NSA3: Tutorial 2 Solutions Guide

1. Synchronous TDM allocates slots in fixed rotation to some maximum number of sources which are currently using a given link. The receiving end of the link knows which slot belongs to each source purely by its position and it follows that a sequence of slots (a channel) must be allocated to any such source before it can use the link. This inherently means that any source wishing to use a synchronous TDM link must request and obtain a channel on the link before it can transmit along it. Such a channel, once obtained, must be released before another source can use it. Both ends of the link must be made aware at setup and release time, which source is using which channel. Thus synchronous TDM implies (temporary) ownership of channels on a link by individual sources. When a source has variable output, it may not, on the one hand, have enough capacity on its channel to cope with a sudden increase in output while, on the other, it may waste bandwidth when it is quiescent for a short period, but has not released its channel. Rapid acquisition and release of channels is typically impractical because of the overhead involved.

   With asynchronous TDM, however, chunks of link bandwidth (whether fixed-size slots or not) are allocated on an individual basis to each fragment of each message ready to be sent on the link, regardless of source. As there is no fixed pre-allocation, and fragments (packets or cells) are queued for the link as they arrive there is total flexibility when data output from any source varies.

2. In the first case, the channel bandwidth is allocated equally between the channels. Each channel (and each source fortunate enough to acquire one) gets

$$\frac{1000000}{32} = 31.25\ kbps$$

   With asynchronous TDM, we assume that a maximum of 32 active sources (equivalent to 32 virtual channels) are allowed to use the line at one time. Each slot must carry an address, which is 5 bits long to allow the receiver to identify which channel a particular piece of data belongs to (the requirement for this address explains why a limit has to be set in the first place. Thus each slot can carry only 11 bits of actual data. Clearly only 11/16 of the total link bandwidth is actually available, and this is divided between 32 channels. In asynchronous TDM a slot may be used by any channel which requires it, but the premise of the question is that all active sources are given an equal division in this particular case. Thus each gets:

$$\frac{1000000 * 11}{32 * 16} = 21.48438\ kbps$$

3. When the medium is an underlying layer, a service access point will typically be implemented in software. There are various ways of implementing layers: for example, in the most general implementation, a user and provider may be separate tasks. Here, a user will exchange primitives with the provider using local inter-process messaging facilities, before which an association must be provided between user and provider (binding). The service access point is thus a message passing port on the provider, associated with some appropriate internal handling procedures and data areas. In other situations, the provider may simply be a library procedure called by the user (this is especially appropriate at upper layers where SAPs are mapped one-to-one onto those of the layer below). Here an SAP is again just a data structure, this time passed as a parameter to the procedure. Clearly an implementation of this type is much more limited

   Where the medium is physical, the SAP may itself be physical, consisting of internal registers and associated logic in some type of interface device. Of course software handlers or drivers may be provided to hide this level of detail from user processes, in which case the actual SAPs, will as above, be implemented as data areas with associated procedures.

4. A name is simply a widely recognised label attached to something. An address is no more than the name (usually numeric) of a SAP. Clearly therefore all addresses are names. A name of anything other than a SAP is not however an address.

5. A connection-oriented service is one which provides users with provider-supported connections (no need for addressing, in-order delivery, etc) and primitives to support these. A connectionless protocol is one where there is no concept of a connection as far as protocol entities are concerned, (no setup or closedown PDUs and no requirement for protocol entities to track PDUs from

different user sources or to pass on such PDUs in any specified order. Suppose a data item is submitted on a service connection. The item will be passed, using one or more PDUs to the remote provider entity. This entity is now supposed to pass each PDU on to the correct SAP, in the correct order. However, since (by definition) there is no record in the PDU's PCI as to which connection the data belongs to, this is not possible. Thus a connection-oriented service cannot be directly supported by a connectionless protocol.

6. In theory open systems should stimulate competition because no single vendor or group of vendors can, through strength of market position, impose proprietary standards which that vendor or group then control. It is easy for a company to exploit a situation where it controls a standard, by, for example, forcing competitors to licence the standard and pay royalties for the privilege of using it, or changing the standard without reference to others thus disadvantaging their product strategies.

7. The role of the data-link function is to impose a frame structure on raw bit streams, so as to create a reliable data transfer service. In all cases indicated there are raw bit streams which require such enhancement, sharing the characteristic of having uniform and relatively short propagation delays (the reasons why this is important will be discussed more fully in Chapter 6). These steams may be treated, in effect, as if travelling over a link with similar properties to a simple wire. Bit streams exist in packet-switched networks between neighbouring nodes (but not end-to-end), end-to-end across circuit switched networks and end-to-end across broadcast networks (the latter streams are often short-lived but generally have short delays with little variation). See Figure 6.2 and associated discussion for more details.

8. A cell relay network is like a packet-switched network but with some special characteristics of its own: cells are all the same size, switches are designed for maximum speed and overheads such as error checking are as much as possible eliminated from switches to enhance performance. End-to-end cell relay networks, like packet switched networks do not guarantee a steady flow of bits, so an end-to-end data-link function is not appropriate. There are bit streams between neighbouring switches and between switches and hosts, which are structured into cells. The layer which creates the cells is similar in some ways to a data-link layer but has more limited functionality. It is worth noting that for some types of traffic, cell relay networks can mimic end-to-end bit streams (see Chapter 9) but this is not general and a true data-link function is not usually present as such in a cell relay network.

9. The idea of running a connectionless layer above a connection-oriented one is not very efficient, but there are circumstances where it might make sense. It is most likely where it is desirable to run a connectionless protocol (typically for compatibility with some standard) over a provider which is inherently connection-oriented. Thus, for example, one might wish to run the connectionless IP protocol over an ATM connection (see Section 9.4.3).

10. If a codeword, say, *a*, is repeated three times, *aaa* and only one instance is corrupted (say by a single error), the destination will receive two copies of *a* and one of some other codeword, say *b*. The destination will immediately be aware that an error has occurred. If (as is generally the case) one corrupt word is more likely than two or three, the destination may justifiably assume on receiving two *a*'s and one *b*, that the *a*'s have arrived correctly and the *b* is wrong. This implies that the message sent was *a*, and error correction has been achieved. Of course, if two copies of the codeword were corrupted correction will fail.

11. Suppose a 7-bit ASCII character is sent , say α and suppose that two characters

$$a_7a_6a_5a_4a_3a_2a_1 \quad \text{and} \quad b_7b_6b_5b_4b_3b_2b_1$$

are received. If there are no errors, these two characters should be identical. Since each bit is subject to error with probability *p*, the probability of no error is:

$$(1-p)^{14}$$

An undetectable error will occur if and only if all the pairs (a_i, b_j) have $a_i = b_i$ but at least one pair is corrupt (otherwise there is no error at all). Since the occurrence of an undetectable error and the occurrence of no error are mutually exclusive, we have:

$$\text{P(undetectable error)} = \text{P(all pairs } (a_i, b_j) \text{ have } a_i = b_i) - \text{P(no error)}$$

Now the probability that a pair (a_i, b_j) has $a_i = b_i$ is

$$\text{P}(a_i \text{ and } b_i \text{ are both changed}) + \text{P}(a_i \text{ and } b_i \text{ are both unchanged}) = p^2 + (1-p)^2$$

The probability that all pairs have $a_i = b_i$ is:

$$[p^2 + (1-p)^2]^7$$

Thus the probability of an undetectable error is:

$$[p^2 + (1-p)^2]^7 - (1-p)^{14}$$

If for example $p = 0.1$ (a very high error rate), the probability of an undetectable error is then:

$$(0.01+0.81)^7-0.9^{14} \approx 0.0205$$