# MODULE -6

# Distributed Embedded Systems

## 1.INTRODUCTION

In a distributed embedded system, several processing elements (PEs) (either microprocessors or ASICs) are connected by a network that allows them to communicate. The application is distributed over the PEs, and some of the work is done at each node in the network.

### 1.1 Reasons to build network-based embedded systems.

1.When the processing tasks are physically distributed, it may be necessary to put some of the computing power near where the events occur.

2. Data reduction is another important reason for distributed processing. It may be possible to perform some initial signal processing on captured data to reduce its volume.

3.Modularity is another motivation for network-based design. For instance, when a large system is assembled out of existing components, those components may use a network port as a clean interface that does not interfere with the internal operation of the component in ways that using the microprocessor bus would.

4.A distributed system can also be easier to debug the microprocessors in one part of the network can be used to probe components in another part of the network. Finally, in some cases, networks are used to build fault tolerance into systems.

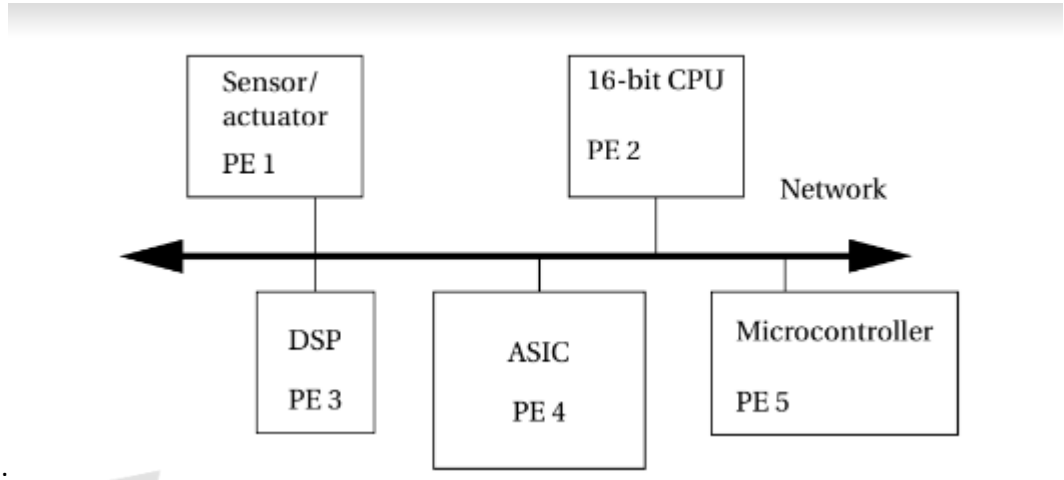
5.Distributed embedded system design is another example of hardware/software co-design, since we must design the network topology as well as the software running on the network.

## 2.DISTRIBUTED EMBEDDED ARCHITECTURE

A distributed embedded system can be organized in many different ways, but its basic units are the PE and the network as illustrated in Figure 8.1.

A PE may be an instruction set processor such as a DSP, CPU, or microcontroller, as well as a nonprogrammable unit such as the ASICs used to implement PE 4. An I/O device such as PE 1 (which we call here a sensor or actuator, depending on whether it provides input or output) may also be a PE, so long as it can speak the network protocol to communicate with

other PEs. The network in this case is a bus, but other network topologies are also possible. It is also possible that the system can use more than one network, such as when relatively independent functions require relatively little communication among them. We often refer to the connection between PEs provided by the network as a communication link. The system of PEs and networks forms the hardware platform on which the application runs.




.

**Fig 8.1 An example of a distributed embedded system.**

## 2.1 Why Distributed?

In some cases,distributed systems are necessary because the devices that the PEs communicate with are physically separated. If the deadlines for processing the data are short, it may be more cost-effective to put the PEs where the data are located rather than build a higher-speed network to carry the data to a distant, fast PE.

An important advantage of a distributed system with several CPUs is that one part of the system can be used to help diagnose problems in another part.Whether you are debugging a prototype or diagnosing a problem in the field, isolating the error to one part of the system can be difficult when everything is done on a single CPU. If you have several CPUs in the system,you can use one to generate inputs for another and to watch its output.

## 2.2 Network Abstractions

Networks are complex systems. Ideally, they provide high-level services while hiding many of the details of data transmission from the other components in the system. In order to help understand (and design) networks, the International Standards Organization has developed a seven-layer model for networks known as Open Systems Interconnection (OSI ) models

The seven layers of the OSI model, shown in Figure 8.2, are intended to cover a broad spectrum of networks and their uses. Some networks may not need the services of one or more layers because the higher layers may be totally missing or an intermediate layer may not be necessary. However, any data network should fit into the OSI model. The OSI layers from lowest to highest level of abstraction are described below.

■ Physical: The physical layer defines the basic properties of the interface between systems, including the physical connections ( plugs and wires), electrical properties, basic functions of the electrical and physical components, and the basic procedures for exchanging bits.

■ Data link: The primary purpose of this layer is error detection and control across a single link. However, if the network requires multiple hops over several data links, the data link layer does not define the mechanism for data integrity between hops, but only within a single hop.

■ Network: This layer defines the basic end-to-end data transmission service.The network layer is particularly important in multihop networks.

■ Transport: The transport layer defines connection-oriented services that ensure that data are delivered in the proper order and without errors across multiple links.This layer may also try to optimize network resource utilization.

■ Session: A session provides mechanisms for controlling the interaction of end-user services across a network, such as data grouping and checkpointing.

■ Presentation: This layer defines data exchange formats and provides transformation utilities to application programs.

■ Application: The application layer provides the application interface between the network and end users.

| | |
|---|---|
| Application | End-use interface |
| Presentation | Data format |
| Session | Application dialog control |
| Transport | Connections |
| Network | End-to-end service |
| Data link | Reliable data transport |
| Physical | Mechanical, electrical |

fig 8.2 The OSI Model layers

## 2.3 Hardware and Software Architectures

Distributed embedded systems can be organized in many different ways depending upon the needs of the application and cost constraints. One good way to understand possible architectures is to consider the different types of interconnection networks that can be used.

A point-to-point link establishes a connection between exactly two PEs. Point-to-point links are simple to design precisely because they deal with only two components.We do not have to worry about other PEs interfering with communication on the link.


**FIGURE 8.3**
A signal processing system built from print-to-point links.

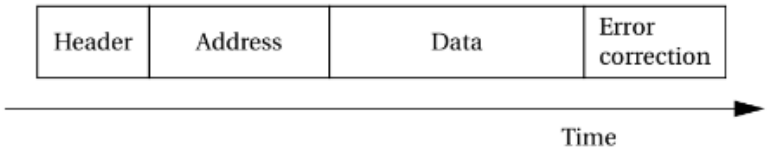



**FIGURE 8.4**
Format of a typical message on a bus.

Figure 8.3 shows a simple example of a distributed embedded system built from point-to-point links. The input signal is sampled by the input device and passed to the first digital filter, F1, over a point-to-point link. The results of that filter are sent through a second point-

to-point link to filter F2. The results in turn are sent to the output device over a third point-to-point link. A digital filtering system requires that its outputs arrive at strict intervals, which means that the filters must process their inputs in a timely fashion. Using point-to-point connections allows both F1 and F2 to receive a new sample and send a new output at the same time without worrying about collisions on the communications network.

A bus is a more general form of network since it allows multiple devices to be connected to it. Like a microprocessor bus, PEs connected to the bus have addresses. Communications on the bus generally take the form of packets as illustrated in Figure 8.4. A packet contains an address for the destination and the data to be delivered. It frequently includes error detection/correction information such as parity. It also may include bits that serve to signal to other PEs that the bus is in use, such as the header shown in the figure. The data to be transmitted from one PE to another may not fit exactly into the size of the data payload on the packet. It is the responsibility of the transmitting PE to divide its data into packets; the receiving PE must of course reassemble the complete data message from the packets.

Distributed system buses must be arbitrated to control simultaneous access,just as with microprocessor buses. Arbitration scheme types are summarized below.

■ Fixed-priority arbitration always gives priority to competing devices in the same way. If a high-priority and a low-priority device both have long data transmissions ready at the same time, it is quite possible that the low-priority device will not be able to transmit anything until the high-priority device has sent all its data packets.

■ Fair arbitration schemes make sure that no device is starved. Round-robin arbitration is the most commonly used of the fair arbitration schemes. The PCI bus requires that the arbitration scheme used on the bus must be fair, although it does not specify a particular arbitration scheme. Most implementations of PCI use round-robin arbitration. A bus has limited available bandwidth. Since all devices connect to the bus,communications can interfere with each other. Other network topologies can be used to reduce communication conflicts.At the opposite end of the generality spectrum from the bus is the crossbar network shown in Figure 8.5.

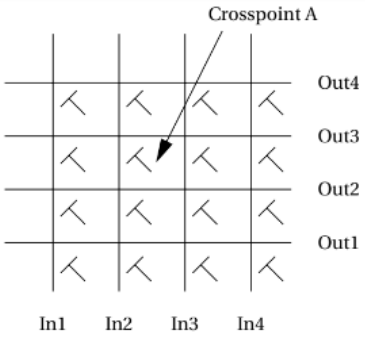



**FIGURE 8.5**
A crossbar network.

A crossbar not only allows any input to be connected to any output, it also allows all combinations of input/output connections to be made. Thus, for example, we can simultaneouslyconnect in1 to out4, in2 to out3, in3 to out2, and in4 to out1 or any other combinations of inputs. (Multicast connections can also be made from one input to several outputs.) A crosspoint is a switch that connects an input to an output. To connect an input to an output, we activate the crosspoint at the intersection between the corresponding input and output lines in the crossbar. For example, to connect in2 and out3 in the figure, we would activate crossbar A as shown. The major drawback of the crossbar network is expense: The size of the network grows as the square of the number of inputs (assuming the numbers of inputs and outputs are equal).

Many other networks have been designed that provide varying amounts of parallel communication at varying hardware costs. Figure 8.6 shows an example multistage network. The crossbar of Figure 8.5 is a direct network in which messages go from source to destination without going through any memory element. Multistage networks have intermediate routing nodes to guide the data packets.

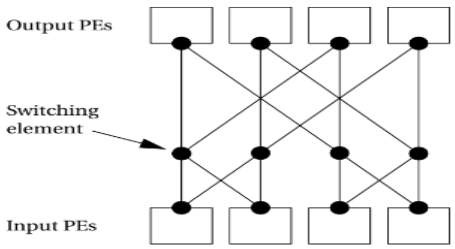



**FIGURE 8.6**
A multistage network.

## 2.4 Message Passing Programming

Distributed embedded systems do not have shared memory,so they must communicate by passing messages.We will refer to a message as the natural communication unit of an algorithm; in general, a message must be broken up into packets to be sent on the network. A procedural interface for sending a packet might look like the following:

**send_packet(address,data);**

The routine should return a value to indicate whether the message was sent successfully if the network includes a handshaking protocol. If the message to be sent is longer than a packet,it must be broken up into packet-size data segments as follows:

**for (i = 0; i < message.length;i=i+ PACKET_SIZE)**

**send_packet(address,&message.data[i]);**

The above code uses a loop to break up an arbitrary-length message into packet- size chunks. However, clever system design may be able to recast the message to take advantage of the packet format. For example, clever encoding may reduce the length of the message enough so that it fits into a single packet. On the other hand, if the message is shorter than a packet or not an even multiple of the packet data size, some extra information may be packed into the remaining bits of a packet. Reception of a packet will probably be implemented with interrupts. The simplest procedural interface will simply check to see whether a received message is waiting in a buffer. In a more complex RTOS-based system, reception of a packetmay enable a process for execution.

Network protocols may encourage a **data-push design** style for the system built around the network. In a single-CPU environment,a program typically initiates a read whenever it wants data. In many networked systems, nodes send values out without any request from the intended user of the system. Data-push programming makes sense for periodic data—if the data will always be used at regular intervals, we can reduce data traffic on the network by automatically sending it when it is needed.

## 3.NETWORKS FOR EMBEDDED SYSTEMS

Networks for embedded computing span a broad range of requirements; many of those requirements are very different from those for general-purpose networks. Some networks are

used in safety-critical applications, such as automotive control. Some networks, such as those used in consumer electronics systems, must be very inexpensive. Other networks,such as industrial control networks,must be extremely rugged and reliable.

Several interconnect networks have been developed especially for distributed embedded computing:

■ The I2 C bus is used in microcontroller-based systems.

■ The Controller Area Network (CAN) bus was developed for automotive electronics. It provides megabit rates and can handle large numbers of devices.

■ Ethernet and variations of standard Ethernet are used for a variety of control applications.

In addition, many networks designed for general-purpose computing have been put to use in embedded applications as well.

## 3.1 The I2C Bus

The ***I 2C bus*** is a well-known bus commonly used to link microcontrollers into systems. I 2C is designed to be low cost,easy to implement, and of moderate speed (up to 100 KB/s for the standard bus and up to 400 KB/s for the extended bus). As a result, it uses only two lines: the serial data line (SDL) for data and the serial clock line (SCL), which indicates when valid data are on the data line. Figure 8.7 shows the structure of a typical I2C bus system. Every node in the network is connected to both SCL and SDL. Some nodes may be able to act as bus masters and the bus may have more than one master. Other nodes may act as slaves that only respond to requests from masters.

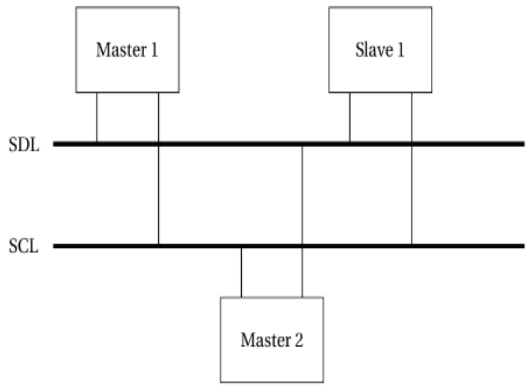



**FIGURE 8.7**

Structure of an $I^2C$ bus system.

The I2C bus is designed as a multimaster bus—any one of several different devices may act as the master at various times. As a result, there is no global master to generate the clock signal on SCL. Instead, a master drives both SCL and SDL .when it is sending data.When the bus is idle, both SCL and SDL remain high.When two devices try to drive either SCL or SDL to different values, the open collector open drain circuitry prevents errors, but each master device must listen to the bus while transmitting to be sure that it is not interfering with another message—if the device receives a different value than it is trying to transmit, then it knows that it interfering with another message.

Every I2 C device has an address. The addresses of the devices are determined by the system designer, usually as part of the program for the I2 C driver. The addresses must of course be chosen so that no two devices in the system have the same address. A device address is 7 bits in the standard I2 C definition (the extended I2 C allows 10-bit addresses).

A bus transaction is initiated by a start signal and completed with an end signal as follows:

- A start is signaled by leaving the SCL high and sending a 1 to 0 transition on SDL.
- A stop is signaled by setting the SCL high and sending a 0 to 1 transition on SDL.

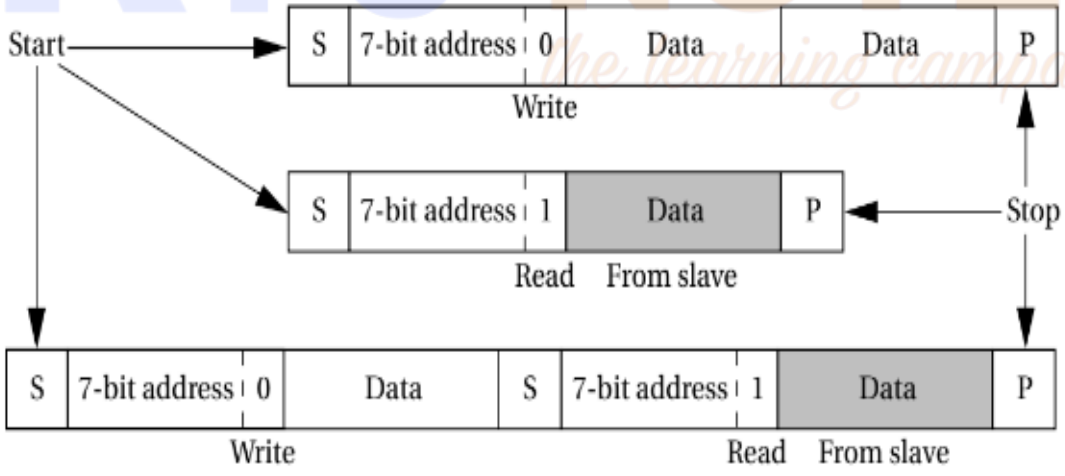



**FIGURE 8.11**

Typical bus transactions on the $I^2C$ bus.

## 3.2 Ethernet

Ethernet is very widely used as a local area network for general-purpose computing. The physical organization of an Ethernet is very simple, as shown in Figure 8.14. The network is a bus with a single signal path; the Ethernet standard allows for several different implementations such as twisted pair and coaxial cable. Unlike the I2 C bus, nodes on the Ethernet are not synchronized—they can send their bits at any time. I2 C relies on the fact that a collision can be detected and quashed within a single bit time thanks to synchronization. But since Ethernet nodes are not synchronized, if two nodes decide to transmit at the same time, the message will be ruined.

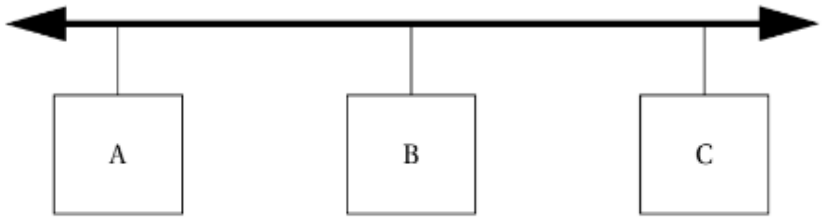
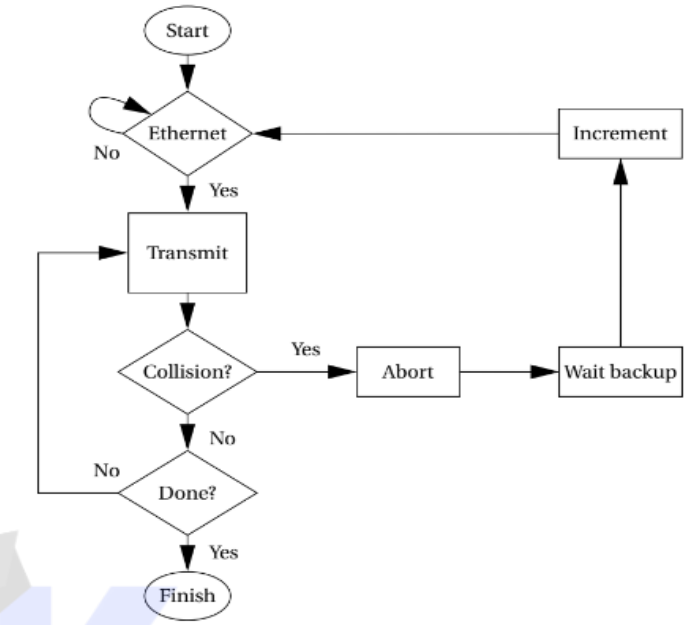



**FIGURE 8.14**
Ethernet organization.

The Ethernet arbitration scheme is known as ***Carrier Sense Multiple Access*** with ***Collision Detection (CSMA/CD)***. The algorithm is outlined in Figure 8.15. A node that has a message waits for the bus to become silent and then starts transmitting. It simultaneously listens, and if it hears another transmission that interferes with its transmission, it stops transmitting and waits to retransmit. The waiting time is random, but weighted by an exponential function of the number of times the message has been aborted. Figure 8.16 shows the expo- nential backoff function both before and after it is modulated by the random wait time. Since a message may be interfered with several times before it is successfully transmitted, the ***exponential backoff*** technique helps to ensure that the network does not become overloaded at high demand factors. The random factor in the wait time minimizes the chance that two messages will repeatedly interfere with each other. The maximum length of an Ethernet is determined by the nodes' ability to detect collisions.

**FIGURE 8.15**

The Ethernet CSMA/CD algorithm.



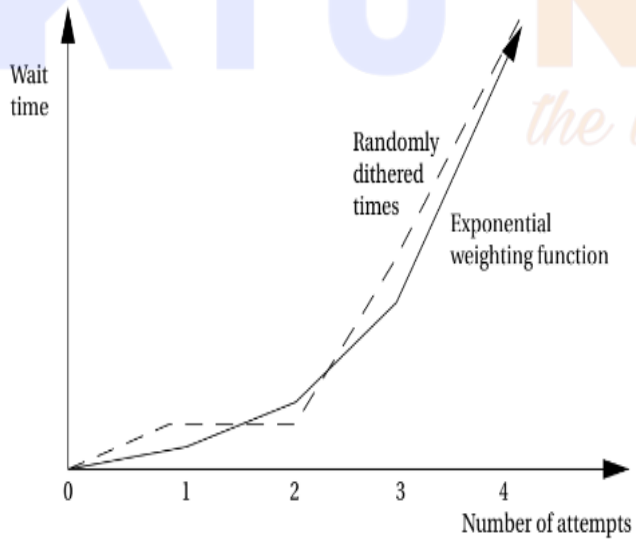




**FIGURE 8.16**

Exponential backoff times.

| Preamble | Start frame | Destination address | Source address | Length | Data | Padding | CRC |
|---|---|---|---|---|---|---|---|

**FIGURE 8.17**

Ethernet packet format.

## 3.4 Fieldbus

Fieldbus is a set of standards for industrial control and instrumentation systems.The H1 standard uses a twisted-pair physical layer that runs at 31.25 MB/s. It is designed for device integration and process control. The High Speed Ethernet standard is used for backbone networks in industrial plants. It is based on the 100 MB/s Ethernet standard. It can integrate devices and subsystems.

## 4. NETWORK-BASED DESIGN

Many embedded networks are designed for low cost and therefore do not provide excessively high communication speed. The ***message delay*** for a single message with no contention (as would be the case in a point-to-point connection) can be modeled as **tm= tx + tn + tr** where *tx* is the transmitter side overhead, *tn* is the network transmission time, and *tr* is the receiver side overhead. In I2 C, *tx* and *tr* are negligible relative to *tn* .

If the network uses fixed-priority arbitration, the network availability delay is unbounded for all but the highest-priority device. Since the highest-priority device always gets the network first, unless there is an application-specific limit on how long it will transmit before relinquishing the network, it can keep blocking the other devices indefinitely.

■ If the network uses fair arbitration, the network availability delay is bounded. In the case of round-robin arbitration, if there are *N* devices, then the worst- case network availability delay is ***ty= td+ tm*** where *t*arb is the delay incurred for arbitration. *t*arb is usually small compared to transmission time.

Even when round-robin arbitration is used to bound the network availability delay, the waiting time can be very long. If we add acknowledgment and data corruption into the analysis, figuring network delay is more difficult.

Our process scheduling model assumed that we could interrupt processes at any point. But network communications are organized into packets. In most networks we cannot interrupt a packet transmission to take over the network for a higher priority packet.. When a low-priority message is on the network, the network is effectively allocated to that low-priority message, allowing it to block higher-priority messages. This cannot cause deadlock since each message has a bounded length,but it can slow down critical communications. The only solution is to analyze network behavior to determine whether priority inversion causes some messages to be delayed for too long.

A round-robin arbitrated network puts all communications at the same priority. This does not eliminate the priority inversion problem because processes still have priorities. Thus far we have assumed a ***single-hop network****:* A message is received at its intended destination directly from the source, without going through any other net- work node. It is possible to build ***multihop networks*** in which messages are routed through network nodes to get to their destinations. (Using a multistage network does not necessarily mean using a multihop network—the stages in a multistage network are generally much smaller than the network PEs.) Figure 8.18 shows an example of a multihop communication. The hardware platform has two separate networks ( perhaps so that communications between subsets of the PEs do not interfere), but there is no direct path from *M* 1 to *M* 5. The message is therefore routed through *M* 3, which reads it from one network and sends it on to the other one. Analyzing delays

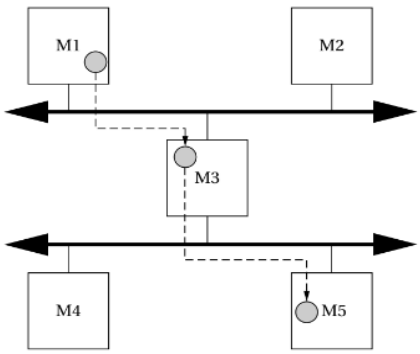



**FIGURE 8.18**
A multihop communication.

## 5. INTERNET-ENABLED SYSTEMS

Some very different types of distributed embedded system are rapidly emerging— the ***Internet-enabled embedded system*** and ***Internet appliances***. The Internet is not well suited to the real-time tasks that are the bread and butter of embedded computing, but it does provide a rich environment for non–real-time interaction.

### 5.1 Internet

The **Internet Protocol (IP)** is the fundamental protocol on the ***Internet*** . It provides connectionless, packet-based communication. Industrial automation has long been a good application area for Internet-based embedded sys- tems. Information appliances that use the Internet are rapidly becoming another use of IP in embedded computing.

Internet protocol is not defined over a particular physical implementation—it is an ***internetworking*** standard. Internet packets are assumed to be carried by some other network, such as an Ethernet. In general, an Internet packet will travel over several different networks from source to destination. The IP allows data to flow seamlessly through these networks from one end user to another. The relationship between IP and individual networks is illustrated in Figure 8.19.

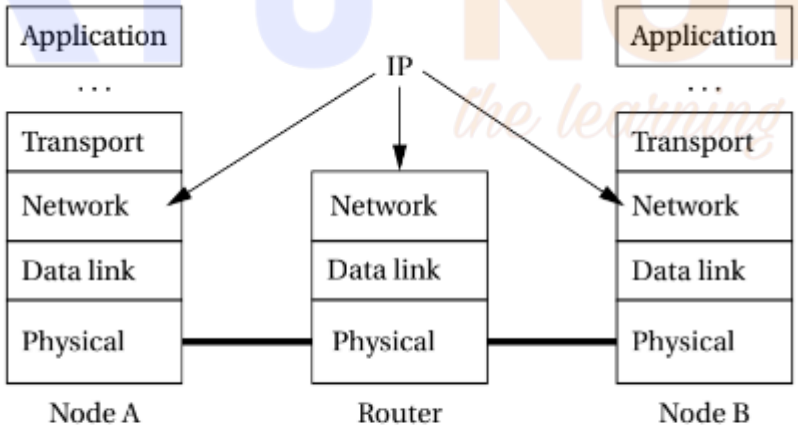



**FIGURE 8.19**
Protocol utilization in Internet communication.

IP works at the network layer. When node A wants to send data to node B, the application's data pass through several layers of the protocol stack to send to the IP. IP creates packets for routing to the destination, which are then sent to the *data link* and *physical* layers. A node that transmits data among different types of networks is known as a ***router*** . The router's functionality must go up to the IP layer, but since it is not running applications, it does not need to go to higher levels of the OSI model. In general, a packet may go through several routers to get to its destination. At the destination, the IP layer provides data to the transport layer and ultimately the receiving application. As the data pass through several layers of the protocol stack, the IP packet data are encapsulated in packet formats appropriate to each layer.

The basic format of an IP packet is shown in Figure 8.20. The header and data payload are both of variable length. The maximum total length of the header and data payload is 65,535 bytes.An Internet address is a number (32 bits in early versions of IP, 128 bits in IPv6). The IP address is typically written in the form xxx.xx.xx.xx.

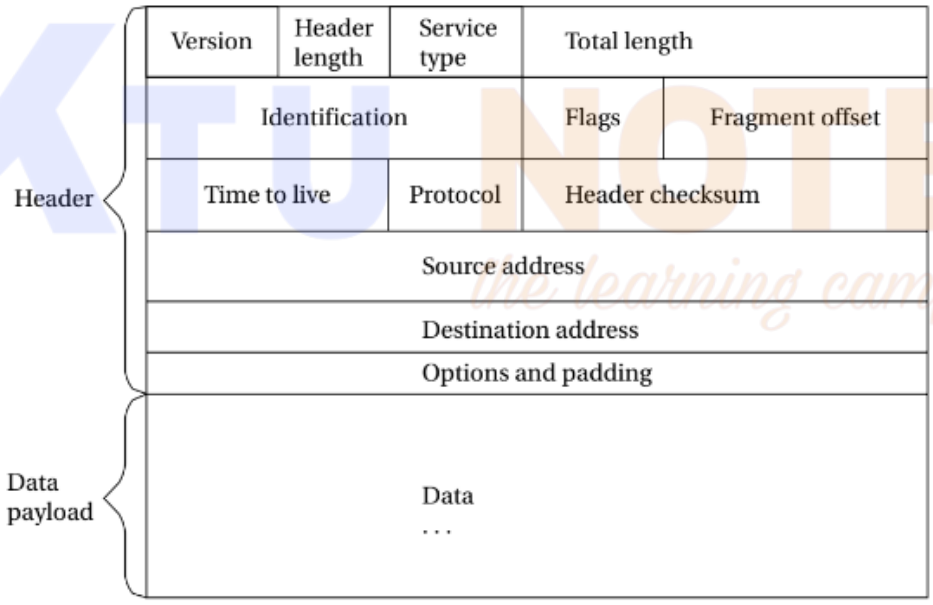



**FIGURE 8.20**
IP packet structure.

The fact that IP works at the network layer tells us that it does not guarantee that a packet is delivered to its destination. Furthermore,packets that do arrive may come out of order. This is referred to as ***best-effort routing*** . Since routes for data may change quickly with subsequent packets being routed along very different paths with different delays, real-time performance of IP can be hard to predict. When a small network is contained totally within the embedded system,

performance can be evaluated through simulation or other methods because the possible inputs are limited. Since the performance of the Internet may depend on worldwide usage patterns, its real-time performance is inherently harder to predict.

The Internet also provides higher-level services built on top of IP. The ***Transmission Control Protocol (TCP)*** is one such example. It provides a connection- oriented service that ensures that data arrive in the appropriate order, and it uses an acknowledgment protocol to ensure that packets arrive. Because many higher- level services are built on top of TCP, the basic protocol is often referred to as TCP/IP.

Wide Web service, ***Simple Mail Transfer Protocol*** for email, and Telnet for virtual terminals. A separate transport protocol, ***User Datagram Protocol*** , is used as The Internet service stack. the basis for the network management services provided by the ***Simple Network Management Protocol*** .

**5.2 Internet Applications**

The Internet provides a standard way for an embedded system to act in concert with other devices and with users, such as:

■ One of the earliest Internet-enabled embedded systems was the laser printer. High-end laser printers often use IP to receive print jobs from host machines.

■ Portable Internet devices can display Web pages, read email, and synchronize calendar information with remote computers.

■ A home control system allows the homeowner to remotely monitor and control home cameras, lights, and so on.

Although there are higher-level services that provide more time-sensitive delivery mechanisms for the Internet, the basic incarnation of the Internet is not well suited to hard real-time operations. However, IP is a very good way to let the embed- ded system talk to other systems. IP provides a way for both special-purpose and standard programs (such as Web browsers) to talk to the embedded system. This non–real-time interaction can be used to monitor the system, set its configuration, and interact with it.

The Internet provides a wide range of services built on top of IP. Since code size is an important issue in many embedded systems, one architectural decision that must be made is to determine which Internet services will be needed by the system. This choice depends on

the type of data service required, such as connectionless versus connection oriented, streaming vs. non- streaming, and so on. It also depends on the application code and its services: does the system look to the rest of the Internet like a terminal, a Web server, or something else?.

**6 .Recent Trends in Embedded Computing**

An embedded system is an application-specific system designed with a combination of hardware and software to meet real-time constraints. The key characteristics of embedded industrial systems include speed, security, size, and power.

**1. Patterns insight from the applications of embedded systems in real life**

Embedded systems are more than part of human life. For instance, one cannot imagine life without mobile phones for personal communication. Its presence is virtually unavoidable in almost all facets of human endeavor. While we search on patterns in each of these application spaces, we can clearly identify the trend as to where the future of embedded systems is heading.

**2. Multicore in embedded**

With a lot functionalities being added, the need for high performance in embedded systems has become inevitable and so developers are increasingly leaning towards multicore processors in their systems design decision.

- It drove to higher power consumption and so the higher thermals;
- Overall cost increased as the peripherals surrounding also needed to operate at matching speed, which was truly not practical in all cases, there by driving the costs.

**3.Embedded operating systems**

Traditionally embedded systems did away with an operating system (OS), it had lightweight control program/monitor to offer limited I/O and memory services, however, as the systems became complex, it was inevitable to have OS which offered low latency real-time response, low foot print both in time and space and give all traditional functionality such as memory protection, error checking/report and transparent interprocess communication, which can be

applied to communications, consumer electronics, industry controls, automotive electronics and aerospace/national defense.

**4.Embedded digital security and surveillance**

Digital security and surveillance is currently in the host of new applications in the embedded arena which is benefiting from multicore phenomenon. Older systems needed more human intervention, but new systems offer intelligent systems to operate multisite, integrated and net centric systems that optimizes the resources needed to complete the job. The applications based on computer vision and tracking offers multiple benefits in capturing, post processing and identification and alerting of security video in realtime.

**6.Healthcare**

Electronic medical device and other technological innovations with the convergence of biotech, nanotech, manufacturing tech, communication tech and device, sensor technologies are making breathtaking transformations in healthcare delivery and creating new health care paradigms.

Bio med devices tech is being applied into wide variety of analytical problems including medicine, surgery and drug discovery, these devices are portable diagnostic imaging and home monitoring such as cholesterol monitors, blood glucose meters and with recent innovations paving way for miniaturization of devices, replacement organs and tissues, earlier use of more accurate diagnostics, and advances in information technology, became available thru Silicon Chip revolution.

**7.Improved Security for Embedded Devices**

With the rise of the Internet of Things (IoT), the primary focus of developers and manufacturers is on security. In 2019, advanced technologies for embedded security will emerge as key generators for identifying devices in an IoT network, and as microcontroller security solutions that isolate security operations from normal operations.

**8.Cloud Connectivity and Mesh Networking**

Getting embedded industrial systems connected to the internet and cloud can take weeks and months in the traditional development cycle. Consequently, cloud connectivity tools will be an important future market for embedded systems. These tools are designed to simplify the process of connecting embedded systems with cloud-based services by reducing the underlying hardware complexities.

A similar yet innovative market for low-energy IoT device developers is Bluetooth mesh networks. These solutions can be used for seamless connectivity of nearby devices while reducing energy consumption and costs.

**9. Optimization for Lower Energy Consumption**

A key challenge for developers is the optimization of battery-powered devices for low power consumption and maximum uptime. Several solutions are under development for monitoring and reducing the energy consumption of embedded devices that we can expect to see in 2019. These include energy monitors and visualizations that can help developers fine-tune their embedded systems, and advanced Bluetooth and Wi-Fi modules that consume less power at the hardware layer.

**10.Visualizations in Real Time**

Developers currently lack tools for monitoring and visualizing their embedded industrial systems in real time. The industry is working on real-time visualization tools that will give software engineers the ability to review embedded software execution. These tools will enable developers to keep a check on key metrics such as raw or processed sensor data and event-based context switches for tracking the performance of embedded systems.

**11.Deep Learning**

Deep learning solutions represent a rich, yet unexplored embedded systems market that has a range of applications from image processing to audio analysis. Even though developers are primarily focused on security and cloud connectivity right now, deep learning and artificial intelligence concepts will soon emerge as a trend in embedded systems.