G22.3023 Special Topics of Industrial Interest
Network Design and Implementation
Lecture Notes Spring 84
with revisions from
G22.2263 Network Design and Implementation
Lecture Notes Spring 91

Herbert J. Bernstein
Max Goldstein

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

251 Mercer Street
New York, New York 10012

Note: This is a resetting of this document in LaTex by the first author in December 2010 to make a reference copy available on the web.

Acknowledgement

Frances C. Bernstein spent many hours reading and rereading these notes and our "Data Communications" notes. Without her efforts the number of errors in spelling, grammar and style would have been far greater than that which remains. Our sincere thanks.

Contents

Chapter 1

Introductory Material

This is a one semester course on the tools and techniques for computer to computer networking. It is complementary to, but not dependent on G22.2262, Data Communications. We will consider network topology, traffic management, local networks, long distance networks, heterogeneous interconnected networks, typical network designs (e.g. CCITT X.25, ARPANET, DECNET, SNA, ETHERNET), satellite systems, datagrams and virtual circuits, data base and file transfer subsystems, security, hardware and software constraints.

The required reading for this course consist of these lecture notes and

Tanenbaum, Andrew S., "Computer Networks," Prentice-Hall, Inc., Englewood Cliffs, N.J. 1981, 517 pp., ISBN 0-13-165183-8. CR 22, 8 (Aug 81) #38,255 and CR 22,5 (May 81) #37,826.

Depending on your background, additional reading may be necessary in order to understand the material presented, to do the homework and to do the term project. If you have no prior exposure to data communications concepts, you may find it helpful to read the lecture notes, "G22.2262 Data Communications – Lecture Notes – Fall 1983" and the references listed therein.

There will be regular homework, a term project and a final. While homework will be accepted late, lateness will have a definite negative effect on your grade. The term project is due on the last day of classes and will not be accepted late.

Since this semester has only 13 Mondays, and one of them April 16, conflicts with Passover, the final will be a take-home final, and there will be a lecture during the scheduled final period.

Organization of the Material

We will begin with an overview of the subject of networks, defining our terms and drawing examples from common experience. We will review some of the history of the subject. Then we will consider two tools used in network design: queuing theory and graph theory. Queuing theory will allow us to optimize a network for minimal traffic delay times. Graph theory will allow us to allocate traffic among paths to make maximal use of capacity and to move as much data as possible. We will consider the conflicting interactions of these two approaches.

Once we have explored the basic tools needed for network design, we will look at the questions of routing and congestion control raised in practical network implementations. We will look in some detail at ARPANET, DECNET, CCITT X.25 and IBM's SNA. Finally we will look at the questions raised in internetting and in local area networks.

1.1 Overview

Computer networks consist of autonomous nodes able to exchange information via some medium. When the medium is private to two communicating parties, we face the classic problems of data communications: how to compensate for the rate limitations and errors of the medium. We may have to resolve some media sharing questions, but with only two parties we can accept solutions, e.g. master-slave relationships, that would prove cumbersome in more general cases.

In this course we assume that adequate techniques are available to reduce media errors to acceptable levels, and address the problems raised by many parties sharing common communications facilities. To see the nature of these problems, let us consider some examples.

People as an Example of Autonomous Communicating Devices

If one seeks autonomous communicating devices, the obvious choice is people. People gather in small groups and converse by sound, sight and feel. People at moderate distances can still communicate directly by sound or sight, but as distances increase other media and methods become useful. Runners, drum relays, heliographs, flag signals, carrier pigeons, postal systems, billboards, telegraphs, telephones, radio and television all have been of value.

Direct Conversation

First consider a group of people in direct conversation. Anything any one of them says is heard by the speaker and all the others at essentially the same time. If more than one person attempts to speak at once, words will be lost. Either one person must have

the authority to distribute access to the common medium, or the people involved must cooperate, keep their messages short and take turns speaking. In the case of cooperation, if two people accidentally start speaking at the same time, both will know there has been a collision and stop on the first word or so. One or the other will then defer his thought until the other has spoken. This is the model for local carrier sense multiple access networks such as Ethernet.

Another common pattern in direct conversation is the formation of subconversations. In this case subsets of the original group tune out the uninteresting conversations, as at a cocktail party, and maintain ties only within each subset. This is a generalization of the idea of virtual circuits among nodes of a network.

Conversation at a Distance

As distance increases, the limited range and increasing propagation delay of sound become important factors. This can be seen on a military parade ground, where communication becomes hierachically structured. Complex commands are broken down into short messages, which are relayed down the levels of organization. Each commander need only communicate with his subordinate commanders. Yet very large groups are able to act as a single entity.

At still greater distances – still assuming sound as the medium – one might introduce runners to go from unit to unit and to headquarters. Each unit may well act with great autonomy, yet the runners allow coordination of their actions. By relaying runners from unit to unit and spawning several outgoing runners for one incoming runner, it is possible to manage hundreds of thousands of people in some concerted effort, albeit slowly.

Substituting telegraph, telephone, radio or light for sound reduces the propagation delay, so greater distances can be handled, but the concepts are similar. With long transit times, communications may require small message sizes, a hierarchical structure, message forwarding, and replication of messages to reach multiple destinations. The problems are also similar. Messages may get lost or back up at relay points. A path to a relay point or receiver may cease to function. Multiple paths to the same destination may cause message duplication or reordering. We will encounter these concepts in computer networks.

1.2 History

The history of networking consists of constant reinvention of the wheel. It is remarkable to read many early papers on the subject which cite only the early reports of the authors of the paper. The result is that it is now difficult to give proper credit to the originators of the concepts we now use, but one feels a moral obligation to try. Corrections are most welcome.

1.2.1 SAGE to ARPANET

The first major long distance computer to computer network was the Semiautomatic Ground Environment system (SAGE) established as an air defense network in 1958. Considerable networking research was spawned by the military's need for reliable and secure communications. From 1962 to 1964, P. Baran of the Rand corporation developed the basics of packet switching. Largely under the guidance of L. G. Roberts, the Department of Defense Advanced Research Projects Agency built on Baran's work and on the lessons of another DoD network, AUTODIN (Automated Digital Information Network), to design ARPANET, which used a homogeneous network of Interface Message Processors (IMPs) to allow networking among heterogeneous computers.

1.2.2 DECNET, SNA and CCITT X.25

In 1973, DEC introduced its data link protocol, Digital Data Communications Message Protocol (DDCMP), which it expanded to a full network architecture, Digital Network Architecture (DNA) in 1975. IBM introduced its Systems Network Architecture (SNA) in 1974 as a computer to terminal protocol and upgraded it for computer to computer networking in 1976. These commercial vendor networks provoked many imitations and work on an international standard, which became CCITT X.25. The European PTTs based national networks on this standard. Telenet, a commercial derivative of ARPANET, and Tymnet, a network for timesharing, adapted to it.

1.2.3 Local Area Networks

Local networking may be considered as a natural evolution from the modular construction of computers or as a specialization of long distance networking. The most viable direction seems to flow from R.M. Metcalfe and D.R. Boggs Ethernet design of 1976. In 1980, DEC, Intel and Xerox announced a joint commitment to local networking via Ethernet. (It should be mentioned that Ethernet owes much to the ideas developed in the ALOHA radio packet broadcasting system started by N. Abramson and others at the University of Hawaii in 1970). The major competition for local area networking comes from the broadband networks, such as WANGNET, and IBM's token passing networks.

Much has been made of the merits of Ethernet versus token passing rings versus empty slot rings. In 1975 design work was started on the Cambridge Distributed Computing System. In their 1982 book on the subject R.M. Needham and A.J. Herbert note that "All three types of local network have energetic and variously monied advocates; for the purposes of applications such as the Cambridge Distributed System (though not for all imaginable purposes), they are almost indistinguishable in practice." (R. M. Needham & A. J. Herbert, "The Cambridge Distributed Computing System", Addison-Wesley, London, 1982, 170 pp.) It should be noted that in tightly coupled systems of multiple CPUs within a few feet of each other, there are many options for network interconnections which would

be impractical at greater distances. Shared memories, busses, daisy chains, rings and crosspoint switching networks have all been used effectively.

Despite this progress, to date most networking has been for simple batch and time-sharing terminal support rather than for complex resource sharing. Costs and software sophistication are just now reaching the point at which the promise of networking may at last be realized.

1.3 The Problems

As we said above, a computer network consists of autonomous nodes connected by some medium. It might seem that we are excluding consideration of the most popular use of networking: connecting terminals to a computer. If one views the terminal as a simple slave of the computer, our interest vanishes. However, if one views the terminal and its user as an entity, we certainly have an autonomous node, and if one is forced to introduce intermediate nodes to forward messages from terminals with some discretion at the nodes, we also have autonomy. Thus our topic will include the problems that arise from the attempts of autonomous cooperating processes to communicate. It is convenient to provide some structure to these problems by adopting the ISO open systems interconnection model (OSI) of seven layers:

1.3.1 The ISO OSI Layers

7. The Application Layer
6. The Presentation Layer
5. The Session Layer
4. The Transport Layer
3. The Network Layer
2. The Data Link Layer
1. The Physical Layer

Each layer is expected to use the ones below to achieve a virtual link among peer processes on that layer. As one might expect from the course title, we will be most concerned with the Network Layer and its relationship to the other layers, though we will touch on all of them.

The Physical Layer is concerned with the actual transmission of raw signals from device to device. The Data Link Layer engages in whatever error checking and correction is necessary to make available to the Network Layer an apparently error-free path among message switching nodes. The Network Layer takes messages and manages the network of intermediate nodes so that each message presented to it is moved a hop at a time to

its ultimate destination. The Transport Layer uses that facility to provide the higher layers with end-to-end connections which will transmit streams of messages in the order given. The Session Layer provides the user with the necessary resources of the lower layers and accounts for them. The Presentation Layer provides such services as encryption, compression, files transfer and data bases. The Application Layer handles the actual user application.

We will not concern ourselves with the details of error detection and correction. That is for a data communications course. However, it must be remembered that networks must handle errors gracefully, and one can never assume the simple determinism of local computing. Thus we will pay much attention to problems of lost and duplicated messages, deadlocks and resource contention. Links ceasing to function or becoming unexpectedly heavily loaded, will make dynamic message routing a serious problem. Hosts refusing to accept data will make flow control a concern.

Since the resources involved are still quite expensive, we will have to consider ways to minmize costs and make the best use of available capacity.

To handle these problems, we will need some tools from graph theory, queuing theory and other areas.

1.4 Remedial Queuing Theory

One of the essential tools in network design is queuing theory, which may be viewed as a subfield of the study of "stochastic processes" (see, for example, S. Karlin, "A First Course in Stochastic Processes", Academic Press, New York, 1969, 502 pp.). Stochastic processes are those governed by the behavior of some family of random variables, R(t), where t is an index variable, often representing time. For a particular value, t0, of t, we do not necessarily have a unique value R(t0). Rather, we know the probabilities that R(t0) will assume particular values. By probability, we mean the limiting value over infinitely many trials of the ratio of the number of times R(t0) assumes the desired values to the total number of trials.

1.4.1 Customers and Servers

The model we use in dealing with queues considers customers and servers. We have a supply of customers requiring service. We have servers to provide that service. We do not know when any particular customer may arrive, nor do we know how long a server will take to service a particular customer. We only know the general pattern of arrival of customers and provision of service. For example, we might know that customers arrive at an overall average rate, rc, with no dependence between the arrival of any two customers, and that servers serve at an overall average rate rs, with no dependence between the service time for any two customers.

Such customer and server random variables are said to obey Poisson distributions, with a mean interarrival time of customers of 1/rc, and a mean service time of 1/rs. If there are m servers, such a model is called an M/M/m system, where M stands for Markov, since Markov chains are an effective tool to study such systems.

1.4.2 Markov Chains

A Markov chain is a special case of a Markov process. A Markov process is a stochastic process in which future behavior is completely determined by the behavior at the present moment rather than by memory of past behavior. A Markov chain is a Markov process in which time is considered only in discrete steps and in which there are only finitely many or at worst countably many possible states of the system. The system can then be analysed by considering the matrix of transition probabilities from state to state. We will return to Markov chains in more detail latter.

In general, if the customer random variables obey a rule X, and the server random variables obey a rule Y, and there are m servers, then the queue model is called an X/Y/m system.

For most networking problems, an M/M/1 system model is adequate, and we will consider only those.

1.4.3 Poisson Distribution

Let us derive the behavior of a Poisson distribution. We assume that events arrive in any time interval at the same rate that they arrive in any other time interval, and we assume that by taking a sufficiently small time interval we can limit our attention to the arrival of no events or one event. It would be reasonable to say that for a small time interval, dt, r*dt events are likely to arrive, where r is the rate of event arrivals. Equivalently, we could say that the probability, P1(dt), of an event arriving in an interval of small length dt is approximately r*dt, and that the probability, P0(dt), of no events arriving is approximately 1-r*dt.

For larger intervals it is reasonable to consider the probabilities of more than one event arriving. Let us generalize our notation to Pk(t) = the probability of exactly k events arriving in an interval of length t and derive Pk(t) by examining its derivative:

$$\frac{d\,Pk(t)}{dt} = \lim_{dt \to 0} \frac{Pk(t+dt) - Pk(t)}{dt}$$

```
                Pk(t)*P0(dt)+P[k-1](t)*P1(dt)-Pk(t)
      = lim     ------------------------------------
                                dt

                Pk(t)*(1-r*dt-1) + P[k-1](t)*r*dt
      = lim     ---------------------------------
                                dt

     = r*(P[k-1](t) - Pk(t))
```

In particular, we have dP0(t)/dt = -r*P0(t), so that P0(t) must be exp(-rt+const).

Since the probability of no events arriving in an interval of length 0 is one, the "const" must be absent, so

```
P0(t) = exp(-r*t),
```

and

```
d(P1(t))/dt = r*(exp(-r*t) - P1(t))
```

whence

```
d(P1(t)*exp(r*t))/dt = r - r*P1(t)*exp(r*t) + r*P1(t)*exp(r*t)

     = r
```

and thus

```
P1(t)*exp(r*t) = r*t + const.
```

Since P1(0) = 0, it is clear that "const" = 0, and

```
P1(t) = r*t*exp(-r*t)
```

Similarly, we see

```
d(Pk(t)*exp(r*t))/dt = r*P[k-1](t)*exp(r*t)
```

and

```
              k  -r*t
Pk(t) = (r*t) *e    /k!
```

by induction.

Expected Number of Events per Unit Time

Let us use this result to compute the expected number of events per unit time:

```
        infinity
        _____
        \
         \
E(t) =   /   k*P (t)
        /____   k
        k = 0

        infinity
        _____
        \
         \          k  -r*t
      =   /   k*(r*t) *e    /k!
        /____
        k = 1
                  infinity
```

```
                -----
                \
          -r*t  \         k-1
= r*t*e    * /   (r*t)   /(k-1)!
                /____
                k = 1


          -r*t  r*t
= r*t*e    *e
```

Thus r, the instantaneous rate, is also the long term rate of event arrivals.

1.4.4 M/M/1 Queue

Now let us consider an M/M/1 queue. Customers arrive according to a Poisson distribution with a rate rc and are served according to a Poisson distribution with rate rs. Let Q(k,n,t) be the probability that a queue of length n becomes a queue of length k in a time interval of length t. Again we compute the derivatives of the probabilities.

First we need the value of Q at a slightly perturbed time:

```
Q(k,n,t+dt) = Q(k,n,t)*Q(k,k,dt) + Q(k+1,n,t)*Q(k,k+1,dt)
              + Q(k-1,n,t)*Q(k,k-1,dt),
```

since, under our assumptions for Poisson distributions, we need only consider changes due to at most one arrival and/or service in a small time interval. Now, if we consider only $k > 0$, we have

```
Q(k,k,dt) = P(no customers \& no sevices)
            + P(1 customer \& 1 service)
          = ~ (1-rc*dt)*(1-rs*dt) + rc*dt*rs*dt
          = ~ 1 - (rc+rs)*dt,
```

and

```
Q(k,k+1,dt) = P(no customers & 1 service)
          = ~ (1-rc*dt)*rs*dt = ~ rs*dt,
```

and

```
Q(k,k-1,dt) = P(1 customer & no service)
           = ~ rc*dt*(1-rs*dt) = ~ rc*dt,
```

so that

```
Q(k,n,t+dt) = ~ Q(k,n,t)
              + dt*(-(rc+rs)*Q(k,n,t) + rs*Q(k+1,n,t)
                                + rc*Q(k-1,n,t)),
```

whence

```
d Q(k,n,t)
---------- = -(rc+rs)*Q(k,n,t) + rs*Q(k+1,n,t) + rc*Q(k-1,n,t).
    dt
```

For the case $k = 0$, the probability of no service = 1 in a short time interval, so that $Q(0, 0, dt) = 1 - rc * dt$. Further, since queues cannot go negative, Q(0,-1,dt)=0.

Thus

```
Q(0,n,t+dt) = ~ Q(0,n,t) + dt*(-rc*Q(0,n,t) + rs*Q(1,n,t)),
```

whence

```
d Q(0,n,t)
---------- = -rc*Q(0,n,t) + rs*Q(1,n,t).
    dt
```

If we note that Q(k,n,0) is zero for $k \neq n$, and one for $k = n$, we can solve this system of differential difference equations for small time. However, we are usually interested in the behavior over large time.

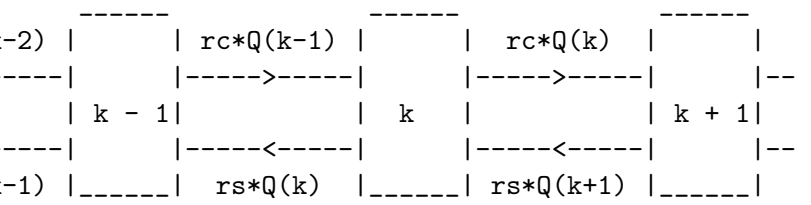
If the queuing system achieves "statistical equilibrium", the initial queue length will no longer matter and the probability of arriving at any particular queue length will become independent of time. Write Q(k) for this long term probability, and we see that

```
(rc+rs)*Q(k) = rs*Q(k+1) + rc*Q(k-1), for k > 0,
```

and

```
rc*Q(0) = rs*Q(1).
```

To understand these equations, consider a queue of length k, k¿0, as a state. We can enter that state from above by a service, and from below by the arrival of a customer. We can leave that state for the next state above by the arrival of a customer, or leave for the next state below by a service. In order to preserve the relative populations of the possible states, the flows into and out of each must balance:



which are precisely the equations we obtained above for k¿0. For k = 0, there are no flows to or from lower states, whence rc*Q(0) = rs*Q(1). A system of states with this property of transitions only to and from the states immediately above and below is called a “birth-death” system.

We can solve for Q(k) by induction, starting with

```
rc*Q(0) = rs*Q(1),
```

to remove Q(1) from the next equation:

```
(rc+rs)*Q(1) = rs*Q(2) + rc*Q(0)
```

implies

```
          rc             rc
Q(2) = (-- + 1)*Q(1) - --*Q(0)
          rs             rs

          rc      rc        rc
       = (-- + 1)*--*Q(0) - --*Q(0)  = (rc/rs)**2 *Q(0),
          rs      rs        rs
```

and, in the general case, by induction,

```
Q(k) = (rc/rs)**k *Q(0).
```

Since

```
     -----       -----
     \           \
      \           \   / rc \ n
1 =   /   Q(n) =  /  |  --  | * Q(0)
     /____       /____\ rs /

        1
= ----------- * Q(0)
  1 - (rc/rs)
```

which implies

```
Q(0) = 1- (rc/rs),
```

and

```
Q(k) = (1 - (rc/rs)) * (rc/rs)**k.
```

The expected queue length is then

```
     -----                -----
     \                    \
      \                rc   \    rc n
L =    / n*Q(n) = (1 - --)* / n*(--)
     /____              rs  /____ rs

  = (1 - (rc/rs)) * (rc/rs) / (1 - (rc/rs))**2

  = (rc/rs) / (1 - rc/rs).
```

From this result, we see that, for an M/M/1 queue, the service rate must be significantly faster than the customer arrival rate, if the queue length is to be kept small. We can see this again by computing the expected waiting time.

1.4.5 Expected Waiting Time

If a customer arrives and finds the queue at length n, he can expect to wait the service time of those already waiting plus his own service time until he is done. Thus the waiting time, W, is

```
    n + 1
W = -----  units of time
     rs
```

on the average. It follows that the expected waiting time is

```
  1    /  rc/rs       \
---- *| ---------- + 1 |
 rs    \1 - rc/rs     /


    1    /    1     \
  = ---- *| ---------- |
    rs    \ 1 - rc/rs/

  = 1/(rs - rc).
```

From this we see that equal service rates and customer arrival rates in an M/M/1 queue produce infinite delays, and the queue cannot be stable if rs ¡= rc. If we wish a queue of expected length 1 and an expected delay time equal to the expected time between customers, we must make the service rate twice the customer arrival rate.

The probability that the long term queue will be of a length greater than, say, m, is

```
infinity
 -----
 \
  \
  /  Q(k)
 /____
 k = m+1
             infinity
             -----
   /      \  \
   |     rc |  \        k
= | 1 - -- |  / (rc/rs)
   |     rs | /____
    \      /  k = m+1

= (1 - rc/rs) * (rc/rs)**(m+1) / (1 - rc/rs)

= (rc/rs)**(m+1)
```

For example, if rc/rs = .5, the probability of having to cope with a queue length of at least 10 is less than .001, while, if rc/rs = .99, the probability of having to cope with a queue of at least 10 is greater than 9/10. This is consistent with the expected queue lengths of 1 and 99 respectively.

1.4.6 Little's Formula

While the assumption of M/M/1 queues works well in practice for networks, it would be reassuring not to require Poisson distributions. D.C. Little ("A proof of the queuing formula L=lambda*W", Operations Research 9, 1961, 383-387) proved the intuitively reasonable result that the expected length of a queue equals the rate of arrival of customers multiplied by the expected delay time for a customer, i.e. during the delay time, W, with r customers per unit time arriving, r*W customers would arrive, while by definition, a customer at

the end of the queue of length, L, would have gotten to the front and been served. If the expected queue length, L, is stable, then clearly we must have r*W = L. Equivalently, we can express the expected waiting time as

```
W = L/r,
```

which can be used to confirm the M/M/1 delay time estimate. Inasmuch as we already have this result, let us make better use of Little's formula.

Consider any set of queues, of expected lengths, Li, arrival rates, ri, and expected delay times, Wi. By Little's formula,

```
Wi = Li/ri.
```

Now define a new queue as follows. The customers for this queue consist of the combined set of all customers for the individual queues. A customer will be considered to have been served by this combined queue when he has been served by his appropriate individual queue. The expected size of this total queue is just the sum of the individual Li, and the average rate of arrival of customers is just the sum of the individual ri.

Thus the expected delay for a customer of the overall queue is

```
                                      ---
                                      \
                                      /__ ri*Wi
W = summation(Li)/summation(ri)  = ---------
                                      ---
                                      \
                                      /__ ri
```

This is intuitively reasonable, since we are simply saying that the expected delay per queue is the average of the individual queue delays weighted by the customer traffic for the queues.

We can now apply this to M/M/1 queues. The delay for the ith queue is

```
Wi = 1/(ui - ri),
```

where ui is the rate of queue services and ri is the rate of queue customer arrivals. In this case, the average delay is

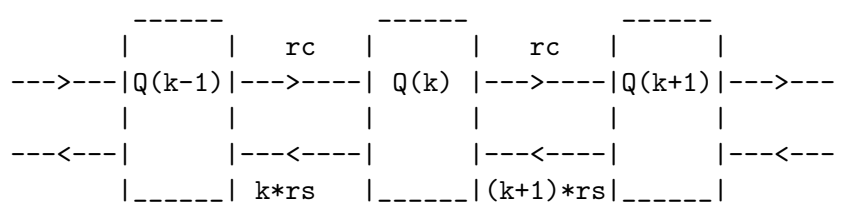
$$W = \frac{\sum_{} ri/(ui - ri)}{\sum_{} ri}$$

1.4.7 Multiple Servers

In the design of networks, the problem of combining multiple servers arises. If we assume one supply of customers at rate rc, and a set of m servers each serving at rate rs, then, if the long term probability of achieving a total queue length k is called Q(k), we can relate these probabilities by:

```
Q(0)*rc = Q(1)*rs
Q(1)*rc + Q(1)*rs = Q(0)*rc + Q(2)*2*rs
Q(2)*rc + Q(2)*2*rs = Q(1)*rc + Q(3)*3*rs
   .
   .
Q(k)*rc + Q(k)*k*rs = Q(k-1)*rc + Q(k+1)*(k+1)*rs
   .
   .
Q(m)*rc + Q(m)*m*rs = Q(m-1)*rc + Q(m+1)*m*rs
   .
   .
Q(n)*rc + Q(n)*m*rs = Q(n-1)*rc + Q(n+1)*m*rs
```

by matching “flows” among the states, and noting that we cannot have a service in an unoccupied queue.



It follows that

```
Q(1) = (rc/rs)*Q(0)
Q(2) = (rc/rs)*Q(1)/2
Q(3) = (rc/rs)*Q(2)/3
   .
   .
Q(k) = (rc/rs)*Q(k-1)/k
   .
   .
Q(m) = (rc/rs)*Q(m-1)/m
   .
   .
Q(n) = (rc/rs)*Q(n-1)/m
```

This implies that

```
Q(k) = (rc/rs)**k*Q(0)/k!, k = 1,..m,
Q(n) = (rc/(m*rs))**n * Q(0) * m**m /m!
```

This result is very similar to the one that we obtained for an M/M/1 queue for the longer queue lengths, if we consider an m*rs rate server, but is quite different for the shorter queue lengths.

We can obtain the actual probabilities by summing Q(0) + Q(1) + ... to 1, but in order compute expected queue lengths, it is sufficient to leave the probabilites in this form to compute

```
E = 0*Q(0) + 1*Q(1) + 2*Q(2) + .. + k*Q(k) + ...

  = Q(0) * ( 1*Q(1)/Q(0) + 2*Q(2)/Q(0) + .. k*Q(k)/Q(0) + ... )
```

$$= \frac{\sum_{k=1}^{m} \frac{1}{(k-1)!} \left(\frac{rc}{rs}\right)^k + m \sum_{k=m+1}^{\infty} \frac{1}{m!} \left(\frac{rc}{m \cdot rs}\right)^k}{\sum_{k=0}^{m-1} \frac{1}{k!} \left(\frac{rc}{rs}\right)^k + m \sum_{k=m}^{\infty} \frac{1}{m!} \left(\frac{rc}{m \cdot rs}\right)^k}$$

which may be evaluated in closed form by noting that the infinite sum in the denominator is just a geometric series, and that the infinite sum in the numerator is the derivative of a geometric series.

However, in many cases, it is a sufficiently accurate approximation to consider the case where m goes to infinity. This leaves us with a numerator which is (rc/rs) times an exponential and a denominator in which the same exponential appears. Thus, in the limit, the expected queue length for such a queuing system is just the ratio of the rate of arrival of customers to the rate of services by a single server.

To see how realistic such an approximation can be, consider the case where rc = 10 and rs = 1. While the queue length is infinite for 10 servers, it stabilizes to a queue length of 10.000000000075 with 11 servers, and is effectively an exact 10 with 13 servers, giving an average delay per customer of one service time. One should not think this a full substitute for a single, faster server, since use of a single server with service rate rs = 12, would give an expected queue length of only 5, i.e. half the service delay of the multiple server queue. On the other hand, when reliability considerations lead one to use of, say, two half speed servers, rather than one full speed server, the delay penalty can be rather small. Say rc = rs = 1, and m = 2, then E = 4/3; while for rc = 1, rs = 2, m = 1, we have E = 1.

Binomial Distribution Queue

It would seem that we have made a serious mistake in our choice of models thus far. Many, if not most realistic communications problems are based on discrete time, not on the continuous time model we have used. Typically, bits arrive, if at all, at some fixed rate. In a given bit time, at most one bit can arrive, not several as in a Poisson distribution. We will now consider the effect of using discrete time intervals and show that the continuous time models are a good approximation for large time.

Suppose time is counted in integral multiples, t = n*t0, of some minimal time unit, t0, in which a single event may or may not occur. If the short-term rate of arrival of events is r, then the probability, p, of the arrival of an event in the interval t0 is r*t0 = p. The probability, Bk(t), of exactly k events arriving in time t is then governed by the binomial distribution. That is,

```
              n!     k      n-k
Bk(t) = ----------*p *(1-p)
        (n-k)!*k!
```

since the binomial coefficient, n!/((n-k)!*k!), is the number of possible patterns of k time slots selected from among the n slots in time t, and p**k*(1-p)**(n-k) is the probability that events will occur in exactly k selected slots. Let us rewrite Pk(t), the probability of exactly k events governed by a Poisson distribution, in the same terms:

```
              k  -r*t
Pk(t) = (r*t) *e    /k!

                 k  -r*n*t0
      = (r*n*t0) *e       /k!

              k  -p*n
      = (p*n) *e    /k!
```

Thus

```
Bk(t)/Pk(t) =

           n!     k      n-k
         --------*p *(1-p)                                  n
         (n-k)!*k!                 n*(n-1)*...*(n-k+1)*(1-p)
       = --------------------   = --------------------------
           k  k                          k      k  -p*n
          n *p   -p*n                   n *(1-p) *e
          -----*e
           k!


                                                      n
         1*(1-1/n)*(1-2/n)*...*(1-(k-1)/n)) *(1-p)
       = ----------------------------------------
                           k  -p*n
                      (1-p) *e
```

Now

```
                n    -z
  lim  (1 - z/n)  = e
n-->inf
```

so, if p*n is bounded as n goes to infinity, we can substitute

```
              n          n      -pn
  (1 - (pn/n))  = (1 - p)  for e
```

and the ratio of Bk(t) to Pk(t) consists of terms all going to 1 and itself goes to one. Thus, for small p = r*t0 and large n = t/t0, the binomial distribution is close to the Poisson distribution. In practical terms, this means that we can use the Poisson distribution when r*t0 is small and the times under consideration are long compared to t0.

Consider a queue in which customers arrive according to a binomial distribution at a rate rc and are served according to a binomial distribution at a rate rs. Suppose the minimal time interval, t0, is the same for both distributions. Let C(k,m,t) be the probability that a queue of length m becomes a queue of length k in an interval of length t. As in the analysis of an M/M/1 queue above:

```
C(k,m,t+t0) = C(k,m,t)*C(k,k,t0) + C(k+1,m,t)*C(k,k+1,t0)
                    + C(k-1,m,t)*C(k,k-1,t0)
```

and, for k¿0,

```
C(k,k,t0) = P(no customers & no services) + P(1 customer & 1 service)

          = (1 - rc*t0)*(1 - rs*t0) + rc*rs*t0**2

          = 1 - (rc+rs)*t0 + 2*rc*rs*t0**2,

C(k,k+1,t0) = P(no customers & 1 service)

          = (1 - rc*t0)*rs*t0 = rs*t0 - rc*rs*t0**2

C(k,k-1,t0) = P(1 customer & no services)

          = rc*t0*(1 - rs*t0) = rc*t0 - rc*rs*t0**2
```

while, for k=0, there can be no service in time t0, so that C(0,0,t0) = 1-rc*t0, and, since the queue cannot go negative, C(0,-1,t0) = 0. Thus

```
C(0,m,t+t0) = C(0,m,t)*C(0,0,t0) + C(1,m,t)*C(0,1,t0)

     = C(0,m,t)*(1 - rc*t0) + C(1,m,t)*(rs*t0 - rc*rs*t0**2)
```

We could analyse these difference equations for small time. However, for our applications, the large time case is most interesting, in which the initial queue length has been forgotten. If we let C(k) be the probability of going to a queue length of k over long time, then we obtain the relations:

```
rc*C(0) = (rs - rc*rs*t0)*C(1)

(rc+rs-2*rc*rs*t0)C(k)
        = (rs-rc*rs*t0)*C(k+1) + (rc-rc*rs*t0)*C(k-1)
```

If we define

```
rs' = rs - rc*rs*t0
rc' = rc - rc*rs*t0
```

then

```
rc*C(0) = rs'*C(1)

C(k+1) = (rc'/rs' + 1)*C(k) - (rc'/rs')*C(k-1)
```

which, as with the M/M/m case above, has a solution similar to that for the M/M/1 queue, but with a slight adjustment in the probabilities of the lower queue lengths.

More on Markov Chains

A detailed study of Markov chains is beyond the scope of these notes. However, some familiarity with the basic terms and concepts is desirable.

The simplest problems in probability theory are those in which the events being studied are totally independent. In a series of papers from 1906 to 1924, A.A. Markov investigated the more difficult problems of dependent events. In our queuing examples, we reached each queue state from ones below or above. In general, one could have a system which might be in any of the states drawn from some set, X, and consider the probabilities of transitions over time from any one of those states, x, to any other one of those states, y. The resulting transition probability, p[x,y], is conditional on starting in state x.

If the set of states is finite, and the transition probabilities do not depend on time, then one can find the probability of going from state x to state y through some unknown intermediate state by computing the sum of all p[x,z]*p[z,y] for all possible intermediate states z. Thus we can form the matrix of all two step transitions by forming the product of the matrix of one step transitions with itself. A Markov chain is a process which we can follow in this manner. The approach can be generalized to continuous time Markov chains, such as the M/M/m queues, above, in which time varies continously, but the states are at worst countable, and further generalized to Markov processes, in which the states may also form a continuum.

Summary

In this chapter we have reviewed the basics of queuing theory. We have examined the Poisson distribution and its discrete time analogue, the binomial distribution. We have

looked at the behavior of queues formed by customers arriving according to a Poisson distribution and being served by servers providing service according to another Poisson distribution, and we have seen that infinite queues will develop if the customers arrive just at the service rate.

Exercises

1. Prove that r*t is the expected number of events in a time interval, t = n*t0, in which event arrival is governed by a binomial distribution with short-term rate of event arrival r over minimal time-step t0.
2. A. K. Erlang studied the behavior of telephone systems, and developed the basic formulae used for allocation of sufficient trunk capacity in telephone systems. Derive the Erlang formulae giving the probability of finding all lines busy in a telephone exchange having m outgoing lines, a single queue of incoming lines, and Poisson distributed traffic, under the assumption that all traffic for unavailable lines is lost, instead of queued.
3. What is the expected delay time for a queuing system composed of 100 smaller queuing systems, fifty of which have a customer arrival rate of 10 per second and a expected delay time of 5 seconds each, and 50 of which have a customer arrival rate of 20 per second and expected queue lengths of 1 each?
4. Suppose an M/M/1 queuing system has a customer arrival rate of 5 per second and an expect queue length of 3. What is the service rate?

1.5 Graph Theory for Network Design

1.5.1 Introduction

The tools of graph theory find extensive application in network design. For a grounding in the subject, see R.G. Busacker & T.L. Saaty, "Finite Graphs and Networks: An Introduction with Applications", McGraw-Hill, New York, 1965, 294 pp.

When we draw a picture of a communications network, it is usually as a set of nodes connected by communications lines. If we record the physical locations of the nodes as points in 3-space and the paths of the lines as curves in 3-space, the sets V, of points and E, of curves, constitute a "geometric graph". While there are communications design problems which require exact knowledge of node locations and line paths, as in cabling a room full of equipment, much can be learned from the abstract topology of the graph of a communications network.

The essential features we require are distinct labels for the nodes and lines, and a description of which lines connect which nodes. We will wish to supplement that information with line capacities and costs. For the nodes, define a set V of abstract vertices v[i]. For the lines, define a set, E, of abstract edges e[i], and a mapping f, from E into VxV, the set of pairs of vertices. If we care about the permitted direction of flow of information, then,

if f(e[k]) = (v[i],v[j]), v[i] will be the origin of the flow, and v[j] will be the sink for the flow. If we do not care about the direction, then (v[i],v[j]) will be considered equivalent to (v[j],v[i]).

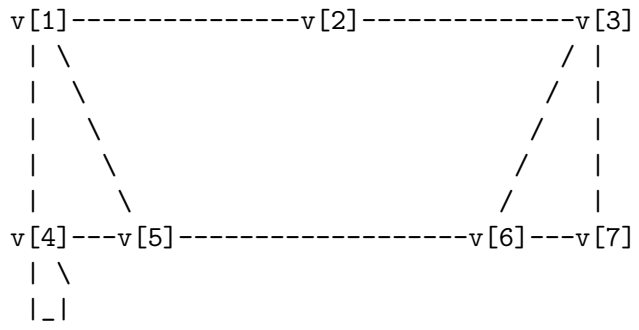
1.5.2 Abstract Undirected Graph

We define an abstract undirected graph, G = $< V, E, f >$, as a set of vertices V, a set of edges E, and a mapping f from E to $(V \times V)/\{(v[i], v[j]) = (v[j], v[i])\}$. Usually we simply call G a graph, and consider f implicit in E. For simplicity, we will confine our attention to undirected graphs at present.

Incidence Matrix

If two edges are incident on the same pair of vertices, those edges are said to be parallel. Since we can treat multiple lines between a pair of nodes as one line of composite characteristics, we forbid the case of parallel edges in the abstract graphs under consideration. With that minimal restriction, we can represent a finite graph by its "incidence matrix", where each row and column corresponds to a vertex, and a non-zero entry, say 1, is made for each edge joining a pair of vertices.

For example, consider the following graph:



which can be represented by:

	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]
v[1]	0	1	0	1	1	0	0
v[2]	1	0	1	0	0	0	0
v[3]	0	1	0	0	0	1	1
v[4]	1	0	0	1	1	0	0
v[5]	1	0	0	1	0	1	0
v[6]	0	0	1	0	1	0	1
v[7]	0	0	1	0	0	1	0

Notice that the only diagonal element which is non-zero is due to the loop from v[4] to itself and that the matrix is symmetric around the main diagonal. (If we had had a directed graph, that symmetry could have been disturbed).

The incidence matrix of a graph can be used to trace paths through sequences of edges. Indeed, the nth power of the incidence matrix contains a non-zero entry for each pair of vertices which can be connected by edge sequences of length n.

Since we are interested in data rates and costs, we may replace the ones in an incidence matrix by some representation of capacity and cost, and use it much the same way. Before doing so, however, we will require some more notation.

Two vertices are called adjacent if some edge joins them. Two edges are called adjacent if some vertex is common to both. A vertex is incident with an edge if it is incident with one of the endpoints of the edge. In that case, we may also say the edge is incident with the vertex.

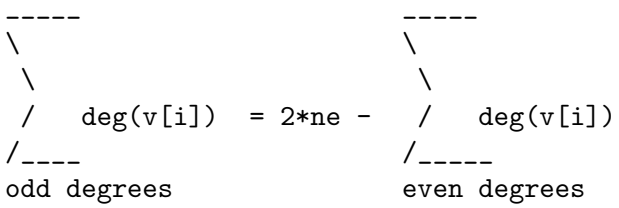
1.5.3 Degree and Euler's Theorem

The number of edges incident with a vertex, v, counting loops twice, is the degree of v, deg(v). An isolated vertex is a vertex of degree zero. A terminal vertex is a vertex of degree one. A graph is regular of degree d if every vertex in the graph is of degree d.

Euler proved that, for a graph G with vertices V = v[i], i = 1,..,nv and edges E = e[i], i = 1,..,ne, the summation of deg(v[i]) = 2*ne. (Proof: each edge adds one to the degree of each of two vertices.)

For a finite graph it follows that the number of vertices of odd degree is even.

Proof:



must be even, since all the terms on the right hand side are even. This implies that there must be an even number of vertices of odd degree, or the left hand side would be odd.

Paths, Circuits, Hamiltonian Circuits

Since we have forbidden parallel edges, there will be no confusion if we write edges, e[k], as the vertex pairs (v[i],v[k]) to which they correspond. A sequence of edges (v[i1],v[j1]), (v[i2],v[j2]), ..., (v[in],v[jn]) is said to form a path of length n if v[jk]=v[i(k+1)], k=1,...,n-1, and v[i1], v[i2], ..., v[in], v[jn] are all distinct. A circuit is similarly defined, except v[jn]=v[i1].

A geodesic is a path of minimal length between two vertices. A graph is connected if there is at least one path connecting every pair of vertices. The diameter of a connected graph is the length of the longest geodesic. A Hamiltonian path is a path which includes every vertex. A Hamiltonian circuit is a circuit which includes every vertex.

A connected graph with no circuits is called a tree. Removal of one edge from a tree must disconnect it, so a tree is, in that sense, a minimal connected graph. Given a finite graph, one can find many trees within it. A tree within a graph which includes all the vertices of the full graph is called a spanning tree. If we assign some cost to each edge, a spanning tree with minimal total cost is called a minimal spanning tree. A spanning tree of a graph need not be unique. A minimal spanning tree of a graph also need not be unique.

Minimal Cost Path

Let us now consider the question of finding a path of minimal total cost between two vertices of a connected graph with costs assigned to its edges. Since loops from vertices to themselves cannot contribute to such a path, we discard them.

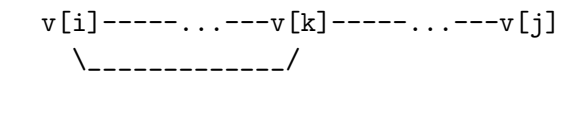
First we will consider a rather inefficient algorithm which has the advantage of being obviously correct. Then we will consider an efficient algorithm due to Dijkstra, which avoids unnecessary work, but which is less obvious.

Form the incidence matrix of the graph, with entries of the form C[i,j] or 0, where C[i,j] is the cost of the edge from vertex v[i] to vertex v[j], and an entry of zero means there is no edge. Now use this matrix to compute a new matrix of least costs of paths of length 2, by forming the minima of the sums C[i,k]+C[k,j] for which both terms exist. Repeat the process until we have cost matrices for all possible paths.

Now a path of minimal cost from vertex v[i] to vertex v[j] must be represented by the least (i,j) entry among these matrices, and, retracing the creation of that entry, we are done.

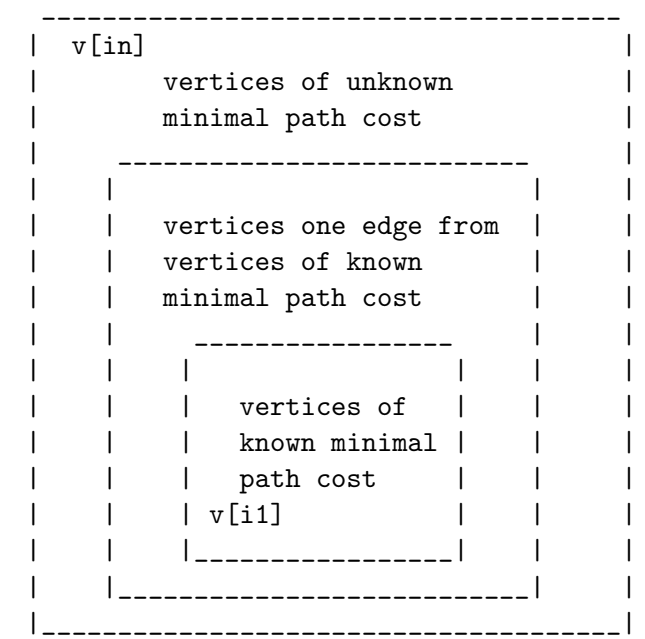
Now that you believe we can find the required path, let us find it more efficiently (E.W. Dijkstra, “A Note on Two Problems in Connexion with Graphs”, Numer. Math. 1, pp

269-271, 1959). To understand this algorithm, one should realize that a path of minimal cost from v[i] to v[j] through v[k] must include a path of minimal cost from v[i] to v[k]. If it did not, we could replace that portion of the original path with a path of lower cost and decrease the total cost of the path.



Thus the inductive step in finding a path of minimal cost between two vertices is to extend previously found paths of minimal cost by oneedge at a time in order of increasing minimal path length.

Let the starting vertex be v[i1] and the ending vertex be v[in]. Mark all vertices other than v[i1] with a tentative total cost of infinity. Mark v[i1] with a permanent cost of zero, and mark its immediate neighbors with tentative costs given by C[i1,j]. Some of these first neighbors will have a minimal cost. We claim that that cost cannot be lowered for those vertices. The only way it could, would be if there was a multiple edge path from v[i1] to such a vertex with lower total cost, but that would imply that the very first edge in that multiple edge path has strictly lower cost than the supposed minimum of the single edge costs – a contradiction. Thus we can permanently mark any (or all) of the first neighbors of minimal cost, noting also the previous vertex on the path.



We will now extend the set of vertices with known minimal path cost from v[i1]. At each stage, we will have tentatively marked a finite path cost to all vertices within one edge of permanently marked vertices. The inductive step is as follows. Suppose we have just marked a vertex with the minimal cost of a path to that vertex, and with the prior vertex in that path. Now, to ensure that we have tentatively marked all vertices within one edge of permanently marked vertices, examine the neighbors of this newly marked vertex. If they are not marked with a finite cost, assign one. If they are already marked with a tentative finite cost, see if the newly introduced vertex allows a lower cost path. If so, lower the tentative cost and make the newly permanently marked vertex the predecessor.

Now find one of the tentatively marked vertices of minimal cost. We claim that its cost cannot be further lowered. If it could, there would be a path of lower cost, say v[i1]–v[i2]–...–v[ik], to this vertex v[ik]. If v[i(k-1)] had been marked permanently earlier, then we would have already known about this lower cost path, a contradiction. If v[i(k-1)] was only tentatively marked, then find the first vertex in this lower cost path which is tentatively marked. Since that vertex is within one edge of a permanently marked vertex, it must have a finite tentative marking, but a finite tentative marking which is strictly less than our claimed minimal tentative marking, again a contradiction.

We continue until v[in] is marked permanently.

Directed Graphs

Now let us turn our attention to directed graphs, so that we can handle data flow problems. An abstract directed graph, G = $< V, E, f >$, consists of a set of vertices V, a set of edges E, and a mapping f from E to $V \times V$. As before, we forbid the case of parallel edges, but, in a directed graph, an edge from v[i] to v[j] is not parallel to an edge from v[j] to v[i], so both may exist. Under this assumption, we may identify an edge by its initial and terminal vertices, and use an incidence matrix to describe the graph.

An edge has positive incidence with its initial vertex and negative incidence with its terminal vertex. The number of edges of positive incidence with a vertex is the positive degree of a vertex. The number of edges of negative incidence with a vertex is its negative degree. Clearly the sum of the positive degrees of the vertices must equal the sum of the negative degrees of the vertices and must also equal the number of edges.

A path in a directed graph is defined as in an undirected graph, but the edge directions must be consistent with one another. A cycle in a directed graph is defined in the same manner as a circuit in an undirected graph, but, again, the edge directions must be consistent.

A directed graph is connected if the equivalent undirected graph is connected. A directed graph is strongly connected if, for every pair of vertices v[i] and v[j], there exists a path from v[i] to v[j] and a path from v[j] to v[i]. A directed graph is strongly k-connected if, for every pair of vertices, v[i] and v[j], there are k distinct paths from v[i] to v[j] which have only v[i] and v[j] in common.

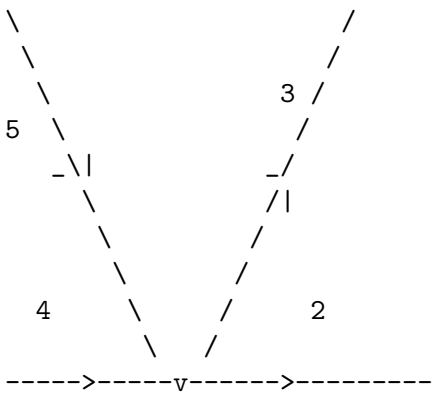
1.5.4 Flows and Networks

In the graph theoretical study of flows, a network is a finite directed graph which is connected and has no loops from vertices to themselves. We will restrict our attention to networks without parallel edges. The major original contributors to the study of flows in networks were L.R. Ford, Jr., D.R. Fulkerson, and G.B. Dantzig.

Consider a network with a limiting flow capacity assigned to each edge. A flow in the graph is an assignment of values to the edges. The value assigned to an edge (v[i],v[j]) indicates a flow from v[i] to v[j] if positive, and a flow from v[j] to v[i] if negative.

A flow is called feasible for the network if it is positive on each edge and no greater than the capacity of the edge.

The total of the flows out from a vertex, v, minus the total of the flows into the vertex is called the net output of v, Q(v). A vertex with positive net flow is called a source. A vertex with negative net flow is called a sink. A vertex with zero net flow, i.e. which conserves flow, is called an intermediate vertex.

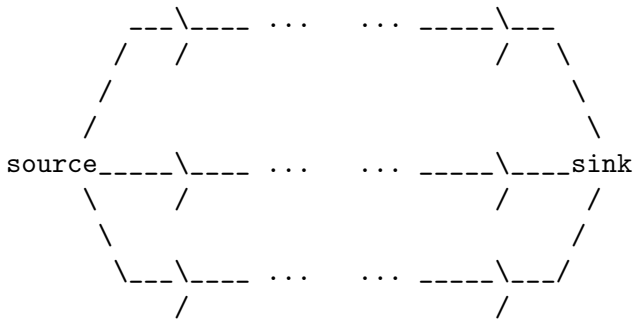


Q(v) = 3+2-5-4 = -4 (a sink)

If there is only one source and only one sink in the network, then the flow is said to be from the source to the sink. By adding new vertices and edges with properly assigned flows, we can change any flow into a flow with just one source and one sink. To do so, we add a new vertex, which will become the single source, with outgoing flows to each of the original sources equal to their net flows. Then we add a second new vertex, which will become the single sink, with incoming flows from each of the original sinks.

If the original flow was feasible and we assign capacities to the new edges equal to the new flow values, the new flow is feasible, has only outgoing flow values from the single source, only incoming flow values to the single sink, and no net flow at the rest of the vertices. Since the total of all the net flows must be zero (each edge makes a balancing contribution to two vertices), the outgoing flow from the source must equal the magnitude

of the incoming flow to the sink. (Notice that Tanenbaum calls these combined, stronger conditions the definition of a feasible flow).



cuts

Let v[1] and v[2] be vertices. A v[1]-v[2] cut is a set of edges, the removal of which disconnects all paths from v[1] to v[2]. A cut is called minimal if restoring any of the edges in the cut will reconnect the vertices in question. The capacity of a cut is the sum of the capacities of the edges in the cut.

We can use judiciously chosen minimal cuts to analyse feasible flows between vertices. It is important to realize that we are only looking at the flow between pairs of vertices, not at some generalized total flow in the network.

Max-Flow-Min-Cut Theorem

If we have a feasible flow in which v[1] is the only source and v[2] is the only sink, we say the flow is from v[1] to v[2]. The magnitude of that flow is defined as Q(v[1]) = -Q(v[2]). The magnitude of the flow cannot exceed the capacity of a v[1]-v[2] cut (a fortiori of a cut of minimal capacity), so the maximal flow from v[1] to v[2] is bounded above by the capacity of a v[1]-v[2] cut of minimal capacity.

Further, if the maximal flow could not reach the capacity of a cut of minimal capacity, then the limitation on flow would be due to a cut of still smaller capacity, a contradiction. Thus we have the "Max-Flow-Min-Cut" Theorem:

The maximal feasible flow from v[1] to v[2] is equal to the capacity of a v[1]-v[2] cut of minimal capacity.

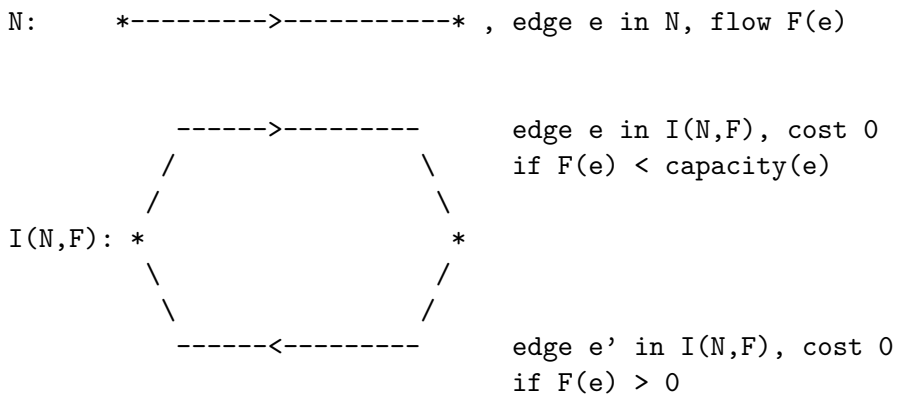
Finding Maximal Feasible Flow

There are many algorithms to find the maximal feasible flow. An alternative to the Malhotra, Kumar, Maheshwari algorithm (Inf. Proc. Letters, vol 7, Oct 1978, pp 277-278),

which relates to minimal cost path finding is given in Busacker & Saaty:

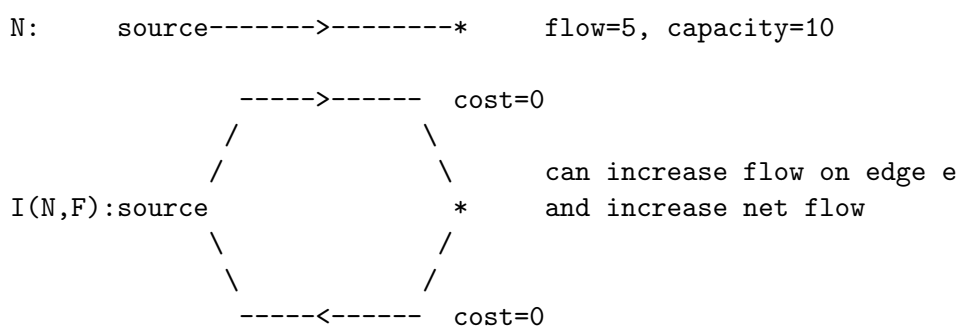
The idea is to start with a reasonable flow and incrementally increase it towards a maximum, with termination guaranteed by restricting flows and capacities to integral values. This restriction is not severe, since we can scale up the capacities to recover any necessary precision.

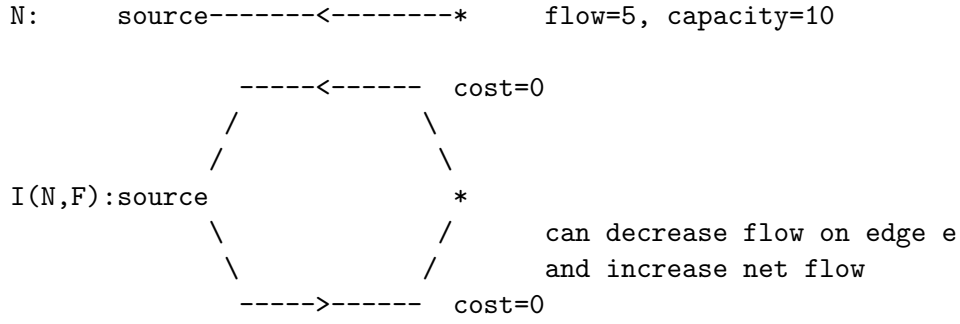
The tool in these incremental flow improvements is the incremental graph, I(N,F), of a network, N, with feasible flow F. I(N,F) has two directed edges for each directed edge of N, using the same set of vertices as in N. If e is an edge of N, use e for the edge in I(N,F) in the same direction, and e' for the edge in I(N,F) in the opposite direction from e in N. Assign a cost to e of zero, if F(e) is less than the capacity of e. Assign a cost to e' of zero, if F(e) is greater than zero. Otherwise assign a cost to e or e' of one.



Thus, if we could add to the flow in an edge of N, and have the flow remain feasible, we would have a cost of zero in the edge of I(N,F) in the same direction, while if we could decrease the flow we would have a cost of zero in the edge of I(N,F) in the opposite direction.

Suppose we have an entire cost zero path from source to sink. If we change the flow in N by one in the indicated direction, it is still feasible. Since the path from the source is outward, the effect of this change is to increase the net flow.



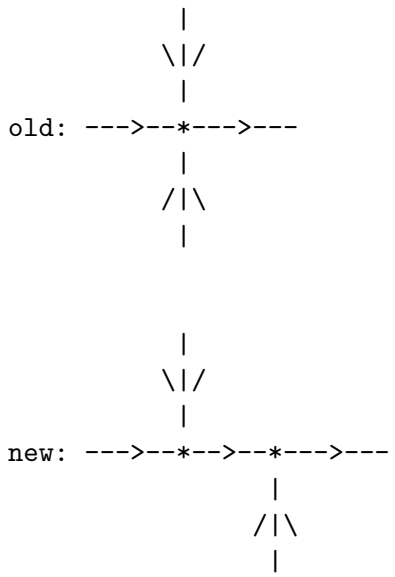


By, say, a directed version of Dijkstra's algorithm, we can find the minimal cost path in I(N,F) from v[1] to v[2]. If the cost of this path is zero, we can increase the net flow along the corresponding, path in N, and start again. If all paths have non-zero cost, the flow is already maximal. Since, when we can increase flow, we can always increase by at least one, the algorithm must terminate. In practice, we would increase the flow by the largest value which would keep the flow along the path feasible.

While this algorithm is far from optimal, it is clear and easy to implement.

Finding the Degree of Redundancy of Paths and Vertices

If we redefine the capacities of the edges of a network to all be one, then the maximal flow from v[1] to v[2] gives the number of alternate paths from v[1] to v[2], i.e. the degree of redundancy for traffic from v[1] to v[2]. Note, however, that the alternate paths may share vertices. In order to find the degree of redundancy in vertices, we have to transform the network by converting each vertex into a pair of vertices, connected by a single directed edge. All incoming edges (negatively) incident with the original vertex are made incident with the initial vertex of the new edge, while all outgoing edges (positively) incident with the original vertex are made incident with the terminal vertex of the new edge.



By splitting the vertices this way, alternate paths can no longer share vertices of the original network. Thus the degree of redundancy of paths in this new network gives us the degree of redundancy of vertices in the original network.

Summary

In this chapter we have looked at the tools from graph theory which are most useful in network design. The incidence matrix gives us a data structure with which to represent graphs. Dijkstra's algorithm provides an efficient means of finding minimal cost paths within a graph. Using the idea of the incremental graph of a network, we can use minimal cost algorithms to find maximal flows within a network which do not violate the capacity constraints of the network. The same approach can be used to find the degree of redundancy in paths or in vertices of a network.

Exercises

1. Consider a finite (undirected) graph of 201 vertices, numbered 1 through 201. Suppose each even numbered vertex has an edge to the two odd numbered vertices above and below it in sequence. Suppose each odd numbered vertex in the range 1 through 199 also has an edge to vertex 201. Is the number of edges incident on vertex 201 odd or even? Why?
2. Take the graph from problem 1 and assign a cost to each edge equal to the square root of the magnitude of the difference in ordinals of the vertices incident on the edge. Find a minimal cost path from vertex 3 to vertex 198, from vertex 8 to vertex 3, from vertex 8 to vertex 198.

3. Take the graph from problem 1 and make it into a directed graph by directing all edges from lower numbered vertices to higher numbered vertices, except at vertex 201. At vertex 201 direct the edges out if they go to odd numbered vertices, and in from even numbered vertices. Assign a capacity to each edge equal to the magnitude of the difference in ordinals of the vertices incident on the edge. Find the maximal feasible flow from vertex 3 to vertex 198, from vertex 8 to vertex 3, from vertex 8 to vertex 198.

Chapter 2

Communications Network Design

2.1 Design of Network Topologies

In this section we will consider first the selection of a suitable network topology, and then the design of algorithms to provide routing and congestion control. The tools of queuing theory and graph theory provide a rigorous basis for network topology design. Unfortunately, however, the complexity of real problems is so great that we will be obliged to fall back on crude approximations and heuristics. That will also be the case in handling routing and congestion control questions.

The Conflict Between Response Delays and Throughput

The tools we have introduced allow us to do two conflicting things. With queuing theory we can optimize a network for minimal delay times. As we have seen, this will require us to provide servers with significantly faster service rates than the customer arrival time to avoid saturating network capacities. Indeed, to avoid building queues, we would have to stay below half the capacity of the servers.

With graph theory, we can allocate traffic among paths to make maximal use of capacity and to move as much data as possible. Ideally we would have no reserve capacity, but that implies infinite queue lengths. This conflict between response delays and throughput optimization is real and has no simple solution. There are customers who require short response delays because they have many independent transactions, and there are customers who require maximal data rates because they are pumping very large streams of data through the system. About all one can do to provide both kinds of service is to logically divide a network into two networks, one optimized for minimal delay times by having reserve capacity, and one optimized for throughput by saturating lines.

As long as one is not guaranteeing the high-throughput service, it is possible to cheat a little and use some of the reserve capacity intended to ensure short delays as capacity available for continuous traffic, provided one can steal back that capacity at will. This is

similar to the idea of running batch production jobs at low priority in the backgound of a time-sharing system. The danger is that, when the time comes to take back the capacity loaned to the high throughput system, it will not be economically or politically feasible to do so.

The Telephone Company as an Example

To make this inherent conflict clear, consider the plight of the telephone company. For call establishment, it must maintain a moderate delay signalling network. For actual voice traffic, it must maintain a close to zero delay voice network. To make best use of its equipment, avoiding idle capacity and attracting digital traffic, it is prudent to convert voice and signalling traffic into digital packets sharing very high bandwidth trunks. Yet, once a conversation is broken into packets, the time between packets cannot be allowed to stretch, or conversations will get out of synchronization, so some capacity, while available most of the time, must be kept idle, just in case there is a burst in the information content of a conversation.

Hierarchical Design

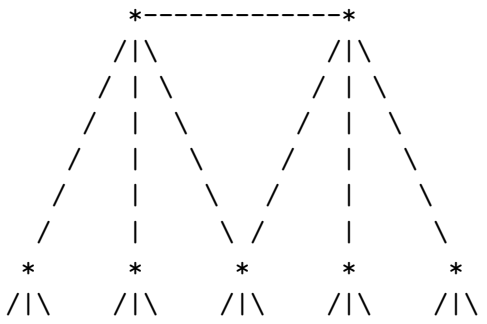
Once one has made the basic design decisions of what characteristics one is trying to optimize and with what relative weights, it is necessary to organize the problem into manageable tasks. In large networks this organizational problem can be reduced by imposing a hierarchical structure, similar to the telephone system of local exchanges and long distance exchanges.

The idea is to consider the traffic in local areas as a separate design problem, and then treat each local area as if it were a single node on a larger network. In the telephone system, each instrument is connected to a local exchange. Traffic among subscribers connected to the same exchange is switched within that exchange. Local exchanges are then grouped into areas. Within an area, exchanges are connected by local trunks. When a subscriber in the area calls another subscriber in the same area, but within a different exchange, the call is switched onto a local trunk to the target exchange by the calling exchange. This might be identified by the first three digits dialed. The remaining, say four, digits are ignored by the calling exchange and used by the target exchange to complete the call. In order to call between areas, long distance exchanges are used, with a similar division of labor, where an area code of, say, three digits is used to start the call on its way to the foreign area, which then handles the remaining digits of the number at that end.

Hierarchical Design

By adhering to such a hierarchy, we can greatly reduce the complexity of the networks to be handled. The cost is that we do not allow arbitrary one-step interconnections among terminal nodes throughout the network, introducing extra switching and forwarding delays.

Thus it might seem that a hierarchical design is only suitable when traffic tends to be localized. In many networks, as in the telephone system, it is the case that much traffic is localized. However, even in networks with heavy non-local traffic, modern hardware allows the costs of efficient hierarchical structures to be low enough to make non-hierarchical schemes unattractive in almost all cases.

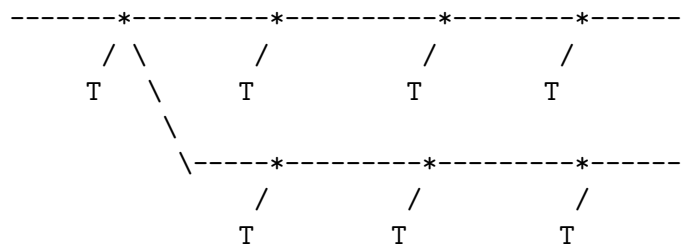


In communications network design, the major division in the hierarchy is usually between the design of the local area networks and some wide area networks joining the local area networks. Some node(s) on the local area networks serve as gateways to the wide area networks.

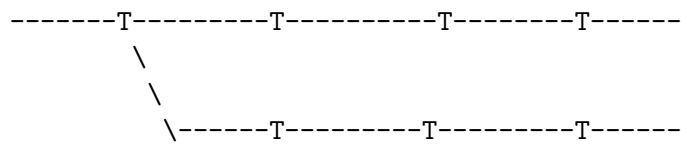
2.1.1 Local Area Networks

Let us look at the local area network problem first, since many network designs never have to go beyond that point. In its simplest form, a local area network might attempt to connect a set of terminal nodes to a single exchange point or message concentrator. Wire routings would be determined by local geography, and the only questions would concern optimal placement of the exchange and grouping of wire bundles to minimize total wire costs. Consider, again, the telephone company example. For each residence with a single telephone, a pair of wires will suffice. For a block of, say, 25 houses, one might run a cable of 25 pairs to the block, and then branch out with single pair wires for the houses. If there are more houses in the next block, however, it might well be more economical to run a much larger cable past the first block.

In the fine detail of the wiring, each house is a terminal node of the network. However, in planning the major cable routings, one might ignore the inexpensive single pair cables to the houses, and work only in terms of the drop-off points from the major cables.



becomes



Minimal Spanning Trees

As a crude first approximation to solving such a problem, it is often useful to find a minimal spanning tree from a tentative exchange location to the terminal nodes. The parameter to be minimized will vary with the application. It might be wire lengths, pole counts, repeaters, or any appropriate cost factors.

Algorithms to find spanning trees of minimal total cost have much the flavor of Dijkstra's path finding algorithm, since essentially one is finding minimal paths within the graph formed by totally interconnecting all vertices. In Prim's algorithm, edges are added in order of increasing path cost from the exchange, always taking the costs from unattached vertices to the tree built thus far. In Kruskal's algorithm, edges are added in order of increasing edge cost, skipping those that make circuits.

These algorithms are only approximately correct when external constraints are added. For example, if 25 pair cables are used, paths must be limited to 25 terminal vertices each. In such cases, one takes a less than perfect tree and tries to make small changes to improve it.

One of the simplest improvement techniques, which we will look at again, is the "branch exchange" technique. One finds an edge which was not used and which seems to have an attractively low cost. That new edge is inserted into the tree, closing some circuit, then other edges are removed until we have a tree again. If the total cost is actually lower, we can retain the new tree, otherwise we undo our changes and try another exchange.

Multiple Local Exchanges

At some point, the number of terminal nodes becomes too large for one local exchange, and some division among local exchanges must be made. Equivalently, one may think of grouping lines with message concentrators. There are two interrelated decisions to make: where to locate the exchanges and which terminals to assign to each exchange. Fortunately, we usually have only a few alternate exchange locations available. In the more general case we would have a constraining continuous multiparameter minimization to do. In the more restricted case at hand we have a combinatorial minimization to do, either starting with a skeletal set of assignments and working up, or with excessive assignments and working down. Depending on the nature of the constraints and the couplings of the cost factors, an optimal or suboptimal solution may be feasible. An optimal solution is always possible by exhaustive enumeration of cases, but may require years of calculations.

Wide Area Network Topology

If the previous discussion seems vague, it is because there is no clearly superior approach to the optimization involved. This is even more the case on the higher levels of the design problem, establishing the topology of wide area networks, i.e. the pattern of interconnections and line capacities of a network of long distance exchanges.

The costs involved in long-line trunks are large. This is true both for the system which runs its own lines and for the system which rents capacity from a common carrier. It is worth considerable effort to optimize the topolgy.

Since many message paths can be disconnected by the loss of one long distance trunk, a reasonable starting point in designing a wide area network is an analysis of the degree of redundancy required for each exchange. Using an estimate of the reliability of the equipment involved and some limit on the number of exchanges through which traffic must pass, one can estimate the required number of vertex-disjoint paths required for communications between various regions, and start with a topology with exchanges having at least the required number of alternate lines.

Steiglitz, Weiner and Kleitman proposed sequentially connecting vertices which are most lacking in the required degree of redundancy, choosing to add the edge of lowest cost when several alternative choices are available. This provides a starting point which may or may not be sufficiently connected.

The resulting network may or may not have the required node-disjoint path connectivity. To achieve it, we must take each vertex pair, compute the actual degree of connectivity, and add sufficient paths to achieve the required degree. If there are no vertices available, extra vertices or extra direct edges may have to be added.

Eventually one achieves a network with the required degree of connectivity. However, this network may well have unnecessarily high connectivity for some vertex pairs and will probably have many uneconomically long paths. Long paths raise reliability and response

time questions. Each extra vertex in a path increases the chance of a path disconnection and adds to the total transit delays. Thus we wish both to prune extra edges and to add new ones which bypass long paths, until we have a network of the required connectivity with sufficiently short paths.

2.1.2 Allocating Traffic and Capacity for Minimal Cost

Now we are ready to match edges to traffic, line capacities and costs. The traffic along edges is implied by the required flow between vertex pairs. It is tempting to use just the shortest path between two vertices to imply the flow. However, in highly connected networks this is not realistic, since traffic may be expected to flow along many equivalent routes. If one used just one such route, and it happened to share edges with a shortest path between two other vertices, one would force an unnecessarily high flow onto the shared edges, instead of making an economical distribution. Therefore, it is necessary to distribute the flows among alternate paths, either equally, or with weights inversely proportional to some cost function.

Since line capacities have not yet been assigned, the second choice would require some preliminary cost assumption, say a linear relationship between cost and distance.

A Linear Approximation

Having flows, we can take a given limit on average delay to compute line capacities.

This requires us to establish a relationship between service rates, rs[i], and line capacities, C[i]. The simplest assumption is that the relationship is linear with the same constant factor of proportionality, u, for all lines:

rs[i] = u*C[i].

If C[i] is in, say, bits per second, and rs[i] is in messages per second, then u must be interpreted as the reciprocal of bits per message. Since the physical bit rate of a line is usually a constant, and most systems run with a few discrete packet sizes in a very limited range of sizes, we do not seem to have a Poisson distribution for the servers involved. Some writers avoid this issue by assuming a Poisson distribution of packet sizes. That just begs the question. The primary reason for using the M/M/1 queuing model is that, in practice, the predictions made from it are in reasonable agreement with reality. One might suspect that this is the case because variations in message handling times at the ends of the lines are sufficiently random to fit a Poisson distribution with C[i] suitably scaled down from the raw line bit rate.

As we saw earlier, the expected delay for a network with line traffic rates r[i] and service rates u*C[i] is given by:

```
      r[i]/(u*C[i] - r[i])
W = ------------------------
                r[j]
```

If the cost is some function of the capacities, g(C[1],C[2],..), then our task is to minimize g subject to some performance constraints. A common choice of constraint is a bound, W[0], on W. The calculation can be simplified by doing the minimization of g for W = W[0], on the assumption that reductions in delay time usually increase cost.

Lagrange Multipliers

In order to solve such a constrained minimization problem, we can apply the technique of "Lagrange multipliers". Suppose we are trying to minimize a function of many variables, g, subject to the constraint equation, h = h[0]. Suppose both g and h are differentiable. Introduce a new variable, lambda, and form

```
f = g + lambda*(h-h[0])
```

Lagrange showed, for reasonable contraints, that, if g has a minimum at some point subject to the constraint h=h[0], then there is a value of lambda for which the partial derivatives of f all vanish at the same point. (We are actually assuming that the constraint implies a smooth surface of dimensionality one less than the number of free variables). In this case the form of f is

```
                 /          /_____              \      \
                |     1    | \                   |      |
                | ___---  *|  \     r[i]         |      |
f = g + lambda*| \        |  /   ---------      | -W[0]|
                | /__ r[j] | /____ u*C[i]-r[i]   |      |
                 \          \                   /      /
```

The equations which result from setting the partial derivatives of f with respect to C[i] equal to zero are:

$$0 = \frac{d f}{d C[i]} = \frac{d g}{d C[i]} - \frac{u*lambda*r[i]}{sum\ (r[j])} *(u*C[i] - r[i])^{-2}$$

If we assume that the cost, g, is linear in the capacities, C[i], with coefficients d[i] (for distance along the edge), we get

$$\frac{1}{u*C[i] - r[i]} = \sqrt{\frac{d[i] * sum\ (r[j])}{u*lambda*r[i]}}$$

which allows us to recompute W[0] without the u*C[i]-r[i] terms as

$$W[0] = \frac{\sum \sqrt{(r[i]*d[i])}}{\sqrt{u*lambda*r[j]}}$$

from which we can derive lambda in terms of W[0], u, r, and d. This then gives

```
C[i] = (1/u)*(r[i] + sqrt((u*lambda*r[i])/(d[i]*sum(r[j]))))
```

a definition of C[i] in terms of known quantities.

The difficulty with this approach, due to Kleinrock, is that the differentiability assumption rarely applies in practice. Usually, cost is a step function is terms of capacity, with only discrete capacities available. However, this approach does provide a quick and simple calculation of a starting point for a combinatorial cost minimization using more realistic cost functions.

Improving the Topology

Once we have a cost for a topology, we can compare it to the costs of alternative topologies. To get alternative topologies, we can take a starting topology, remove edges of low utilization or high cost and restore connectivity between vertices without direct paths but with high flows. Gerla, Frank and Eckl proposed a more efficient heuristic, the "Saturated Cut Heuristic", in which one first finds minimal cut with maximal flow per unit capacity. The vertices incident on the cut are called primary vertices. The non-primary vertices within one edge of the primary vertices are called secondary vertices. Call the rest tertiary.

Tertiary vertices which have traffic to cross the cut face multiple edge paths for that traffic, and thus are likely candidates for a new edge to bypass the cut, especially if an inexpensive edge is possible and path delays are high. After reconnecting the network, edges with low utilization can be deleted and the process continued.

2.1.3 When Indirect Routing is Better than Direct Routing

It is sometimes tempting to solve network cost optimization problems by adding more and more direct paths. In order to understand the cases in which direct connection is more or less cost effective than indirect routing, consider a simple network of three nodes, A, B, C, in which A and B each have traffic, rc[a] and rc[b], for C. Let the distance from A to B be d1, the distance from B to C be d2, and the distance from A to C be d3. Suppose we either route the traffic from A directly to C or via B.

```
case 1:
     A ---------->-------- C ---------<---------B
          traffic = rc[a]       traffic = rc[b]
          distance = d3         distance = d2

case 2:
     A ---------->-------- B ---------->--------C
          traffic = rc[a]       traffic = rc[a]+rc[b]
          distance = d1         distance = d2
```

In case 1, the expected node to node delay for traffic is the same as the end to end delay, while in case 2, the expected delays differ. Let W[e] be the end to end delay in case 2. Let W[n] be the node to node delay in case 2. Since multiple hops are involved in case 2, W[e] is scaled up from W[n] by the expected number of hops:

$$W[e] = \frac{2*rc[a] + rc[b]}{rc[a] + rc[b]} * W[n]$$

With this in mind, we can calculate the optimal cost in each case for a given end to end delay, W, from the formulae above which use the node to node delay.

In case 1:

$$W = \frac{sqrt(rc[a]*d3) + sqrt(rc[b]*d2)}{sqrt(u*lambda*(rc[a]+rc[b]))}$$

$$cost = (1/u)*(rc[a]*d3 + rc[b]*d2 + sqrt(u*lambda)*(sqrt(rc[a]*d3)+sqrt(rc[b]*d2))/sqrt(rc[a]+rc[b]))$$

$$= (1/u)*(rc[a]*d3 * rc[b]*d2 + (1/W)*(sqrt(rc[a]*d3) + sqrt(rc[b]*d2))**2/(rc[a]+rc[b]))$$

In case 2:

$$W[n] = \frac{sqrt(rc[a]*d1) + sqrt((rc[a]+rc[b])*d2)}{sqrt(u*lambda*(2*rc[a]+rc[b]))}$$

$$cost = (1/u)*(rc[a]*d1 + (rc[a]+rc[b])*d2 + (1/W[n])*(sqrt(rc[a]*d1)+sqrt((rc[a]+rc[b])*d2))**2/(2*rc[a]+rc[b]))$$

$$= (1/u)*(rc[a]*d1 + (rc[a]+rc[b])*d2 + (1/W)*(sqrt(rc[a]*d1)+sqrt((rc[a]+rc[b])*d2))**2/(rc[a]+rc[b]))$$

The difference, delta, of the cost in case 1 minus the cost in case 2 is then given by:

$$delta = (1/u)*(rc[a]*(d3-d1-d2) + (1/(W*(rc[a]+rc[b])))*(rc[a]*(d3-d1-d2) +2*sqrt(rc[a]*rc[b]*d3*d2)$$

```
                    - 2*sqrt(rc[a]*(rc[a]+rc[b])*d1*d2))

      >= (1/u)*(-2*d1*rc[a]
        + (1/(W*(rc[a]+rc[b])))*(-2*d1*rc[a] + 2*sqrt(rc[a]*rc[b]*(d2-d1)*d2)
                    - 2*sqrt(rc[a]*(rc[a]+rc[b])*d1*d2))
```

by the triangle inequality, d2 ¡= d1+d3. For example, if d1 is small compared to d2, and W*(rc[a]+rc[b]) is small, then this lower bound for delta is approximately

```
(1/(u*W*(rc[a]+rc[b])))*(2*d2*sqrt(rc[a]+rc[b])).
```

Thus we have cases in which, since delta is positive, it is more cost effective to route A-C traffic through B than to send it directly from A to C.

In general, however, there will be a range of both positive and negative delta. Consider the straight line case, d3 = d1+d2. Then

```
delta = (1/(W*(rc[a]+rc[b])))*
             (2*sqrt(rc[a]*rc[b]*(d1+d2)*d2)
               - 2*sqrt(rc[a]*(rc[a]+rc[b])*d1*d2))
```

is positive if and only if

```
rc[b]*d2 >= rc[a]*d1.
```

This is reasonable, in that it pays to go directly from A to C when A is far from B or has a great deal of traffic for C. That the cross-over point is the one given may not be so obvious.

In the general case, we can make a rough estimate for a lower bound on delta by using the inequality

```
sqrt(v) - sqrt(w) >= sqrt(v/2 - w), for v >= 4*w >= 0,
```

to obtain

```
delta >= (1/u)*(-2*d1*rc[a]
              + (1/(W*(rc[a]+rc[b])))*
                    (-2*d1*rc[a] +
                      sqrt(2*rc[a]*rc[b]*(d2-d1)*d2
                            -4*rc[a]*(rc[a]+rc[b])*d1*d2)))




       = (1/u)*(-2*d1*rc[a]
              + (1/(W*(rc[a]+rc[b])))*
                    (-2*d1*rc[a] +
                      sqrt(2*rc[a]*(rc[b]*d2**2 -2rc[a]*d1*d2
                             -3rc[b]*d1*d2))))

      >= (1/u)*(-2*d1*rc[a]
              + (1/(W*(rc[a]+rc[b])))*
                    sqrt(rc[a]*(rc[b]*d2**2 - 4rc[a]*d1**2
                          -2rc[a]*d1*d2 - 3rc[b]*d1*d2))))

      >= (1/(W*u*(rc[a]*rc[b])))*
              sqrt(.5*rc[a]*(rc[b]*d2**2 - 4rc[a]*d1**2
                       -2rc[a]*d1*d2 - 3rc[b]*d1*d2
                       -8rc[a]*d1**2*W**2*(rc[a]+rc[b])**2))))
```

Suppose d1 ¡= d2/6, and W*(rc[a]+rc[b]) is less than 1, then delta will be positive if rc[b] ¿ 5*rc[a].

Plotting delta in other common cases is left as an exercise for the reader.

Summary

Network topology design problems can be addressed by combining the tools of queuing theory and graph theory with reasonable heuristics. The scope of the problem can be greatly reduced by adopting a hierarchical approach, laying out local area networks and placing local exchanges as problems separate from the design of a backbone wide area network. In local area networks line paths, line capacities, and exchange locations are usually sufficiently constrained to make the problem combinatorial rather than continuous. We may start with Prim's or Kruskal's algorithm to find a spanning tree of minimal cost and improve it with branch exchange.

Wide area network topology design usually raises additional questions of redundant paths and continuous variations in traffic and line capacities. Starting with a topology of sufficiently high connectivity, we allocate traffic flows to the lines and use Kleinrock's approach to allocate capacities which will minimize costs subject to theconstraint of some maximum traffic delay. Having such a starting topolgy we can attempt to improve it by branch exchange or the saturated cut heuristic. While perfect solutions are unlikely, and many different starting topologies may have to be tried before one is satisfied, such an approach does produce economically viable solutions.

Exercises

1. Design a network for New York State, Connecticut, New Jersey, Pennsylvania, and Rhode Island. Assume one network node at the geographic location of each state capitol. Assume that each state has traffic with each other state in direct proportion to the product of their populations, at a rate of 1 bit per second per 1000000 people**2. Assume that wire costs are directly proportional to the product of capacity and distance at a rate of $.02 per foot per megabit per second. Assume that wire is available in all capacities, instead of in discrete units of capacity.

Assume that service rate is equivalent to outgoing capacity and Poisson distributed.

Limit average service delay to 100 microseconds per bit. Assume that every state must have two alternate paths to every other state.

2. Design a local area network for a community consisting of two sets of 21 houses each, each set of houses uniformly distributed on a circular road of diameter one mile. Do the problem in three cases: first when the two circular roads touch at one point, sharing one house, second when the two circular roads are two miles apart (center to center), and third when the two circular roads are ten miles apart. Assume a uniform distribution of traffic among pairs of houses and a wire cost proportional to distance. Assume local exchanges are free. First assume that house-to-house or house-to-exchange traffic cannot share wire capacity. Then assume that party-lines are acceptable. In all cases, assume that exchange-to-exchange traffic can share a common wire.

3. Suppose A, B and C are three nodes along a straight line, in which A and B each have traffic for C of 100 bits per second. Assume cost is proportional to the product of traffic and distance. Assume the B-C distance is fixed. Compute the range of A-C distances for which it is more cost-effective to send A-C traffic through B, rather than send it directly.

4. Change the cost assumption to be that cost is proportional only to distance and independent of traffic. Redo problem 3.

2.2 Network Layer Routing and Congestion Control

Having defined the major characteristics of network topology, we will now consider the techniques for moving data within the constraints of a given topology. We assume that

we have the necessary physical connections (the physical layer) and have taken whatever steps are necessary to achieve a suffciently low error rate in those connections (the data link layer). We are about to address the design of network layer processes. In the network layer, we are concerned with the problems which arise from the general interconnection of communicating nodes with possible intermediate nodes in the message paths and possible multiple paths for the same message.

Virtual Circuits and Datagrams

A network layer which provides a "virtual circuit" service is expected to act like a perfect wire for its end-point users, while a network layer which provides "datagram" service is responsible only for the integrity of individual messages, not for ensuring their order, or even their arrival.

The extreme example of a virtual circuit service is a network, such as the older analogue telephone system, which provides physical circuits. Before a call is placed, a phone is connected only as far as its local exchange. When the call is placed, a sequence of trunk lines to the target exchange is reserved for the life of the call, creating a temporary direct connection between communicating phones. The only involvement of the network during the call is to monitor the traffic for a disconnect signal (i.e. the calling party hangs up) and to do accounting. Newer phone systems may use channel sharing on trunk lines, analogue to digital conversions, etc., but the effect of a physical circuit is still provided.

The extreme example of a datagram service is, as the name suggests, the old telegram system, where each message was sent to its destination with no attempt to relate it to any other message. For many years, businesses made effective use of large "tear-tape" shops, which could get in messages on paper tape from some source, tear them off of the incoming punch, and manually transfer them to the appropriate outgoing reader. These shops were the prototypes of what is now done electronically in packet switching networks providing datagram service.

For most applications, if the network layer does not provide virtual circuits, some higher layer will have to do the job. However, there are cases where datagrams are all that are required. In particular, when interconnecting heterogeneous networks, each using internal datagrams, there is little value in forcing sequencing at the interface. Also, in applications, such as military command and control, where the network topology may be very unstable, datagrams may be all that can be provided.

Circuit Switching and Packet Switching

A related distinction is between "circuit switching" and "packet switching" networks. In circuit switching, a complete sequence of lines is provided to a data stream for some period of time, as in the telephone example above. The data stream can be quite arbitrary, since it does not have to share the lines. In packet switching, all data is broken into

packets of limited size, and intermediate nodes may share a given line among the packets of many data streams. Clearly a circuit switched network is well suited to providing virtual circuits, and a packet switched network is well suited to providing datagram service, but the distinction is not rigid. As the necessary hardware becomes cheaper, packet switching has found application as the internal mechanism in more and more formerly circuit switched networks. When the applications require it, a circuit switched network can be used for datagrams.

Routing

In any case, either during circuit establishment for a virtual circuit or physical circuit, or with every message in getting a datagram to its destination, one must provide a routing mechanism. Since many messages will be handled by a single virtual circuit, one can usually accept a higher routing overhead than with datagrams. One should not, however, take this as a license for unnecessary delays.

In addition to providing a routing mechanism the network layer must also provide means to regulate message flow, and to avoid congestion.

Let us first consider routing. In designing the network topology we made flow assignments for pairs of communicating nodes. When the topology allowed alternate routes, we distributed the traffic among those routes. One of the simplest techniques to handle routing is to provide each of the communicating nodes with precisely the same information used in designing the network, so that messages may be sent along the anticipated routes. This is called static routing.

Static Routing

Each node needs a table with an row of alternate next nodes for each ultimate destination, possibly with an estimate of the desirability of using that route. Each message must contain its destination address. When a node gets a message, it takes the destination address, looks up the appropriate row of the routing table, and using a suitable algorithm, sends the message on to one of the acceptable next nodes.

One possible algorithm is to use a random number generator weighted by the desirability of each route and randomly select from among the row entries. An alternative is to use the best route to saturation, then the next best, etc. In a network with high connectivity, the random selection corresponds best to the original design criteria and reduces the chances of saturating a route needed for other traffic. If connectivity is low, or the best route is dramatically better than the alternates, saturating the best route first may be the best choice. In all cases, there is an important distinction between the first hop a message is to take and all later hops. On the first hop, the message is being started on one of many alternate routes, and the design analysis holds. On later hops, the message is already on one of the node disjoint paths selected, and should be diverted only if the primary path is

not available. Usually, that primary path to the destination is just the continuation of the best paths to that destination from earlier nodes (the optimality principle), making a tree, called the sink tree, leading from all other nodes to the destination.

Flooding

When a message simply must get through at all costs, one might go to the boundary case of static routing, flooding, i.e. sending a message on all alternate routes simultaneously. This, however, creates the risk of having a single message saturate the entire system forever. To prevent that, one adds a hop-counter to the message, which is incremented each time the message moves one link. When the hop counter reaches some limit, the message is discarded. In networks of modest size, an alternative to the hop counter is a bit map, with a bit position for each node. The originating node sets only its own bit. Each node that gets the message sends it only to nodes whose bits are not set. This limits the number of copies that one node will see to the number of incoming lines, and is considerably more efficient than a hop counter.

Adaptive Routing

When one moves beyond static routing and flooding to routing algorithms which adapt to changes in topology and traffic, it is important to maintain a clear view of the design criteria of the routing algorithm chosen. As we saw above, there is an inherent conflict between optimizing for minimal delay and optimizing for maximal utilization. Further, once one is doing network design on the fly, it is difficult to guarantee the correctness of the routings, which places a premium on simplicity and clarity. In networks of high connectivity, one would also like the algorithm to make the best use of that connectivity, i.e. to be robust in the presence of line and node failures.

2.2.1 Centralized versus Distributed Routing

The primary choice in adaptive routing is between centralized routing control and distributed routing control. In centralized routing control, one or a few routing control centers make the routing decisions for the network. The routing control centers periodically sample the state of the network and distribute new routing tables to the nodes. To avoid conflicts among nodes as to the state of their tables, each distributed table must have an "effective time" stamp. Then each node will start using the newly updated tables at the same time. The primary advantage of this technique is that massive computing resources can be brought to bear on the routing problem, achieving near optimal solutions. The disadvantages are that the centralized routing adds its own sampling and update traffic to the network load, and, to preserve synchronization, cannot respond very rapidly to load and topology changes. Further, the routing control centers and the links to them are potential points of failure of the system.

In distributed routing, each node makes its own decisions about modifications to its routing tables. This allows very rapid response to changes in load and topology, but can lead to uncoordinated and conflicting decisions. The result can be an effective loss of connectivity, even though the network is physically connected. The extreme case of distributed routing is isolated routing, in which no information is exchanged among nodes in making routing decisions, other than noticing how well traffic is moving on various lines. The probability of poor routing choices can be very large. In more general distributed routing schemes, nodes explicitly probe their neighbors for routing information, causing major routing errors to gradually be corrected. Still, in all its forms, because distributed routing lacks time synchronized topology updates, it is more prone to routing errors and gross inefficiencies than centralized routing.

It seems the best one can do is to combine the two approaches, by having some centralized source(s) inform the nodes about the basic network topology, while the individual nodes bias that information with locally gleaned information about which lines don't seem to be working well and are producing large backlogs. The French Transpac system uses such a scheme. ARPANET uses a variation in which all nodes start with a centrally distributed topology and then periodically broadcast line delay information to all other nodes.

Most of these routing problems are greatly simplified if the network is hierarchically structured, so that only a small number of nodes have to be considered on each level. This keeps the table sizes small and allows rather thorough optimization of routes. The penalty paid is in ignoring possible better routes which violate the hierarchy. In practice, that penalty is small.

2.2.2 Highly Dynamic Network Routing

The following material is abstracted from Some Comments on Highly Dynamic Network Routing, by H. J. Bernstein, Technical Report 371, New York University, Department of Computer Science, CIMS, NY, NY, May 1988.

In a large network there can be a significant benefit in avoiding poor choices of routing. When a high speed path of few hops is available, it would seem to be sheer folly to send any traffic along slow multi-hop back-door paths. However, the addition of traffic to a route remove some of its capacity, and the next set of messages might well be better sent along an alternate path. The assignment of a set of alternate paths with an allocation of portions of the offered load among them gives rise to the bifurcated routing problem. Under reasonable constraints it is possible to find routings which optimize an appropriate payoff function, e.g. average end-to-end delay [M. Gerla, The Design of Store and Forward Networks for Computer Communications,, Ph. D. Thesis, Dept. of Computer Science, UCLA, 1973, and L. Fratta, M. Gerla and L. Kleinrock, The Flow Deviation Method: An Approach to Store and Forward Communication Network Design, Networks, vol 3, pp. 97-133, 1973]

Congestion and Flow Control

Assuming we have some acceptable routing mechanism, the remaining network layer questions are the regulation of message flow and the avoidance of congestion.

The message flow control question arises from the finite resources of receivers. At some point, no further buffers are available for incoming messages, and the sending of more messages must be stopped. The simplest approach is to design the network so that sender and line capacities are sufficiently below receiver capacities to prevent the problem from ever arising. An example is a 30 character per second printer used on a 300 Baud line. However, the desire to avoid wasted capacity usually makes this an unacceptable solution. The next level of control is provided by having the receiver inform the sender of the currently acceptable rate of transfer. As long as the estimates are conservative, this will work, but one cannot use all the capacity because one must allow for the time delay between the receiver's perception of its capacity and the arrival of that information at the sender. To go beyond such rate limiting, one must introduce a positive acknowledgement scheme, in which no more than a certain number of messages may be sent before the sender waits for an acknowledgement of the earliest messages. Such handshaking is common in high speed data communications, and also provides the basis for reliable error recovery.

Congestion is a more difficult problem to handle than simple flow control. In congestion, an increasing number of messages are moving among nodes, but a decreasing number are arriving at their final destinations. Virtual circuits avoid the problem by preallocating the necessary resources to keep traffic moving to its destination. In a datagram system, since arrival is not guaranteed, one can back off from congestion by simply discarding packets to reduce the load. There are, however, intermediate solutions, which amount to restricting the total load on the network below the point of congestion.

2.2.3 Packet Discarding

Let us consider packet discarding. If congestion is moderate and localized, discarding a few packets might clear the problem. Clearly, it is most desirable to discard packets which are involved with thecongested lines. I.M. Irland ("Buffer Management in a Packet Switch", IEEE Trans. Commun., vol COM-26, pp 328-337, Mar 1978) suggested that the designation of a line as being congested be based on a queue using more than (number of out going lines)**-.5 of the output capacity of a node. Kamoun ("Design Considerations for Large Computer Communications Networks", Ph.D. thesis, Computer Science Dept., UCLA, 1976) suggested also leaving each uncongested line with some capacity, even in the absence of traffic. This keeps congestion on a few lines from stopping traffic on the others.

For example, suppose we have a node with 16 outgoing lines and buffer space for 160 messages. We might decide to reserve 2 message buffers per line in all cases. If we run short on buffers, any line using more than 40 buffers will be considered congested, and have further messages discarded, rather than taking more buffers. As long as only three lines

are building large queues, we can go even higher than 40 without cutting into the reserve. When many lines are building large queues, we may never reach 40, since there are only an average of 10 buffers per line. In that case, we discard messages for all lines which don't have a reserved buffer free.

This last situation can have serious consequences. Extensive discarding of packets simply generates more traffic which has to be discarded since the discarded packets must eventually be retransmitted. If the only option available is packet discarding, one should at least discard the packets as early in their lives as possible. It would have been better never to have accepted too many packets to begin with.

Isarithmetic Congestion Control

In the special case of ring networks, it is possible to achieve this result by filling the ring with a constant number of circulating dummy packets. A message may be introduced into the ring only in place of a dummy message. When a real message is finally delivered, a dummy packet must be reintroduced into the ring. This "isarithmetic" approach can be extended to networks of linked rings. As long as no packets are lost, the system can be taken to full load without congestion. If packets are lost, as they must eventually be, a loss of capacity similar to that of congestion occurs.

In a simple ring, or a network built of linked simple rings, lost packets can be reinserted on the basis of open time slots, so networks made of linked rings can use this approach effectively. Attempts to apply the idea to generate networks without a uniform circulation of packets fail, because there is no simple way to decide when to reinsert a lost packet.

2.2.4 General Congestion Control

In the general case, one must fall back on the original design of the network, and limit flow to the network capacity. In the absence of traffic variations, node or line failures, this could be done by table lookups. Traffic does, however, vary, and components do fail, so a more dynamic approach is required. One approach is to use the flow control mechanisms considered above, to insure that no sender is allowed to swamp any receiver. The triggers can be similar to those resorted to in packet discarding, but with thresholds set so that, if the sender responds as required, buffers will be available for all traffic. If the sender being slowed is an intermediate node in a message flow, the action of slowing down as a sender will probably trigger its own flow control mechanisms to slow down the prior nodes in the chain sending the traffic. Thus, unlike packet discarding, use of flow control does not tend to generate extra traffic.

Both with simple flow control, and with packet discarding, it is important to design to avoid deadlocks. Deadlocks occur when two customers each need service and each of them is holding resources needed to serve the other. There are two major approaches to avoiding this situation. First, one can avoid giving any customer part of his required resources until

all of his required resources are free. Second, one can include timeout, after which an unserviced customer holding scarce resources is forced to release them. In networks, the resource most prone to deadlocks is the buffer pool. Preallocation, as in virtual circuit design and in the line reservation scheme above, prevents deadlocks at the expense of idle capacity. Resource release after a timeout can be made to work with less wasted capacity, but requires a careful case analysis to avoid infinite loops among states.

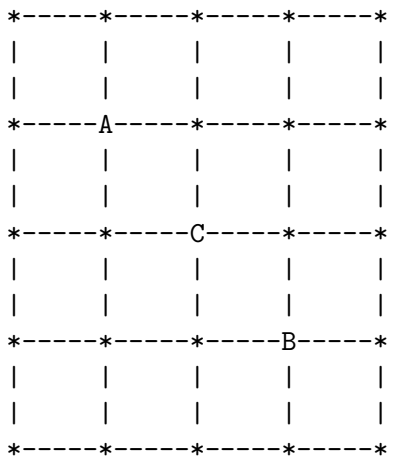
Summary

Network layer routing is required either for circuit establishment in a virtual circuit network, or for each packet in a datagram service. If the network traffic and topology are stable, static routing can be used. However, if the traffic changes or lines and nodes drop out of the network, dynamic routing becomes essential. The rerouting decisions could be made by a central facility or by each node in a distributed fashion. The first choice insures correct choices. The second choice saves considerable time. In practice, a combination of both centralized and distributed routing works well.

Unless we are dealing with a very deterministic message pattern, some form of congestion control is needed. We can discard packets in a datagram service. We can reserve some capacity for each essential message route. In ring networks, we can fill the system with dummy traffic and require that new traffic wait for a dummy message to replace. We can use explicit flow control mechanisms to keep the message traffic below network saturation.

Exercises

1. Consider a network made up of a 5 by 5 rectangular grid of 25 nodes connected by lines of equal capacity:



Provide routing tables for all nodes involved in traffic from A to B.
2. Now suppose node C wishes to broadcast a message to all nodes in the network above, except to A and B. Compare the amount of traffic generated by the following:

a. Send an individual message to each target node. b. Send the message by flooding with a minimal hop counter. c. Send the message to each appropriate neighbor with a 25-bit wide bit map indicating which destinations are to be reached.
3. Suppose we wished to apply isarithmetic congestion control to the network in problem 1, above. Assume each line can be fed at most one packet per second. Treat each small square of the network as a ring in which the dummy packets are to circulate. How fast can packets circulate in each square? How long will it take to get a packet from A to B? How long would it take without the dummy packets in circulation?

Chapter 3

Examples of Network Designs

We will now consider some actual network designs. ARPANET was the testbed for many of the concepts used in current network designs. Digital Equipment Corporation's DNA and International Business Machines SNA are two very successful commercial network architectures. CCITT X.25 is the international standard for packet switching networks. We will look in some detail at each of these. Unfortunately, the terminology used is not uniform. CCITT uses the term network layer for tasks that IBM lists under "virtual and explicit route control" in SNA, and DEC lists under "routing layer" in DNA. ARPANET breaks the same subject into two categories: IMP to IMP protocols and source to destination IMP protocols. Since the ARPANET design contributed so much to current thinking on networks, we will start with ARPANET.

3.1 ARPANET

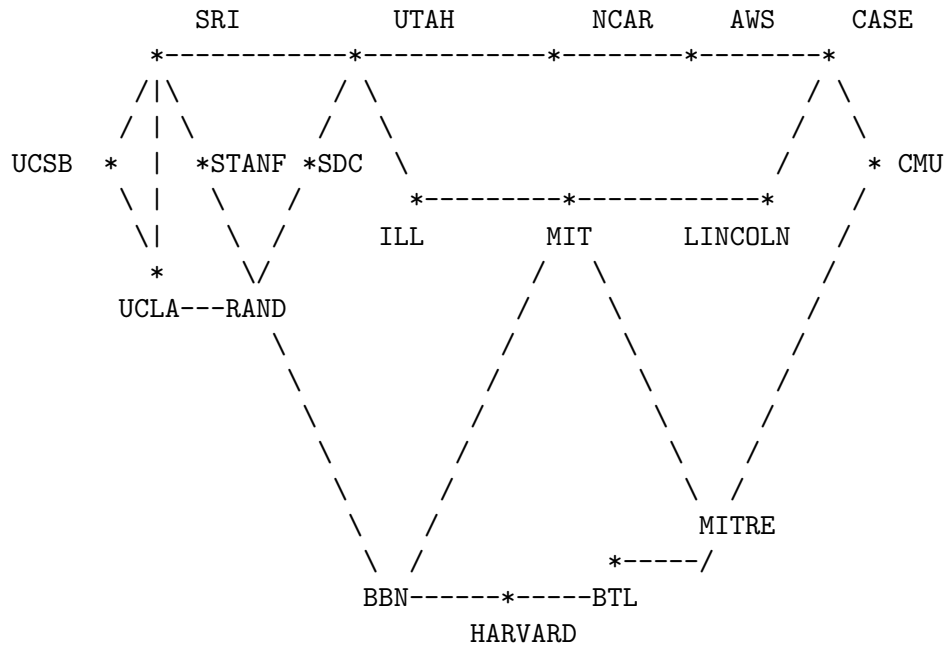
ARPANET was created as by the Advanced Research Projects Agency (now the Defense Advanced Research Projects Agency) of the U.S. Department of Defense both as a functioning resource sharing network for its geographically dispersed research efforts and as a testbed for network design concepts. It has changed greatly over the years.

A conveniently organized summary of early ARPANET design can be found in a series of five papers in the 1970 Spring Joint Computer Conference Proceedings, AFIPS Conference Proceedings, Volume 36, AFIPS Press, Montvale, New Jersey, 1970, 717+ pp. L. Roberts and B. Wessler wrote an overview in "Computer network development to achieve resource sharing" (pp 543-550). F. Heart, R. Kahn, S. Ornstein, W. Crowther, W. Walden wrote about "The interface message processor for the ARPA computer network" (pp 551-568). L. Kleinrock wrote on "Analytic and simulation methods in computer network design" (pp 569-580). H. Frank, I. Frisch, W. Chou wrote on "Topological considerations in the design of the ARPA computer network" (pp 581-588). S. Carr, S. Crocker, V. Cerf wrote on "HOST-HOST Communication protocol in the ARPA network" (pp 589-598). A de-

scription of the more recent ARPANET design can be found in DoD Military Standard Protocols, DDN Protocol Handbook, vol. 1-3, DDN Network Information Center, SRI International, Menlo Park, California, December 1985. (601, 1178 and 601 pp.)

The objective of the ARPANET design was to achieve a resource sharing network among geographically dispersed, existing heterogeneous computers. The designers focused on the creation of an autonomous message carrying service which would appear to each computer much like an interactive time-sharing terminal, so that users on one computer could make use of the resources of the other computers without significant disruption of existing operating system structure.

With that model in mind, the designers selected a human-like range of message sizes of from 1 to 1000 characters, and a message transit time of 1/2 second or less for 1000 bit blocks, to match the responsiveness of typical time-sharing terminals of the day. The inital network had to connect 18 computers across the United States:



3.1.1 Store and Forward Packet Switching Structure

Roberts explicitly credits Baran's Rand studies "On Distributed Communications" for the ARPANET choice of a distributed store and forward structure in preference to full interconnection with leased lines, dial-up service, or a star configured message switching service. The main computers, called HOSTs in ARPANET, were each connected to "small general purpose computers" called Interface Message Processors, or IMPs. 50 kilobaud

leased lines were used to connect the IMPs, with a requirement of at least "two transmission paths" between any two nodes. The two-connectedness of the network was expected to provide a loss of connectivity between any two nodes of about 30 seconds per year, based on 10-12 hours of transmission down-time per line per year. The IMPS were chosen to use no moving parts, i.e. no disks, to achieve a mean time between failures of over 10000 hours, but that implied a very limited buffer capacity for messages. The resulting network had a delay characteristic similar to that of a 20 Kilobaud total interconnect at a tenth the cost.

Services to Hosts

From the point of view of the HOSTs, ARPANET provided up to 63 simplex "links" (vitual circuits) from each HOST out to other HOSTs. In this first form, only one message at a time could be in transit on a link. When the destination HOST accepted the message the source HOST be be given a "Ready for Next Message" (RFNM) to inform it that the link was available again. Because only one message could be outstanding, messages could be kept in their original order.

Within the network of IMPs, messages, which could range up to 8095 bits, were broken up into packets of up to 1000 bits, which were routed independently. The IMP for the destination HOST had to accumulate the packets forming a message, before it could send the message on to the destination host, but intermediate IMPs did not have to worry about such reassembly. Thus, within the network, packets were handled as datagrams.

Datagram Routing

The routing of these datagrams was based on simple routing tables built by the IMPs. Each IMP was assigned a unique number, and each IMP started only with the knowledge of its own number. Using either real traffic, or dummy "hello" packets, each IMP could discover the numbers of the live IMPs directly connected to it, and estimate the delays for traffic over those lines. Thus each IMP could extend its routing table to include its one hop topology. Now each IMP could send that information to each of its neighbors, so that the tables could be extended to two hops. On the next cycle, the tables would grow to three hops, etc., until all IMPs would have a picture of the network topology, including delays and number of hops to any destination. By exchanging routing tables with neighbors every half second, the tables could respond to traffic and topology changes.

Fault Control

Among the conditions an IMP had to notice were: Failure to get all the packets forming a message within a reasonable time, failure of a destination HOST to accept a completed message within a reasonable time, and failure to provide a source HOST with a RFNM within a reasonable time. Reasonable was defined as 15 minutes in the first two cases and 20 minutes in the last.

Each IMP would monitor its own condition. Power failures would, cause a restart and failure to reset a special "watchdog" timer every minute would force a reload from a randomly selected neighboring IMP.

One of the HOSTs served as the "network measurement center", collecting statistics from the IMPs to monitor the performance of the network. One special site served as the "control center", also monitoring the network, but for malfunctions, rather than performance, and provided the needed scheduling of modifications to the network.

IMPs and TIPs

Each IMP was a ruggedized Honeywell DDP-516 with 12K 16-bit words, providing room for program, routing tables and 70 packet buffers.

This simple design worked, but with success came more demands and the need for improvements. Special IMPs which could accept user terminals directly, without a local HOST, were introduced. They were called Terminal Interface Processors (TIPs). The IMPs themselves grew to 16K Honeywell DDP-316s and multiprocessor Lockheed SUEs, and then BBN Butterfly computers. The limit of one outstanding message per virtual circuit proved too inefficient, and was first replaced by a limit of four outstanding messages per source-destination IMP pair, and then by a limit of eight messages per HOST pair, bringing us much closer to the flavor of usual network flow analysis.

Reassembly Lockup

The simple timeout detection of incompletely assembled messages was inadequate, because the failure might well be due to the exhaustion of buffers rather than to the loss of data (reassembly lockup). To improve the situation, messages requiring multiple packets could be sent only after requesting and getting a reservation of the necessary buffers from the destination IMP. The allocation would be held for a while afterwards to help in sending long streams of multiple packet messages.

Retreat from Fully Distributed Routing

The idea of having each IMP built its own map of the network topology simply by exchanging data with its neighbors worked well enough when the network was small and lightly loaded, but became unresponsive and a source of excessive traffic as ARPANET grew. Now, each IMP starts with a full map of the network and computes shortest paths within it, using delay measurements made by each IMP and then broadcast to all others every 10 seconds.

ARPANET Architecture

Until 1980, ARPANET built its communications architecture in an informal manner, gradually converging on a layered structure. For example, in 1977 (J. Davidson, W. Hathaway, J. Postel, N. Mimno, R. Thomas, D. Walden, "The ARPANET TELNET Protocol: Its Purpose, Principles, Implementation, and Impact on Host Operating System Design", Proceedings, Fifth Data Communications Symposium, September 27-29, 1977, Snowbird, Utah, IEEE, New York, pp 4-10 - 4-18) ARPANET protocols had the following structure:

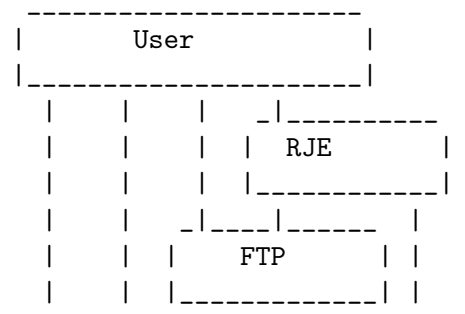
7. Ad Hoc user protocols
6. Remote Job Entry protocol (RJE)
5. File Transfer protocol (FTP)
4. Telecommunications protocol (TELNET)
3. Initial Connection protocol (ICP)
2. Host to Host protocol
1. Communications Subnet (Host-IMP, IMP-IMP)

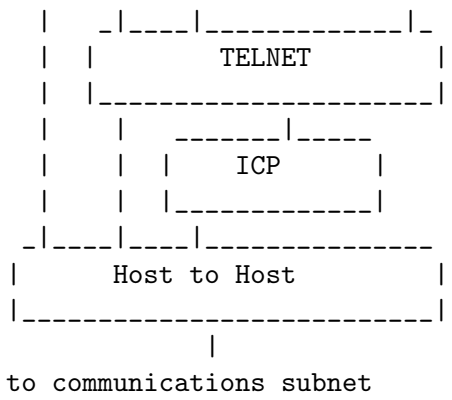
The Host to IMP and IMP to IMP protocols correspond to the ISO network layer protocols, providing a rudimentary virtual circuit service via datagrams in a communications subnet.

The Host to Host protocols correspond to the ISO Transport Layer protocols. As of 1977, ARPANET had its original Network Control Program (NCP) protocol and was developing a more general and robust Transmission Control protocol (TCP).

The Initial Connection protocol (ICP) used the Host to Host protocol to establish the required facilities for a communications session, and thus corresponds best to the ISO Session Layer, but was little used.

The TELNET protocol used the ICP and Host to Host protocol to provide virtual terminal access to remote hosts. The File Transfer protocol (FTP) used the TELNET and Host to Host protocols to move files of data among hosts. The RJE protocol used the FTP and TELNET protocols to submit batch jobs to remote hosts and capture output. TELNET, FTP and RJE fit best into the ISO Presentation Layer.





3.1.2 Change to TCP/IP

From 1980 to 1982, ARPANET went through a change to an architecture more suited to interconnection of networks and more strictly layered. NCP was dropped entirely in favor of TCP, an Internet Protocol (IP) was introduced between TCP and the communications subnet, and ICP was dropped. (See V.G. Cerf, "Protocols for Interconnected Packet Networks," Computer Communications Review, Volume 10, Number 4, October 1980, pp 10-132).

This protocol is also used in some local area network designs to link heterogeneous machines.

It has become the defacto standard for academic communications. Using it, a very wide variety of computers have interoperable electronic mail systems, file transfer protocols, and virtual terminal service.

IP

IP, the DoD Internet Protocol, is datagram oriented. It does not provide a virtual circuit service. That is the responsibility of higher level protocols. This simplifies the design of an IP gateway and allows it to work with a wide variety of heterogeneous networks without significant redesign of those networks. However, because IP is not providing a virtual circuit service, the transport layers above which have to provide that service, must either be identical, or at least very tightly coordinated in design.

Pouzin [L. Pouzin, A Proposal for Interconnecting Packet Switching Networks, in Proceedings of EUROCOMP, pp. 1023-36, Bronel University, May 1974] introduced the term catenet for a network of networks. In this context, a local network is one with sufficient protocol homogeneity not to require an explicit gateway.

Each node in the total IP Catenet, is thought of as a station on a local network which contains at least one IP gateway. As long as that node has traffic for other nodes on the same local network, it need do nothing special relative to IP. When the node has

traffic for a node on another network, it forms a global catenet address. Typically, the addresses are hierarchical, specifying the destination network and then the destination node within the destination network. The destination node address may require further levels of hierarchical decomposition to find the actual destination. Rather than maintain one global name directory for all possible addresses, one can form domains with local name servers to provide the needed addressing information.

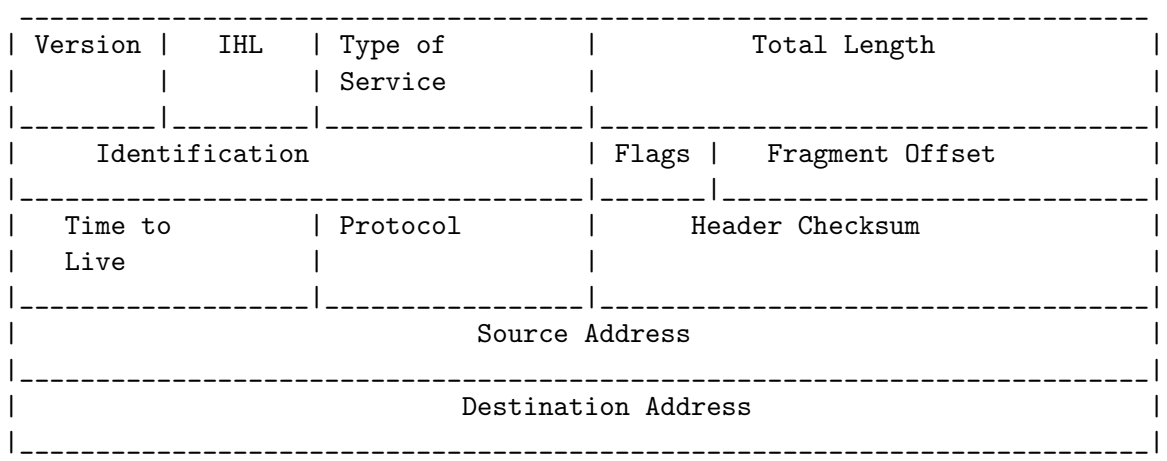
Having a valid address specifying at least the destination network, the IP process in the sending node encapsulates its datagram with header and trailer information appropriate to the network connecting it to the gateway as well as with the IP header information giving the global catanet address.

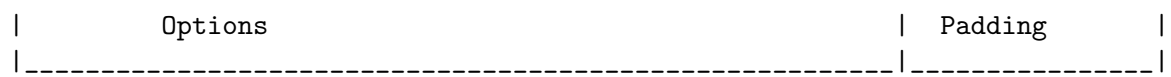
The gateway strips its local network header off the datagram, and examines the remaining catanet adressing information. It must now make a routing decision. The essential information is the portion of the global catenet address specifying the destination network. If that destination network is not known to the gateway, it rejects the datagram, sending an error message back to the sending node. Otherwise, the gateway looks up the appropriate next network to use in a routing table. If that network requires a smaller packet size, the orginating gateway breaks up the original datagram into smaller ones, flagging them for eventual reassembly into the original datagram, and sends them out individually. If queues for a particular network are overloaded, packet dropping may be used to avoid congestion. A hop counter is used to extinguish datagrams which spend too much time looking for their destination.

The protocols in IP are divided into three parts: the Intenet Protocol (IP), the Gateway to Gateway Protocol (GGP) and the Internet Control Message Protocol (ICMP).

The Internet Header

The Internet Protocol uses an Internet Header of the following form:





FieldText Version (4 bits)

This field specifies the version of the internet header format. As of September 1981, the version number was 4.

IHL (4 bits)

This is the Internet Header Length in units of 32 bit words. The minimum value is 5.

Type of Service (8 bits)

This field specifies the type of service the gateway requires of the network. The first (leftmost) three bits specify the precedence of the datagram:

111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine

The next bit, bit 3, is set to 1 when low delay is required. Bit 4 is set to 1 when high throughput is required. Bit 5 is set to 1 when high reliability is required. Each specific network may have its own response to these requests.

The field gives the total length of the datagram in units of 8 bit octets, including the internet header and data. The maximum length possible is 65,535 octets, but hosts are only required to accept datagrams of up through 576 octets.

Identification (16 bits)

This should be a number unique to a datagram as originally sent, so that the destination can reassemble the fragments into which it may be divided.

Flags (3 bits)

The leftmost bit is always zero. The next bit is 1 if the sender will not permit the datagram to be fragmented into smaller datagrams. The rightmost bit is 1 if there are more fragments making up the original datagram, or 0 if this is the last fragment.

Fragment Offset (13 bits)

This gives the offset in units of 8 bit octets of the placement of this fragment within the original datagram. When a datagram has to be further fragmented, the new offsets are always relative to the original datagram, not the fragment which was being divided.

Time to Live (8 bits)

When this field contains zero, the datagram has expired and should be destroyed. The field is supposed to be in units of seconds, but each gateway that passes on the datagram is expected to decrement the Time to Live by at least 1, making it function more as a hop counter.

Protocol (8 bits)

This field specifies the protocol for which the data field of the datagram was formatted. The acceptable values of this field are specified in RFC790 by J. Postel, "Assigned Numbers", USC/Information Sciences Institute, September 1981.

Header Checksum (16 bits)

This is the 16 bit ones complement of the ones complement sum (i.e. with end-around carry), of all 16 bits words in the header with the checksum field set to zero.

Source Address (32 bits)

There are three valid address formats, all of which correspond to a hierarchical net.node format. If the high order bit of the address is zero, this is a class "a" address, in which the next seven bits specify the net and the last 24 bits specify the node on the net. If the high order two bits are 10, then this is a class "b" address, in which the next 14 bits specify the net and the last 16 bits specify the node. If the high order three bits are 110, then this is a class "c" address, in which the next 21 bits specify the net and the last 8 bits specify the node. Other bit patterns are reserved for extending the addressing scheme. The node addresses for IP purposes can and should be mapped many to one, so that one destination node can have many equivalent addresses.

Destination Address (32 bits)

The same rules apply here as for the Source Address.

Options (variable length)

All IP implementations must support the options field, but not all datagrams need have this field. There may be zero or more options in a datagram. Each option may be a single 8-bit octet, or may be a group of 8-bit octets, starting with an option-type octet and option length octet, followed by the option data octets. The option length octet is two greater than the number of octets in the data.

If the first bit of the option type is 1, the option will be copied into all fragment datagrams on fragmentation. The next two bits give an option class:

0 - control 1 - reserved for future use 2 - debugging and measurement 3 - reserved for future use

The next five bits give the option number:

0 - end of option list (1 octet long) 1 - no operation (1 octet long) 2 - security, carried security information (11 octets long) 3 - loose source routing (variable length) 9 - strcit source routing (variable length) 7 - record route, traces the route taken (variable length) 8 - stream ID (4 octets long) 4 - Internet Timestamp (variable length, class 2)

The user of IP needs to have available a SEND and a RECV process. The send process specifies source, destination, type of service, time to live, the data, the Identifier, the Don't

Fragment Flag, and the option data. The receive process get the data, the source and destination, the type of service, and the option data.

An Internet Control Message Protocol (ICMP) is specified as a part of IP. ICMP uses IP to inform gateways and sending stations of catenet status. The possible ICMP message types are:

Destination Unreachable: The gateway does not have the destination network in its routing tables, or has been informed that the destination node was not reachable, or the IP header required a particular route which was not available, or breaking up into subpackets was forbidden by the IP header but required by some network along the route.

Time Exceeeded: The hop counter was exhausted or the destination station did not get all the subpackets needed for reassembly of the full datagram within its time limit.

Parameter Error: There was invalid or inconsistent information in the IP header.

Source Quench: The sending node is asked to send less traffic.

Redirect: The sending node is informed that a better route through another gateway exists.

Echo and Echo Reply: This is a loop-back test facility.

Timestamp and Timestamp Reply: These messages are used to probe the transit times of the catenet.

3.2 DECNET

Digital Equipment Corporation, having a popular line of timesharing computers as well as a line of minicomputers suitable for use as nodes in a distributed computing network, introduced their corporate data link protocol, Digital Data Communications Message Protocol (DDCMP), in 1973, and expanded it to a full network architecture, Digital Network Architecture (DNA), in 1975. There has been a gradual evolution of DECNET, through what DEC calls "phases". As of late 1982, DECNET was up to Phase IV, which is described in "DECnet DIGITAL Network Architecture (Phase IV) General Description", order number AA-N149A-TC, May 1982, Digital Equipment Corporation, Maynard, Mass. Portions of what follows are copied from that document (copyright 1982, Digital Equipment Corporation).

In DECNET Phase IV, the architecture is built around 8 layers:

8. The User Layer 7. The Network Management Layer 6. The Network Application Layer 5. The Session Control Layer 4. The End Communication Layer 3. The Routing Layer 2. The Data Link Layer 1. The Physical Link Layer

These layers are much closer in spirit to the ISO Open Systems Interconnect model than were the earlier DECNET phases.

The DECNET Physical Link and Data Link layers correspond to the ISO Physical and Data Link layers. DEC now supports DDCMP, Ethernet, and CCITT X.25 on a variety of media.

The DECNET Routing Layer corresponds to the ISO Network Layer, providing a datagram service. In the earlier DECNET phases, this layer was called the Transport Layer.

The DECNET End Communication Layer corresponds to the ISO Transport Layer, providing end-to-end virtual circuits. In the earlier DECNET phases, this layer was called the Network Services Layer.

The DECNET Session Control Layer corresponds to the ISO Session Layer.

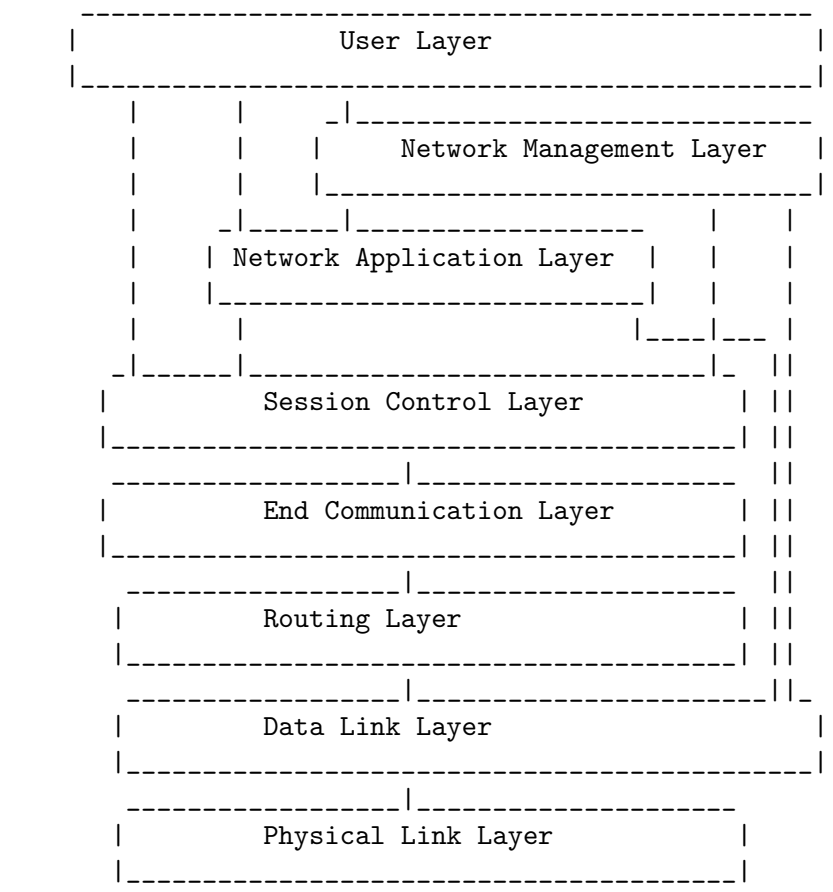
The DECNET Network Application Layer is intended to correspond to the ISO Presentation and Application Layers, but might better be considered the repository of all functions not located elsewhere in the architecture.

The Network Management Layer and the User layer provide the tools needed for configuration and control of the network. In DECNET, user programs also reside in the User Layer, rather than in the Network Application Layer.

The architecture is not strictly hierarchical. Processes in the User Layer may request services directly from the Network Management Layer, the Network Application Layer or the Session Control Layer. Processes in the Network Management Layer may request services directly from the Network Application Layer, the Session Control Layer and the Data Link Layer and may exercise management control over all layers other than the User Layer. Processes in the Network Application Layer may request services directly from the Session Control Layer and from the Data Link Layer.

The usual path for data from user processes is from layer 8 to 6 to 5, etc., without using layer 7, with processes in each layer working as peers of remote processes in the same layer. Thus, for most applications, DNA is hierarchical. The extra paths and layer 7, the Network Management Layer, are meant to improve reliability and provide the tools needed to control the system.

Leaving out the management links from the Network Management Layer, the resulting interactions are as follows:



Within each layer, DEC defines a set of protocols for peer process communications.

In the User Layer, the protocols are left to the user.

In the Network Management Layer, there are three protocols. The Network Information and Control Exchange protocol (NICE) provides for loading and dumping other nodes over the network, setting network control parameters, and system testing. The Event Logger Protocol makes a record of important changes in all lower layers, such as line failures. The Maintenance Operation Protocol performs actual line tests (hence the need for direct access to the Data Link Layer), acts as a virtual operator console for remote systems, loads and dumps remote systems.

In the Network Application Layer, there are five protocols. The Data Access Protocol (DAP) provides access to remote file directories and means for the transfer of files among nodes. The Network Virtual Terminal protocols allow a user at a terminal on one node to appear to be on a terminal at a remote node. The X.25 Gateway Access Protocol and SNA Gateway Access Protocol provide a means for DECNET systems to be interfaced to

foreign networks. The Loopback Mirror Protocol is a line testing protocol provided as a service to the Network Management Layer.

The Session Control Layer has a Session Control Protocol for establishment and termination of "logical links", i.e. virtual circuits.

The End Communication Layer uses a Network Services Protocol (NSP) to manage traffic on logical links.

The Routing Protocol in the Routing Layer makes decisions on datagram routing and responds to congestion.

The Data Link Layer uses DDCMP, CCITT X.25, or Ethernet.

Routing Through Hosts

DECNET differs drastically from ARPANET in not having a division between the devices using the communication network and the nodes which participate in routing data through the network. Every node may be both an end-user of the communications services provided and may provide routing services for traffic among other nodes. This approach evolved naturally from the first DECNET phases, which through phase II had no indirect routing of messages. All communicating nodes had to have direct paths to all possible destinations. With phase III, DECNET added the ability to route messages indirectly, purely as a change in software for the existing base of installed systems. There is, of course, nothing to stop an organization from building a DECNET which has dedicated switching nodes servicing hosts which do no indirect routing.

The DNA Routing Layer provides a pure datagram service. There is no guarantee that messages will not be lost, duplicated or delivered out of order. It is the responsibility of the End Communication Layer to correct such problems.

In the construction of a DECNET implementation, each node is given a unique, permanent numeric address. The Network Management Layer constructs a topology of connections among these nodes with line costs, line types, and node types specified. The Routing Layer uses the costs to determine minimum cost paths for messages. The line types (DDCMP, X.25 or Ethernet) determine the details of message handling. The node type (routing or non-routing) determines whether a node can be used to pass traffic on to other nodes or not.

Routing Layer Responsibilities

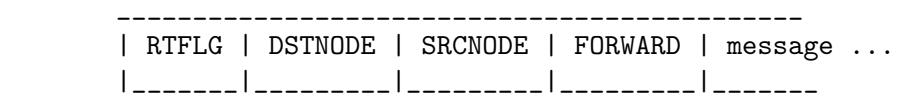
The responsibilities of the Routing Layer are:

Accept and deliver End Communication Layer messages Select routes for messages Store and forward messages for other nodes Transfer routing information among nodes Inform the End Communication Layer of unreachable nodes Control congestion and packet looping Verify adjacent node passwords Manage buffers, monitor errors, and collect statistics

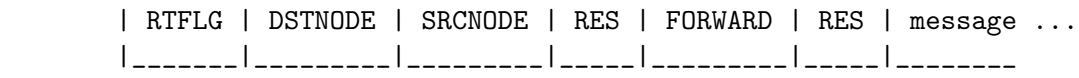
3.2.1 Routing Layer Message Formats

The Routing Layer supports data packets and distinct control messages. Data packets are formed from End Communication Layer messages by adding a Packet Route Header. If the message is going directly to an Ethernet non-routing node, an Ethernet Endnode Data Packet Routing Header is used. Otherwise, a Phase IV Data Packet Routing Header is used:

Phase IV Data Packet Routing Header



Ethernet Endnode Data Packet Routing Header



RTFLG is a 16 bit field containing routing flags: Return to Sender (indicating a packet coming back to the sender), Return to Sender Request (indicating that a packet that cannot be delivered should be returned to the sender), Intra-Ethernet Packet (indicating to a receiving Ethernet end-node that the originating node is on the same Ethernet). There is also a choke bit available in this field. The choke bit provides a congestion control mechanism.

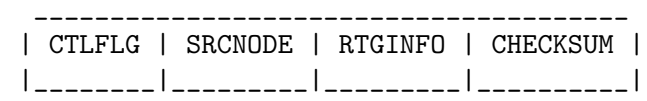
DSTNODE and SRCNODE are the numeric addresses of the destination and source nodes, respectively. In the Phase IV header they are 16 bit fields. In the Ethernet header they are 64 bit fields.

FORWARD is an 8 bit field specifying a limit on the number of hops this message may make.

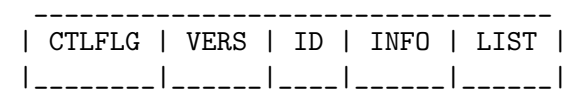
RES are an 8 and 16 bit field in the Ethernet header reserved for future use.

Routing Layer control messages are created within the Routing Layer to transfer traffic management information among nodes. On all types of lines, a general Routing message is used to update the routing tables of adjacent nodes. On Ethernet lines, an Ethernet Router Hello message and an Ethernet Endnode Hello message are used for initialization and periodic monitoring of Ethernet nodes. On non-Ethernet lines, a Hello and Test Message, an Initialization message and a Verification message are used for similar functions.

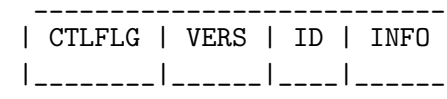
Routing Message



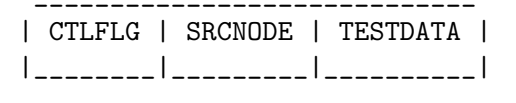
Ethernet Router Hello Message



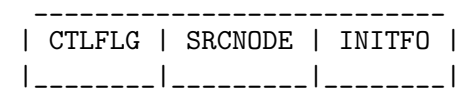
Ethernet Endnode Hello Message



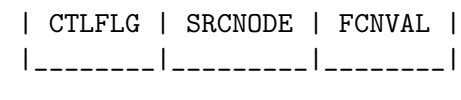
Hello and Test Message



Initialization Message



Verification Message



CTLFLG is an 8 bit (in Ethernet messages) or 16 bit (in all other messages) field with values indicating the message type.

SRCNODE is a 16 bit field containing the address of the node originating the message.

RTGINFO is a consecutive set of n 16 bit fields giving the number of hops and cost to each of the n destinations.

VERS is a 24 bit field giving the routing layer software version number.

ID is a 48 bit field containing the Ethernet identification of the node originating the message.

INFO is a 64 bit or 168 bit (in endnode messages) field giving the node type (routing node or endnode), maximum receive block size, and a hello timer.

LIST is a set of m 56 bit fields giving the known routing nodes on the Ethernet circuit with a 7 bit priority for each routing node and a flag bit for two-way connectivity.

TESTDATA is an 8 to 1024 bit field of circuit test data.

INITFO is a 56 bit field giving node type, required verification message, maximum receive block size, and routing layer software version number.

FCNVAL is an 8 to 64 bit field giving the required response to an Initialization message.

When a Routing message is received, a node recomputes the connectivity and traffic assignment algorithms for all message paths affected and updates its routing tables. If adjacent nodes are affected, they are sent Routing messages. Routing messages are also sent periodically to ensure the consistency of routing tables throughout the network. Notice that only path length and cost information are sent, not traffic intensity. This allows the routing tables to change less frequently than in, say, ARPANET, where load affects the definition of best path. A DECNET network manager can, however, respond to heavily loaded circuits by artificially increasing their costs.

Congestion is controlled mainly by packet discarding with thresholds for the transmit buffer queue length of each outgoing line, and a limit on the ratio of locally originated packets to forwarded packets. The FORWARD field, the contents of which usually is limited to 63, is decremented by one on each hop, so that packets are limited to 63 hops before being considered undeliverable.

The Ethernet Router Hello message is broadcast periodically by each node on a DECNET Ethernet which is itself a routing node. The content of the message is a list of all nodes on the Ethernet which recently sent Ethernet Router Hello messages. This allows each node to update its own information about the current topology. The Ethernet Endnode Router Hello message comes from nonrouting nodes to provide the routing nodes with information about the status of nonrouting nodes.

Since DECNET allows three very different Data Link Layers, the Routing Layer is divided into two sublayers: the Routing Control sublayer, which is fairly independent of the characteristics of the Data Link Layer, and the Routing Initialization sublayer, which provides Data Link specific initialization and circuit monitoring.

Routing decisions, routing table updates, store and forward buffer management, and End Communication Layer servicing are Routing Control sublayer tasks. Routing messages

are generated and processed by this sublayer.

Determination of the identities of adjacent nodes, of proper Data Link transmission block size dependencies, and of circuit reliability and availability are Routing Initialization sublayer tasks. The exchange of Ethernet Hello and Ethernet Endnode Hello messages is done in this sublayer for Ethernet lines. For DDCMP and X.25 lines, Initialization, Verification and Hello and Test messages are handled by this sublayer. Some extra work is required of this layer for X.25 lines.

CCITT X.25 provides a virtual circuit service. The DECNET Network Management layer establishes a database which maps the circuits to be used into X.25 permanent or switched virtual circuits. Once the virtual circuit is established, DECNET treats it as an available line for its datagram service. If the virtual circuit is broken, the Routing Initialization sublayer tries to reestablish it. This sublayer also blocks small DNA datagrams into larger X.25 packets or subdivides large DNA datagrams into smaller X.25 packets, as appropriate. An additional checksum is added to every X.25 packet to help detect virtual circuit transmission errors.

Note that in a DECNET using X.25 lines, we may find ourselves in the peculiar position of having a higher level virtual circuit implemented by Routing Layer datagrams transmitted over automatically created X.25 virtual circuits.

We will consider some of the details of DDCMP, X.25 and Ethernet before turning to SNA.

The original DECNET protocol, DDCMP, is described in "DNA Digital Data Communications Message Protocol (DDCMP) Functional Specification" Version 4.1.0, order number AA-K175A-TK, Digital Equipment Corporation, Maynard, Massachusetts 01754. DDCMP is a character oriented protocol, which DEC prefers to call "byte oriented". It is intended to work over a wide variety of line types and a variety of transmission speeds with reasonable reliability and efficiency.

DDCMP assumes transmission over media which may fail to faithfully reproduce data and which may have large transit times for messages. It allows for up to 256 stations on a single line, but only one has control of the line, as master, and the slaves do not send messages directly to one another. In a DECNET configuration, DDCMP would most likely be used for both long-distance low speed lines, either leased or direct dial, and very high speed local connections between pairs of machines.

DDCMP is a positive acknowledgement protocol with retransmission. When a message is sent, it is held for possible retransmission until an acknowledgement of receipt is received. Each message is identified by a sequence number in the range 0 - 255. Depending on the availability of buffers, additional messages may be sent while waiting for acknowledgement of earlier ones. Acknowledgement of a later message implies acknowledgement of all earlier ones. This helps to compensate for large transit times, by allowing the sender to place many messages into the pipeline, without having to be certain the first has arrived. The price to be paid for this efficiency is that many messages must be buffered for possible retransmission.

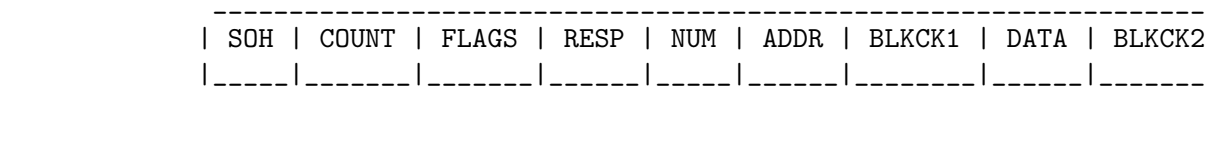
Flow control is also provided by the positive acknowledgement scheme. The network manager specifies a maximum number of receive buffers for each line. When that number of messages are outstanding, no more are sent until some of them are acknowledged.

Errors are detected by including cyclic redundancy checks on both headers and data. The CRC-16 polynomial x**16 + x**15 + x**2 + 1 is used. On lines subject to single-bit independent errors of probability p, this will give an expected undetected error rate of less than (n*p)**3, where n is the length of the field being protected. On lines subject to large burst errors, we can expect at least 99.997 percent of those errors to be detected.

DDCMP uses three types of messages: data messages, control messages, and maintenance messages.

Data messages are formed from Routing Layer messages by encapsulating them with a header and a trailing checksum. On some line types, there is further framing by synchronization characters introduced by the appropriate device drivers.

Data Message



SOH is an ASCII even parity start of header character (129 decimal).

COUNT is a 14 bit count of the number of 8 bit bytes in the DATA field.

FLAGS is a 2 bit field, one bit for line synchronization and one for transfer of control from master to slave and back.

RESP is an 8 bit field giving the message sequence number of the last data message correctly received by the sender of this data message. This piggybacked acknowledgement avoids the need to send a separate acknowledgement message for traffic in the reverse direction.

NUM is the message sequence number of this data message.

ADDR is the address of the station which should accept this message. This field is used when more than two nodes share the same line, so that each slave can recognize its own traffic.

BLKCK1 is a CRC-16 checksum on the previous fields. This is needed to ensure the reliability of the count field.

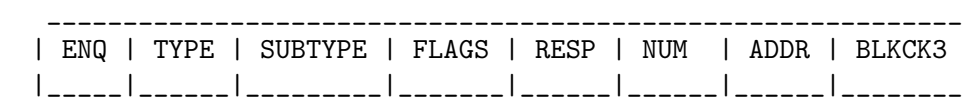
DATA is a field of 1 to 16383 8 bit data characters of arbitrary content. Since the COUNT field gives the number of bytes in the DATA field, there is no need to scan for an end message field. This is the reason DEC prefers the term “byte oriented” to character oriented. On some line types this approach creates the risk that a garbled header will cause subsequent messages to be merged into this one. For local lines, the error rates are sufficiently low for this not to be a problem. For remote lines, one usually uses synchronous

modems with discontinuous carriers, which provide an additional indication of message termination, avoiding the concatenation problem.

BLKCK2 is a CRC-16 checksum of the DATA field.

Control messages are created within the Data Link layer to control the data message traffic.

Control Message



ENQ is the ASCII enquiry character (decimal 5), flagging a control message.

TYPE is

1 for ACK, the Acknowledge message, 2 for NAK, the Negative Acknowledge message, 3 for REP, the Reply to Message Number message, 6 for STRT, the Start message, or 7 for STACK, the Start Acknowledge message.

The ACK message is sent when no reverse traffic is available onto which to piggyback acknowledgements. The NAK message is sent to provide an immediate indication of an error in a previously received message, and to unambiguously acknowledge the last correctly received message before the error. The REP message is a request for the status of a particular previously sent message. The STRT message is an initialization and synchronization request. When the node receiving STRT has reinitialized itself, it replies with STACK.

SUBTYPE is the error type in a NAK message, and zero otherwise. This field is 6 bits wide.

FLAGS are as in a Data message.

RESP is as in a Data message for ACK or NAK, and zero in REP, STRT and STACK.

ADDR is the address of the station involved on a shared line.

BLKCK3 is the CRC-16 checksum of the message.

Maintenance messages have the same format as data messages, except DLE is used in place of SOH and the message sequence numbers are zero. These messages are used for loading, dumping and controlling remote nodes.

On a shared line, the long message length and master/slave control of DDCMP can cause problems. Traffic from master to one slave can preempt the line for long periods, locking out other slaves. A master/slave relationship also requires intervention by the master in slave to slave traffic. A more efficient alternative for local area networks which share lines is Ethernet.

Ethernet

Ethernet is a local baseband broadcast system started by Xerox and jointly developed by Xerox, Intel and DEC. The specifications of Ethernet can be found in "The Ethernet, A

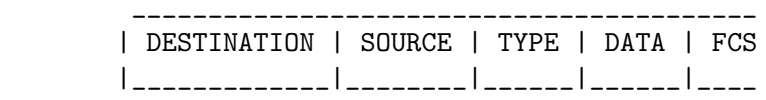
Local Area Network, Data Link Layer and Physical Layer Specification", version 1.0, order number AA-759A-TK, Digital Equipment Corporation, Maynard, Massachusetts 01754.

In an Ethernet, up to 1024 stations share a 10 megabit coaxial cable on which no two stations are more than 2.5 kilometers (approximately 8000 feet) apart. The cable must consist of segments no more than 500 meters in length, with no more than 100 stations per segment. Repeaters and remote links of no more than 1000 meters (totalled over all remote links) of coaxial cable connect the segments. Using such repeaters, the cable may branch, but will not loop. A maximum of two repeaters may be inserted between any two stations.

Each station on an Ethernet is autonomous, and may decide to broadcast messages to other stations at any time, without permission of some central arbitrator. Instead, each station listens to the line until it seems silent. It then may start its own transmission. While it transmits, it listens to the signal for signs of a collision with the signal from any other station. As long as no collision is detected within the time it would take for a signal to reach any other station and come back (less than 10 microseconds), the transmitting station knows no further collision will occur. If a collision does occur, both stations transmit a short jamming signal to ensure that the collision is detected, and then back off for a random time delay. Increasing delays are used as repeated collisions occur. Messages are kept long enough to ensure that the line contention overhead is a small fraction of the transmission time, and short enough to avoid line hogging. The use of a minimum also guarantees that receivers will discard the fragments of a collision. The minimum is 64 bytes (512 bits). The maximum is 1518 bytes (12144 bits). Notice that the longest message would take 1.2144 milliseconds to transmit. The technique used by Ethernet is called Carrier Sense Multiple Access with Collision Detect (CSMA/CD).

In contrast to DDCMP, as used in DECNET, Ethernet provides error detection without error recovery. Messages may be totally lost. Since the frequency of this event is not high, and the layer above, the Routing layer, is only providing a datagram service, without any guarantee of delivery, this approach is not fatal. Error recovery is provided by the End Communication layer instead.

In DECNET, an Ethernet message has the following form:



where

DESTINATION is the address of the destination of this message. Some addresses are reserved to indicate messages destined for more than one station or for all stations (multicast and broadcast). In DECNET the only multicast addresses are of all routing nodes on the Ethernet and of all non-routing nodes on the Ethernet. Otherwise, this 48 bit field is composed of a 32 bit prefix and the 16 bit DECNET node address, so that every node within a larger interconnected set of networks will have a unique address.

SOURCE is the address of the node originating the message. It is also 48 bits long.

TYPE is a 16 bit field used by higher layers to distinguish types of data which may follow. It is used to distinguish three cases in DECNET. The first case is a message which is too short for Ethernet and therefore includes padding. In that case the DATA field starts with a 16 bit count of the valid data and zero pad follows the data. The second case is a normal message from the Routing layer, and the third case is a maintenance message. In these two cases, the data field is just the message itself.

The DATA field is 368 to 12000 bits long (46 to 1500 bytes). It may contain arbitrary data, since the end of message is flagged by loss of carrier.

FCS is a 32 bit CRC.

For a local area network with moderate average demand for capacity and infrequent but essential requirements for bursts of high capacity, Ethernet may provide a cost effective solution. In DECNET, wider areas or high bandwidth connections may be handled by either DDCMP or CCITT X.25. The latter has the additional feature of allowing interconnection with public networks providing X.25 service.

3.3 CCITT X.25

The International Telegraph and Telephone Consultative Committee of the International Telecommunications Union has, since 1976, been establishing a series of "recommendations" on public data networks. These recommendations have, for good reason, been treated as standards by many public and private organizations throughout the world. One of the most important has been CCITT recommendation X.25, "Interface between data terminal equipment (DTE) and data circuit equipment (DCE) for terminals operating in packet mode on public data networks". A fairly accessible reference is in Computer Communication Review, Volume 10 (January/April 1980) number 1 and 2, in the article "CCITT Recommendation X.25 as part of The Reference Model on Open Systems Interconnection" by T. Jacobsen and P. Thisted (pp 48-55) and the copy of the 1980 draft of CCITT X.25 which follows (pp 56-129).

Strictly speaking, CCITT X.25 specifies only the interface between an external user of a packet switching network and the network. Internally the network is free to do anything it wishes, as long as it preserves the required external effect. In practice, X.25 has proven a sensible model for both the external interface and the internal working of a packet switched network.

CCITT X.25 provides two levels of protocols. The Link Level provides the necessary error handling and minimal flow control to allow the Packet Level to assume a reliable connection between devices.

The Packet Level defines three services: a virtual call service, a permanent virtual circuit service, and an optional datagram service. The Data Link layer protocol is called the link access procedure. There are two link access procedures in X.25, LAPB, the current

version, and LAP, the original version retained only for existing older systems, but not intended for use in any new systems. We will consider only LAPB. LAPB is intended to conform to the ISO High Level Data Link Control procedure (HDLC) and is similar to IBM's Synchronous Data Link Control (SDLC) and ANSI's Advanced Data Communication Control (ADCCP) procedures.

The packets of the Packet Level will be carried within the "information fields" of Link Level frames. We will consider the Link Level in some detail first.

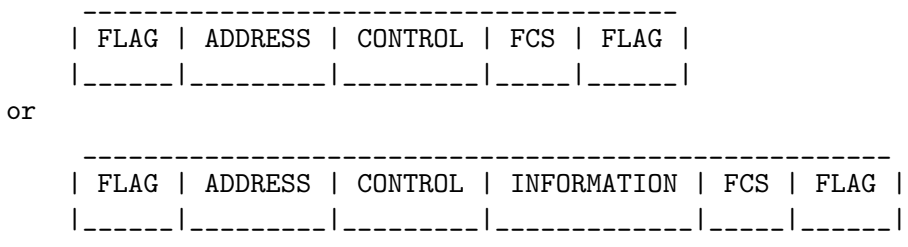
3.3.1 Link Level Protocol (LAPB)

LAPB is a bit oriented protocol. Rather than working in terms of 8 bit characters, this protocol works in terms of arbitrary bit streams, which may or may not be a multiple of 8 bits long. In order to provide a way to distinguish between control information and data in the bit stream, LAPB uses a convention that data fields will never contain more than 5 consecutive one bits. To achieve this distinction without restricting possible data patterns, all user data is examined for sequences of 5 consecutive one bits. When such a sequence is found, a zero bit is inserted on transmission. The receiving end has only to recognize the pattern 0111110 and convert it to 011111 on reception.

This allows the pattern 01111110 to be used as a message framing flag, and any sequence of seven or more one bits to be recognized as a message abort flag.

Frames

A message in X.25 is called a frame. A frame starts and ends with the flag sequence 01111110. The flag sequence which ends one frame may also start the next. There are two frame formats allowed:



FLAG is the 8 bit sequence 01111110.

ADDRESS is an 8 bit field containing either the pattern 11000000 (A) or the pattern 10000000 (B), where A is used for DCE to DTE commands and DTE to DCE responses and B is used for DTE to DCE commands and DCE to DTE responses. The patterns are given in the order of transmission of bits. In many byte oriented computers, this would be the reverse of the order of storage of the bits in memory.

CONTROL is an 8 bit field specifying the type of frame: information transfer (I frame), numbered supervisory function (S frame), or unnumbered control function (U frame). We will describe the control field in more detail later.

INFORMATION is a field with arbitrary data. Packet level information goes into this field.

FCS is the frame checking sequence, a cyclic redundancy check using the polynomial x**16 + x**12 + x**5 +1 of the frame, not including the FLAG field, times x**16 plus x**k * (x**15 + x**14 + ... + x**2 + x + 1), where k is the number of bits in the frame.

The CONTROL field may have one of the following formats:

Frame type \ Bits	1	2	3	4	5	6	7	8
I	0	N(S)			P/F	N(R)		
S	1	0	S	S	P/F	N(R)		
RR	1	0	0	0	P/F	N(R)		
RNR	1	0	1	0	P/F	N(R)		
REJ	1	0	0	1	P/F	N(R)		
U	1	1	M	M	P/F	M	M	M
SARM	1	1	1	1	P/F	0	0	0
DM	1	1	1	1	P/F	0	0	0
SABM	1	1	1	1	P	1	0	0
DISC	1	1	0	0	P	0	1	0
UA	1	1	0	0	F	1	1	0
CMDR	1	1	1	0	F	0	0	1
FRMR	1	1	1	0	F	0	0	1

N(S) is a sequence number (0 -7) for the frame being sent.

N(R) is the sequence number (0 - 7) of the next frame expected. This amounts to a positive acknowledgement of all earlier frames.

P/F is a poll bit when issued in a command, and a final bit when issued in a response. It is used mainly to request a status response without waiting for reverse traffic.

RR is the supervisory receive ready command or response. It indicates that the DTE or DCE is ready to receive an I frame, and carries a positive acknowledgement in N(R) for receipt of frames through N(R)-1.

RNR is the supervisory receive not ready command or response. It is used by the DTE or DCE to indicate that it is temporarily not ready to receive further I frames.

REJ is the supervisory reject command or response, used to request retransmission of the I frame numbered N(R).

SARM is the unnumbered set asynchronous response mode command. The proper response is the UA (unnumbered acknowledge) response, and a reset of the next expected receive sequence number to zero.

DM is the unnumbered disconnected mode response, given by a device waiting to be reset by an SABM command.

SABM is the unnumbered set asynchronous balanced mode command. The proper response is the UA response, and a reset of both the next expected receive sequence number and the next transmit sequence number to zero.

DISC is the unnumbered disconnect command. The proper response is a UA response, after which the device which sent the DISC enters the disconnected mode, and will respond with DM until reset by an SABM.

UA is the unnumbered acknowledgement response, indicating receipt of an unnumbered command.

CMDR is the unnumbered command reject response, and is identical to FRMR, the unnumbered frame reject response. It is issued to indicate that an error has been detected which cannot be recovered by retransmitting the prior frame, e.g. an invalid command or response frame format has been detected, or an I frame too long to process has been received, or an impossible N(R) field was found, or an INFORMATION field was included in a frame which cannot allow it. There is an information field in this frame, giving the control field of the reject frame and the internal state of the responding device. CMDR and FRMR are distinguished by the INFORMATION field.

3.3.2 Establishing a LAPB Link

A LAPB link is set up between DTE and DCE as follows. Initially, the DCE transmits contiguous flags, until it receives an SABM command from the DTE. Then it responds with a UA response. The devices may then exchange I and S frames, until the link is disconnected by the DTE sending a DISC command. The DCE responds with a UA response and enters the disconnected phase. It is permissible for the DCE to initate these transactions instead of the DTE doing so. Collisions are resolved by having the DCE act as if the DTE went first, if the commands agree, and by having the DCE disconnect if the commands do not agree. X.25 also defines a timer, T1, to timeout responses to commands.

Packet Level Protocol

Having established a link, the INFORMATION fields of I frames can carry packets for up to 4095 virtual circuits on the same link. The possible packet types are:

Call Initiation and Termination Packets (used in Virtual Call service)

DCE to DTE Incoming Call Call Connected Clear Indication DCE Clear Confirmation DTE to DCE Call Request Call Accepted Clear Request DTE Clear Confirmation

Data and Interrupt (used in Virtual Call and Permanent Virtual Circuit services)

DCE to DTE DCE Data DCE Interrupt DCE Interrupt Confirmation DTE to DCE DTE Data DTE Interrupt DTE Interrupt Confirmation

Datagram

DCE to DTE DCE Datagram Datagram Service Signal DTE to DCE DTE Datagram

Flow Control and Reset (used on all services)

DCE to DTE DCE RR (Receive Ready) DCE RNR (Receive Not Ready) Reset Indication DEC Reset Confirmation DTE to DCE DTE RR (Receive Ready) DTE RNR (Receive Not Ready) DTE REJ (Reject) Reset Request DTE Reset Confirmation

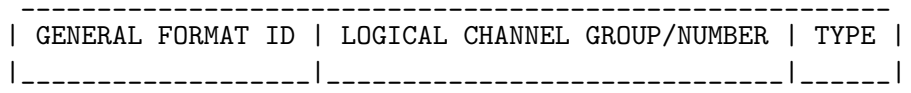
Restart (used on all services)

DCE to DTE Restart Indication DCE Restart Confirmation DTE to DCE Restart Request DTE Restart Confirmation

Diagnostic (from DCE to DTE on all services)

3.3.3 Packet Format

The format of packets in X.25 is described in terms of "octets", i.e. eight bit characters. A packet consists of at least three octets:



where the GENERAL FORMAT ID is a four bit field giving the broad class of the packet involved, LOGICAL CHANNEL GROUP/NUMBER is a twelve bit field giving the circuit involved, and TYPE is the specific packet type (8 bits).

The GENERAL FORMAT ID specifies the category of packet and whether a packet sequence numbering scheme modulo 8 or 128 is being used. If the low two bits of the field are both ones, then this is a general format identifier extension (for future expansion), otherwise 01 indicates the modulo 8 numbering scheme and 10 indicates the modulo 128 numbering scheme. The high order bit is set to 1 for datagram service signal packets and for data packets being used by higher level protocols for control information, and to zero otherwise. The next to high order bit is used for a delivery confirmation procedure. It is set to one for end-to-end confirmation of delivery.

General Format Identifier

```
  Call set-up packets                     0dss

  Data packets                            qdss

  Datagram service signal packets         10ss

  Clearing, datagram, flow control,
  interrupt, reset, restart and
  diagnostic packets                      00ss

  General Format Identifier extension     **11\\

ss = 01 for modulo 8 sequencing, 10 for modulo 128
d = 1 for end-to-end delivery confirmation
q = 1 for higher level protocol control information in data
```

The LOGICAL CHANNEL GROUP/NUMBER is divided into a 4 bit group number and an 8 bit channel number in the X.25 documentation, but is actually just a single field. The lowest range of logical channels, starting with 1, are reserved for permanent virtual circuits and datagrams. The next range is for one-way incoming logical channels for virtual calls. The next range is for two-way logical channels for virtual calls, and the last range is for one-way outgoing channels for virtual calls.

The TYPE field gives the packet type, and in data packets and some control packets carries some status information:

```
TYPE        Packet Type
00001011    Incoming Call or Call Request
00001111    Call Connected or Call Accepted
00010011    Clear Indication or Clear Request
00010111    DCE/DTE Clear Confirmation
pppmsss0    Data (modulo 8 sequencing)
sssssss0    Data (modulo 128 sequencing)
00100011    DCE/DTE Interrupt
00100111    DCE/DTE Interrupt Confirmation
ppp00001    DCE/DTE RR
ppp00101    DCE/DTE RNR
ppp01001    DTE REJ
00011011    Reset Indication or Reset Request
```

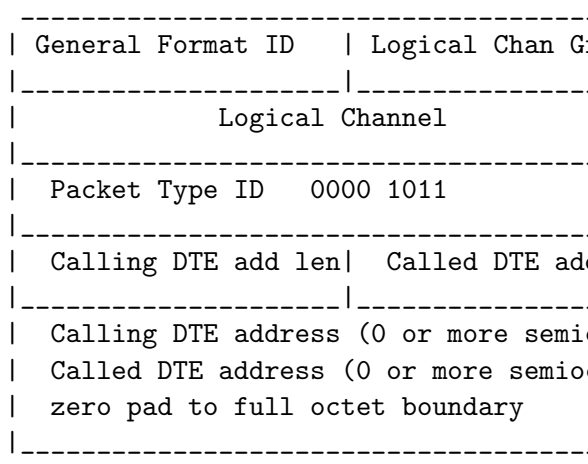
```
00011111    DCE/DTE Reset Confirmation
11111011    Restart Confirmation or Restart Request
11111111    DCE/DTE Restart Confirmation
11110001    Diagnostic
```

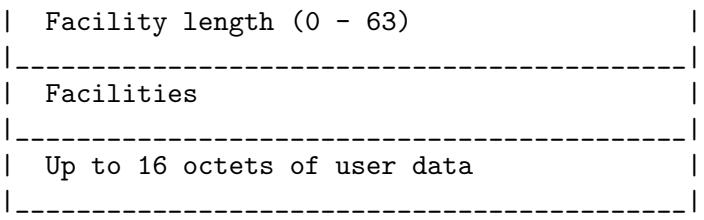
When using modulo 8 packet sequencing, the ppp field is a piggyback acknowledgement field, giving the next acceptable packet sequence number. When using modulo 128 sequencing, the next octet is of the form ppppppm. In either case, the m field indicates that the next packet will contain a continuation of the data in this one. A datagram may not ever have this field set. The sss or sssssss field contains the sequence number of the packet being sent. In the RR, RNR and REJ packets, the ppp field is a piggyback acknowledgement when using modulo 8 packet sequencing. When using modulo 128 packet sequencing, ppp is zero, and an octet of the form ppppppp0 follows. EndField

The sequencing fields provide for positive acknowledgement and flow control on a virtual circuit by virtual circuit basis, independent of the LAPB acknowledgement and flow control.

When setting up calls or sending datagrams, the next octet consists of two 4 bit fields. The high order field gives the length in decimal digits of the calling address (0-15), and the low order field gives the length in decimal digits of the called address. If the corresponding field is non-zero, each address then follows encoded as Binary Coded Decimal (BCD) in one 4 bit field per decimal digit. After these addresses a facilities length field and a facilities field follow, to allow requests for special network facilities, such as non-standard packet data field lengths, special flow control, special throughput, fast select (i.e. rapid dialing). Up to 16 octets of user data may then follow, e.g. for a password.

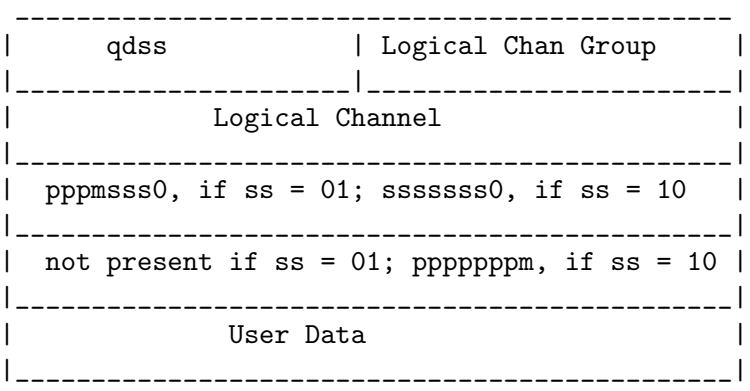
Call Request and Incoming Call packet format





When sending data on a permanent virtual circuit or a previously established virtual call circuit, the data field follows the sequencing fields. The default maximum length of the User Data field within a packet is 128 octets, and some networks require that this field be an integral number of octets long. Some networks may allow selection of other maxima: 16, 32, 64, 256, 512 or 1024 octets for all virtual calls and for each permanent virtual circuit.

Data Packet Format



Virtual Call Setup

Virtual calls are made by an exchange of Call Request and Call Accepted packets. For clarity, let us designate a local DTE, a local DCE, a remote DCE and a remote DTE, and assume that the local DTE wishes to originate a call to the remote DTE.

First the local DTE must select a channel which it believes to be free, usually the highest numbered such channel, and must issue a Call Request packet with the appropriate addresses on that channel. The local DCE gets that packet and uses the network to send it to the remote DCE. The remote DCE will select a channel which it believes to be free, usually the lowest numbered such channel, and issue an Incoming Call packet to the remote

DTE. If at the same time the remote DTE selects the same channel for an outgoing Call Request, the remote DCE will cancel the Incoming Call. Once the remote DTE gets the Incoming Call packet, it may reply with a Call Accepted packet to accept the call or with a Clear Request packet to reject the call. If the remote DTE does nothing, a timeout will eventually clear the call. When the remote DCE gets the Call Accepted packet, it passes it back to the local DCE, which provides a Call Connected packet to the local DTE. The virtual circuit created is then ready for use.

A virtual call may be cleared by a DTE or by a DCE using the Clear Request or Clear Indication packets respectively. A Clear Confirmation packet is required in response.

The normal sequencing of packets may be bypassed to send higher priority data by use of Interrupt packets. A virtual circuit may be forced to a clean initial state by a Reset packet. All virtual circuits may be simultaneously cleared by a Restart packet. Since a Reset or Restart may occur at any time, a user of an X.25 virtual circuit must always be prepared to reestablish it, and to consider as lost any packets in transit at that time.

In DECNET, virtual call service is called switched virtual circuit service. Switched virtual circuits and permanent virtual circuits are used within a DECNET. Datagrams are not used. Access to external X.25 networks is provided by an X.25 Gateway Access Protocol handled by a gateway server. The server allows nodes on a DECNET to be addressed as if they were on the X.25 network. Datagrams and end-to-end delivery confirmation are not supported.

3.4 SNA

IBM's structure for a corporate network architechture is called IBM System Network Architecture (SNA). It began in 1974 with a simple single host structure, followed by multiple, root linked trees in 1976, and general interconnections in 1979. For a historical survey of SNA, see "Doll on the Evolution of SNA" by Dixon Doll, Datamation, Volume 26, number 3 (March 1980), pp 135-138.

3.4.1 Early SNA

Let us start with an early (ca. 1975) SNA. All that is involved is a single host computer talking to its terminals through front end processors and cluster controllers. There are five layers:

5. Application layer
4. Transmission Control Functions layer
3. Path Control Functions layer
2. Data Link Control Functions layer
1. Physical Link layer

The nodes of an SNA network consist of Hosts (type 5), communications front ends (type 4), terminal cluster controllers (type 2) and terminals (type 1). Each node may contain multiple devices or processes able to communicate on the network. Each distinct communicating entity is called a Network Addressable Unit (NAU). An NAU may be a Logical Unit, a System Services Control Point, or a Physical Unit. In order to use the network, a user process must attach itself to a Logical Unit (LU). Within the host, a single System Services Control Point (SSCP) manages all traffic between the host, front ends, concentrators, and terminals. Within each node, a single Physical Unit (PU) handles all activation, termination, and error logging for data links. The PUs are logically extensions of the SSCP in the host.

The Applications layer consists simply of an applications program in the host and a user at a terminal. The messages they exchange are called Request Units (RUs). The Transmission Control Functions layer adds a Request-Response Header (RU) to identify the origin and destination of the message. The combined RH and RU is called a Basic Information Unit (BIU). The Path Control Functions layer selects a link to the next node on a path to the destination and adds a Transmission Header (TH) to the BIU, forming a Basic Transmission Unit (BTU). The Data Link Control layer adds a Link Header (LH) and Link Trailer (LT) to the BTU to form a Basic Link Unit (BLU) to send over the physical medium involved.

Applications programs used a Virtual Terminal Access Method (VTAM) to deal with terminals via the network. Programmable communications controllers used a Network Control Progam (NCP), and devices would communicate using a Synchronous Data Link Control (SDLC) protocol.

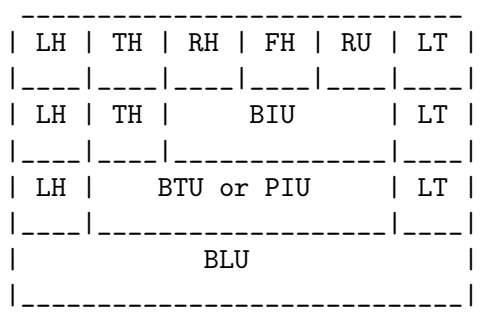
Later SNA

Within current SNA, we can identify at least eight layers:

8. End User layer
7. Network Addressable Units Services layer
6. Data Flow Control Functions layer
5. Transmission Control Functions layer
4. Virtual and Explicit Route Control layer
3. Transmission Group Control layer
2. Data Link Control layer
1. Physical Link Control layer

In order to preserve the context of the original SNA, each LU acts as if there were a single SSCP in the network. The multiple SSCPs, each of which is monarch of its own subarea, cooperate as autonomous processes. The NAU Services layer adds Presentation, Session, and Network services functions to the original SNA structure, with a Function Header (FH) added between the Request-Response Header and the Request Unit. The Basic Transmission Unit may now consist of one or more Path Information Units (PIUs),

i.e. packets. The Virtual and Explicit Route Control layer and Transmission Group Control layer comprise the functions of the earlier Path Control Functions layer.



Virtual and Explicit Route Control

The closest approximation to the ISO Network Layer in SNA is the Virtual and Explicit Route Control layer. To understand it, we must consider the layer above, the Transmission Control Functions layer, which correpsonds to the ISO Transport layer, and the layers below.

The Transmission Control Functions layer provides a session oriented virtual circuit service. The addressing used is hierarchical. The upper level is a backbone of type 5 and type 4 nodes, each controlling a subarea of type 2 and 1 nodes. The system managers prepare static routing tables with alternate routes along the backbone. Up to 8 alternate routes may be specified for each of 3 service classes (e.g. interactive, batch, and file transfer). Each hop of a virtual route is given in terms of sets of transmssion groups, i.e. sets of equivalent alternative physical lines. The Virtual and Explicit Route Control layer tries to balance load by selection of a valid virtual route, i.e. a sequence of backbone nodes which are connected for the class of traffic. This sequence is then mapped to a given explicit route of transmission groups for the session. The Transmission Group Control layer distributes traffic among lines of a transmission group, reestablishing packet ordering at each node.

A session may continue with some dynamic rerouting around failed lines as long as some valid virtual route is available. Subareas exchange information about changes in the status of transmission groups to facilitate rerouting.

The Transmission Group Control layer both disassembles long messages into multiple packets and groups short messages into more efficient packet sizes.

Flow Control and Pacing

Flow control in SNA is provided both in SDLC by a message sequence number scheme similar to that used in X.25 LAP, and by a higher level pacing scheme. In pacing, a

Transmission Control layer process which is ready to receive data sends a pacing message to its cooperating sending process indicating that it can accept some fixed number of RUs. As long as such messages are sent before the pipeline runs dry, a steady flow of data without the danger of overflow can be achieved. The pacing scheme is also used to control congestion. Intermediate nodes handling traffic pass along warnings of congestion which cause the pacing counts to be reduced.

SDLC

The major data link protocol in SNA, SDLC, is an ancestor of the CCITT X.25 LAPB protocol. The major differences are that SDLC uses the ADDRESS field of a frame to identify a destination or source slave terminal on a multidrop line, and the control field allows some more frame types:

Frame type \ Bits	1	2 3 4	5	6 7 8
I	0	N(S)	P/F	N(R)
S	1	0 S S	P/F	N(R)
RR	1	0 0 0	P/F	N(R)
RNR	1	0 1 0	P/F	N(R)
REJ	1	0 0 1	P/F	N(R)

Frame type \ Bits	1	2 3 4	5	6 7 8
U	1	1 M M	P/F	M M M
NSI	1	1 0 0	P/F	0 0 0
RQI	1	1 1 0	F	0 0 0
SIM	1	1 1 0	P	0 0 0
SNRM	1	1 0 0	P	0 0 1
ROL	1	1 1 1	F	0 0 0
DISC	1	1 0 0	P	0 1 0
NSA	1	1 0 0	F	1 1 0
CMDR	1	1 1 0	F	0 0 1
XID	1	1 1 1	P/F	1 0 1
NSP	1	1 0 0	0/1	1 0 0
TEST	1	1 0 0	P/F	1 1 1

NSI stands for Nonsequenced Information and is used to send a command or response that has an information field but does not use the normal sequencing.

RQI stands for Request Initialization. The proper reply is SIM, for Set Initialization Mode.

SNRM stands for set normal response mode, to reinitialize a slave.

ROL identifies an offline station, and corresponds to the LAP DM response.

DISC means disconnect, as in LAP.

NSA means nonsequenced acknowledge, i.e. the LAP UA.

CMDR agrees with the LAP CMDR.

XID provides a system identification in the information field.

NSP flags an optional response when there is no P bit.

TEST sends a check pattern in the Information field.

BISYNC

Because of a large customer base with older equipment, IBM also supports its older BISYNC protocol under SNA. It is a character oriented protocol for synchronous data links. Messages are started by a stream of synchronization (SYN) characters, of which at least two must be received to start a valid message. A Start of Header (SOH) then marks the start of a header field. The header ends and the message proper begins with a Start of Text (STX) character, followed by data and an End of Text (ETX) or End of Transmission Block (ETB) character. A two character CRC finishes the message.

Provision is made for transmission of arbitrary bit patterns which might be confused with message delimiting characters by use of a Data Link Escape (DLE) character. Transparent mode is entered by using DLE STX in place of STX. After that control characters are ignored unless preceded by a DLE. DLE DLE is used to send DLE as data.

BISYNC provides positive acknowledgement for error recovery and flow control, and allows interleaving of multiple data streams on one communications line. Its only serious disadvantage compared to SDLC is that it does not make as effective use of the bandwidth of a full duplex line, since it was designed for older half-duplex communications modems.

3.5 Internetting

Since there are such a variety of networks, one may wish to connect two heterogeneous networks. Aside from the obvious political and financial difficulties, there are considerable technical problems to face.

Consider two possible approaches to merging networks. First, one might wish to use a foreign network simply to join two disconnected pieces of an otherwise homogeneous network. This is the approach in using CCITT X.25 virtual circuits within DECNET.

Indeed, some manufacturers avoid the need to provide routing of packets by assuming use of an X.25 utility. Assuming one provides proper management of circuit establishment and reestablishment, and is willing to block and deblock messages to fit X.25 packet constraints, this approach is not much worse than introducing, say, an automatically dialed telephone line into a network.

The second, considerably more difficult, problem is allowing nodes on heterogeneous networks to actually talk to each other. If one of the nodes is very simple, e.g. an asynchronous terminal, the task can be handled by mapping mismatched characteristics. This has been done in grafting the American commercial data networks to the European and Canadian networks, so that foreign terminal users can have access to American host computers, and for the reverse connection. Each network creates a gateway node, which is effectively on both networks. Painful decisions must be made in creating such gateways. Is the gateway a full routing node of either or both networks, or is it to act simply as a host on each? Is virtual circuit or datagram service to be provided through the gateway? How are different message sizes to be matched up?

Those are, however, easy questions compared to the full problem of connecting heterogeneous networks. If one wishes to have fully functional nodes on different networks cooperating, there must be a matching of presentation and application layer functions. The only solution which seems to resolve all the problems of doing such a matching is standardization of operating systems, a far from completed task. For the moment, cooperation among network designers in refining their understanding of the ISO Open Systems Interconnect model is a step in the right direction.

When the heterogeneous networks are under the control of one parent organization, another solution presents itself: rewrite the incompatible parts of the code of the various networks. The U.S. Department of Defense has adopted this approach in part, in order to bring together its various networks. For example, ARPANET's NCP was replaced by TCP for this reason.

TCP/IP

TCP in ARPANET interfaces to the DoD Standard Internet Protocol, which provides a simple datagram forwarding scheme. It is the responsibility of the hosts to provide sequencing, flow control, and other services needed for reliable communications. Since datagrams are involved, a large header with full addressing is required for each packet. For more details, see V. G. Cerf, "Protocols for Interconnected Packet Networks", Computer Communication Review, Volume 10, Number 4 (October 1980), pp 10-11, and the description of the DoD Standard Internet Protocol on pp 12-51 of the same issue.

CCITT X.75

If the networks are not truly heterogeneous in design, just physically separate, a clean internetting standard is possible. CCITT has provided such a standard for X.25 networks with its X.75 internetwork protocol, which carries virtual circuits between networks via half-gateways, i.e. gateways which are split into two parts, each one truly a part of its own network, connected via a communications line. Virtual circuits can be established end-to-end through mutiple X.25 networks with X.75. Since virtual circuits are used, only small headers are needed on data packets.

There is some divergence in views about the desirability of internetting using virtual circuits. Some would prefer the use of datagrams, to avoid the overhead of forcing packet sequencing at network boundaries. As with the networks themselves, it is likely that both services will be needed for internetworking in the future.

3.6 Distance Considerations

Networks may be classified as local area networks (LANs) or wide area networks (WANs). Approaches may work well for one, the other, or both. Local area networks tend to have short transit delays, allowing designs which require frequent exchanges of administrative information. For example, in a LAN it is feasible to have a central arbitrator of resources, to distribute control by passing a token around a ring, etc. In wide area networks, especially when satellite links are involved, the transit delays are usually long, making it necessary that nodes work independently, without central control at every step. As speeds in local area networks increase, they face similar problems.

Packet switching has proven cost effective in both LANs and WANs. However, in LANs there is more chance of using packet switching on a medium shared by autonomous contending nodes. Resolution of that contention may be done in many ways. At the moment, the two most popular are the Ethernet CSMA/CD and the IBM token passing ring.

In Ethernet the stations resolve contention by listening for collisions and backing off for a while when they occur. In a token passing ring, stations avoid contention by waiting for a token (i.e. a dummy message) which they replace with their own message. The advocates of token passing feel that such networks handle very heavy loads in a graceful round-robin manner which they think difficult to achieve with CSMA/CD designs. The experience with Ethernets as of this writing does not seem to support this viewpoint. Indeed, in the February 1984 issue of Datamation, Jan Johnson wrote, in “IBM’s Two-LAN Plan”, that IBM would support both a token passing ring and a CSMA/CD system (Datamation, volume 30, no. 2, pp. 120-128). The token passing ring would be used for be used for short distances on inexpensive twisted pair wiring, while the CSMA/CD component would use broadband channels for greater distances on coaxial cable.

3.6.1 Broadband

Ethernet makes rather inefficient use of coaxial cable bandwidth. Broadband networking makes better use of the available bandwidth. Frequency division multiplexing is used to provide multiple channels. Each channel could be used for a CSMA/CD system, or for a dedicated master/multiple slave link, or for voice or video. Standard Ethernet is called a baseband system. While less hardware is required for an Ethernet system, and the larger customer base reduces unit costs, the intrinsic costs of baseband and broadband network connections are similar. Over the long run, it is likely that Ethernet baseband will become a special variation of broadband networks.

Typical broadband systems use commercial cable television (CATV) or master antenna television (MATV) components, making it natural to divide the frequency spectrum into 6 megaHertz television channels. This is not quite enough bandwidth for a reliable full 10 megabit per second use with inexpensive equipment, so broadband CSMA/CD is sometimes redone on the basis of 5 megabits per second.

3.6.2 Repeaters

Once a baseband or broadband cable becomes too long, some form of repeater is required to restore the signal strength. In baseband systems this can be tricky, since the repeater must not treat its amplified output in one direction as an input to be amplified in the other direction. For CSMA/CD networks this problem can be resolved by having the repeater act in only one direction at any given time. If it hears a signal on only one side the repeater can shut down its amplifier in the other direction, since any signal coming the other way must represent a collision. If the repeater has signals on both sides it has a definite collision and need only broadcast the jamming signal on both sides, not amplify either incoming signal.

General broadband repeaters which allow arbitrary traffic in both directions simultaneously do so by using two different channels, one for each direction, with a unidirectional amplifier for each. This requires that the system contain a frequency translation circuit at one end to move the incoming traffic on one of the two channels onto the corresponding outgoing channel.

3.6.3 Digital PBX Systems

In an attempt to avoid major recabling for local area networks, some institutions have tried to use their existing telephone cables for local area networks. Telephone cables consist of twisted pairs which can be driven with signals of over 1 megabit per second at distances of up to a mile. Fairly conservative designs allowing for 64 kilobits per second over a few miles are in common use. One can redesign a voice exchange system (PBX) for use with such signals and even leave a part of the capacity for voice traffic. In the March 1983 issue of Datamation, F. H. Harris, F. L. Sweeney Jr. and R. H. Vonderohe describe such a

digital PBX system for the University of Chicago ("New Niches for Switches", Datamation, volume 29, no. 3, pp. 109-114).

3.7 Data Base and File Transfer Subsystems

While most current use of data communications networks for computers is for timesharing and interactive processing, data communications networks have seen significant use for data base and file transfer problems.

Airline reservation systems, military attack and defense coordination systems, and weather forecasting systems all have required integration of data communications with data bases. However, most such systems have simply grafted a communications system onto a centralized data base system. The demands of such applications on the communications system are similar in nature to those of ordinary interactive applications. When the data base is distributed, however, new problems do arise.

A process at a given node wishing to do a search of the data base must coordinate with search processes at remote nodes. For ordinary sequential searches, this amounts to little more than distributing the query and waiting for the answers to arrive. It is more efficient to use indices to speed searches. The indices can be duplicated at many, or even all, nodes, so that the index search is done locally, and only the final data records are gathered remotely. This greatly reduces the traffic involved in searches, but creates an update coordination problem and requires duplication of storage. The flavor of this problem is similar to that of managing the routing tables of the network itself, since that is also a distributed data base. In the general problem one cannot permit the imperfect approaches used in routing, since end users expect perfect retrievals of data, while network routing errors are accepted. Thus, in the data base problem, one must sacrifice statistical cost optimization to ensure reliable operation.

In file transfer, two hosts are exchanging some large block of data, e.g. a batch job stream, a report, a data base. This application differs from interactive use in requiring long stable sessions, but allowing long transit times. Despite this, at present the most common form of file transfer is to use a terminal simulator (e.g. UNIX uucp) which acts as if the file were being typed in by the terminal user. The lack of end-to-end error checking and automatic session recovery makes this a human labor intensive solution to the problem. In an attempt to provide a better solution, ARPANET defined a File Transfer Protocol (FTP), which removes much of the user intervention and allows arbitrary data streams. DECNET goes even further, allowing the user full access to the file system of remote hosts on a record by record basis.

3.8 Security

No data communications system which allows any access by unauthorized persons to any portion of the system can be assumed to be secure. Cummunications lines can easily be tapped so that data streams can be recorded and examined. Passwords can be misappropriated or guessed at. Terminal screens can be observed by strangers. System listings, tapes and disks can be gleaned from trash. The most one can do is to make penetration difficult, expensive and time-consuming.

Encryption of all communicated and stored data is a minimal necessary step. To avoid information gathering from the pattern of transmission itself, it is also important to force a uniform level of activity on all lines by insertion of dummy traffic when no true traffic is available. Data storage should by organized with both hierarchical and zoned access permissions so that no greater degree of access need be provided than is required for the task at hand. All transactions with any security implications should be permanently logged, so that attempts at penetration can be analysed and stopped.

Hopefully, this sounds as paranoid as intended. It is not that there are hordes of industrial spies ready to pounce on every vulnerable network. Rather, the costs of sercurity breaches can be so high, that, while the probability of attempts at penetration is low, the expense of avoiding them if possible is well justified.

Conclusion

The time constraints of a single semester leave much to be said about the details of network design and implementation. It is hoped that the reader has at least gained some appreciation of the magnitude of the problems involved and of the nature of the tools needed to attack them. Network design has in the past often been characterized by ad hoc mixtures of heuristics and difficult to apply abstractions. The problems are too complex to admit fully rigorous solutions in reasonable time, but it is hoped that future designs of networks show an ever improving craftsmanship.