Unit 5- Transport Layer

The transport layer is not just another layer. It is the heart of the whole protocol hierarchy. Its task is to provide reliable, cost-effective data transport from the source machine to the destination machine, independently of the physical network or networks currently in use. Without the transport layer, the whole concept of layered protocols would make little sense. In this chapter we will study the transport layer in detail, including its services, design, protocols, and performance.

6.1 The Transport Service

- Services Provided to the Upper Layers
- Transport Service Primitives
- Berkeley Sockets

6.1.1 Services Provided to the Upper Layers

The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective service to its users, normally processes in the application layer. To achieve this goal, the transport layer makes use of the services provided by the network layer. The hardware and/or software within the transport layer that does the work is called the transport entity. The (logical) relationship of the network, transport, and application layers is illustrated in Fig. 6-1.

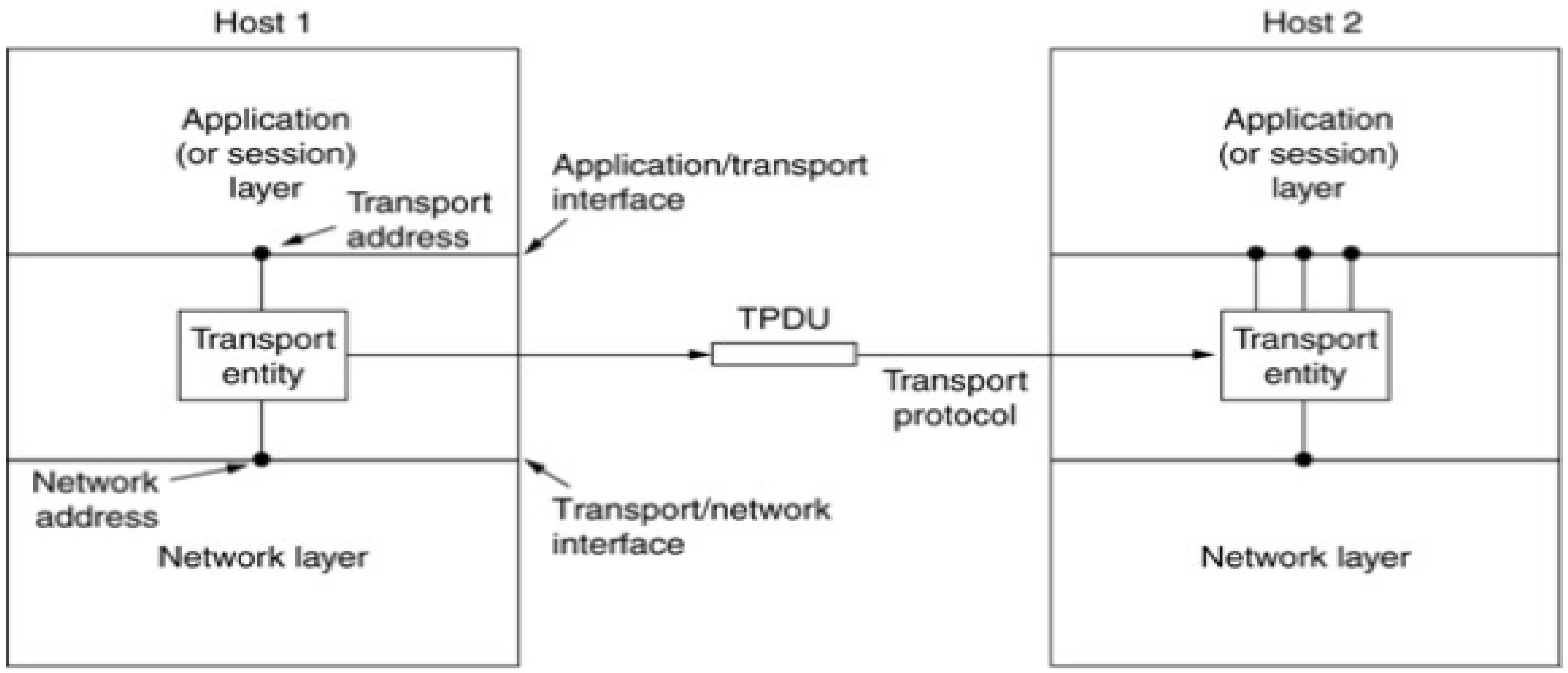

Fig. 6-1 The network, transport, and application layers.

- **The connection-oriented and the connectionless**
- **Why need the transport layer.**

Just as there are two types of network service, connection-oriented and connectionless, there are also two types of transport service. The transport service is similar to the network service in many ways.

The transport code runs entirely on the users' machines, but the network layer mostly runs on the routers, which are operated by the carrier (at least for a wide area network). What happens if the network layer offers inadequate service? Suppose that it **frequently loses packets** ? What happens if routers **crash from time to time** ?

Problems occur, that's what. **The users have no real control over the network layer,** so they cannot solve the problem of poor service by using better routers or putting more error handling in the data link layer. The only possibility is to put on top of the network layer another layer that improves the quality of the service.

In essence, **the existence of the transport layer makes it possible for the transport service to be more reliable than the underlying network service** . Lost packets and mangled data can be detected and compensated for by the transport layer. Furthermore, the transport service primitives can be implemented as calls to library procedures in order to make them independent of the network service primitives.

Thanks to the transport layer, application programmers can write code according to a standard set of **primitives** and have these programs work on a wide variety of networks, without having to worry about dealing with different subnet interfaces and unreliable transmission.

For this reason, many people have traditionally made a distinction between layers 1 through 4 on the one hand and layer(s) above 4 on the other. The bottom four layers can be seen as the **transport service provider** , whereas the upper layer(s) are the **transport service user** . This distinction of provider versus user has a considerable impact on the design of the layers and puts the transport layer in a key position, since it forms the major boundary between the provider and user of the reliable data transmission service.

6.1.2 Transport Service Primitives

Review primitives.

Transport primitives are very important, because many programs (and thus programmers) see the transport primitives. Consequently, the transport service must be convenient and easy to use.

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Fig. 6-2 The primitives for a simple transport service.

TPDU

TPDU (Transport Protocol Data Unit) is a term used for messages sent from transport entity to transport entity. Thus, TPDUs (exchanged by the transport layer) are contained in packets (exchanged by the network layer). In turn, packets are contained in frames (exchanged by the data link layer). When a frame arrives, the data link layer processes the frame header and passes the contents of the frame payload field up to the network entity. The network entity processes the packet header and passes the contents of the packet payload up to the transport entity. This nesting is illustrated in Fig. 6-3.

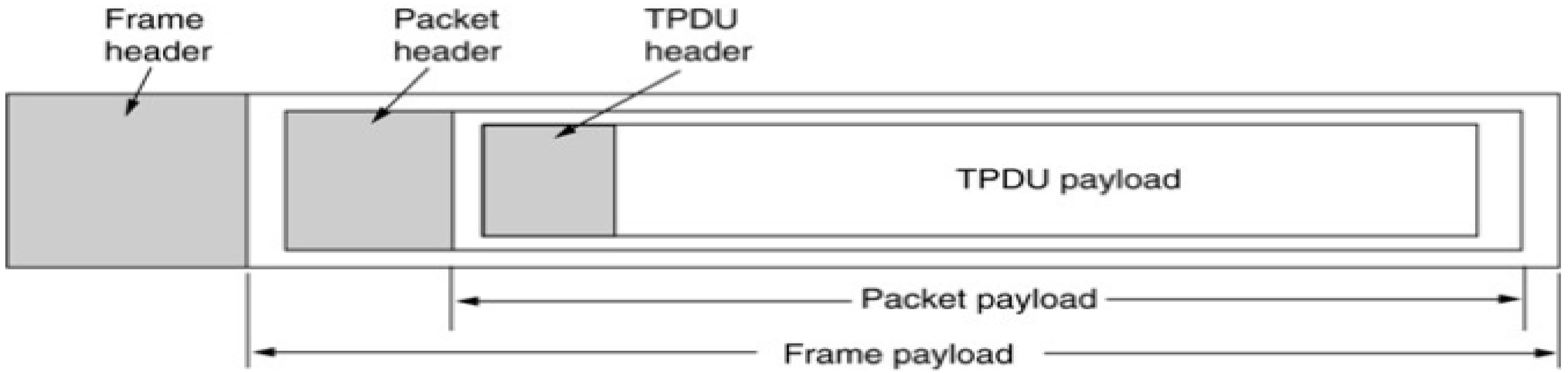

Figure 6-3. The nesting of TPDUs, packets, and frames.

6.1.3 Berkeley Sockets

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Figure 6-5. The socket primitives for TCP.

6.1.3 Berkeley Sockets

- The first four primitives in the list are executed in that order by servers. The SOCKET primitive creates a new endpoint and allocates table space for it within the transport entity.
- The parameters of the call specify the addressing format to be used, the type of service desired (e.g., reliable byte stream), and the protocol. A successful SOCKET call returns an ordinary file descriptor for use in succeeding calls, the same way an OPEN call on a file does.
- Newly created sockets do not have network addresses.
- These are assigned using the BIND primitive. Once a server has bound an address to a socket, remote clients can connect to it.
- The reason for not having the SOCKET call create an ad▯dress directly is that some processes care about their addresses (e.g., they have been using the same address for years and everyone knows this address), whereas others do not.
- Next comes the LISTEN call, which allocates space to queue incoming calls for the case that several clients try to connect at the same time. In contrast to LISTEN in our first example, in the socket model LISTEN is not a blocking call.