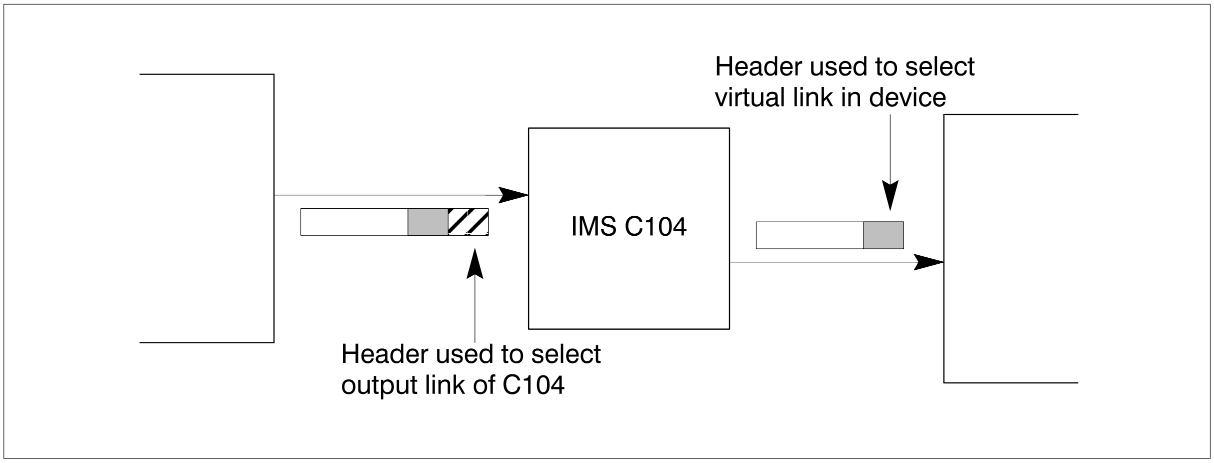


Figure 3.9 Header deletion

As can be seen from figure 3.10, by using separate headers to identify the destination device and a channel within that device, the labelling of links in a routing network is separated from the labelling of virtual channels within each device. For instance, if the same 2 byte header were used to do all the routing in a network, then the virtual channels in all the devices would have to be uniquely labelled with a value in the range 0 to 64K. However, by using two 1 byte headers, all the devices can use virtual channel numbers in the range 0 to 255. The first byte of the header will be used by the routing system to ensure that the packets reach the appropriate device before the virtual channel number is decoded.

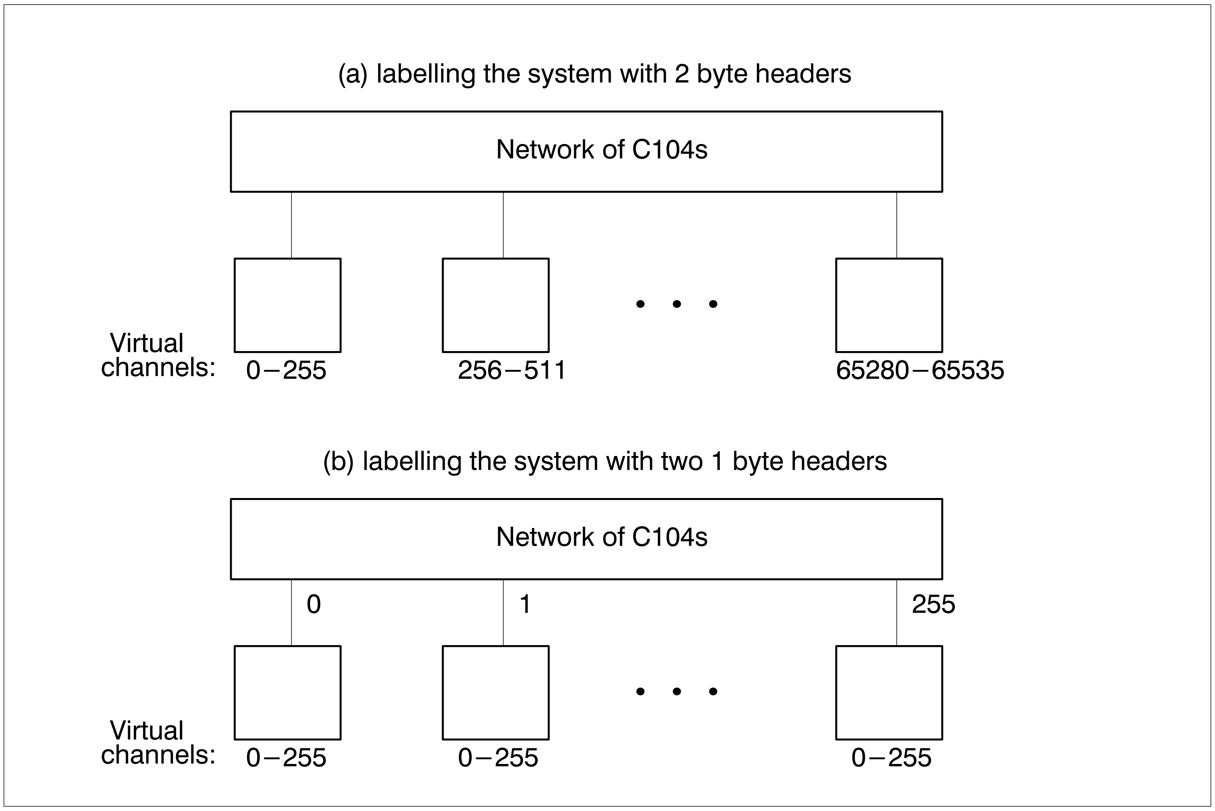


Figure 3.10 Using header deletion to label a network

The advantages of using header deletion in a network are:

- It separates the headers for virtual channels from those for the routing network.
- The labelling of the network can be done independently of the application using the network.
- There is no limit to the number of virtual channels that can be handled by a system.
- By keeping the header for routing short, routing latency is minimized.

Any number of headers can be added to the beginning of a packet so that header deletion can also be used to combine hierarchies of networks as shown in figure 3.11. An extra header is added to route the message through each network. The header at the front of each packet is deleted as it leaves each network to enter a sub-network. This is just like the local–national–international hierarchy of telephone numbers. Since the operation of the IMS C104 is completely independent of the length of the packets, the fact that header deletion changes the length of a packet as it passes through the network causes no problem at all.

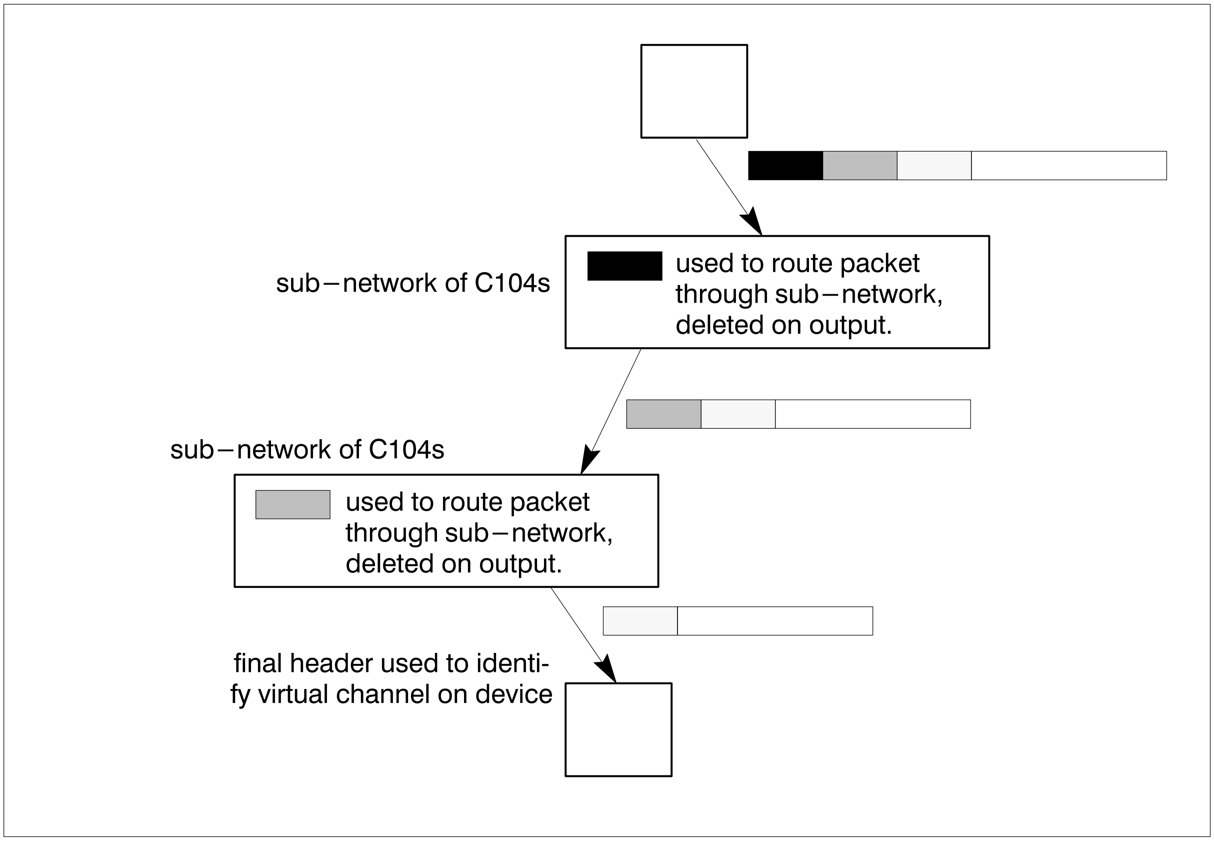
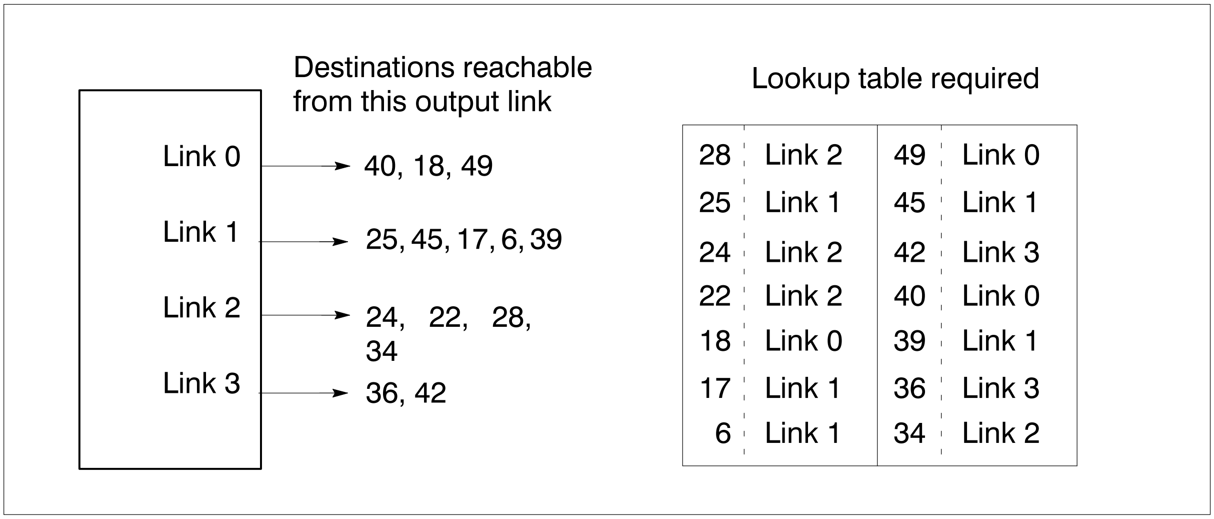

sub−network of C104s
used to route packet through sub−network, deleted on output.
sub−network of C104s
used to route packet through sub−network, deleted on output.
final header used to identify virtual channel on device

Figure 3.11 Using header deletion to route through sub-networks

3.6.3 Labelling networks

For each IMS C104 there will be a number of destinations which can be reached via each of its output links. Therefore, there needs to be a method of deciding which output link to use for each packet that arrives. The addresses that can be reached through any link will depend on the way the network is labelled. An obvious way of determining which destinations are accessible from each link, is to have a lookup table associated with all the outputs (see figure 3.12). In practice, this is difficult to implement. There must be an upper bound on the lookup table size and it may require a large number of comparisons between the header value and the contents of the table. This is inefficient in silicon area and also potentially slow.



28	Link 2	49	Link 0
25	Link 1	45	Link 1
24	Link 2	42	Link 3
22	Link 2	40	Link 0
18	Link 0	39	Link 1
17	Link 1	36	Link 3
6	Link 1	34	Link 2

Figure 3.12 Labelling a network

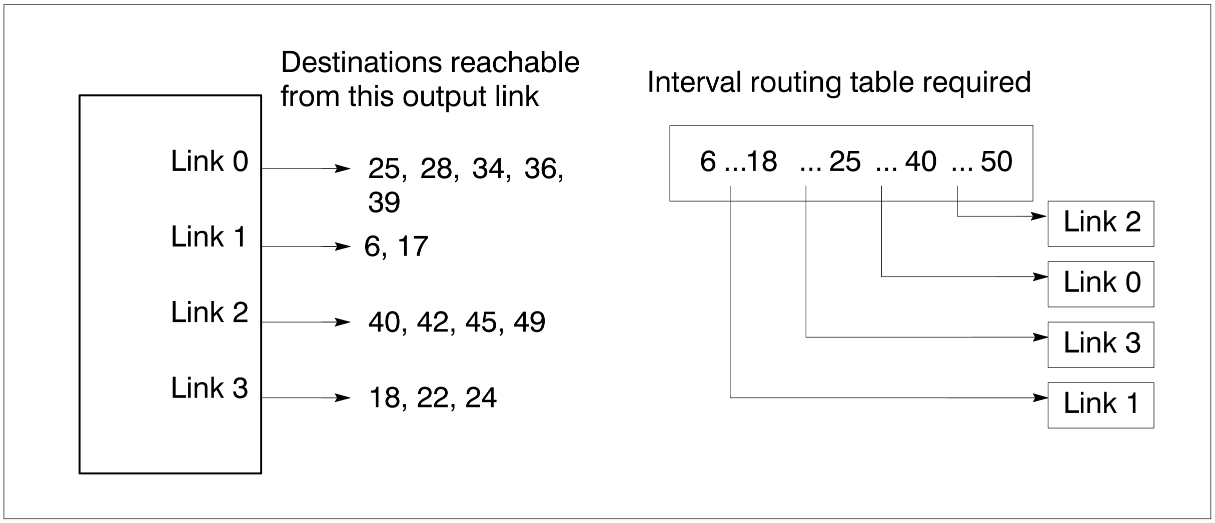


Figure 3.13 Interval labelling

However, a labelling scheme can be chosen for the network such that each output link has a *range* of node addresses that can be reached through it. As long as the ranges for each link are non-overlapping, a very simple test is possible. The header just has to be tested to see into which range, or interval, it falls and, hence, which output link to use. For example, in figure 3.13, a header with address *n* would be tested against each of the four intervals shown below:

Interval	**Output link**
$6 \leq n < 18$	1
$18 \leq n < 25$	3
$25 \leq n < 40$	0
$40 \leq n < 50$	2

The advantages of interval labelling are that:

- It is ‘complete’ – any network can be labelled so that all packets reach their destinations.
- It provides an absolute address for each device in a network, so keeping the calculation of headers simple.
- It is simple to implement in hardware – it requires little silicon area which means it can be provided for a large number of links as well as keeping costs and power dissipation down.
- Because it is simple, it is also very fast, keeping routing delays to a minimum.

Figure 3.14 gives an example of interval routing for a network of two IMS C104’s and six IMS T9000 transputers showing one virtual link per transputer. The example shows six virtual channels, one to each transputer, labeled 0 to 5. The interval contains the labels of all virtual channels accessible via that link. The interval notation [3,6) is read as meaning that the header value must be greater than or equal to 3 and less than 6. If the progress of a packet with the header value 4 is followed from IMS T9000$_1$ then it is evident that it passes through both IMS C104s before leaving on the link to IMS T9000$_4$.

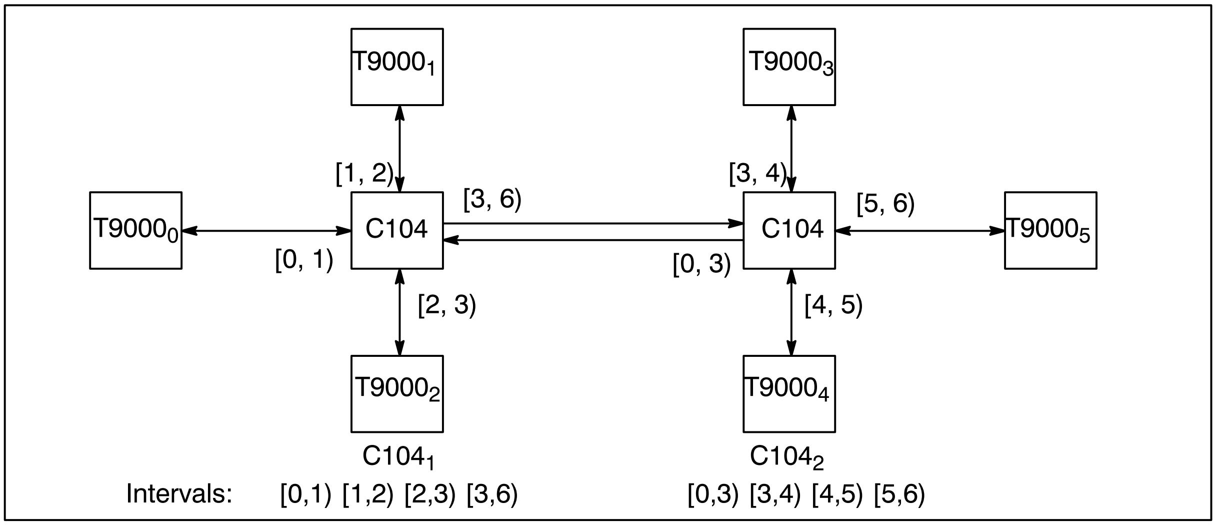


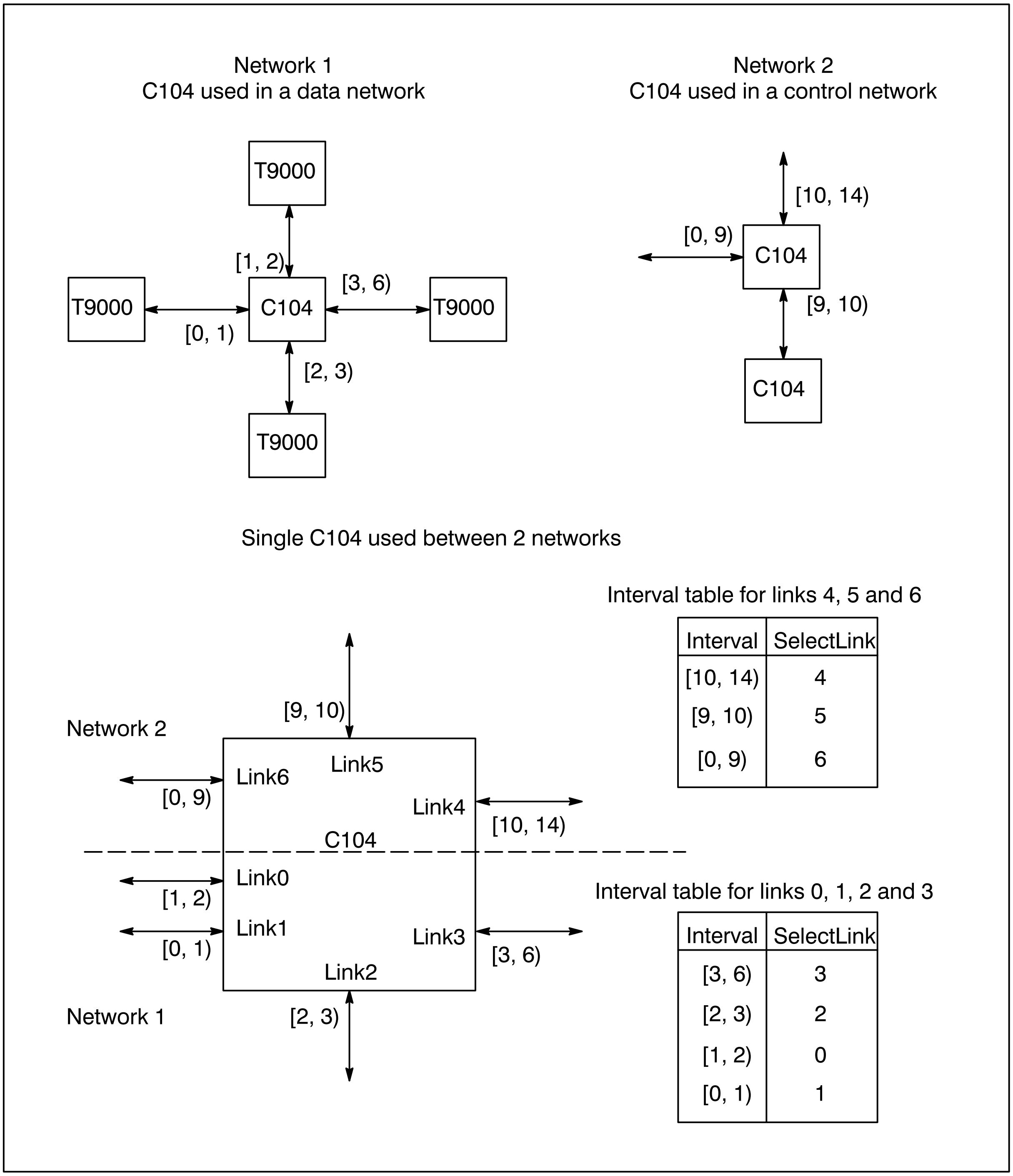
Figure 3.14 Interval routing

It is possible to label all the major network topologies such that packets follow an optimal route through the network, and such that the network is deadlock free. Optimal, deadlock free labelings are available for grids, hypercubes, trees and various multi–stage networks. A few topologies, such as rings, cannot be labeled in an optimal deadlock free manner. Although they can be labeled so that they are deadlock free, this is at the expense of not using one or more of the links, so that the labeling is not optimal. Optimal deadlock free labelings exist if one or more additional links are used.

3.6.4 Partitioning

All the parameters determining the routing are programmable on a per link basis. This enables an IMS C104 to be used as part of two or more different networks. For example, a single IMS C104 could be used for access to both a data network and a control network (see figure 3.15).

Partitioning provides economy in small systems, where using an IMS C104 solely for a control network is not desired, whilst maintaining absolute security. By ensuring that no link belonging to one partition occurs in any interval routing table in another partition, it is guaranteed that no packet can be routed from one partition to another, whatever the value of its header.



Interval table for links 4, 5 and 6

Interval	SelectLink
[10, 14)	4
[9, 10)	5
[0, 9)	6

Interval table for links 0, 1, 2 and 3

Interval	SelectLink
[3, 6)	3
[2, 3)	2
[1, 2)	0
[0, 1)	1

Figure 3.15 Using partitioning to enable one C104 to be used by two different networks

3.6.5 Grouped adaptive routing

The IMS C104 can implement *grouped adaptive routing*. Sets of consecutive numbered links can be configured to be grouped, so that a packet routed to any link in the set would be sent down any free link of the set[11]. This achieves improved network performance in terms of both latency and throughput.

Figure 3.16 gives an example of grouped adaptive routing. Consider a message routed from C104$_1$, via C104$_2$, to T9000$_1$. On entering C104$_2$ the header specifies that the message is to be output down **Link5** to T9000$_1$. If **Link5** is already in use, the message will automatically be

[11] This is also sometimes called a *hunt group*.

routed down **Link6**, **Link7** or **Link8**, dependent on which link is available first. The links can be configured in groups by setting a bit for each link, which can be set to 'Start' to begin a group and 'Continue' to be included in a group.

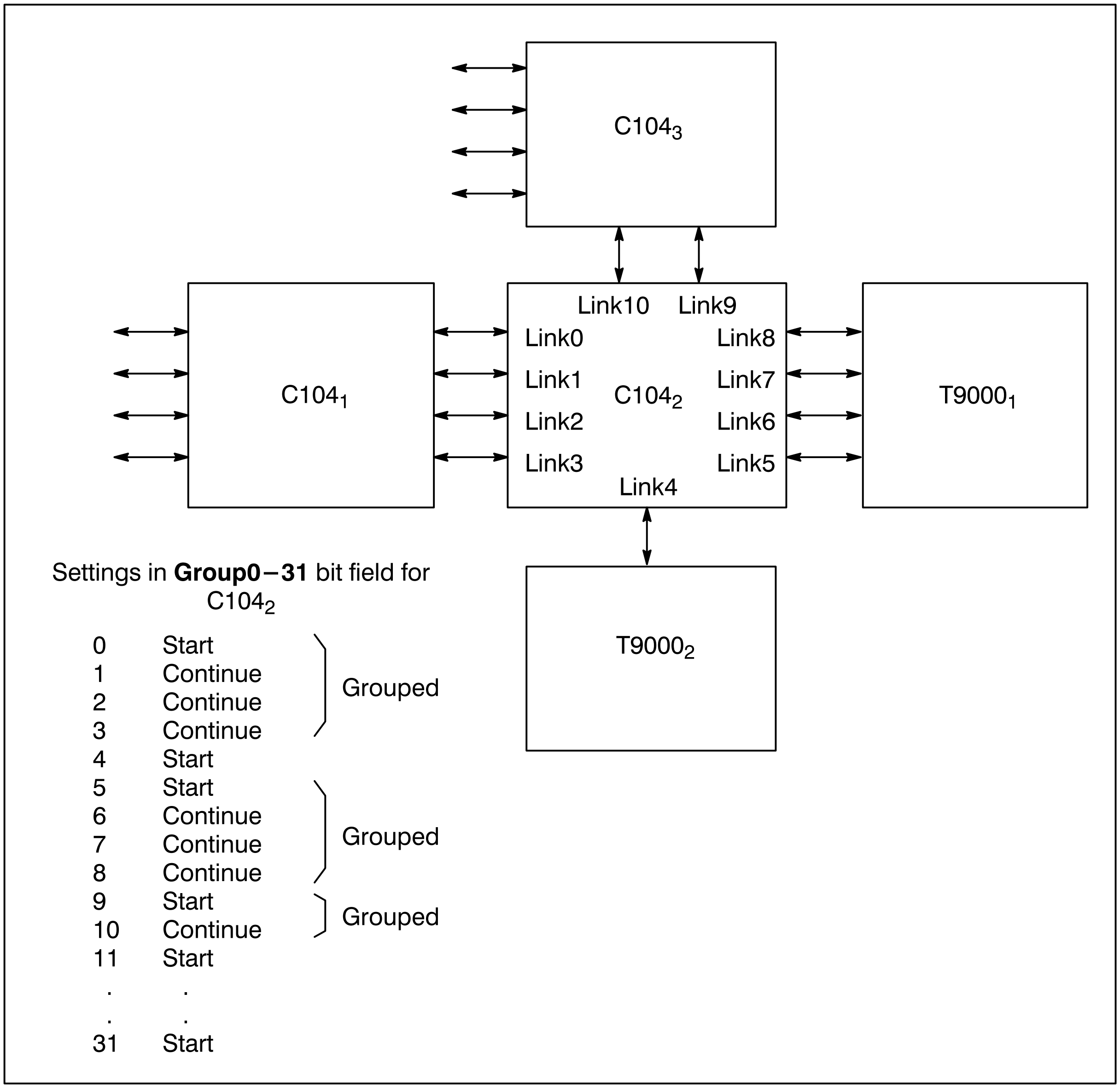


Figure 3.16 Grouped adaptive routing

Grouped adaptive routing is also very effective in multi–stage networks such as those illustrated in figures 7.1 to 7.4. Since all the centre–stage switches are equivalent, all the links from each first–stage switch towards the centre can be grouped together, allowing a high degree of adaption to dynamic traffic conditions.

3.7 Conclusion

DS-Link technology provides reliable, high–speed serial communications at low cost, in a simple form which is suitable for a wide range of applications. A simple protocol, implemented in hardware, keeps overheads down whilst allowing more complex functions to be layered on top of it. It also permits high–performance routing devices to be constructed, from which efficient systems of any size can be built to provide very high system bandwidth and fault–tolerance.