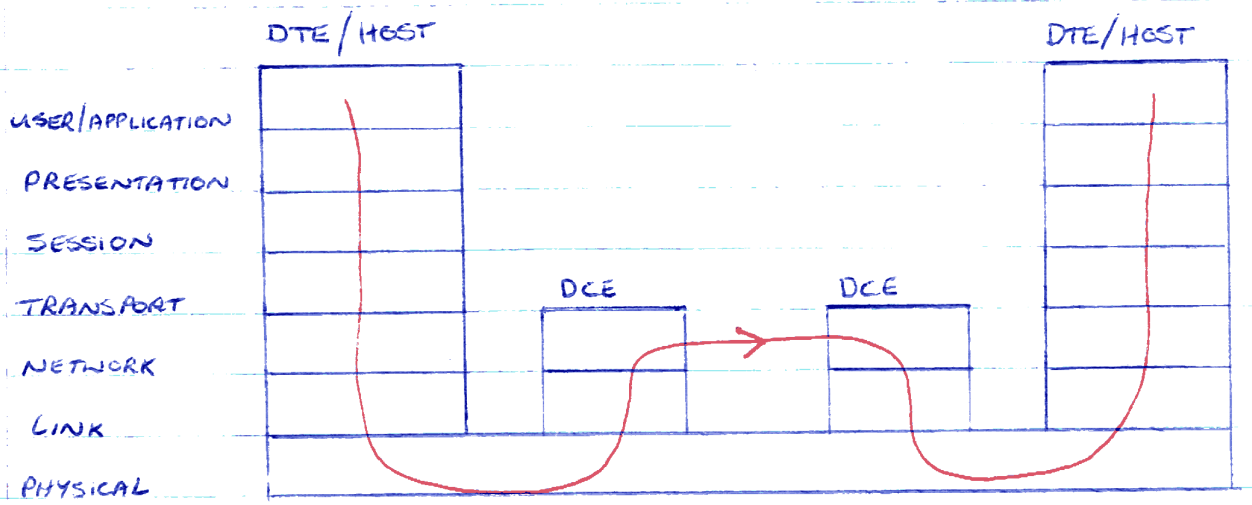
Contents.

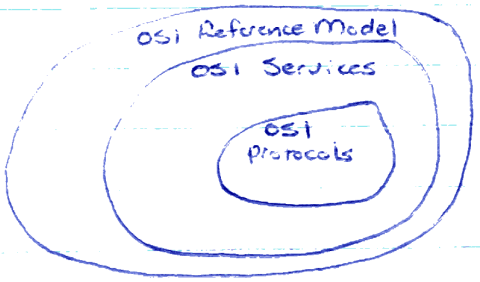
ADVANCED DATA COMMUNICATIONS

Lectured by: P.S. Kritzinger.

From 21-7-86 to 22-8-86

1 OSI (OPEN SYSTEM INTERCONNECTION) REVISITED. (ISO 7498)





The OSI Reference Model, Services and Protocols are successively more detailed specifications and thus more constraining. There are many services and protocols that satisfy the Reference Model. There are fewer that satisfy both the Reference Model and Service Specification, etc. The OSI Reference Model cannot be implemented – it is a model for describing the concepts for coordinating the parallel development of interprocess communication standards.

1.1 ELEMENTS OF THE ARCHITECTURE.

SYSTEMS, LAYERS AND ENTITIES

OSI is concerned with standards for communication between systems. A system is considered to be one or more autonomous computers and their associated software, peripherals and users that are capable of information processing and/or transfer. Layering is used as a structuring technique to allow the network of open systems to be logically decomposed in independent, smaller subsystems. Each individual layer is viewed as being logically composed of a

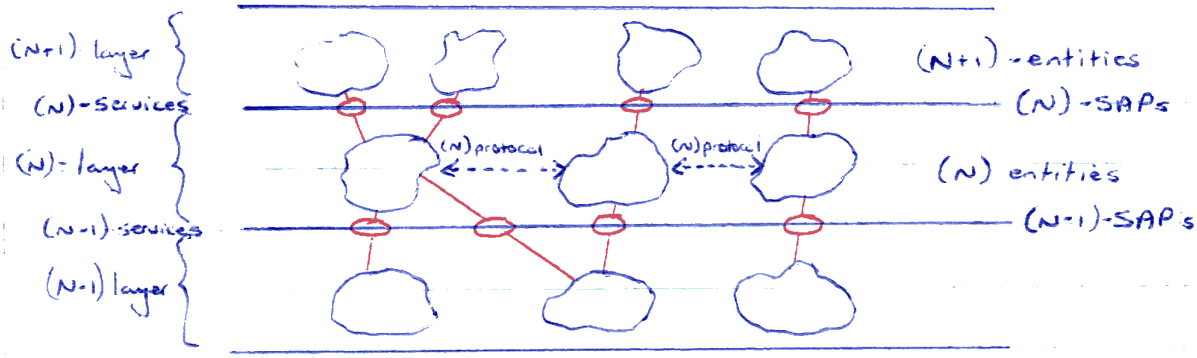
succession of subsystems, each corresponding to a layer. A layer, therefore, comprises many entities distributed among interconnected open systems. Entities in the same layer are called peer entities.
The basic idea of layering is that each layer adds value to services provided by the set of lower layers in such a way that the highest layer is offered the full set of services needed to run distributed applications. Layering also ensures independence of each layer by defining services provided by a layer provided to the next higher layer, independent of how those services are performed.

Services and Service Access Points.

A service is a capability of the (N)-layer which is provided to the (N+1)-entities. (NB only those capabilities that can be seen from the layer above are services).
The (N)-services are offered to the (N+1)-entities at the (N)-service access points ((N)-SAP's) which represent the logical interfaces between the (N)-entities and the (N+1)-entities.
An (N)-SAP can be served by only one (N)-entity and used by only one (N+1)-entity, but one (N)-entity can serve several (N)-SAP's and one (N+1)-entity can use several (N)-SAP's. An (N)-SAP is located by its (N)-address.

Functions and Protocols

An (N)-function is part of the activity of an (N)-entity. Cooperation among (N)-entities is governed by one or more (N)-protocols (sets of rules and formats governing the communication between (N)-entities).



ENTITIES, SAP's AND PROTOCOLS

NAMING

The OSI archi. defines identifiers for entities, SAP's, and connections, as well as relations between these.
Each (N)-entity is identified with a global title which is unique and which identifies the same (N)-entity anywhere in the network of open systems. Within more limited domains, an (N)-entity can be identified with a local title which uniquely identifies the (N)-entity only in that domain.
Each (N)-SAP is identified by an (N)-address (unique)
Bindings between (N)-entities and the (N-1) SAP's they use are defined in an (N)-dictionary which indicates correspondence between global titles of (N)-entities and (N)-addresses through which they can be reached.

CONNECTIONS

A common service offered by all layers consists of providing connections between peer SAP's which can be used to transmit data. Most common is point-to-point, but multi-endpoint also allowed. The end of an (N)-connection at an (N)-SAP is an (N)-connection-endpoint, or (N)-CEP. Each (N)-CEP is uniquely identified within its (N)-SAP by an identifier used on both sides of the (N)-SAP to identify the (N)-connection. The Reference Model currently restricts communications between entities to "connection mode"

11 LOCAL NETWORKS

11.1 LOCAL NETWORK TECHNOLOGY

11.2 THE BUS/TREE TOPOLOGY

11.3 THE RING TOPOLOGY

11.4 MEDIUM ACCESS CONTROL PROTOCOLS

11.5 LAN PROTOCOL PERFORMANCE

12 PROTOCOLS AND ARCHITECTURES

12.1 PROTOCOLS

12.2 THE LAYERED APPROACH : OSI

12.3 THE HIERARCHICAL APPROACH: DOD

12.4 EXAMPE ARCHITECTURES: SNA, DNA

13 NETWORK ACCESS PROTOCOLS

13.1 THE NETWORK INTERFACE

13.2 CIRCUIT-SWITCHED NETWORK ACCESS

13.3 PACKET-SWITCHED NETWORK ACCESS

13.4 BROADCAST NETWORK ACCESS

14 INTERNETWORKING

14.1 PRINCIPLES

14.2 THE BRIDGE

14.3 X75

14.4 INTERNET PROTOCOL (IP)

14.5 PROTOCOL TRANSLATION

ie the (N-1) service requires that an (N-1) connection between (N-1) SAPs before any communication between (N)-entities can take place. Conversely, when the (N)-entities need no longer communicate, the (N-1)-connection may be released.

1) ESTABLISHMENT & RELEASE OF CONNECTIONS

When an (N+1) entity requests the establishment of an (N)-connection from one of the (N)-SAPs it uses to another (N)-SAP, it must provide at the local (N)-SAP the (N)-address of the distant (N)-SAP. When the (N)-connection is established, both the (N+1)-entity and the (N)-entity will use the (N)-CEP identifier to designate the (N)-connection. (N)-connections must be established and released dynamically on top of (N-1) connections, and so on downwards.

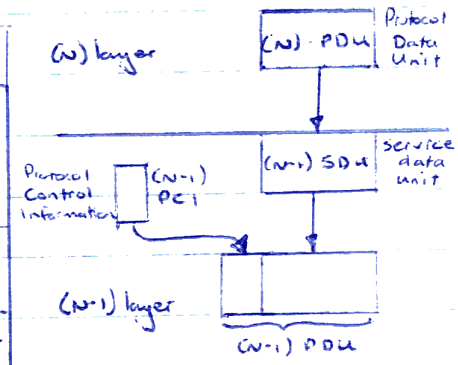
2) DATA TRANSFER ON A CONNECTION

Information is transfered in various types of data units between peer entities and between entities attached to a specific SAP.

INTERRELATIONSHIP BETWEEN DATA UNITS

	Control	Data	Combined
(N)-(N) Peer entities	(N)-protocol-control-information	(N)-user-data	(N)-Protocol-data-units
(N)-(N-1) Adjacent layers	(N-1)-interface control-information	(N-1)-interface data	(N-1) Interface-data-unit

LOGICAL RELATIONSHIP BETWEEN DATA UNITS IN ADJACENT LAYERS



(N)-PCI is information exchanged between two (N)-entities, using an (N-1) connection, to coordinate their joint operation

(N)-user-data are the data transfered between two (N)-entities on

behalf of the (N+1) entities for whom the (N)-entities are providing services.

An (N)-PDU is a DU which contains (N)-PCI and possibly (N)-user-data

(N)-interface-control information is information exchanged between an (N+1) entity and an (N)-entity to coordinate their joint operation.

(N)-interface-data-unit is the amount of (N-1) interface data whose identity is preserved from one end of an (N-1) connection to the other. Data may be held within a connection until a complete SDU is put into the connection.

3) ELEMENTS OF LAYER OPERATION.

Functions that are part of layer operation include such things as multiplexing, flow control, and error control.

Three particular types of constructions of (N)-connections on top of (N-1) connections are distinguished:

- one-to-one (one (N)-connect to one (N-1)-connect)
- multiplexing (several (N)-connects to one (N-1) connect)
- splitting (one (N)-connect to several (N-1) connects)

Two forms of flow control are recognised: a peer flow control which regulates the flow of (N)-PDU's between peer entities, and interface flow control which regulates the flow of (N)-interface-data through an (N)-SAP.

A variety of error functions are recognised by the model including acknowledgement, error detection, and error notification mechanisms, as well as a reset function to allow recovery from a loss of synchronisation between communicating (N)-entities.

1.2 The Seven Layer Model

Application Layer

The primary concern of the application layer is with the semantics of the application - all application processes reside in this layer. However, only part of this layer is in the real OSI system - those aspects of the application process which are concerned with interprocess communication (called the application entity)

Presentation Layer

Primary purpose is to provide independence to application processes from differences in data representation. The PL protocol allows the user to select a "Presentation Context", which may be specific to an application or hardware.

Session Layer

Primary purpose is to provide the mechanisms for organising and structuring the interactions between application processes. The mechanisms provided in the session layer allow for two-way simultaneous and two-way alternate operation, the establishment of major and minor synchronisation points, and the definition of special tokens for structuring exchanges.

Transport Layer

Purpose is to provide transparent transfer of data between end systems, thus relieving the upper layers from any such concern.

NETWORK LAYER

The NL masks from the Transport layer all the peculiarities of the actual transfer medium. The NL also handles relaying and routing data through as many concatenated networks as necessary while maintaining the quality of service parameters requested by the transport layer. The NL functions can be categorised into three sublayer groupings:

3C - the Concatenation and Routing Functions (called Internetting)
3B - the Subnetwork Convergence Functions
3A - the Subnetwork Access Functions

DATA LINK LAYER

Purpose is to provide the functional and procedural means to transfer data between network entities and to detect and possibly correct errors which may occur in the physical layer. The protocols used here are very dependent on the transfer technology.

PHYSICAL LAYER

This provides the mechanical, electrical, functional, and procedural standards to access the physical medium.

2 Elementary Queuing Theory

2.1 Some Important Random Processes.

2.1.1 Notation & Structure for Basic Queuing Systems.

We first consider a very general queuing system G/G/m, whose interarrival time distribution $A(t)$ is completely arbitrary, and whose service time distribution $B(x)$ is also completely arbitrary and independent of $A(t)$. The system has m servers and service order is also quite arbitrary. Let

C_n denote the n^{th} customer to enter the system

$N(t) \triangleq$ number of customers in the system at time t.

$U(t) \triangleq$ unfinished work in the system at time t.

$\triangleq$ the remaining time required to empty the system of all customers present at time t.

If $U(t) > 0$ the system is busy; otherwise idle.

$\tau_n \triangleq$ arrival time for C_n

$t_n \triangleq$ interarrival time between C_{n-1} and $C_n = \tau_n - \tau_{n-1}$

Since we have assumed that all interarrival times are drawn from the distribution $A(t)$, we have that:

$$P[t_n \leq t] = A(t) \quad \text{(independent of } n\text{)}$$

$x_n \triangleq$ service time for C_n

and then $P[x_n \leq x] = B(x)$.

$W_n \triangleq$ waiting time in Q of C_n.

$S_n \triangleq$ system time for $C_n = W_n + x_n$

2. Delay Analysis

Two major constraints of design are delay and reliability. When network traffic is light, packet delay is primarily due to the time each node needs to store and forward the packet, plus possibly a propogation delay. As traffic increases, the principle delay becomes the queuing delay within each node.

2.1 Introduction to Queuing Theory.

Queuing systems can be characterised by five components:

- interarrival-time probability density function
- service-time probability density function
- number of servers
- queuing discipline
- amount of buffer space in the queues.

We are considering only systems with an infinite number of customers (ie long queues do not so deplete the population that the input rate is reduced - this is in contrast with eg time-sharing models.)

We will concentrate on infinite-buffer, single-server systems using first-come, first-served. The notation A/B/m is widely used in the queuing literature for these systems. where A is the interarrival-time prob. dens., B the service-time prob. dens., and m the number of servers.

A and B may be:

M - exponential probability density (Markov)
D - all customers have same value (Deterministic)
G - arbitrary probability density (General)

For a system with a large number of independent customers, we may assume an exponential interarrival probability, ie, the probability of exactly n customers arriving during an interval of length t is given by the Poisson law

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

where λ is the mean arrival rate.

Now we will show that Poisson arrivals generate an exponential interarrival probability density. The probability $a(t)\,\Delta t$, that an interarrival interval is between t and $t + \Delta t$ is the probability of no arrivals for time t times the probability of one arrival in the interval Δt:

$$a(t)\,\Delta t = P_0(t)\,P_1(\Delta t)$$

Now, $P_0(t)$ is $e^{-\lambda t}$, and $P_1(\Delta t)$ is $\lambda \Delta t\, e^{-\lambda \Delta t}$. In the limit $\Delta t \to 0$, the exponential factor in P_1 approaches unity, so:

$$a(t)\,dt = \lambda e^{-\lambda t}\,dt$$

Note that the integral from 0 to ∞ of this equation is 1, as it should be.

Although the assumption of an exponential interarrival probability density is usually reasonable, the assumption of exponential service times is harder to defend on general grounds. Nevertheless, for situations in which increasingly long service times are increasingly less likely, M/M/1 may be an adequate approximation.

2.2 The M/M/1 Queue in Equilibrium

The state of an M/M/1 queue is completely described by the number of customers currently in the system. - the exponential density function has no memory; the probability of the remaining service time requiring t seconds is independent of

how much service the customer has already received.

Let p_k be the equilibrium probability that there are exactly k customers in the system (queue + server). Even when the system is in equilibrium, transitions between states take place. If the system is in state 4 (ie 4 customers) and a new customer arrives, the system moves into state 5, etc. Queueing systems in which the only transitions are to adjacent states are known as birth-death systems.

STATE DIAGRAM FOR A SINGLE-SERVER QUEUING SYSTEM.

If the mean arrival rate is λ customers/sec, the transition rate from state k to state $k+1$ is λp_k. Similarly, if the server is capable of processing μ customers/sec, the transition rate from state $k+1$ to state k is μp_{k+1}

In equilibrium we must have $\lambda p_k = \mu p_{k+1}$*. We can solve this to get $p_k = \rho^k p_0$, where $\rho = \lambda/\mu$ is the traffic intensity. The traffic intensity must be less than 1 for the queue to be stable.

To eliminate p_0 from our equation, we use the fact that the probabilities must sum to 1, ie $\sum_{k=0}^{\infty} \rho^k p_0 = 1 \Leftrightarrow \sum_{k=0}^{\infty} \rho^k = \frac{1}{p_0}$

Using the sum of a geometric series formula $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$ ⊕, we get $p_0 = 1 - \rho$, and finally: $p_k = (1-\rho)\rho^k$.

Note that $\rho = 1 - p_0$ is the probability that the system is not empty.

We can now find N, the mean number of customers in the system.

$$N = \sum_{k=0}^{\infty} k p_k = (1-\rho) \sum_{k=0}^{\infty} k \rho^k$$

The value of the summation can be found by differentiating both sides of ⊕ wrt x and multiplying through by x.

* called detailed balancing

This gives us $N = \frac{\rho}{1-\rho}$

Each customer is in the system for an average time T (the mean time for all customers). Thus the mean number of new arrivals subsequent to the arrival of a specific customer and just prior to his/her departure is λT. At the instant of the customer's departure, all λT of these customers, and no others, are in the system. This gives the result $N = \lambda T$, known as Little's law.

Using the two equations above, we have:

$$T = N/\lambda = \frac{\rho/\lambda}{1-\rho} = \frac{1/\mu}{1-\rho} = \frac{1}{\mu - \lambda}$$

For the sake of generality, we will state the formula for the mean number of customers in an M/G/1 system:

POLLACZEK-KHINCHINE EQUATION $N = \rho + \rho^2 \frac{1 + C_b^2}{2(1-\rho)}$ where $C_b = \frac{\text{std. dev (PF)}}{\text{mean (PF)}}$ $\left(= 1 \text{ for Poisson}\right)$

PF = service time prob. dens. fn.

2.3 NETWORKS OF M/M/1 QUEUES.

Let the probability density function for packet size in bits be $\mu e^{-\mu x}$ with a mean of $1/\mu$ bits per packet. Let the capacity of communication channel i be C_i bits/sec. The product μC_i is then the service rate in packets/sec. The arrival rate for channel i is λ_i packets/sec. Our equation for T can be rewritten for channel i as

$$T_i = \frac{1}{\mu C_i - \lambda_i}$$

where T_i includes both queuing and transmission time as

can be seen by taking the limit $\lambda_i \to 0$.

It has been shown that if the outputs of several M/M/1 servers feed into the input queue of another server, the resulting input process is also a Poisson process, with mean equal to the sum of the means of the feeding processes. Furthermore, an open network of M/M/1 queues can be analysed as though each one was isolated from the others - all that needs to be known is the mean input rate.

We have to make one artificial, but nevertheless necessary and reasonable, assumption before we can continue. When a packet moves round a network, it maintains its size, introducing some non-random correlations into the system. We assume that every time a packet arrives at a node it loses its identity and a new length is chosen for it at random. - the Independence Assumption.

To calculate the mean packet delay for a network with n nodes and m channels, it is convenient to define:

$$\gamma = \sum_{i=1}^{n} \sum_{j=1}^{n} \gamma_{ij} \quad \text{and} \quad \lambda = \sum_{i=1}^{m} \lambda_i$$

where γ_{ij} is the number of packets to be sent from node i to node j, and λ_i is the total traffic in line i. The ratio $\bar{n} = \lambda/\gamma$ is the mean number of hops per packet.

As an aside, it is interesting to note that λ can be related to the traffic matrix γ_{ij} by: $\lambda = \sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij} \gamma_{ij}$ where h_{ij} is the number of hops in the route from node i to node j. If shortest path routing is used, the λ so obtained is the theoretical minimum achievable for the given topology and traffic matrix.

The mean delay per hop is simply the sum of the individual line delays, weighted by the amount of traffic in the line, λ_i. To normalise the result, we divide by the total traffic λ, yielding mean delay per line $= \sum_{i=1}^{m} \frac{\lambda_i T_i}{\lambda}$

However, the mean packet delay T is longer, as many packets must make several hops, i.e.

$$T = \bar{n} \sum_{i=1}^{m} \frac{\lambda_i T_i}{\lambda} = \bar{n} \sum_{i=1}^{m} \frac{\lambda_i / \lambda}{\mu C_i - \lambda_i}$$

Note that this is only an approximation to reality, eg we have neglected processing time and propogation delay.

3 Formal Models of Communication Protocols.

We use formal models so that we can prove network properties, such as liveness, freedom from deadlock, etc. We consider three types here - extended finite-state machines (EFSM's), Petri-nets and variations and temporal logic methods

3.1 Extended Finite State Machines

An EFSM is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

where: Q is a set of possible states
Σ is a finite input alphabet
δ is a state transition function
q_0 is an initial state
F is a set of final states.

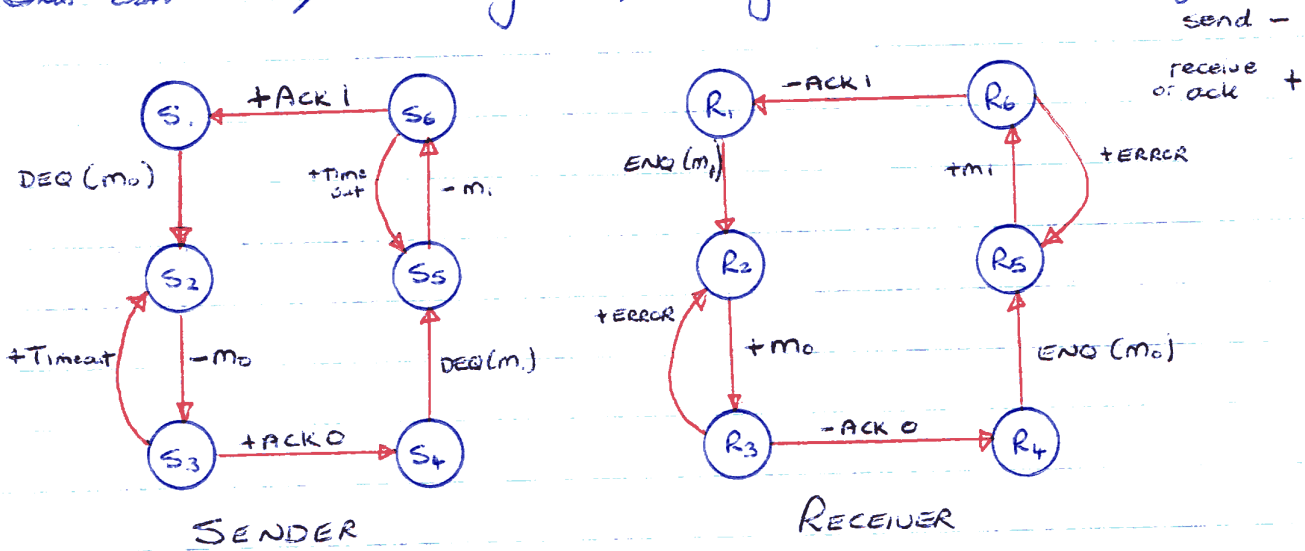
An EFSM is extended, as we allows predicates on inputs (eg IC1 (MSG.SEQ-NO = 1 .AND. NO-ERROR)), and output actions on state transitions (eg MSG.SEQ-NO = MSG.SEQ-NO MOD 2)

As an example, we consider the alternating bit protocol, a simple data link protocol with a window size of 2 and positive acknowledgement. No error checking is involved but the protocol can handle lost frames by timing out. Data frames are transmitted in one direction only.
Each protocol machine (ie sender or receiver) is always in a specific state at every instant in time, the state being the value of all its variables. In most cases, a large number of states can be grouped together (partitioned) to simplify analysis.

We first consider the sender and receiver separately. A sequence of possible events using our protocol is:

- de-queue message on sender side
- give it a sequence number 0 or 1 (alternating bit)
- transmit message SEQ-NO = 0 to receiver
- message is lost / corrupted
- sender times out and retransmits if no acknowledgement received after T_0 seconds.
- receiver accepts and acknowledges message
- receiver enqueues message on out queue
- sender receives acknowledgement
- sender de-queues next message

This can be represented by the following state-transition diagrams



Note that the state of the channel has been ignored. To include this, we define states (XYZ), where:

X ≙ 0 or 1 depending on what sender is transmitting

Y ≙ 0 or 1 depending on what receiver expects

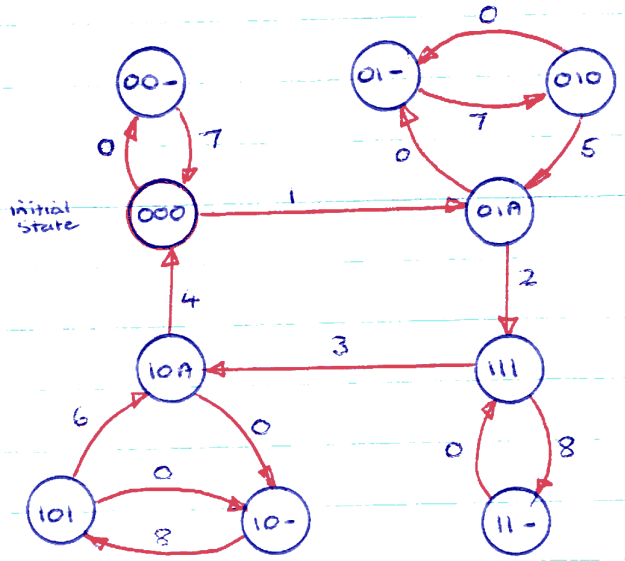
Z ≙ 0 or 1 or A (ack) or − (empty) depending on channel.

These states represent the state of the entire system. From each state, there are zero or more possible transitions to other states; these transitions occur when an event happens.

Given the initial state and a directed graph representing the possible states / transitions, we can use graph theory methods

(eg computing transitive closures) to determine which states are reachable and which are not.

Our model of the system now becomes



STATE GRAPH FOR THE ALTERNATING BIT PROTOCOL

The following transitions are possible:

Active	Tranition	Recvd Frame	Recvd Tid	Start State	End State	Explanation
-	0		lost	(ABC)	(AB-)	Channel loses frame C
R	1	0	A	(000)	(01A)	Frame 0 delivered & acked
S	2	A	1	(01A)	(111)	Frame 1 sent
R	3	1	A	(111)	(10A)	Frame 1 delivered & acked
S	4	A	0	(10A)	(000)	Frame 0 sent
R	5	0	A	(010)	(01A)	Recovery from lost frame 0
R	6	1	A	(101)	(10A)	Recovery from lost frame 1
S	7	timeo.	0	(01-)	(010)	Time out & retransmit frame 0
S	8	timeo.	1	(10-)	(101)	Time out & retransmit frame 1

△ channel

Note our protocol does not cater for checksum errors. During normal operation, transitions 1 thru 4 are effected repeatedly, each cycle delivering two frames.

Once we have our graph, we can use it to demonstrate properties of the protocol. We can also program the FSM as an actual protocol implementation (see over)

```
/*
-------------------------------------------------------------------------
|              FINITE STATE MACHINE USED BY THE SENDER                  |
|    in the version of the Alternating Bit protocol specified here      |
-------------------------------------------------------------------------*/

:fsm name=SENDER_FSM
   states='ACK_1_RCVD,MSG_0_FTCD,MSG_0_SENT,ACK_0_RCVD,MSG_1_FTCD,MSG_1_SENT'
     pass='MSG_NR,ACK_NR'
    passt='char(*) varying,char(*) varying'
  process=SENDER
  signals='FETCH,DISPATCH'. for enqueuing & dequeuing.
:cbox. comment box
This SENDER Finite State Machine models the behaviour of the sender
in a version of the Alternating Bit protocol
:ecbox.
:function.Sender Finite State Machine

:fsminput name=ABPSMS_INPUT  prolog=no.
 ACK_0    ACK_NR=ZERO & MSG_NR=RESET;
 ACK_1    ACK_NR=ONE & MSG_NR=RESET;
:efsminput.

:start.
 dcl SENDER process;
 dcl #trace procedure(integer,char(*),integer);
 dcl MSG_NR char(*) varying;
 dcl ACK_NR char(*) varying;
 dcl ONE char(4) constant('1111');
 dcl ZERO char(4) constant('0000');
 dcl RESET char(4) constant('bbbb');  blanks?

:matrix.                    /* definition of the state transition matrix */
:s id=1 name='ACK 1 RCVD'
:s id=2 name='MSG 0 FTCD'
:s id=3 name='MSG 0 SENT'
:s id=4 name='ACK 0 RCVD'
:s id=5 name='MSG 1 FTCD'
:s id=6 name='MSG 1 SENT'
:fsmseg.
:i ic=ACK_0                  a=/  a=/  a=4C a=/  a=/  a=/.
:i ic=ACK_1                  a=/  a=/  a=/  a=/  a=/  a=1C.
:i ic='signal(FETCH)'        a=2A a=/  a=/  a=5B a=/  a=/.
:i ic='signal(DISPATCH)'     a=/  a=3D a=/  a=/  a=6D a=/.
:ematrix.

:outcode.                                   /* define the output code */
:o name=A.
   call #trace(75,'State(0001->0002)  Time(50)',27);
   MSG_NR=ZERO;                                      /* message 0 fetched */
:o name=B.
   MSG_NR=ONE;                                       /* message 1 fetched */
   call #trace(75,'State(0004->0005)  Time(50)',27);
:o name=C.
   select anyorder;  /* acknowledgement received*/
      when (ACK_NR=ZERO)
           do;
           ACK_NR=RESET;
           call #trace(75,'State(0003->0004)  Time(10)',27);
           end;
      when (ACK_NR=ONE)
           do;
           ACK_NR=RESET;
           call #trace(75,'State(0006->0001)  Time(10)',27);
           end;
     otherwise call #trace(75,'SENDER01: Now how did we get here?',34);
   end; /*select*/
:o name=D.                                           /* message dispatched */
 select anyorder;
  when (MSG_NR=ZERO) call #trace(75,'State(0002->0003)  Time(30)',27);
  when (MSG_NR=ONE) call #trace(75,'State(0005->0006)  Time(30)',27);
  otherwise call #trace(75,'SENDER02: Now how did we get here?',34);
 end; /*select*/
:eoutcode.
:efsm.
```

3.2 PETRI NET MODELS.

Petri nets have been developed from the early work of Carl Petri (1962), who used them to model the class of discrete-event systems with concurrent parallel events. With the advent of parallel computing and concurrent processing, PN's are becoming increasingly popular.

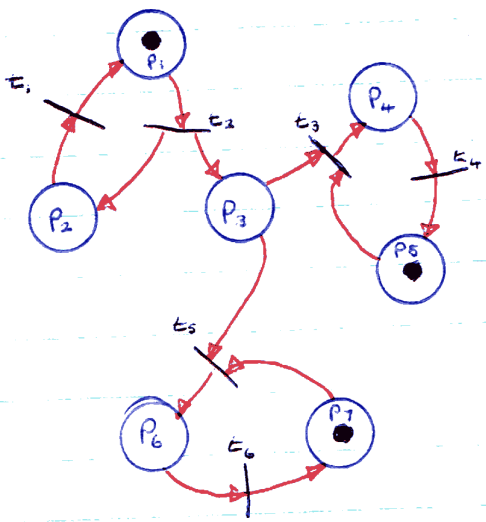
A PN has four fundamental components:

- places (representing states)
- transitions (representing events)
- arcs
- tokens (a PN with tokens is called marked)

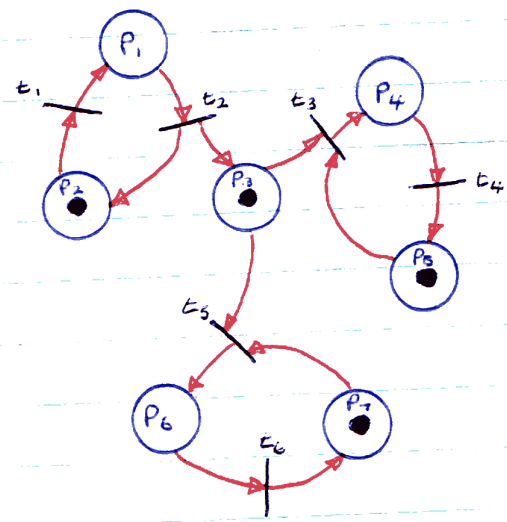
A PN with two places A, B and two transitions 1, 2. A has a token.

A transition is enabled if there is at least one input token in each of its input places. Any enabled transition may fire at will, removing one token from each input place, and depositing a token in each output place. If two or more transitions are enabled, any one of them may fire, the choice being indeterminate.

Eg a producer-consumer problem, with one producer (P_1 & P_2) and two consumers (P_4, P_5) & (P_6, P_7), modelled by the tokens "produced" by transition t_2 and "consumed" by transitions t_3 and t_5:



BEFORE FIRING

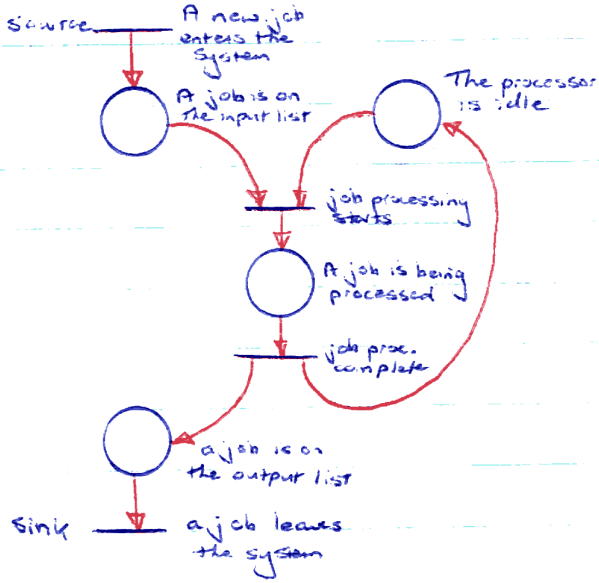


AFTER FIRING

As another example, consider the following computer system

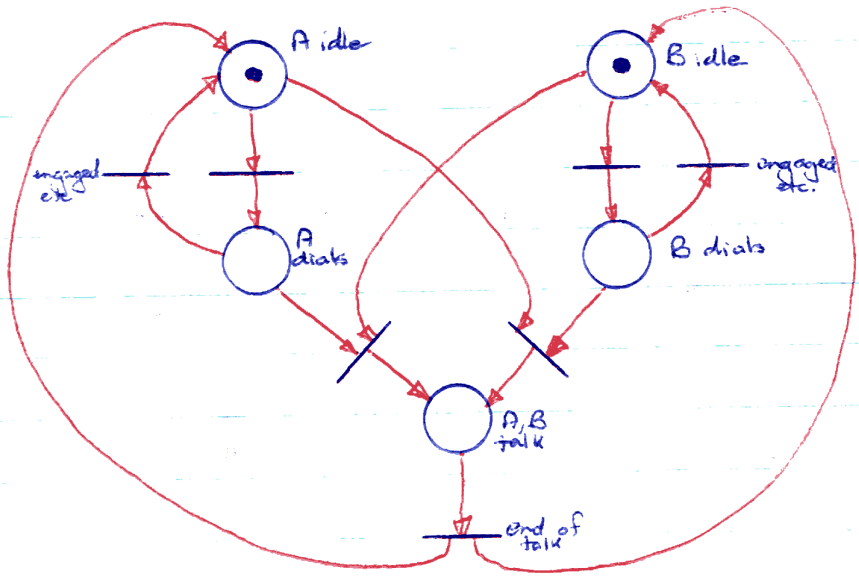
- jobs appear and are put on an input list. When the processor is free, and there is a job on the input list, the processor starts to process the job.
- When the job is complete, it is placed on an output list, and if there are more jobs on the input list, the processor continues with another job, otherwise it waits for another job.

This can be modelled as follows:

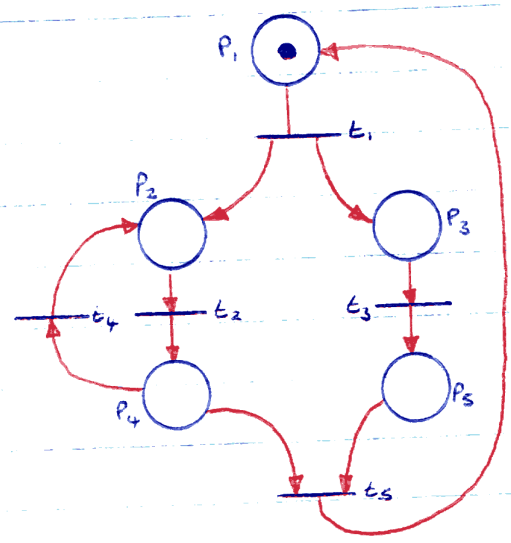


MODELLING OF A SIMPLE COMPUTER SYSTEM WITH PETRI NETS.

As a final example, consider a protocol for a field telephone



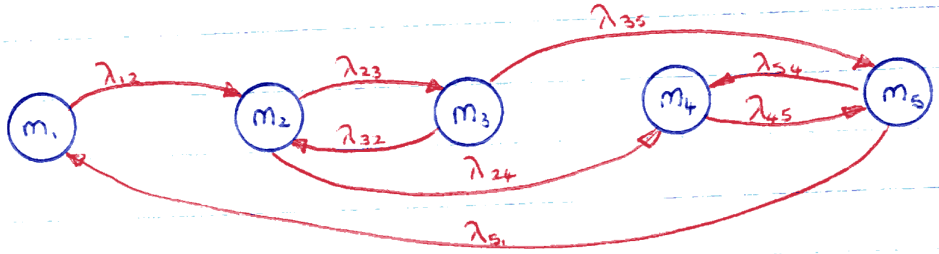
3.2.1 TIME IN PETRINETS.



MARKINGS: (note bit vector representation)

	P_1	P_2	P_3	P_4	P_5
m_1	1	0	0	0	0
m_2	0	1	1	0	0
m_3	0	0	1	1	0
m_4	0	1	0	0	1
m_5	0	0	0	1	1

We assume by convention that the time spent in a marking is the time spent between its boundary transitions. In our example above, we can draw a marking transition graph:



Here, λ_{12} is defined as the firing rate of transition t_1, etc. We can solve this ergodic markov MTG to obtain (for eg.)

$P[m_1]$ (probability of system being in marking m_1) etc.

3.2.2 PROPERTIES OF PETRI NETS.

Defn: A PN is <u>elementary loop free</u> iff for all transitions, the intersection of the set of input places and the set of output places is empty.

Defn: A PN is <u>bounded by n</u> for an initial marking m_0 iff, whatever the current marking reachable from m_0, each place has at most n tokens.

An unbounded PN implies that the corresponding system may have an infinite number of markings (ie states), whence its implementation is not possible.

Def: A PN is <u>live</u> for an initial marking M_0 iff, whatever the current marking, for any transition t there is a legal firing sequence which will enable t.

Note that when a PN is live, the associated system is deadlock-free.

Def: A 1-bounded PN is called <u>safe</u>.

3.2.3 VARIATIONS OF PETRI NETS.

"Ordinary" petri nets are also called place-transition nets. Other types of petri-nets are also used.

PLACE-COLOURED NETS.

The tokens are split into classes, each class having a different colour. Formally, a place coloured net consists of:

- a bipartite graph whose vertices are places and transitions
- the arcs are labelled with expressions denoting functions on bags (sets with repetitions) of coloured tokens
- an initial marking M_0, which associates with every place P a bag of given values of the initial colours

PREDICATE TRANSITION NETS.

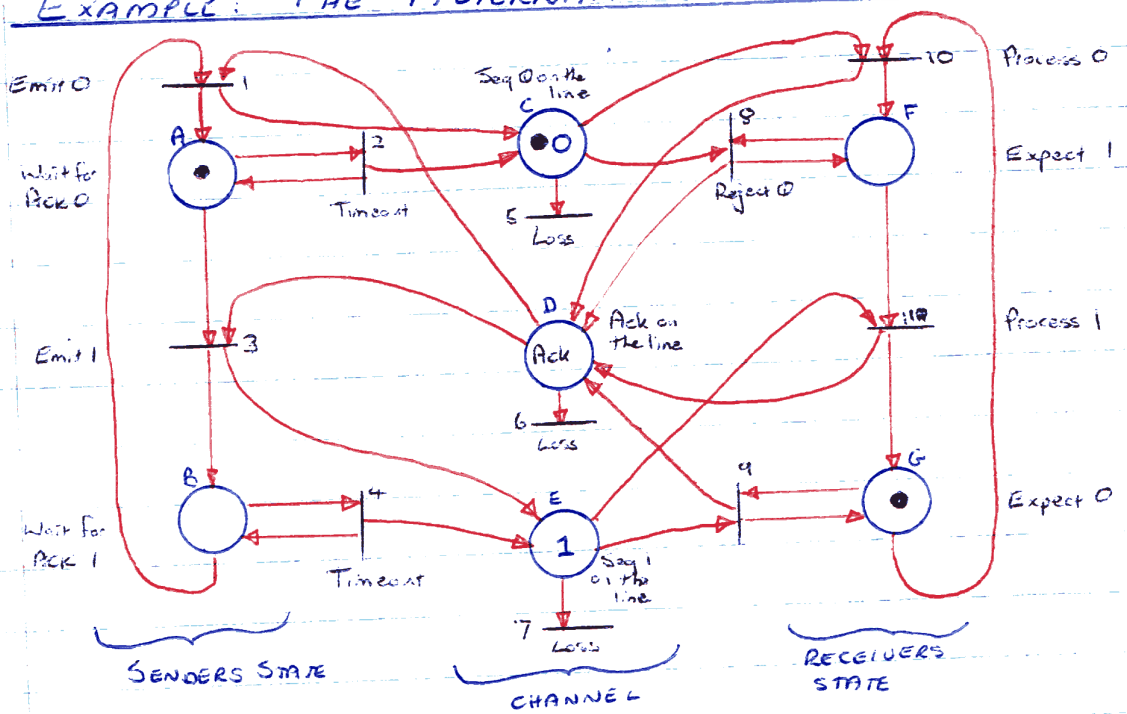
A predicate transition net is a place-coloured net with a set of predicates associated with every transition which describes the relationship(s) which must hold between input tokens

for the transitions to fire.

PREDICATE ACTION NETS.

A predicate action net is a safe PN with a set of labels attached to the transitions; each transition t has an expression of the form WHEN $P_t(x)$ DO $x' \leftarrow F_t(x)$ for predicate $P_t(x)$ and output action $F_t(x)$, both defined on a vector of program / system variables x. Note the similarity with extended FSM's.

3.2.4 EXAMPLE: THE ALTERNATING BIT PROTOCOL.



Unlike the FSM, there are no composite states. The transitions are:

1 - Frame 0 transmit
2 - Frame 0 retransmit after timeout.
3 - " 1 "
4 " 1 " " "
5 - Loss of frame 0
6 Loss of ack
7 - Loss of frame 1
8, 9 Wrong seq. number received
10, 11 arrival of next frame in sequence.

Obviously a real, sophisticated protocol would be far more complex.

3.3 Temporal Logic

We use temporal logic in the verification of protocols by proving:

- safety properties (invariants - bad things will not happen)
- liveness properties (commitments - good things will happen).

which have been described using temporal logic formalisms.
We draw on propositional and predicate calculus (variables, operators (and proof methods to an extent), and modal logic.
Modal logic involves propositions which happen to be true, ones which are necessarily true, and ones which are possibly true. Necessity and possibility are regarded as duals $\Box p \equiv \sim \Diamond \sim p$. The modal universe consists of many states and an accessibility relation between them.
The semantics of $\Box$ and $\Diamond$ are: ($\Diamond p \equiv \sim \Box \sim p$)

$$|\Box \omega|_s = \forall s' \, [R(s,s') \supset |\omega|_{s'}]$$

$$|\Diamond \omega|_s = \exists s' \, [R(s,s') \wedge |\omega|_{s'}]$$

To be useful, our formal system needs axioms, inference rules and theorems.

Eg:

A1	$(p \vee p) \supset p$
A2	$q \supset (p \vee q)$
A3	$(p \vee q) \supset (q \vee p)$
A4	$(q \supset r) \supset ((p \vee q) \supset (p \vee r))$
A5	$\Box p \supset p$
A6	$\Box (p \supset q) \supset (\Box p \supset \Box q)$
I1	Uniform substitution
I2	Modus Ponens $[\alpha \wedge (\alpha \supset \beta)] \supset \beta$
I3	$\alpha \supset \Box \alpha$

T1. $p \equiv \sim\sim p$ (double negation)

$\Box p \equiv \sim\sim \Box p$ ($\Box p / p$)

$\Box p \equiv \sim\sim \Box \sim\sim p$ ($\sim\sim p / p^{(one)}$)

$\boxed{\Box p \equiv \sim \Diamond \sim p}$ (from def $\Diamond$)

On temporal logic, we include the interpretation of passing time, make the accessability relation reflexive and transative, and consider execution sequences (of states). We also allow axioms defining the semantics of programming language constructs (cf Hoare). Instead of a formal system, we use a verification system of axioms, inference rules, theorems and verification conditions (defining properties of a specific program). The assertions (VC's) should be proven as theorems.

Problems with this are that proofs are long and complex and may contain errors, and that the protocol must be written before the prog, ie it is no help in the design of programs.

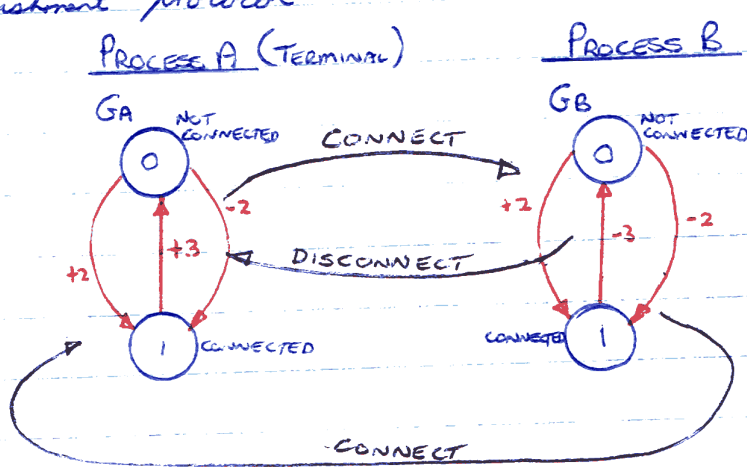
4. Office Automation

5 PROTOCOL VALIDATION identifying & localising logical design errors in protocols

5.1 PROTOCOL VALIDATION BY DUOLOGUE-MATRIX ANALYSIS

5.1.1 THE PROTOCOL MODEL.

We model a protocol as a pair of interacting processes, each represented by a directed graph. Eg a simplified connection-establishment protocol:



Events are the basic unit of communication between the processes, each one represented by an integer (-ve ⇒ transmission; +ve ⇒ reception). Events cause state transitions (eg DISCONNECT causes the 1→0 transition). State transitions that are not related to event reception / transmission are "nonevents" and are represented by 0 (zero). Protocols require external control (eg CONNECT must be initiated by some external control). Event transmissions that are not initiated by external actions are modelled by nonevents.

5.1.2 THE VALIDATION PRINCIPLE

We assume the processes begin and end in state 0. Furthermore, we only consider interactions which do not cause state 0 traversals. (eg Process A: -2 +3 Process B: +2 -3)

A uniloque is a path beginning and ending in state 0 but

not crossing state 0 eg $A_1 = \{-2, +3\}$, $B_1 = \{+2, -3\}$
A tuple of unilogues is a dualogue eg $[A_1, B_1]$.
A dualogue is an interaction sequence which implies the traversal of one unilogue in each graph. The dualogue matrix $[D]$ is the set of all conceivable dualogues for a given protocol. $[D] = S_A \times S_B$, where S_x is the set of all unilogues in the graph g_x.
We represent this as:

$$D = \begin{bmatrix} [A_1, B_1] & [A_1, B_2] \\ [A_2, B_1] & [A_2, B_2] \end{bmatrix}$$

in our example, where: $A_1 = \{-2, +3\}$ $B_1 = \{+2, -3\}$
$A_2 = \{+2, +3\}$ $B_2 = \{-2, -3\}$.

A dualogue is "well-behaved" if events transmitted by one unilogue are received by the other, and the dualogue always terminates. A dualogue is "non-occurable" if it can either never begin or if any attempt to execute it invariably results in the execution of another dualogue. If a dualogue is neither well-behaved nor non-occurable, it is "erroneous".

We introduce a validation function val, where

$$val([A_i, B_k]) = \begin{cases} +1 & \text{if } [A_i, B_k] \text{ is well-behaved} \\ 0 & \text{if } [A_i, B_k] \text{ is non-occurable} \\ -1 & \text{if } [A_i, B_k] \text{ is erroneous} \end{cases}$$

We apply val to our matrix

$$val(D) = \begin{bmatrix} val\,[A_1, B_1] & val\,[A_1, B_2] \\ val\,[A_2, B_2] & val\,[A_2, B_2] \end{bmatrix}$$

It can easily be seen that $[A_1, B_1]$ and $[A_2, B_2]$ are well-behaved. Consider $[A_2, B_1] = [\{+2, +3\}, \{+2, -3\}]$. Both processes wait indefinitely to receive events that were never generated; this dualogue is thus non-occurable.
Consider $[A_1, B_2] = [\{-2, +3\}, \{-2, -3\}]$. Both processes transmit 2 simultaneously (collision) but there is no provision for

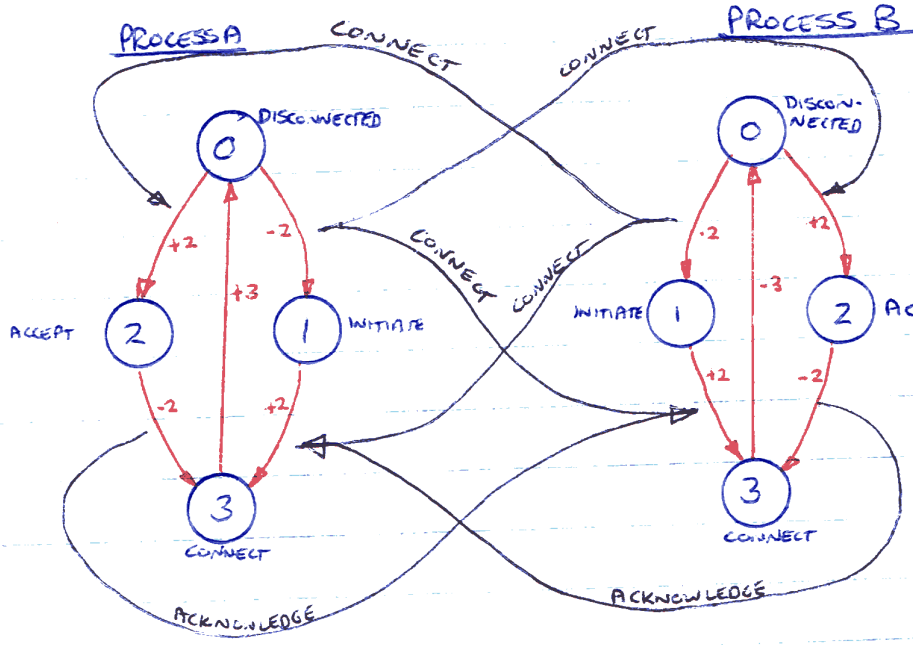
simultaneous reception of event 2. This contradicts both the well-behaved and non-accusable criteria. We thus have:

$$Val[D] = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$, the validation matrix

This shows one error, $[A_1, B_2]$. Typically these errors are events the designer believes are non-occurable.

5.1.3 APPLYING THE VALIDATION PRINCIPLE

Rule 1: a protocol contains design errors if Val [D] contains any -1 elts.
Rule 2: the position of the -1 elts identify the protocols erroneous dialogues.
Below is a first correction of our protocol. We introduce an acknowledge event. Now $A_1 = \{-2, +2, +3\}$ $A_2 = \{+2, -2, +3\}$, and $Val[D] = \begin{bmatrix} +1 & +1 \\ 0 & +1 \end{bmatrix}$

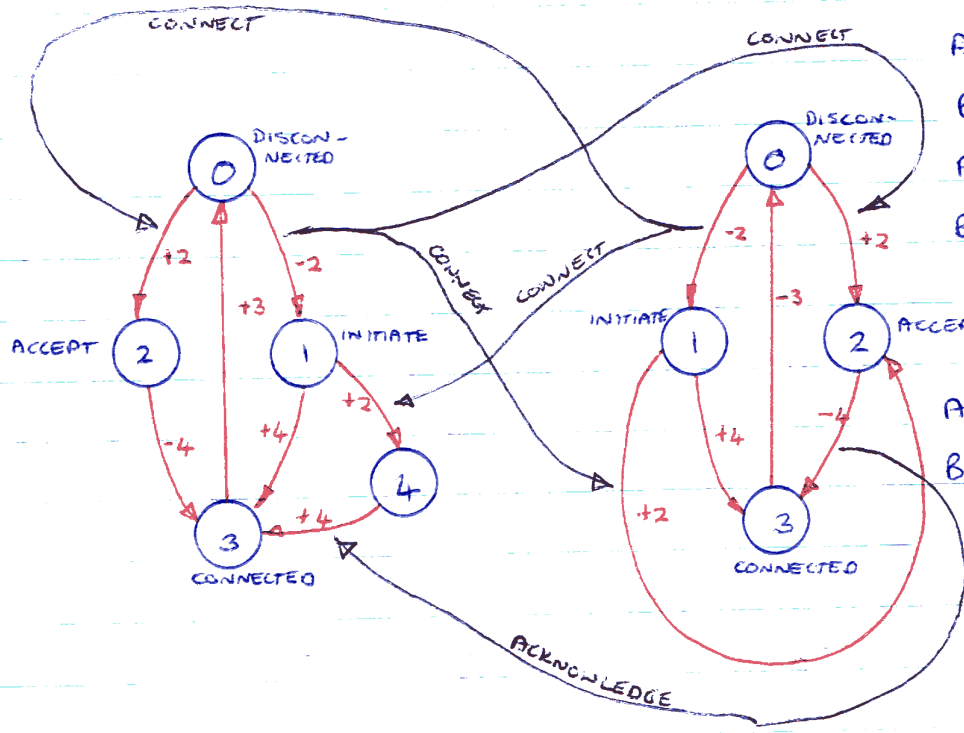


ambiguous
A_1 : -2 +2 +3
B_1 : +2 -2 -3
A_2 : +2 -2 +3
B_2 : -2 +2 -3

Rule 3: If the i^{th} row (k^{th} column) of Val [D] contains more than one +1 elt, then the behaviour of process B (process A) is ambiguous w.r.t process A (process B) when traversing unilogue A_i (B_k)
Ambiguities need not always be design errors but may be and so must be discovered anyway.
We extend our example by choosing an acknowledge event (4) different from the connect event (2). Furthermore when a collision occurs reception of event 2 sets process A into state 4 and process B

into state 2 so that a collision is resolved in favour of process A initiating the connection. The dialogue matrix expands to 3×3, and the validation matrix is

$$Val[D] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$A_1 = \{-2, +4, +3\}$

$B_1 = \{+2, -4, -3\}$

$A_2 = \{+2, -4, +3\}$

$B_2 = \{-2, +4, -3\}$

$A_3 = \{-2, +2, +4, +3\}$

$B_3 = \{-2, +2, -4, -3\}$

Nonsquare dialogue matrices occur when there are more dialogues in one graph than in the other. eg we could have

$$Val[D] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Either this means that B_3 and B_4 may be discarded as they were never meant to be traversed (giving $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$) or they were intended to be executed, in which case the matrix can always be extended to yield

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

5.4 THE VALIDATION FUNCTION.

We now introduce some conditions (examples omitted)

The Post-Transmission Condition: when $[A_i, B_k]$ is executed, if for all event transmissions in $[A_i, B_k]$, if A_i (B_k) contains an event transmission it must be the transmission of an event subsequently received in B_k (A_i)

The Pre-Reception Condition: when $[A_i, B_k]$ is executed, then for all event transmissions in $[A_i, B_k]$, if $A_i^{(B_k)}$ contains an event reception it must be the reception of an event previously transmitted by B_k (A_i)

The Completeness Condition: for all event transmissions in $[A_i, B_k]$, if $A_i^{(B_k)}$ transmits an event x that can collide (occur simultaneously) with an event or non-event y from $B_k^{(A_i)}$, then $g_B^{(g_n)}$ must be designed to receive x independently of whether or not y is generated (are y traversed or not).

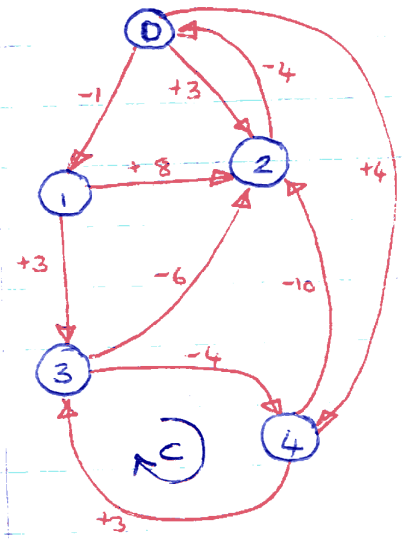
We can now give precise definitions:

- A dialogue is occurable if it fulfils the post-transmission and pre-reception conditions
- A dialogue is well-behaved if it fulfils all 3 of the above conditions

In a dialogue $[A_i, B_k]$ let $+x, +y$ be the first event receptions in A_i, B_k respectively that contradict the pre-reception conditions, and let $[A_i', B_k']$ be the $[A_i, B_k]$ section preceding $+x$ and $+y$.
Then the dialogue $[A_i, B_k]$ is non-occurable if $+x$ and $+y$ exist and $[A_i', B_k']$ is a subset of a larger section of some other dialogue, whereby no event belonging to that larger section contradicts one three conditions
Our definition of erroneous (neither well-behaved nor non-occurable)

5.1.5 The Duologue Matrix.

For more complicated protocols it becomes difficult to obtain S_A and S_B by inspection. Graph theory comes to our aid here. Consider the graph g_A and its state transition matrix $[a]$:



Current State \ Next State	0	1	2	3	4
0	λ	-1	+3	λ	+4
1	λ	λ	+8	+3	λ
2	-4	λ	λ	λ	λ
3	λ	λ	-6	λ	-4
4	λ	λ	-10	+3	λ

(NB $\lambda \Rightarrow$ no arc)

Let $[a]^j$ be the product of $[a]$ with itself j times, and let $a_{ik}(j)$ be the "a_{ik}" of $[a]^j$

i.e. $a_{ik}(j) = \bigcup_n a_{in}(j-1) \times a_{nk}(1)$

where $\bigcup_n$ represents the union over the products for all n, and $\times$ represents the Cartesian product between sets, and where any event sequence containing λ's are dropped.

$a_{00}(j)$ is thus the set of all possible paths starting and ending in state 0 with arclength j, i.e. $a_{00}(j)$ contains all unologies of length j

To eliminate all non-unologies from $a_{00}(j)$ we use a matrix with the same order as $[a]$ called $[I']$;

$[I']$ is an identity matrix except that $I'_{11} = 0$.

and we define $[a']^j = (((([a].[I'])[a]).[I'])[a]\ldots$

with j matrices $[a]$ and $j-1$ matrices $[I']$

Then $S_A = \bigcup_{j=1}^{n} a_{oo}(j)$ $\quad S_B = \bigcup_{j=1}^{m} b_{oo}(j)$

If g_A contains no cycles that are both reachable from state 0 and bypass state 0, then for n equal to the number of states in g_A, the expression for S_A yields all umlogues in g_A. (similarly for g_B). Testing for this condition is simple, namely, if the first row of $[a_i]^{n+1}$ is empty, then no such cycles are present.

We always require that g_A and g_B be strongly connected ie each state is reachable from every other (including state 0 reachable from state 0, obviously) Strong connectivity is fulfilled if the only states in g_A and g_B that are not traversed by any elt of S_A and S_B are states 0.

If g_A contains cycles that are both reachable from state 0 and bypass it, then $n = \infty$. We can choose n sufficiently large so that all such cycles are traversed by at least one or more umlogues, although this does not completely eliminate the possibility of undetected errors. A possible way to avoid this difficulty is to specify that any given cycle in one graph be matched by an equivalent cycle in the other graph so that a single cycle traversal suffices to validate the cycle.

6. Network Analysis and Design

6.1 Introduction

The network topology design program can be characterised as:

Given
- location of the hosts and terminals
- traffic matrix γ_{ij}
- cost matrix c_{ij}

Constraints
- Reliability
- Delay / throughput

Variables
- Topology
- line capacities
- flow assignment

Goal
- Minimise cost.

For most design purposes hosts and terminals are equivalent, and may be lumped together as locations or sites. The traffic matrix tells how many packets per second on average must be sent from site i to site j. It is usually unknown, but is commonly assumed to be proportional to the product of the populations of the two sites, divided by the distance between them. The probability density function for packet length is assumed known and the same for all sites.

The network design problem is huge. For a n site network, there are $\frac{n(n-1)}{2}$ potential lines, and hence $2^{\frac{n(n-1)}{2}}$ topologies (Eg $n = 10$, topologies $\approx 3 \times 10^{13}$).

For this reason network design is usually done in a hierarchical fashion - a backbone connecting nodes is designed, and concentrators and multiplexers used to connect the local access networks to the backbone. This reduces the topology design problem into several subproblems:

- Backbone Design
  - Topology
  - Line capacities
  - Flow assignment
- Local Access Design
  - Location of concentrators
  - Assigning customers to concentrators
  - Terminal interconnection within a site.
- Assignment of concentrators to nodes (difficult?)

6.2 CONNECTIVITY ANALYSIS.

6.2.1 CUTS AND NETWORK FLOW

An X-Y Cut is a set of arcs whose removal disconnects node X from node Y. A minimal cut is one in which replacement of any of its members reconnects the graph. In a weighted graph (weights representing line capacities), the capacity of a cut is the sum of the arc weights in the cut. A cut with the minimum capacity is a minimum cut (not necessarily unique. A minimum cut is a minimal cut, but not nec. vice-versa).

We make our graphs directed to indicate transmission (thus for a full-duplex connection we use two arcs). Consider a flow of information from node X to node Y. The flow is feasible iff

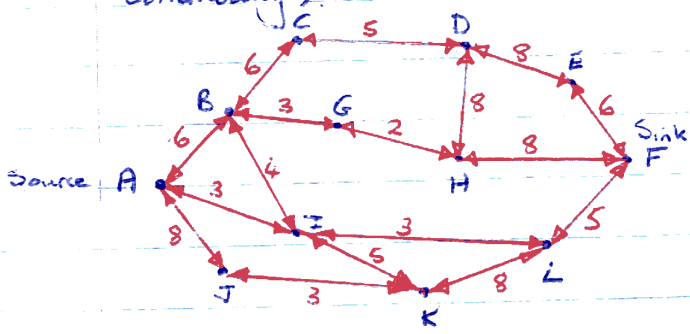
- the source X has no inward arcs (data move away from X)
- the sink Y has no outward arcs (data not emitted by Y)
- no arc has more flow than its capacity
- inflow and outflow balance at all nodes except the source and sink

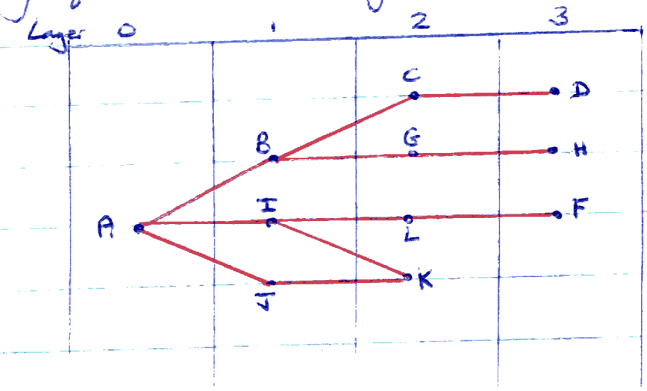
- the outflow at the source equals the inflow at the sink

If there are multiple sources and sinks, the single source-sink graphs may be linearly superimposed.
Note that the cut with the smallest capacity between two nodes forms the bottleneck between them. i.e.
The maximum flow between any two arbitrary nodes in any graph cannot exceed the capacity of the minimum cut seperating those two nodes. (in fact is equal to)
(as all traffic between the two nodes must flow across this cut, or it wouldn't be a cut)

6.2.2 THE MAX-FLOW ALGORITHM (MALHOTRA ET AL 1978)

The algorithm takes as input a weighted directed graph, and determines the maximum feasible flow between a given source and sink. It builds a layered network from the source to the sink. The source is placed alone in layer 0. Any node connected directly to the source by a "useful" arc goes in layer 1; any node connected to a layer 1 node by a "useful" arc goes in layer 2, etc (see below)
A useful (directed) arc from X to Y has either a currently assigned flow from X to Y less than the capacity or there is some flow currently assigned along the arc from Y to X. Either way the effective (net) flow from X to Y can be increased. (although in the latter case only if we are dealing with a single commodity)



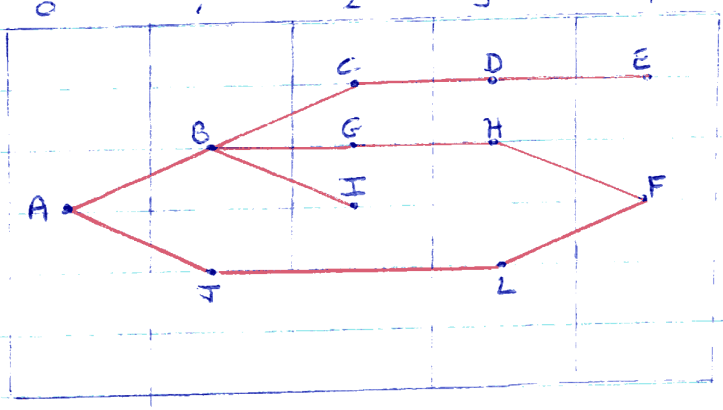


The next step is to prune off all the dead ends, those paths that don't terminate at the sink. After pruning, every node remaining in the layered network has one or more outgoing arcs leading to the sink and one or more incoming arcs from the source. In our example, only AILF remains.

The next step is to examine each node in the layered network in turn to find the potential flow increase it can handle. For each outgoing arc from X to Y, this is the capacity$_{XY}$ + flow assigned$_{YX}$ - flow assigned$_{XY}$. The total output potential is the sum of the potential of all arcs communicating with the next higher layer. The input potential can be found similarly. If, for example, the node has o.p. 7 and i.p. 4, the node can accept 4 more units of flow through it (ie it has sufficient input & output capacities)
The layered network node with the smallest potential is called the reference node, having the reference potential. (neither source nor sink may be ref. node).
In our example, both I and L have pot. 3, so we can choose either.
We then start at the reference node and push a flow equal to the reference potential forward to the sink, using as many arcs as necessary. Any flow pushed in to a node must be pushed out again towards the sink - only the sink can accept extra flow without getting rid of it. It is always possible to push and pull the requisite quantity of flow from the reference node, 'cause every other node in the layered network has at least as much potential, both input and output. Having augmented the flow in the selected arcs, we begin building a new layered network. Arcs which are no longer useful as they are saturated from the first layered network are not used;

this means we get a different network:

Layer 0 1 2 3 4



After pruning, only ABGHF and AJKLF remain. The reference node is H (G is as good) with potential 2.

The cycle of building a layered network, finding a reference node and augmenting the flow continues until it is not possible to build a layered network (ie there is no useful path from source to sink). When this happens, the current flow is the maximum flow, and the set of saturated arcs form a cut set.

It should be noted that it is not necessary to rebuild the layered network until all paths from source to sink have been saturated. After having found the reference node, a check should be made to see if another node with nonzero potential can be found.

6.2.3 DISJOINT PATHS

The arc-connectivity between two nodes X and Y is the minimum number of arcs that must be removed from the graph to disconnect X and Y. Two paths are arc-disjoint if they have no arcs in common (although they may have nodes in common). If there are k arc-disjoint paths between X and Y, then the arc-connectivity of X and Y is at least k. The arc-connectivity C_a of a graph is the minimum of the arc-connectivities of all pairs of nodes in the graph

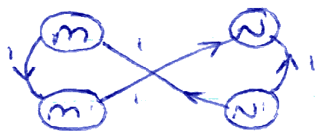
Note that we can analogously define the node-connectivity between nodes and of a graph, and node-disjoint paths. Node-disjointness is a stronger condition than arc-disjointness, as node-disjoint paths are necessarily arc-disjoint, while the converse is often not true.

between x & y.

To determine the number of arc-disjoint paths, we use the max-flow min-cut theorem. Each edge is given weight 1. This means that flow splitting is impossible (as weights must be integral values), as is flow combination (as capacity would be exceeded). Therefore the only utilised paths must be arc-disjoint. Each of these must cross the minimum cut. Thus the number of arc-disjoint paths is equal to the minimum cut.

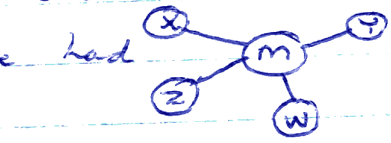
Determining node-connectivity is more complex. Assume the graph has n nodes and a arcs. If X and Y are adjacent, the node-connectivity is defined to be $n-1$, as removing the other $n-2$ nodes will not disconnect X and Y. Otherwise we convert this to a problem of finding arc-connectivity.

To do this we convert the original undirected graph into a directed graph with $2n$ nodes and $2a+n$ arcs, by replacing each node (m) with two nodes (m) → 1 → (m'), and replacing each arc mN with two arcs $m'N$ and $N'm$, i.e. (m)—(N) becomes

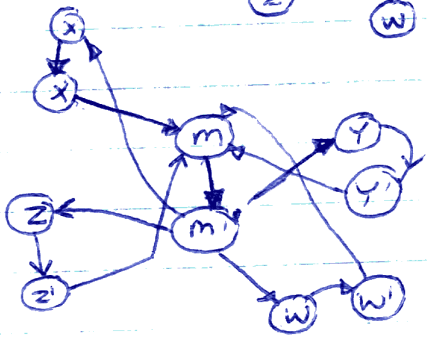


. Having done this, the number of node-disjoint paths is the maximum flow (i.e. minimum cut) between X and Y.

This is because, as splitting and combining is not allowed, each node can only occur on one path (as there is only one arc out of each of our original nodes to its prime node, and this arc occurs on an arc-disjoint path)

eg if we had 

we now have



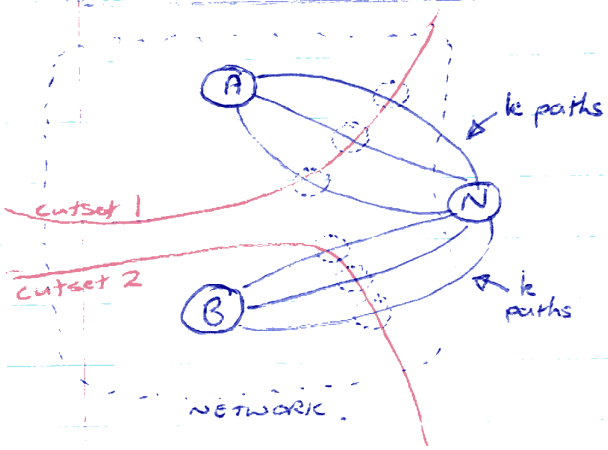
and while in the original graph there were arc-disjoint paths from X to Y and from Z to W, now only one of the two is possible

The arc-connectivity tells us how many lines we can have failing without disconnectivity, while the node-connectivity tells how many nodes may fail.

We thus have methods to determine the arc- and node-connectivities of the two nodes. When we wish to determine them for the whole graph (ie find the minimum over all pairs of nodes), we have the problem of having to calculate $\frac{1}{2}n(n-1)$ connectivities. We need more efficient methods.

We already know that $C_n \leq C_a \leq d$ (Whitney's theorem) where d is the minimum degree taken over all nodes. We thus use the strongest condition, node-connectivity, rather than arc-connectivity. If a network can withstand the loss of k nodes, it can also withstand the loss of k arcs. We also, instead of trying to find the connectivity of the network, rather try to show it has connectivity of at least k, for some k.

THE KLEITMAN CONNECTIVITY TEST (1969)



Choose a node (N), and "pull it out" of the network. Then we must first show that between (N) and any other node in the network, there are at least k node-disjoint paths.

Then, for each pair of nodes in the network (except (N)) (say (A), (B)), we must show that there is no node-cutset separating (A) and (B) with less than k elts. Neither cutset 1 nor cutset 2 will do, as each has at least k elts. Thus, if such a cutset exists, it must include (N) itself. Thus, if we delete (N) from the graph, we have reduced this to a problem of proving that the remaining graph has connectivity k-1.

We continue by selecting from the remaining graph another node (N') and testing that it is connected to every node of the remainder with connectivity k-1, and then prove the graph with both (N) and (N') deleted has connectivity k-2. After repeating this process for k-1 nodes, we are left with the question - is the remaining graph connected? This can easily be tested.

Eg for a 100-node graph, to show 4-connectivity we would have to perform 99+98+97 = 294 pairwise connectivity tests, plus show the remaining 97-node graph is connected; a substantial improvement over full pairwise testing.

EVEN'S CONNECTIVITY TEST (1975)

1) Number the nodes 1, 2, ..., n
2) Form the subset consisting of nodes 1, 2, .. k.
   Check that each node in the subset has k node-disjoint paths to each of the other nodes in the subset
   (if not, the test obviously fails)
3) For each node j, where $k < j \leq n$, do:
   3.1) Form the subset $L = 1, 2, \ldots, j-1$
   3.2) Add a new node X and connect it to each node in L
   3.3) Check that there are k-node disjoint paths between j and X. If not, fail

Even's algorithm requires n applications of the max-flow algorithm, while Kleitman's requires kn applications (where k is typically 2 or 3). Thus Even's algorithm is more efficient.

6.3 SUBNET DESIGN.

The problem is so complex that we have to use lots of trial and error, and computer time, in an iterative process, ie generate potential network topologies until one is found which satisfies connectivity and delay constraints. Compute its cost. Now make minor modifications to generate several new networks. If any of these is better, adopt it as the basis for further modifications, and soon. Thus the main tasks involved are:

- choosing starting topologies
- assigning flow and capacities
- generating slightly modified networks from a given one.

Session
Transport
Network
Link
Physical

7. LAYER 4 - THE TRANSPORT LAYER.

The lowest three layers are concerned with the transmission, framing and routing of packets between machines. The TL, in contrast, has the task of providing a reliable and efficient end-to-end transport service between (user) processes rather than just machines. In general, transport layer software resides in the hosts, not the nodes (the lower layers have software on both hosts and nodes). The technical issues of ensuring reliable end-to-end communication center on the user interface to the TL: naming and addressing of users, connection establishment and termination, buffering and flow control, multiplexing, error recovery, and synchronisation.

The network layer delivers packets error-free and without duplicates, but not necessarily in sequence. The functions of the TL are then:

- delivering messages (S-SDU's) end-to-end
  (ie packets are delivered error-free, without duplicates and in sequence)
- responsibility for flow control (using windows)
- recovery from network crashes
- addressing and network interconnection.

The TL has the overall and final responsibility for controlling the transfer of data between two users.

The TL provides three types of network service:

TYPE A: acceptable error rates and acceptable signalled failure rates

TYPE B: acceptable error rates and unacceptable signalled failure rates

TYPE C: error rates acceptable

7.1. TRANSPORT LAYER PRIMITIVES

The following service primitives have been defined and accepted by ISO and CCITT :

PRIMITIVE	PARAMETERS
T-CONNECT request	(called transport address, calling transport address, options, service quality, TS-user data)
T-CONNECT indication	(called transport address, calling transport address, options, service quality, TS-user data)
T-CONNECT response	(responding address, options, service quality, TS-user data)
T-CONNECT confirm	(responding address, options, service quality, TS user data)
T-DISCONNECT request	(TS user data)
T-DISCONNECT indication	(disconnect reason, TS-user data)
T-DATA request	(TS-user data)
T-DATA indication	(TS-user data)
T-EXPEDITED DATA request	(TS-user data)
T-EXPEDITED DATA indication	(TS-user data)

7.2 TRANSPORT LAYER PROTOCOLS

7.2.1 PROTOCOL CLASSES

Both CCITT and ISO have agreed on the following five classes in the Transport Protocol Specification:

Class 0 : Simple class
Class 1 : Basic class error recovery class
Class 2 : Multiplexing class

Class 3 : Error recovery and multiplexing class
Class 4 : Error detection and recovery class

With the exception of Classes 0 and 1, transport connections of a different class may be multiplexed together onto the same network connection. Classes and options within classes are negotiated during the connection establishment phase. Options define additional functions which may be associated within a class.

The choice of class will be made by the transport entities according to :

- the users requirement expressed via T-CONNECT request and T-CONNECT response service primitives
- the quality of the available network service
- the user-required service versus cost ratio acceptable for the transport user.

7.2.2 Common Elements of the Protocol Classes

The basic protocol unit is the TPDU (transport protocol data unit), which may be control or data. Unless there is a way to indicate the length of data in data TPDU's, concatenation of Control and Data TPDU's can only be achieved by placing Control TPDU's in front of a Data TPDU in a concatenated set of TPDUs.

All classes use the T-Connect Request / Indication and T-Connect Response / Confirm. This protocol exchange provides for :

- identification of calling and called session entities by use of reference numbers
- negotiation of classes and options
- negotiation of size of Data TPDU
- connection identification
- exchange of parameters

Since the TSDU (an arbitrary amount of data from the higher layer but which has to be treated as a single unit) is not constrained in size, it has to be segmented into manageable protocol sizes. The integrity is maintained by an end of TSDU indicator carried in every TPDU. This allows TPDU's to be chained together to represent a complete TSDU.

7.2.3. CLASS 0 - SIMPLE CLASS.

This is a bare bones service for use over type A network connections. Only functions available are for connection establishment, data transfer with segmenting, and error reporting. No functions exist for multiplexing, disconnection, flow control, error recovery, or expedited data transfer. Furthermore, no exchange of user data is permitted during connection establishment, only address and TPDU size parameters are allowed. As there is no explicit disconnection procedure, the Transport connection lifetime is the same as the Network connection lifetime. The standard maximum class 0 Data TPDU length is 128 octets including the TPDU header.

7.2.4 CLASS 1 - BASIC ERROR RECOVERY CLASS

This class is intended for use over type B connections and provides recovery from network signalled errors (network disconnect or reset). Class 1 provides Transport Connections with error recovery, expedited data transfer, disconnection and flow control based on the underlying Network Service provided flow control. Data may be exchanged during connection establishment. No functions are provided for multiplexing.

7.2.5 Class 2 - Multiplexing Class

This provides the capability of multiplexing several Transport Connections within a single network connection. It has been designed for type A Network Connections.
Optional flow control is provided, although no functions are provided for error detection and recovery. If the network resets or disconnects, the Transport Connection is terminated without an explicit end-to-end exchange and the TS users are informed. In addition to class 0 functions, the following are provided:

- multiplexing
- flow control
- exchange of user data during connection establishment
- credit mechanism (with flow control option)
- expedited data transfer
- explicit disconnection.

7.2.6 Class 3 - Error Recovery Class

Intended for type B Network Connections, this is Class 2 enhanced with functions to recover from network signalled errors.

7.2.7 Class 4 - Error Detection and Recovery Class

This is designed for type C connections (i.e. connections considered to have an unacceptable residual error rate relative to high-level requirements), and detects lost TPDU's, misequenced TPDU's, and duplicated TPDU's or parts of TPDU's (control or data type). The main differences from class 3 are the addition of time-out and checksum mechanisms. 3 timeouts provided: "try again" after failure (~2 minutes)

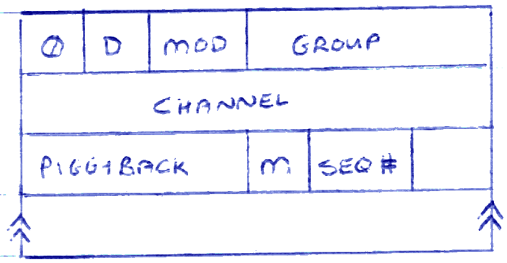
retransmission of TPDU, inactivity (network died)

7.2.8. General Structure of TPDU's.

TPDU's are divided into four parts, each containing an integral number of octets. The parts are:

- Length Indicator Field (LI)
- Fixed part
- Variable part
- Data field.

7.2.9 The Transport Layer & the D Bit in X.25



X.25 Data Packet

If D=0 then ACK means local DCE has received packet

If D=1 then ACK means packet has been delivered to remote DCE.

7.3 Transport Protocol Design Issues

The most important single factor affect TP design is the kind of service the subnet provides - if this is a virtual circuit service (guaranteeing that messages are delivered in order without error, loss or duplication) the TP is simple, if however the subnet provides a datagram service, the TP needs to be much more sophisticated.

7.3.1 Transport Service (provided collectively by layers 1-4)

The TL defines an set of transport addresses or sockets through which communication occurs. These form a network-wide name space allowing processes to refer to one another. To

ensure these are globally unique, each consists of a network number, host number and a port assigned by the host. Users of the TS can request a T. connection between a local T address and one on a remote machine.

7.3.2 ADDRESSING AND CONNECTION ESTABLISHMENT.

A generic service performs some set of actions for its customers (eg compilation, DB management, etc). A servrice belonging to a particular host can be accessed by remote hosts. A process called a server provides the servrice to any (authorised) process that requests it.
One way to implement servrices is to have each server process execute a LISTEN (connect request) call on some transport address. To use a servrice, a user then does a CONNECT specifying the appropriate remote transport address. A mapping between names and addresses is performed by a name server, which address never changes.
Although there are no general rules about how servrices are named, there are two general strategies for allocating transport addresses, hierarchical addresses and flat addresses.
With a hierarchical address space, the address consists of a sequence of fields used to disjointly partition the space, eg
not portable - routing more difficult
<address> = <galaxy> <star> <planet> <country> <net> <host> <port>
Flat addresses have no particular relation to any hierarchy (eg student numbers)
CCITT recommendation X.121 has decreed that public computer networks will use a numbering system similar to the PSTN, with each host identified by a decimal number consisting of a country code (3 digits), network code (1 digit) and a network address (10 digits, typically 7 for host & 3 for port, although this is not specified). If 10 digits do not provide

sufficient resolution, the user data field at the end of the X.25 CALL REQUEST packet can be used to provide subaddressing, essentially more address bits.

7.3.3 FLOW CONTROL & BUFFERING

A sliding window or equivalent scheme is needed to keep a fast transmitter from overrunning a slow receiver. For an unreliable subnet the sender must buffer messages until ACKed. Receivers may or may not use buffers. If the receiver knows that the sender buffers all messages until ACKed, the receiver may, for example, maintain a single buffer pool shared by all connections. When a message is received an attempt is made to allocate it buffer space - if this fails, the message is discarded (and the sender will retransmit).

On the other hand, if the subnet is very reliable, other trade-offs become possible. In particular, if the sender knows that the receiver always has buffer space, it need not retain copies of the messages it sends. However, if the receiver cannot guarantee that every incoming message will be accepted, the sender will have to buffer anyway. In the latter case, the sender cannot trust the subnet's acknowledgement, because the ACK means only that the message arrived, not that it was accepted.

Buffers can be

- chained fixed size (chained ⇒ connected in a linked list)
- chained variable size (better memory utilization, but more complex)
- circular (usually one large circular buffer per connection) (good for heavily loaded connections)

The optimum trade-off between source and destination buffering depends on the type of traffic carried by the connection.

- Low bandwidth, sporadic traffic ⇒ better to allocate dynamically rather than dedicate. Sender buffers must retain copy 'till ACK'ed.
- High bandwidth, heavy traffic ⇒ best for receiver to dedicate a full window of buffers. receiver buffers.

As connections are made and broken, and as the traffic pattern changes, the sender & receiver need to dynamically adjust their buffer allocations. Consequently, the T-protocol should allow a sending host to request buffer space at the other end.

A reasonably general way to manage dynamic buffer allocation is to decouple the buffering from the ACKs. Dynamic buffer management means, in effect, a variable sized window. Initially the sender requests a certain number of buffers, based on its perceived needs. The receiver then grants as many of these as it can afford. Every time the sender transmits a message, it must decrement its allocation, stopping altogether when the allocation reaches zero. The receiver then piggybacks both ACK's and buffer allocations onto the reverse traffic.

Potential problems with buffer allocation schemes of this kind can arise in datagram networks if control messages get lost. If the receiver allocates more buffers after running out, but the allocation message it transmits to the sender gets lost, the sender is deadlocked. To prevent this, each host should periodically send control messages giving the acknowledgement and buffer status on each connection.

With memory prices falling, it may become feasible to equip hosts with so much memory that lack of buffers is rarely, if ever, a problem. However, another limit on the sender's data rate is the carrying capacity of the subnet. If adjacent nodes can exchange at most x frames/sec and

there are k disjoint paths between a pair of hosts, there is no way that those hosts can exchange more than k packets/sec, no matter how much buffer space is available at each end. If the sender pushes too hard, the subnet will become congested.

What is needed is a flow control mechanism based on the subnet's carrying capacity rather than the receiver's buffering capacity, which must be applied at the sender. Belsnes (1975) has proposed a sliding window scheme in which the sender dynamically adjusts the window size to match the carrying capacity. If the network can handle c packets/sec, and the cycle time (incl. transmission, propagation, queuing, processing & ACKing) is T, then the sender's window should be cT. With a window of this size the sender normally operates with the pipeline full; any small decrease in network performance will cause it to block. The sender can monitor both parameters using moving averages and periodically recompute and adjust the window size.

7.3.4 MULTIPLEXING

When connect time forms the major component of the carrier's bill, the multiplexing of different transport connections onto the same network connection becomes attractive. In such upward multiplexing, it is up to the transport layer to group transport connections according to their destination and map each group onto the minimum number of virtual circuits.

Another form of multiplexing, downward multiplexing, is useful when a user requires a higher bandwidth channel than the flow control mechanism will allow. The TL opens multiple network connections, and distributes the traffic among them on a round-robin basis. With k network connections open, the effective bandwidth is increased by a factor of k. If multiple

host-node lines are available downward multiplexing can increase performance even more.

7.3.5 Synchronisation in the Presence of Delayed Packets

Delayed duplicates occur when the routing algorithm gets in a loop (A thinks best route is via B and vice-versa) or when congestion simply causes a long delay. If the new retransmitted packet takes a different route, it may arrive before the original, at which time both sender and receiver forget about the extra packet. After a while the sequence numbers will wrap around, and if the delayed packet now arrives at the receiver, it may be accepted and the correct packet rejected as a duplicate. This error will not be detected.

or after host crashes.

One way to attack this problem is to use extended transport addresses; however this means each host must maintain a certain amount of history information indefinitely, and this may be lost by host crashes. We must instead devise a mechanism to kill off very old packets. Packet lifetime can be restricted to a known maximum using one of the following techniques:

- restricted subnet design (preventing looping, and thus having a known max path length)
- putting a hop counter in each packet
- time stamping each packet (requires clock syncing - non trivial)

In practice we must guarantee that not only is the packet dead, but so are all acks to it. Let T be some small multiple of the true maximum packet lifetime. If we wait for T before retransmitting a packet, we can be sure that neither the original packet nor its ACKs still exist. We also have the requirement that no host may ever send more than $2^k/T$ packets per second (where k is the number of bits in the sequence number, and T is in seconds).

There are several ways we can now deal with delayed duplicates. One way is to wait for T secs after closing a connection

before opening another connection with the same connection identifier. Two serious disadvantages - requests to open connections may be delayed, and hosts must maintain information about connections that no longer exist.

Another method, which still suffers from the second disadvantage, is to remember the last sequence number used for each connection for a time T after the connection is closed. If the connection is reopened while old packets may still exist, there will be no overlap between old and new sequence numbers.

A minor variation of the above is to include an incarnation number in each packet. Each time a connection identifier is reused, the incarnation number is incremented, distinguishing new packets from those of previous incarnations. However, this is just as vulnerable to being lost in a crash.

All three of these methods is that it is difficult to accurately estimate T for large networks where local details may not be known.

We now describe a synchronisation method due to Tomlinson (1975). that solves many of these problems but introduces some peculiarities of its own (most detail omitted - see Tanenbaum pp 347 - 349).

To establish a connection, Tomlinson introduced the three-way handshake. This establishment protocol does not require both hosts to begin sending with the same sequence number. A chooses a sequence number somehow, say x, and sends it to B. B replies with a SYN2 message ACK'ing x and announcing its own initial sequence number y. Finally A ACK's B's choice of an initial sequence number in its first data message.

If the first message is a delayed duplicate SYN1 from a connection since closed, B reacts by sending A a SYN2 message. When A rejects B's attempt to establish, B realises that it was tricked and abandons the connection.

The worst case is when both a delayed SYN1 and an ACK to a SYN2 are floating around. As in the previous example B gets a delayed SYN1 and replies to it. At this point it is crucial to realise that B has proposed using y as its initial S#, knowing full well that no messages containing S# y or ACK's to y are still in existence. When the second delayed message arrives at B, the fact that z has been acked instead of y tells B that this too is invalid. (see below for these examples)

#	A	Packet	B	Comment
1	→	<type = SYN1, seq = x>	→	A wishes to initiate
2	←	<type = SYN2, seq = y, ack = x>	←	B accepts A's request
3	→	<type = DATA, seq = x, ack = y>	→	A acks B and starts

#	A	Packet	B	Comment
1	···	<type = SYN1, seq = x>	→	Delayed dup. SYN1 arrives at B.
2	←	<type = SYN2, seq = y, ack = x>	←	B accepts A's request
3	→	<type = REJECT, ack = y>	→	A rejects B's connection

#	A	Packet	B	Comments
1	···	<type = SYN1, seq = x>	→	Delayed dup SYN1 arrives at B
2	←	<type = SYN2, seq = y, ack = x>	←	B accepts it
3	···	<type = DATA, seq = x, ack = z>	→	Delayed dup. B sees $z \neq y$
4	→	<type = REJECT, ack = y>	→	A rejects B's attempts as before

7.3.6 CRASH RECOVERY

If the subnet goes down, information about the status of packets currently in the net is lost. Recovery can be done by enquiries and retransmissions.
If a host crashes, the receiver can ask for information about any open connections, and the sender can retransmit messages. This can lead to missing messages or duplicates.

7.4 INTERCONNECTION OF PACKET SWITCHING NETWORKS.

We have discussed communication between hosts in the same network; we now examine inter-network connections. This introduces problems (eg one network may be datagram, the other virtual circuit). Thus some transport protocols designed with internetworking in mind assume worst case and put full responsibility for flow and error control in the hosts.

7.4.1 GATEWAYS

Gateways are "black boxes" connecting networks whose function is to perform protocol conversions. Gateways may be bilateral or multilateral. If we consider the collection of gateways and networks as an undirected graph, with gateways as nodes and networks as arcs, we get a supernetwork in which each "line" may have a different speed, delay, price and protocol.
An alternative to this approach is to build a completely new network (of gateway nodes) exclusively for the purpose of carrying internet traffic. Whenever a packet must be passed from one net to another, the source net delivers it to the interconnection net, where it is routed internally among the gateways until it is delivered to the destination network. Since the interconnection net's

only subscribers are the carriers who run the individual nets, it is sometimes called the carrier's carrier.

If the superset is highly connected (desirable for performance and reliability), writing all the gateway software is a major undertaking. A simpler approach is to use a standard internetwork packet format. Even if it is impossible to get all the networks to agree to a single internet format, there is much to be gained if most of them agree. The agreeing networks can use it, whereas connections to other networks will have to be made pairwise.

To solve the problem of who owns and maintains the gateway, each network operates a half-gateway responsible for converting to and from its own internal format only. In fact it may be possible to put this software in one of the nodes or hosts, thus avoiding a special machine altogether.

7.4.2 The Level of Interconnection

Should the gateway be a node or a host? If it is a node, addressing is a problem (the network must be specified adding information and possibly contravening the sending nets internal format) and if the two networks have different subnet protocols even more problems occur.

In contrast, using a host is much easier. Since the gateway is indistinguishable from any other host, but is simply a host common to both networks, internetting requires no changes to the subnet. To send an internetwork message, a host just addresses the message to the gateway host using the standard transport protocol. The gateway performs the format conversion and addresses it to the next network/half-gateway using the second networks' transport protocol.

7.4.3 The X.75 Model vs. the Datagram Model

The CCITT internetwork protocol X.75 is a VC model, whereas the research community prefers the datagram model.
The CCITT X.75 model is based on the idea of building up an internetwork connection by concatenating a series of intranetwork and half-gateway-to-half-gateway virtual circuits. The essential feature of this approach is that the transport layer explicitly asks the network layer to open a connection to a specific destination, which results in a sequence of virtual circuits being set up. Each gateway maintains tables telling which VC's pass through it, where they are to be routed, and what the new VC number is. To close a connection, the TL must be given an explicit command. The whole arrangement is similar to fixed routing, except only the sequence of gateways is necessarily fixed (intranet routes may be datagrams).
Although the X.75 protocol itself only applies to the gateway-to-gateway lines, it effectively dictates the concatenated VC architecture by requiring all packets on a connection to pass over the same gateway-to-gateway VC in sequence.
The alternative datagram model only allows the TL to inject datagrams into the subnet and hope for the best. It does not guarantee that the packets will travel the same route or arrive in order, assuming they arrive at all. Each method has associated advantages and disadvantages (see pp 363-4).

7.4.4 Internetwork Packet Fragmentation

Each network imposes some maximum size on its packets. A problem arises when a large packet wants to travel through a network whose maximum packet size is too small. Basically, the only solution to the problem is to allow gateways to break packets up

into fragments, sending each fragment as a seperate internet packet. Two opposing strategies exist for the more difficult problem of recombining fragments. The first is to send all packets through the constraining network to the same exit gateway where they are immediately recombined, making the fragmentation transparent to subsequent networks. This may result in lost performance due to a single exit gateway being allowed, as well as encoding info allowing the exit gateway to reconstitute the packet. Other problems are possible reassembly lockup, and the processing overhead.

The other technique is to recombine at the destination host. This means that multiple exit gateways can be used (except if a VC model eg X.75 is used) but means that the excess information must be carried all the way, and every host must be able to reassemble.

For ways of numbering fragments, see Tanenbaum p 367.

8 Layer 5 - The Session Layer.

8.1 What is a Session?

Before end-users can communicate, their logical units (LU's - applications, etc) must be connected in a session (LU-LU session). The session allows interaction synchronisation, data exchange and resource allocation (eg buffers).
The format of the data, and the amount of data that can be sent before an ACK is received, is governed by the protocol.
Network names are used for LU's; the network address is determined from the name. Messages must contain both source and destination addresses.

8.2 What does the Session Layer do?

- adds user-oriented services to the transport layer
- managing alternation
- synchronising, checkpointing, and crash recovery.
- segmenting files for transfer
- makes TL crashes transparent.

8.3 Session Concepts.

The session layer includes the following functions
- normal data exchange
- expedited data exchange
- token management (allowing exclusive rights to certain functions)
- dialogue control (full ≠ half-duplex) (only token owner can transmit)
- synchronisation (marking of synchronisation points)
  - 'resync' works with serial numbers
  - major marks
  - minor marks (optionally acked)

- resynchronisation
  - 'moving back' in dialogue
  - not allowed to synchronise to a point before last major mark
- activity management (breaking dialogue into discrete 'activities')
- exception reporting
- typed data function
- capability data function.

8.4 STEPS IN ACTIVATING A SESSION.

- S-CONNECT request sent in S-PDU
  - authorised to communicate?
  - path available?
  - LU's capable of meeting needs of end-users?
- Class of service required can be indicated in the parameters of the request.

8.5 CONTROLLING DATA FLOW

User can:

- transfer data
- control the dialogue (if half-DUP)
- synchronisation marks insertion
- resynchronisation of session to a previous mark
- activity management (start/end/interrupt activities)

8.6 SEQUENCING OF DATA EXCHANGE.

See examples opposite. (§8.8)

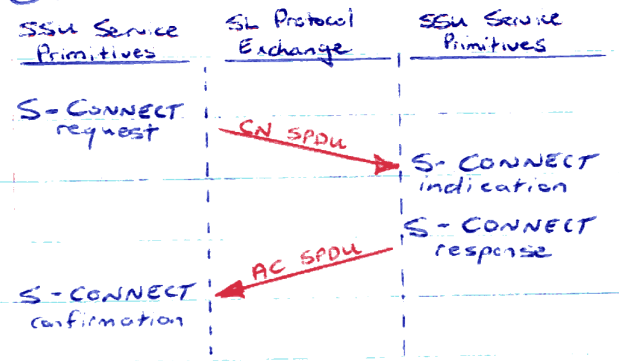
8.7 Connection Release.

This can be orderly when data transfer is complete and there is no data loss, or disorderly, where the session connection is aborted with possible loss of data.

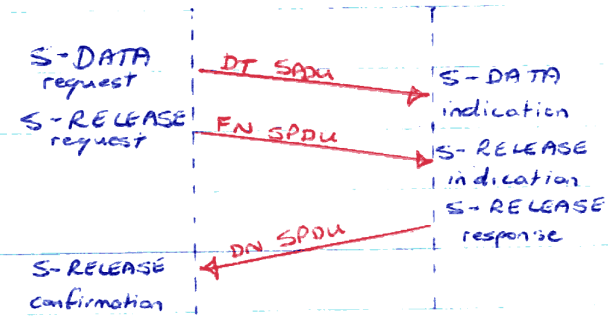
8.8 Examples of Protocol Exchanges

Abbreviations : CN - connect SPDU
AC - accept SPDU
DT - data SPDU
GT - give token SPDU
MKD - major mark SPDU
MCD - major mark confirmation SPDU
FN - finish SPDU
DN - disconnect SPDU.

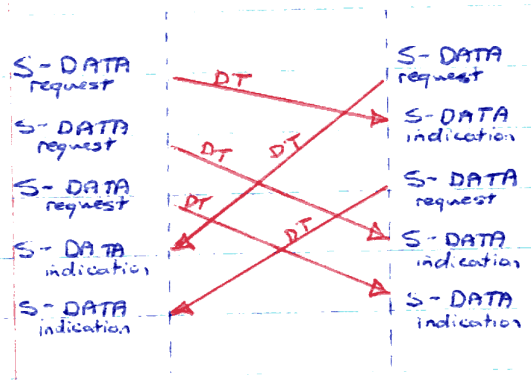
(A) Session Connection Establishment



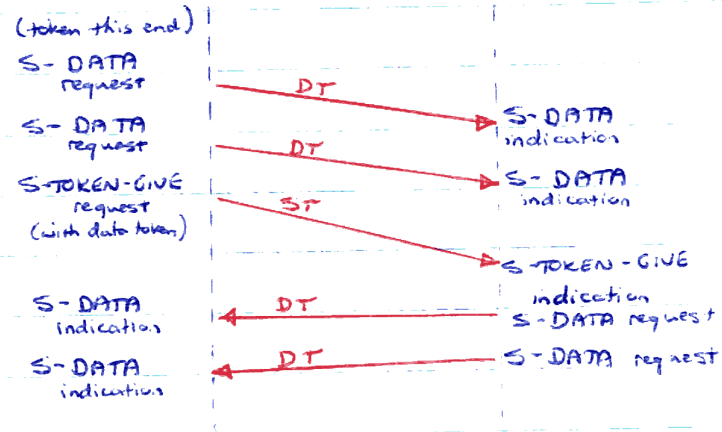
(B) Orderly Connection Release



(C) Duplex Data Transfer



(D) Half-Duplex Data Transfer

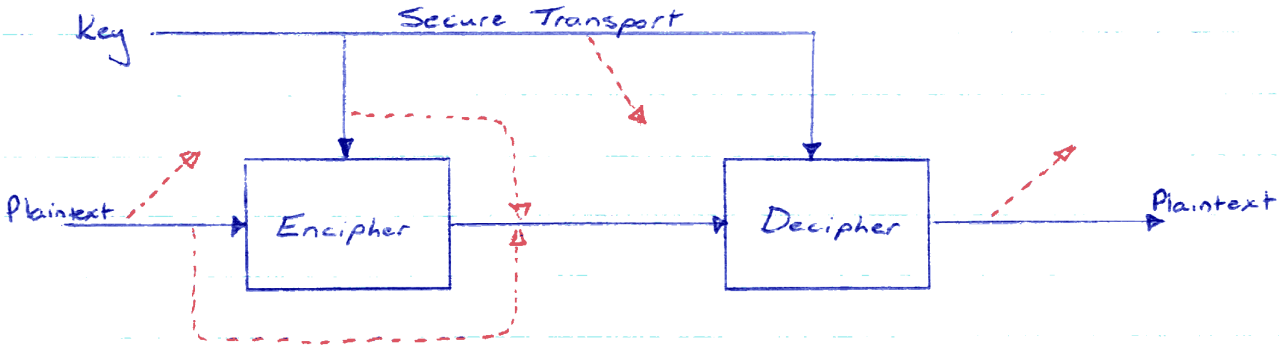


9. Layer 6 - The Presentation Layer

The function of the presentation layer is to provide the user process with certain useful but not always essential services. Which may be part of the O/S, but are preferably implemented as user-callable library routines.

9.1 Network Security and Privacy

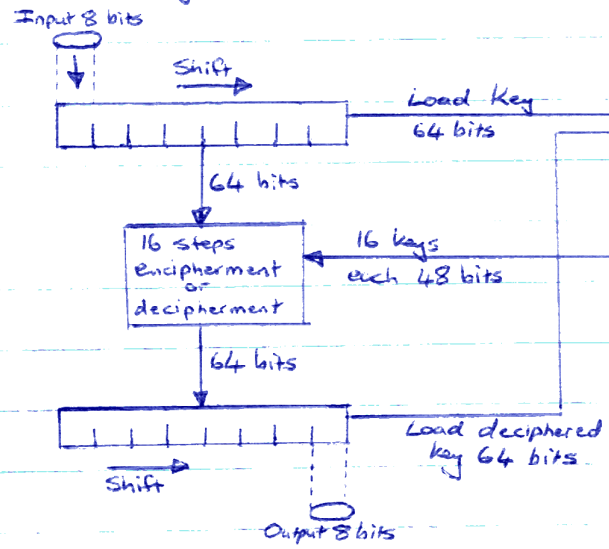
Cryptography (encipherment) is the technical means by which the traffic on a data network can be concealed and modification prevented. It can be performed on any level, eg by a hardware device between node and host (data link encryption - encrypts physical content) or by the presentation or application layer (end-to-end encryption; encrypts specific sessions; more flexible).
Attacks on the network can be through disclosure, modification, data insertion, or repeating old transmissions. The diagram below indicates potential points of attack:



A modern cryptosystem must be secure even when the algorithm and some examples of enciphered/deciphered text are known. For this reason a key is used; this is traditionally secret although public key cryptosystems exist.

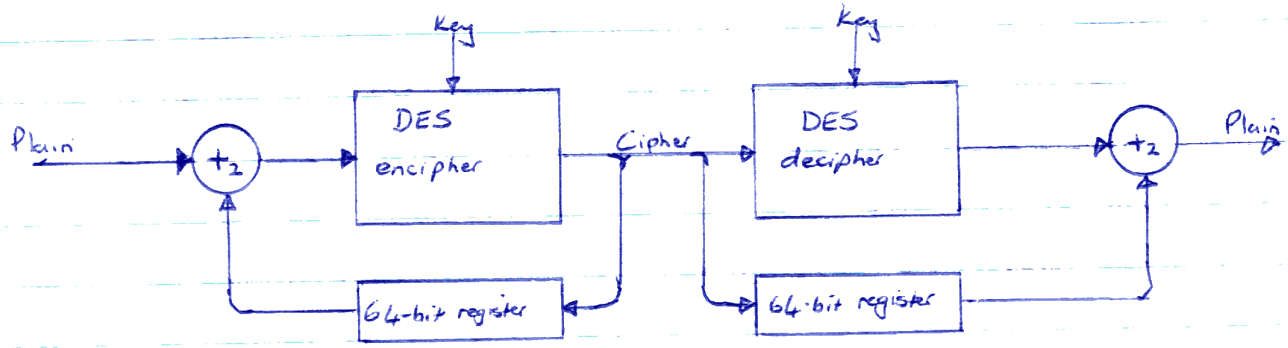
9.1.1 THE DATA ENCRYPTION STANDARD (DES).

This is a 64-bit block cipher, with a 64 bit key (of which 8 are parity bits) available as a chip.



SCHEMATIC OF A TYPICAL DES DEVICE. (operation p393).

There is a danger with block encipherment that repetitive patterns in the plaintext will show through in the ciphertext. One way to get around this is to make each block encipher depend on the value of previous blocks (chaining). Before each plaintext block is enciphered, the contents of the previous ciphertext block are added to it by modulo 2 addition (XOR) over the 64 bits. If we denote the i^{th} message and cipher blocks by M_i and C_i respectively, and the encipherment by E, then: $C_i = E(M_i + C_{i-1})$ and $M_i = E^{-1}(C_i) + C_{i-1}$



9.1.2 Personal Identification. (p395-9)

Three classes depending on:

- something person knows (eg PIN)
- something person possesses (eg electronic key)
- some personal characteristic (eg voice)

9.1.3 One-way Functions and their Applications. (p399-401)

A one-way function $F(x)$ is a function which is easy to compute, but whose inverse $F^{-1}(x)$ is extremely difficult to compute. These are useful for example for storing passwords on file in the encrypted form, so that it is not feasible to determine the original passwords, but by enciphering a password and comparing it to the file entry we can still determine if the password is correct.

9.1.4 Message Authentication. (p401-403)

Encipherment generally provides authentification itself, so here we are concerned with authentification without encipherment. We send a message authentification field with the message, which is encrypted with a secret key known by both sender and receiver. The receiver generates an authentification field from the message and key, and compares this with the field received.

An essential requirement of the authentification process is that it should not be possible to construct a message to fit a given authenticator (generally a secret key will prevent this) It must also not be possible to modify a message by following special rules which leave the authenticator unchanged.

9.1.5 USER AUTHENTICATION (p403/4)

User authentication can be obtained as a byproduct of encipherment or message authentication. This depends on the user's key or PIN entering into the secret key or some other part of the authentication process.

We often need a simple way to check keys before a dialogue begins. This should be an active check (not the same each time). For example, if A and B are starting to converse on a secure communication link, A can verify that B is a live terminal by sending a random number r over the link, to have r+1 returned by B. Then B should verify A in the same fashion.

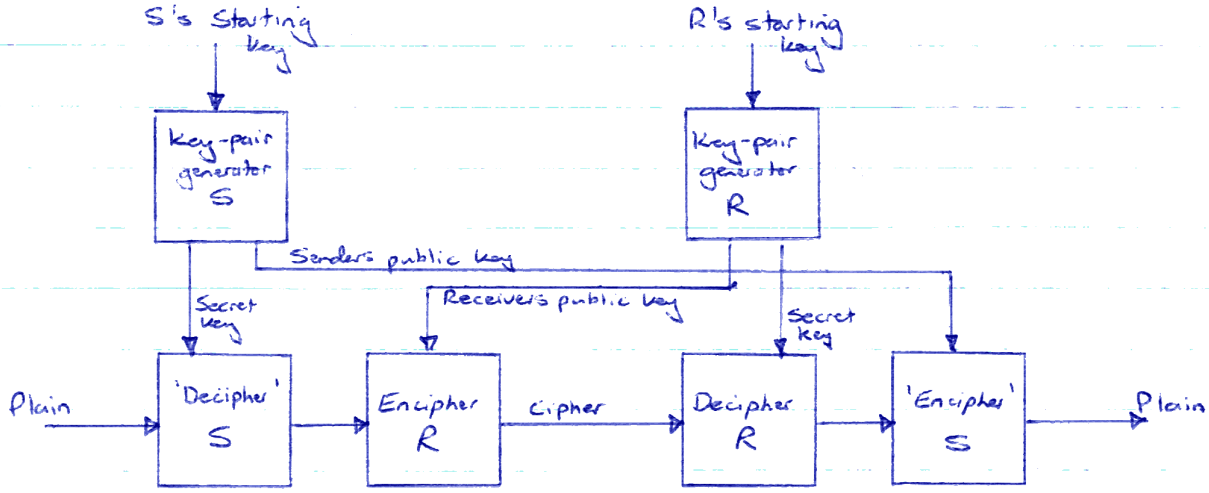
One problem that can arise is if the sender claims they never sent the message. A method of resolving disputes has been found as a byproduct of the public key cryptosystem.

9.1.6 THE PUBLIC KEY CRYPTOSYSTEM

This is different from traditional systems, in that the keys/algorithms for encipherment/decipherment are may be different. Clearly the two keys must be related. The process of calculating the keys is done by the receiver, who has a secret starting key from which are generated two keys: the public enciphering key, and R's secret deciphering key. Obviously the public key must be a one-way function of the starting key, and the ciphertext must be a one-way function of the cipher key (public), with this one-way function having a kind of 'trapdoor' in that knowledge of the secret key enables it to be inverted. This means that anyone can encipher text, but only R can decipher.

This system can also be used for user authentication by having

the sender do a similar thing, with a secret cypher key and public decipher key. Then only the sender can encipher an authentication message, although anyone can see that the authentication message does come from the sender. We can combine these two to get a public key encipher and authentication scheme combined as below:



(see p eg Rivest, Shamir & Adleman's method, p 409 - 412)

9.1.7 AUTHENTIFICATION IN PUBLIC NETWORKS.

In a public network, the best method is probably to use a central authentication server, whose public key never changes. A user can then find the latest public key for any other user from the server. The server can also be used as an independent third party to aid authentication (see p 413/4).

9.2 TEXT COMPRESSION

Text compression is useful to save bandwidth across channels, as well as having a slight security value.
Data sent over a transport connection can be viewed as a sequence of symbols $S_1, S_2, \ldots, S_N$, drawn from a (possibly infinite) set of symbols. Text compression can be approached in three general ways, respectively based on:

- the finiteness of the set of symbols
- the relative frequencies with which the symbols are used
- the context in which a symbol appears.

9.2.1 ENCODING A FINITE SET OF EQUALLY LIKELY SYMBOLS

Eg a library. A typical book has about 20 characters in its title; expressed in ASCII such a title requires 140 bits. Yet no library in the world has anywhere near 2^{140} titles. Instead a sequence number could be used (eg the Library of Congress has about 2^{26} titles, so a 26-bit number could be used).

9.2.2 HUFFMAN CODING.

1) Write down all the symbols, together with the associated probability of each. As the algorithm proceeds, a binary tree will be built up, with these symbols as the terminal nodes. Initially, all nodes are unmarked.
2) Find the two smallest nodes and mark them. Add a new node, with arcs to each of the nodes just marked. Set the probability of the new node to the sum of the probabilities of the two nodes it is connected to.
3) Repeat step 2 until all nodes have been marked except one (the prob. of this node will always be one).

4) The encoding for each symbol can now be found by tracing the path from the unmarked symbol to that symbol, recording the sequence of left and right branches taken. The code is just the path, with left = 0 and right = 1.

9.2.3 CONTEXT DEPENDENT ENCODING.

Huffman encoding implicitly assumes that the probability of a symbol occuring is independent of its immediate predecessor. However, this is not the case, and can be exploited in a number of ways; for example:

BAUDOT VARIATION

Baudot code is an old 5-bit telegraph code. We use 4 alphabets: uppercase, lowercase, numbers & special symbols, and control characters. 28 ~~bits~~ codes are allocated to each alphabet, with the remaining 4 codes being alphabet selection codes.

RUN LENGTH ENCODING

To encode long binary strings containing mostly zeroes. Each k-bit symbol tells how many 0 bits occurred between consecutive 1 bits. To handle longe zero runs, the symbol consisting of all 1 bits means the distance is $2^k - 1$ plus the value of the following symbol (or symbols).

REPEATED RUNS COMPRESSION.

Squash runs of repeated symbols into a count plus the symbol.

9.3 VIRTUAL TERMINAL PROTOCOLS.

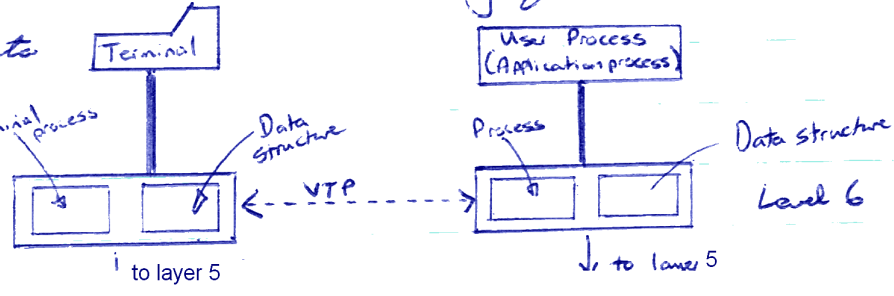
Protocols have been devised to hide the differences between different kinds of terminals.

9.3.1 CLASSES OF TERMINALS

Terminals fall into three broad classes - scroll mode, page mode and form mode. Scroll mode terminals have no local editing capability - when a key is hit the corresponding character is just sent (and possibly also displayed); when a character comes in over the line it is just displayed (eg. hardcopy terminals). Page mode terminals allow screen movement and editing on the current screen. Form mode terminals are usually dedicated data entry terminals.

9.3.2 THE DATA STRUCTURE MODEL

We now describe a model for form mode terminals (the other simpler types can be seen as special cases). The terminal is driven by a terminal process either in the host or terminal. This process communicates with its peer on the remote machine. The central idea is that of a network virtual terminal, represented by a data structure. This data structure is modified by the terminal user, and messages are sent to and from the terminal processes keeping the data structures identical on each end. This data structure also determines the display. The VTP must ensure the consistency of the two data structures after every update



9.4 File Transfer Protocols.

10 Integrated Services Digital Networks.

10.1 Aims of ISDN

- to combine all communication services currently offered over seperate networks into a single network which can be accessed over common facilities by any subscriber through a single connection - a worldwide public telecom network, uniformly accessible.
- conventional services - voice telephony, circuit and packet switched data coms, text message service and facsimile
- new services - enhanced telephony, home telemetry and videotex variations
- isolation of terminal and network evolution (smooth migration)
- provision of value added services
- charging according to capacity rather than connect time
- standardised, transparent transmission service
- support for multiplexing, switched services, and dedicated point-to-point services
- seperation and provision of competitive functions.
- tailoring of service purchased to actual needs.

10.2 Methods of ISDN

- based on digital transmission and digital switching
- built on top of existing IDN's
- PCM digitising of analogue signals
- TDM multiplexing of digital signals
- multiplexing incorporated into time division digital switches
- user access thru local interface to digital pipe of suitable size.
- control signals multiplexed onto same pipe
- basic service consists of two 64 Kbps B channels for digital voice/data, and one 16 Kbps D channel for control signals and data packets.

- primary service (digital PBX & LAN) via 24-channel or 31-channel types.
- digital central office connects ISDN loops to IDN, dedicated lines, packet-switched networks and time-shared transaction-oriented services.

10.3 CONFIGURATIONS

- digital PBX attached to multiple ISDN terminals on local side and multiple network termination devices on carrier side
- LAN connected via a gateway node to a single network termination device (carrier side)
- multiple ISDN terminals connected to network via a network termination device combining function of both local & carrier side devices

4 Digital Data Communication Techniques

4.1 Asynchronous and Synchronous Transmission

Asynchronous Transmission

Start-stop or asynchronous transmission is character-oriented (char = 5 to 8 bits) with resynchronisation at the start of each character. When no character is being transmitted the line is 'idle' (typically '1', MARK). The beginning of a character is signaled by a start bit ('0', SPACE). This is followed by the character data, and possibly a parity bit. Finally, at least 1, 1.5 or 2 stop bits are transmitted (MARK).
This form of communication can handle about 5% drift between receiver and transmitter without loss of synchronisation. Synchronisation errors are called framing errors (the character, start and stop bits constitute the frame), and are due to noise and excessively high drift. The disadvantage is the overhead of transferring at least two framing bits per character.

Synchronous Transmission

This can be achieved by using a separate clock line between receiver and transmitter, or by embedding the clocking information in the data signal (eg with biphase encoding). Each block begins with a preamble bit pattern and usually ends with a postamble bit pattern; these may include control information for data link control. The data plus control info constitute the frame, the exact format depending on whether the transmission scheme is character- or bit-oriented.
In the former case the block is treated as a character stream, etc.

4.2 Error Detection Techniques

When a frame is transmitted, three classes of probabilities can be defined at the receiving end:

- Class 1 - frame arrives with no bit errors (P_1)
- Class 2 - frame arrives with undetected bit errors (P_2)
- Class 3 - frame arrives with detected errors (P_3)

Error-detection techniques are based on the method of adding additional bits (error-detecting code) to a frame, the code being a calculated function of the other bits. Thus P_3 is the probability that if a frame contains errors, these will be detected. P_2 is the residual error rate.

Parity Checks

Character parity checks only detect errors of odd numbers of bits. We can improve on this considerably by viewing the characters as though arranged as a 2-d block, and generating a parity character for the various bit positions - vertical redundancy check (VRC), combined with longitudinal redundancy check (LRC), detects most, but not all patterns of even numbers of errors.

Cyclic Redundancy Checks (CRC's)

Given a k-bit frame, the transmitter generates an n-bit frame check sequence (FCS), so that the resulting k+n bit frame is exactly divisible by some predetermined number.

Let: (using modulo-2 arith, ie no carries)

T = $(k+n)$-bit frame to be transmitted ($n < k$)
M = k-bit message (first k-bits of T)
F = n-bit FCS, the last n bits of T
P = pattern of $n+1$ bits (predetermined divisor)

Obviously $T = 2^n M + F$

Now: $T/P = (2^n M + F)/P$
$= (2^n \cdot M/P) + F/P$
$= (Q + R/P) + F/P$

We want to eliminate the remainder term $(R+F)/P$.
Using modulo-2 arithmetic, we can just set $F = R$.
So: $F = 2^n \cdot M/P - Q$ (ie remainder of $2^n M/P$).

The exact pattern P chosen depends on the type of errors expected. At minimum, both the high- and low-order bits of P must be 1.
An error results in the reversal of a bit; thus:

$$T_r = T + E$$

where: T = transmitted frame
T_r = received frame
E = error pattern with 1's in positions where errors occur.

The receiver will fail to detect an error iff T_r is divisible by P, ie iff E is divisible by P.

We can express a binary number $b_k \ldots b_1 b_0$ as a polynomial $F(x) = b_k X^k + \ldots + b_1 X + b_0$
An error $E(x)$ will only be undetectable if it is divisible by $P(x)$. All of the following are detectable:

(a) all single-bit errors

(b) all double bit errors, as long as $P(X)$ has a factor with at least three terms

(c) any odd number of errors, as long as $P(X)$ contains a factor $(X+1)$

(d) any burst error for which the length of the burst is less than the length of the FCS

(e) most larger burst errors

Four versions of $P(X)$ are widely used:

$CRC\text{-}12 = X^{12} + X^{11} + X^3 + X^2 + 1$ (used for 6-bit transmission)

$CRC\text{-}16 = X^{16} + X^{15} + X^2 + 1$ } (used for 8-bit transmission)

$CRC\text{-}CCITT = X^{16} + X^{12} + X^5 + 1$ }

$CRC\text{-}32 = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$

The CRC process can be implemented with shift registers and exclusive-or gates very easily.

4.3 INTERFACING

We now examine layer 1 of OSI, the physical layer, or interface between the data terminal equipment (DTE) and data circuit-terminating equipment (DCE). The three most important standards are:

- RS-232-C
- RS-449/422-A/423-A
- X.21

RS-232-C

This uses a 25-pin connector, with a voltage more negative than -3V interpreted as binary 1 and a voltage more positive than +3V interpreted as binary 0. The signal rate is rated at < 20 kbps and < 15 m range.

TABLE 4-1 RS-232-C Circuit Definitions

Name	Direction to:	Function
Data Signals		
Transmitted Data (BA)	DCE	Data generated by DTE
Received Data (BB)	DTE	Data received by DTE
Control Signals		
Request to Send (CA)	DCE	DTE wishes to transmit
Clear to Send (CB)	DTE	DCE is ready to transmit, response to request to send
Data Set Ready (CC)	DTE	DCE is ready to operate
Data Terminal Ready (CD)	DCE	DTE is ready to operate
Ring Indicator (CE)	DTE	Indicates that DCE is receiving a ringing signal on the communication channel
Carrier Detect (CF)	DTE	Indicates that DCE is receiving a carrier signal
Signal Quality Detector (CG)	DTE	Asserted when there is reason to believe there is an error in the received data
Data Signal Rate Selector (CH)	DCE	Asserted to select the higher of two possible data rates
Data Signal Rate Selector (CI)	DTE	Asserted to select the higher of two possible data rates
Timing signals		
Transmitter Signal Element Timing (DA)	DCE	Clocking signal, transitions to ON and OFF occur at center of each signal element
Transmitter Signal Element Timing (DB)	DTE	Clocking signal, as above, both leads relate to signals on BA
Receiver Signal Element Timing (DD)	DTE	Clocking signal, as above, for circuit BB
Ground		
Protective Ground (AA)	NA	Attached to machine frame and possibly external grounds
Signal Ground (AB)	NA	Establishes common ground reference for all circuits.

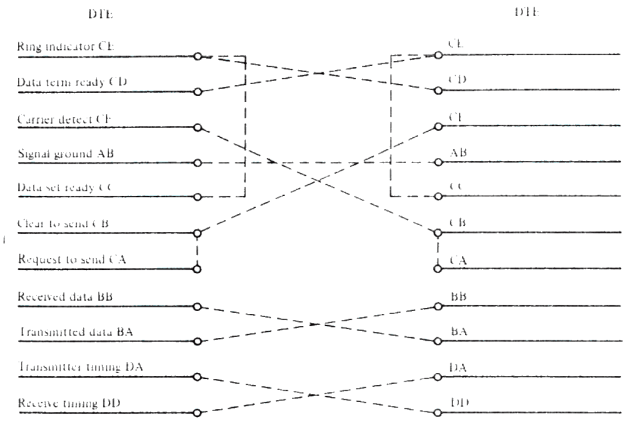


FIGURE 4-10. The null modem.

RS-449/422-A/423-A

RS-232-C has limited distance/speed characteristics, and very little DTE modem control. A newer standard has thus been introduced. RS-449 defines the mechanical, functional and procedural characteristics; RS-422-A and RS-423-A define the electrical characteristics.

Functionally, RS-449 retains all of the interchange circuits of RS-232-C (except protective ground) and adds 10 new circuits

TABLE 4-2 RS-449 and RS-232-C Interchange Circuits

RS-449		RS-232C	
		AA	Protective Ground
SG	Signal Ground	AB	Signal Ground
SC	Send Common		
RC	Receive Common		
IS	Terminal in Service		
IC	Incoming Call	CE	Ring Indicator
TR*	Terminal Ready	CD	Data Terminal Ready
DM*	Data Mode	CC	Data Set Ready
SD*	Send Data	BA	Transmitted Data
RD*	Receive Data	BB	Received Data
TT*	Terminal Timing	DA	Transmitter Signal Element Timing (DTE source)
ST*	Send Timing	DB	Transmitter Signal Element Timing (DCE Source)
RT*	Receive Timing	DD	Receiver Signal Element Timing
RS*	Request to Send	CA	Request to Send
CS*	Clear to Send	CB	Clear to Send
RR*	Receiver Ready	CF	Received Line Signal Detector
SQ	Signal Quality	CG	Signal Quality Detector
NS	New Signal		
SF	Select Frequency		
SR	Signaling Rate Selector	CH	Data Signal Rate Selector (DTE source)
SI	Signaling Rate Indicator	CI	Data Signal Rate Selector (DCE source)
SSD	Secondary Send Data	SBA	Secondary Transmitted Data
SRD	Secondary Receive Data	SBB	Secondary Received Data
SRS	Secondary Request to Send	SCA	Secondary Request to Send
SCS	Secondary Clear to Send	SCB	Secondary Clear to Send
SRR	Secondary Receiver Ready	SCF	Secondary Received Line Signal Detector
LL	Local Loopback		
RL	Remote Loopback		
TM	Test Mode		
			Pins 9 and 10 Test Function
SS	Select Standby		
SB	Standby Indicator		

Category I circuits

- Terminal In Service (IS) indicates to the DCE that the DTE is operational and available to answer calls
- New Signal (NS) tells DCE to prepare to aquire a new line signal, improving overall response time on a polling line.
- Select frequency (SF) allows selection of the DCE frequency mode in a multipoint configuration.
- Local loopback (LL)