On the Modeling and Analysis of Computer Networks

LEONARD KLEINROCK, FELLOW, IEEE

In this paper we view the landscape of analytic models for computer network performance evaluation. We present the basics of network analysis, discuss the network design procedure, and introduce some of the fundamentals of network control. Finally, we present some of the issues surrounding the design and behavior of gigabit networks.

I. Introduction and Apology

Can you predict the way a network will perform? Can you do it easily, exactly, correctly? Indeed, what metrics should you use to evaluate performance? These are the kinds of questions a network analyst asks himself all the time.

The fact is, there are a number of ways in which you can go about doing network system performance analysis. In order of increasing ugliness, they are as follows:

1) Conduct a mathematical analysis which yields explicit performance expressions.
2) Conduct a mathematical analysis which yields an algorithmic or numerical evaluation procedure.
3) Write and run a simulation.
4) Build the system and then measure its performance!

Occasionally we are lucky enough to develop the first type of solution. Let us discuss this case. The fact is that we often work rather hard at trying to solve the mathematical model we develop. Indeed, we often "fall in love" with our mathematical model and assume that our careers depend upon providing an exact solution to the model generated. However, it must be recognized that our mathematical models *never* do an exact job in representing the actual network being analyzed. *Thus we always end up with an approximation, even if our analysis is exact.* Sometimes, all we can do is to provide an approximate solution to our mathematical model; this often takes cleverness on the part of the analyst. A different approach, and one which is often overlooked is to generate a different (and possibly *more* approximate) mathematical model whose solution may be more tractable than for our original model. It is important for the analyst to recognize that this is a perfectly legitimate approach to consider. After all, the final test of our analysis is when we compare its predictions to actual measurements of the real network. Only if the predictions are within an acceptable tolerance can we claim to have done our job as analysts.

In this paper, we will concentrate on the use of such mathematical models of network performance. Of necessity we will be making approximations in a variety of ways.

Manuscript received May 13, 1993.
The author is with the Computer Science Department, University of California, Los Angeles, CA 90024-1956.
IEEE Log Number 9211347.

II. The Early Network Analysis Models

The first models of computer networks were developed in the early 1960's [1] ,[2]. However, it was not until the late 1960's when the United States Department of Defense Advanced Research Projects Agency (DARPA) funded the development of the ARPANET that serious effort began in this area. Indeed, it was the infusion of a relatively small amount of government money and a large amount of vision in funding network research that propelled the entire field of networking and packet switching forward and produced the extensive analytical work and physical networks in which we find ourselves swimming today.

One of the first general results was an *exact* expression for the mean delay experienced by a message as it passed through a network. The evaluation of this delay required the introduction of an assumption (the author's Independence Assumption) without which the analysis remains intractable, and with which the analysis becomes quite straightforward. Let us begin our journey through the analytical thread with this model [1]. Consider the network shown in Fig. 1.

Here we see a network in which we assume there are N nodes corresponding to the network switches and in which we assume there are M links corresponding to data channels connecting the switches. We assume that messages arrive from a Poisson process at origin node j and headed for destination node k at a rate γ_{jk}; that is, the variables γ_{jk} correspond to the entries in the network traffic matrix. Furthermore, we assume that the messages are exponentially distributed with mean length $1/\mu$ bits per message. In Fig. 2 we show a detailed view of a node in the network.

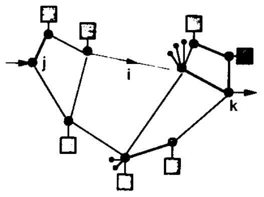

Fig. 1. Model of a computer network.

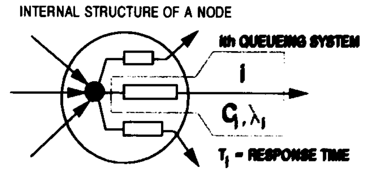

Fig. 2. Detailed view of a network switching node.

Since each channel is, in fact, a full-duplex channel, we choose to represent a channel as two simplex channels. Thus we see there are three ways in and three ways out of the node shown (for the moment, we ignore any attached hosts). Now, when a message arrives to such a node, a routing decision must somehow be made which determines over which outgoing channel the message will travel next. Once this decision is made, the message is placed on the tail of a queue of other messages waiting to be transmitted out over this channel (say channel i). In the figure we have identified a queueing system consisting of such a queue and its corresponding channel. We denote by λ_i the traffic carried on this channel (in messages per second) and we let C_i be the capacity of this channel (bits per second). In addition, we let T_i be the mean response time of this little queueing system.

Let us also define some global quantities. First, we define the total (external) traffic carried by the network as

$$\gamma = \sum_{j,k=1}^{N} \gamma_{jk} \tag{1}$$

and the total (internal) network traffic carried by the channels as

$$\lambda = \sum_{i=1}^{M} \lambda_i \tag{2}$$

Moreover, if we let $\overline{n}$ be the average number of hops that a message must take in its journey through the network (averaged over all origin-destination pairs), then it is easy to show that the following is true [1]:

$$\overline{n} = \lambda / \gamma \tag{3}$$

Lastly, and most importantly, we define T to be the mean delay of all messages (again, averaged over all origin–destination pairs). T is the mean response time of the network and is one of the most important performance variables for a network. It can easily be seen (by two applications of Little's result [3]) that [1]

$$T = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} T_i \tag{4}$$

This is an exact equation and is an important general result for networks of all sorts.

The only difficulty with this last equation is that we have not given an explicit expression for T_i. In general, this turns out to be an insurmountable problem. It is probably true that we will *never* be able to give a tractable expression for T_i! This may come as a surprise to those familiar with queueing networks; indeed, it appears that we have set up a perfect Jackson open queueing network whose solution is simple and well-known [4]. The problem comes from the fact that a key assumption in queueing network theory is that the "service time" experienced by a "customer" in any node of the network is independent of all service times of all customers in all nodes. Our problem comes about since the "service time" a customer (the message) requires from the server (the channel) is simply transmission. Now it is clear that the time it takes to transmit a message over one channel is exactly the time it will take to transmit that same message over a different channel of the same capacity. This generates a major dependence among the successive service times seen by a message as it makes its way through the network. In fact, this also creates a dependency among the interarrival and service times seen by a message at a given node. It turns out that this causes grief beyond belief in the analysis. Indeed, the case of two nodes in tandem where the channel capacity of each of the two attached channels is identical was solved (many years after this problem was first posed in [1]) by Boxma [5], and that solution was extremely messy.

It turns out that if we change the model to one which is, in fact, more representative of data traffic, then we can give an exact solution for a restricted topology. The modification is to assume that the message lengths are all the same (rather than the exponential assumption above) and that the topology is a tandem network; in this case, Rubin [6] was able to give an exact solution for the response time when all traffic enters at one end of the tandem and exits at the other end. Unfortunately, the analysis does not extend to the case of nontandem networks.

Let us now return to our original model using exponentially distributed message lengths. As mentioned above, extreme analytical difficulty comes from the dependence among message service and interarrival times. The Independence Assumption assumes that the length of a message is chosen independently from the exponential distribution each time it enters a switching node in the computer network! This is clearly a sweeping assumption which is patently false; however, it turns out (from measurements and simulation) that this assumption has a negligible effect on the mean message delay. AND, if we do make the assumption, then our analysis becomes trivial since we will have then reduced the system to a Jackson open queueing network model.

If we make the Independence Assumption, then the expression for T_i for the ith queueing system shown in Fig. 2 reduces to an M/M/1 queueing system [4] whose solution is simply

$$T_i = \frac{1}{\mu C_i - \lambda_i} \tag{5}$$

Note that μC_i is the capacity of the ith channel expressed in messages per second. When we substitute this into (4), we end up with an explicit and simple expression for the mean message delay T as follows:

$$T = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} \left(\frac{1}{\mu C_i - \lambda_i} \right) \tag{6}$$

This equation is very effective for design calculations as we shall see below. However, it ignores certain realities which become important when one wishes to give a more precise prediction of network delay. For example, we have assumed $K = 0$ and $P_i = 0$ (where K = nodal processing time and P_i = propagation delay). When we come to apply this analysis to any realistic network, we must include these variables as well as other considerations. Specifically, not only is there message traffic moving through the network, but there is also a certain amount of *control* traffic. The basic M/M/1 result gives the message delay of the true message traffic for a single channel; of course, this delay is composed of two quantities, namely, a waiting time on queue and a service time. The service time has an average value related to the average length of the true data traffic; the waiting time, however, is due to the interference of all other traffic in the network and is composed partly of data traffic and partly of control traffic. Therefore, it behooves us to separate these two contributions to delay and to use the appropriate parameters for each. If we let $1/\mu$ denote the average length of a data packet and if we let $1/\mu'$ represent the average length of all packets, then we see that a more accurate expression for T_i is

$$T_i = \frac{\lambda_i/\mu' C_i}{\mu' C_i - \lambda_i} + \frac{1}{\mu C_i}. \tag{7}$$

Of course, if we set $\mu' = \mu$, this will reduce to (5). If we now account for the nodal processing time K and the channel propagation time P_i we may then write down the following approximation for the average message delay:

$$T = K + \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} \left[\frac{\lambda_i/\mu' C_i}{\mu' C_i - \lambda_i} + \frac{1}{\mu C_i} + P_i + K \right]. \tag{8}$$

The term in square brackets is just our new expression for T_i and the additional term K comes from the fact that messages pass through one more node than they do channels in their travels through the network. Note that in the case $K = P_i = 0$ and $\mu' = \mu$ the expression is reduced to our basic expression in (6).

From this delay analysis we may predict quantitative as well as phenomenological behavior of the average message delay in networks. In particular, if we assume a relatively homogeneous set of C_i and λ_i, then as we increase the load on the network, no individual term in the sum for delay as given in (4) will dominate the summation until the flow in one channel (say channel i_o) approaches the capacity of that channel; this channel corresponds to the network *bottleneck.* At that point, the term T_{i_o}, and hence T, will grow rapidly. The expression for delay will then be dominated by this term and T will exhibit a threshold behavior; prior to this threshold, T remains relatively constant and as we approach the threshold, T will suddenly grow. Thus we expect an average delay in networks that has a *much sharper behavior* than the average M/M/1 delay.

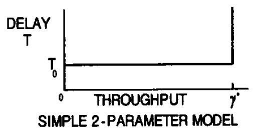

Fig. 3. The threshold behavior of average network delay.

This threshold behavior suggests a simplified deterministic (that is, fluid flow) model for delay in computer networks. The essence of the phenomenon is that delay remains essentially constant up to the critical threshold at which point the delay grows in an unbounded fashion. Such a simplification is shown in Fig. 3. In this figure we represent the delay characteristic in terms of two basic system parameters. The first parameter, T_0, is the constant delay experienced by messages so long as the network load γ is in the stable region ($\gamma < \gamma*$); the second parameter, γ^* is the network saturation load at which point $T \to \infty$. T_0 is simply calculated as the "no-load" delay and corresponds to the delay messages would experience in traveling through an unloaded network. For example, were we to use (6) as our model then

$$T_0 = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} \frac{1}{\mu C_i} \tag{9}$$

The calculation for $\gamma*$, the network saturation load, is quite straightforward. It corresponds to the smallest value of γ at which some (critical) channel is saturated; in terms of our earlier notation this is the point at which $\lambda_{i_0} = \mu C_{i_0}$ where i_0 is that critical (bottleneck) channel. Given a fixed routing procedure, one may simply calculate the set $\{\lambda_i\}$ for any value of γ; one must then examine all the ratios $\lambda_i/\mu C_i$ and identify i_0 as that channel with the largest such ratio. Recognize that we are talking about scaling all terms γ_{jk} (and therefore λ_i) by a common factor. The scaling factor is now adjusted to force $\lambda_{i_0} = \mu C_{i_0}$; the value of throughput γ at which this occurs is precisely $\gamma*$.

III. Design Issues

Whereas network analysis turns out to be "reasonably" straightforward, network design is not. In fact, network design is, in some sense, a "black art." Indeed, the difficulties come from a number of directions, including, for example, the following:

1) Cost is not linear with link capacity.
2) Transmission line tariffs are weird and illogical.

3) Performance is not linear with load.
4) The number of topologies to consider is enormous.
5) Data are usually missing and/or inaccurate.

The net result is that the solution to network design is to use heuristic design procedures that cleverly search over the space of possible topologies using fast evaluation techniques. The ability to evaluate a candidate design quickly is key to such a procedure. Fortunately, in Section II we have given analytical equations for such an evaluation; as mentioned in Section I, this is the best we can hope for in speed and simplicity of evaluation. The cleverness of the topological search procedure is the other key element in design. Here we are not as fortunate, and a number of such procedures have been proposed, each with its own "secret" internal heuristics, special data structures, etc. This is where the "black art" part of design comes in. For example, there is the Concave Branch Elimination Method (CBE) described in [9], [10] and the Cut-Set Saturation Method (a special case of Branch Exchange described in [8]). We describe the ideas behind these methods below. But first, let us set up the problem and discuss some manageable subproblems.

One way to state the formal design problem is as follows: Let us assume that we are given the locations of the N switching nodes. Let us also assume we are given the elements of the traffic matrix $\{\gamma_{jk}\}$. (Of course, it is almost never the case that these traffic values are known, and even if they are, they are certain to be time-varying with hour of day, day of week, week of year, etc. The way the designer gets around this is usually to assume a uniform traffic matrix if little is known about the entries, or to use as good an approximation as possible. Already we begin to see the "ad-hocness" of the design method manifest itself.) Our design task (i.e., the degrees of freedom which the design must resolve) is to choose a topology, to select the capacity of the links in this topology, and to design a routing procedure which will move the traffic along certain paths on the way from its origins to its destinations.

Our objective is to select these design parameters in a way which optimizes an objective function while meeting all the system constraints. If we choose to minimize the mean message delay T, while meeting a maximum allowable cost constraint, then the problem becomes:

Design Problem:

Minimize

$$T = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} T_i$$

By selecting:	Topology
	Channel Capacity Assignment
	Routing Procedure
Such that:	No more than D dollars are spent
	Traffic matrix is satisfied

That is, we are given D dollars to spend on the channel capacity assignment according to some given tariff structure.

The *dual* version of this problem is to minimize cost while not exceeding a maximum allowable mean message delay T_{MAX}, that is,

Dual Design Problem:

Minimize

$$D = \sum_{i=1}^{M} d_i(C_i)$$

By selecting:	Topology
	Channel Capacity Assignment
	Routing Procedure
Such that:	$T \leq T_{\max}$
	Traffic matrix is satisfied

Here, the number of dollars required to supply C_i units of channel capacity to the ith channel is denoted by $d_i(C_i)$, and is dictated by the tariff being used in the design process.

As stated above, both forms of this design problem are intractable for a number of reasons. First, the exact expression for T_i is unavailable in general; we get around this by using our Independence Assumption which results in an approximate expression for delay. Second, the number of topologies to consider is far too large and, in addition, we are dealing with the infamous multicommodity flow problem [8]; we get around this by using an appropriate heuristic (suboptimal) design procedure.

It turns out that there are some simpler subproblems which can be derived from this general design problem which *are* manageable, and which lead us to a heuristic solution to the general problem. First we consider the

Flow Assignment Problem:

Given:	Topology
	Channel Capacity Assignment

Minimize

$$T = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} T_i$$

By selecting:	Routing Procedure
Such that:	Traffic matrix is satisfied

Note that no cost constraint appears since all the money has already been spent on the given channel capacity assignment. This turns out to be a straightforward minimization problem of a convex function over a convex space. Solutions to this kind of problem abound in the literature. One particular solution which works especially well is the Flow Deviation Method [7], but many others also work here.

Next let us consider the

Capacity Assignment Problem:

Given:	Topology
	Routing Procedure

Minimize

$$T = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} T_i$$

By selecting:	Channel Capacity Assignment
Such that:	No more than D dollars are spent
	Traffic matrix is satisfied

This turns out to be a solvable problem whose difficulty depends upon the cost function $d_i(C_i)$. In particular, if the cost function is linear with capacity, that is, $d_i(C_i) = d_i C_i$, then the optimal capacity assignment turns out to be the following [1]:

$$C_i = \frac{\lambda_i}{\mu} + \left(\frac{D_e}{d_i}\right) \frac{\sqrt{\lambda_i d_i}}{\sum_{j=1}^{M} \sqrt{\lambda_j d_j}}, \quad i = 1, 2, \cdots, M \quad (10)$$

where

$$D_e \triangleq D - \sum_{i=1}^{M} \frac{\lambda_i d_i}{\mu} \quad (11)$$

We note that in this assignment, the ith channel is given its minimum required capacity (namely, λ_i/μ bits per second, which is the average traffic it must carry) plus an additional amount which is proportional to the cost-weighted traffic on this channel. If this optimal assignment is made, then the optimized (minimal) mean delay is given by

$$T = \frac{\overline{n}}{\mu D_e} \left[\sum_{i=1}^{M} \sqrt{\left(\frac{\lambda_i d_i}{\lambda}\right)} \right]^2. \quad (12)$$

If the cost function is not linear, then other methods may be used as described in [9].

Lastly, let us consider the combination of the last two subproblems, namely, the

Capacity and Flow Assignment Problem:

Given:	Topology

Minimize

$$T = \sum_{i=1}^{M} \frac{\lambda_i}{\gamma} T_i$$

By selecting:	Channel Capacity Assignment
	Routing Procedure
Such that:	No more than D dollars are spent
	Traffic matrix is satisfied

This turns out to be a much more difficult problem since we are trying to find the minimum of a concave function which contains many local minima. One solution to this is described in [10].

Now, the astute reader will notice that we have been cheating. Specifically, in all three of the subproblems, we assumed that someone else carried out the really difficult task of finding the best topology for us! This is truly the interesting part of the problem. However, we exonerate ourselves by noting that in the solution to the *Capacity and Flow Assignment (CFA)* subproblem, the topology that we begin with changes in the course of the solution, and certain channels are eliminated from the original topology. This is the essence of the Concave Branch Elimination (CBE) Method, whose name you now understand. The steps in the CBE method are, roughly, to begin with any initial topology, to pick a random initial flow and to carry out the CFA algorithm. This finds one minimum. We then repeat with different random flows until we run out of computational dollars. We then ask our boss for more money, which, if granted, we expend by beginning with a different initial topology. We repeat this until our boss gets stubborn and refuses to fund the project any further. At this point, we find the "best" of all the topologies we ended up with and declare this our suboptimal heuristic solution! Such is the nature of heuristic design.

As it turns out, other design procedures, such as the Cut-Saturation Method, lead to nearly identical cost/performance profiles for the suboptimal network designs as shown in Fig. 4.

Here we are plotting the throughput one can achieve if one spends the indicated number of dollars; all these network designs meet the delay constraint $T \leq T_{\text{MAX}}$. The interesting thing about this profile is that it clearly demonstrates that there is an economy of scale in network design. To see this, imagine that we wish to design a "small" network, namely, one which supports only a small amount of throughput. Such a network is shown at point A in the figure. We note that the slope of a line drawn from the origin to point A has a slope whose value is simply the throughput per dollar one can achieve by expending the number of dollars corresponding to point A. We see that

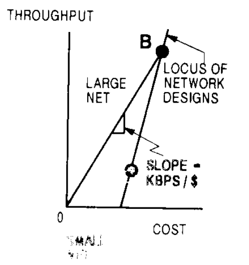

Fig. 4: Locus of suboptimal network designs.

the small network has a small slope. On the other hand, if we design a "large" network such as shown at point B, then we see that we achieve far more units of throughput for each dollar spent. Thus we have an economy of scale in network design.

IV. Control Issues

One of the most important considerations in computer networks is that of control. It is necessary to impose control procedures on networks for a variety of reasons in a variety of places with a variety of mechanisms. The two major forms of control are: routing control and flow control [11] and [12].

A. Routing Control

The routing control procedure in a network is simply a decision rule which determines where next to send a packet as it travels from switching node to switching node; the control of routing takes place at layer 3 of the 7-layer architecture.[1] This control procedure may be distributed or centralized, and quite a number of different types have been considered in the literature and in operational networks.

The original ARPANET routing procedure [9] was totally distributed such that all nodes in the network participated in the decision process through the periodic exchange of routing tables. The table in a given node contained estimates of delay from that node to all others in the net. Upon receiving a table from a given neighbor, a node would then estimate the delay from itself to this neighbor and add this estimate to every entry in that neighbor's received table. Having done this for all neighbors, a node would then select the minimum estimate across all neighbors for a given destination, and use that delay and that neighbor as its own estimated delay and routing decision for that given destination. This process would repeat periodically; it had the lovely property that it would converge to excellent paths in an adaptive fashion in the face of changing network conditions. This ability to adapt was a major benefit which was present in the early ARPANET and was also adopted by the Telenet public packet switched network.

The more recent routing procedures in the Internet use a centralized procedure whereby nodes set a threshold against which changed estimates are compared; once the threshold is exceeded, that node then floods the network with its new information, informing every network node of this change. As a result, every node has the same information as every other, and so when they all respond to the flood from a change, they calculate a new routing procedure using identical databases and arrive at the same routing values.

A centralized routing procedure is sometimes used, as for example in the early Tymnet data network. In this case, a special node known as the Supervisor is responsible for setting up paths for a session to use for its entire discourse between attached end-user devices. The Supervisor makes a central calculation, sets up the path, and then allows the two parties to communicate without further interference. The calculation is a shortest path calculation based on traffic, link capacity, and nature of the session.

In the solution to the *Flow Assignment Problem* of the previous section (whose solution really solves the routing control problem analytically), the Flow Deviation (FD) Method referred to uses a procedure which appeals to one's intuition. In particular, it selects which paths a flow should follow based on a shortest path calculation. The length of a link, as used in this shortest path calculation, is taken to be the incremental increase in network delay when an incremental amount of additional flow is placed on that link. In particular, the length, l_i of the ith channel, given that it is carrying an amount of flow equal to λ_i/μ is simply

$$l_i \triangleq \frac{\partial T}{\partial(\lambda_i/\mu)} = \frac{C_i}{\gamma[(C_i - \lambda_i/\mu)]^2}. \qquad (13)$$

These lengths are used in the shortest path calculation, and the resulting paths represent the "cheapest" (i.e., marginally best for reducing T) paths to which some of the current flow can be deviated. One must then determine how much of the current flow to deviate to these new paths. Once this is determined, the process may be repeated by recalculating new lengths using the updated flows, solving a new shortest-path problem, finding the correct flow to deviate, and so on. This (FD) iterative procedure continues until an acceptable performance tolerance is reached. Of course, the *Flow Assignment Problem* is posed as a design problem and is solved "off-line" in the design process. However, it may also be used as the basis for a distributed dynamic ("on-line") routing control procedure as described in [14] and [15]. In the dynamic case, the link lengths are calculated locally by each node and used to solve the shortest flow problem in a distributed fashion repeatedly.

B. Flow Control

We typically have a number of disparate devices attached to a computer network. As an example, consider the network shown in Fig. 5, where we show only two attached devices, namely, a slow-speed user terminal and a high-performance host processor. Now it does not take too much imagination to see that we face a problem here! These two devices behave very differently, and since they use a network between them for connectivity, we may run into

[1] Its appearance at the third layer is the usual place for routing control; however, it may appear at layer 2 as in the Frame Relay LAPD protocol which includes a subset of the full routing functionality at layer 2 [13].

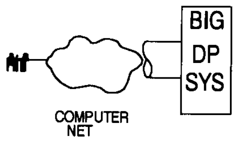

Fig. 5. The flow control problem.

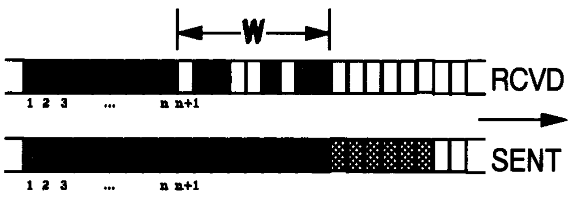

Fig. 6. Window flow control.

some serious problems. For example, one certainly does not want to interrupt the operating system of the host every time the terminal sends a byte across the net. (That is, you do not want to "nickel-and-dime" the host to death!) On the other hand, you certainly would not want to turn on a multimegabit per second host I/O channel and try to feed it to our low-speed terminal. (That is, you do not want to "hose" the terminal to death!) For indeed, the terminal will not take data at the I/O channel speed, but rather it will take them out at the low- speed rate it can handle, and the network will get totally stuffed with a huge data backlog very quickly. (And the network is a very expensive medium to be used for storage!) The point is that you must protect the terminal from the host, the host from the terminal, and the network from both of them!

The way to provide this protection is to *throttle* the input flows at the boundary of the network. This is called "flow control." There are a number of ways in which one can throttle the flow through a network. We will discuss a few of them here. First, there is the simple procedure of setting a maximum rate at which a user may input data; this is called *rate flow control.* A second method is to allow the user to input data in blocks by turning the user on and off in some fashion; this is called *batch flow control.* These two are really simple and unsophisticated methods.

More common methods for controlling the flow permitted between two communicating processes are based on a window or token control scheme as follows. Let us begin with *window flow control.* In Fig. 6 we show two time axes each broken into numbered blocks (which correspond to numbered messages). The lower time axis shows the history of messages which have been transmitted into the network. Each solid block identifies a message which has indeed been transmitted. Each cross-hatched block corresponds to messages which are waiting at the source but which are being held back from entering the network due to the flow control procedure. The upper time axis shows the history of messages which have been end-to-end acknowledged; that is, the source has been informed that the message was delivered. The solid blocks identify those which have been acknowledged. We note that all messages up to message n have been acknowledged; message $n + 1$ is the oldest message not yet acknowledged. In window flow control, we are allowed to transmit up to $W - 1$ messages into the network beyond the first unacknowledged message. Now, when the acknowledgment for message $n + 1$ eventually arrives, then we are allowed to advance the window by 3 and may then transmit messages $n + 1$, $n + 2$, and $n + 3$. By selecting the value of W, and by adjusting the speed with which acknowledgments are returned, we can clearly control the flow.

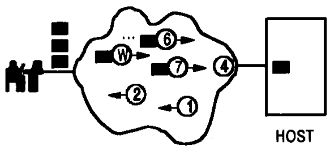

Fig. 7. Token flow control.

In Fig. 7, we show how a *"permit"* or *"token" flow control* scheme works. Here we see a flow between a terminal and a host. We note that there are three messages "in flight" toward the host; messages are represented by rectangles. Note that three messages are waiting at the source, but are not allowed into the network because the rule is that a message may not enter the network until one of W "free" tokens (permits) appears at the source terminal. Once a free token appears, it may be attached to a waiting message (at which time the token is marked as "busy") and then the two travel across the network to the destination. Once a message (with its token) arrives at the destination, the message is stripped off and delivered to the destination, the token is released (and marked "free") and then migrates its way back to the source to be reused, etc. In the figure, we see one message that has just arrived, and two free tokens migrating back to the source.

How do these two schemes differ? The answer is that in window flow control, the "tokens" must be used in sequential order (mod W); for example, we see that five tokens have already arrived at the source (the acknowledged messages beyond n), but cannot be used because token $n+1$ (mod W) must be used next. Both of these schemes are used widely in today's packet networks. The schemes we just described control the flow for a given process-to-process pair. Another method, known as Isarithmic Flow Control, is simply token flow control where the number of tokens in the entire network are shared among all communicating pairs in some fashion [16]; the problem with this scheme is that there is a major problem in ensuring that the tokens migrate to needy sources in a reasonable and fair fashion. For further discussion of flow control methods, the reader is referred to [9] and [12].

C. Flow Control Analysis and Design Issues

It turns out that routing procedures are relatively easy to design, but are hard to analyze in a dynamic environment. On the other hand, flow control procedures are difficult to design and are also difficult to analyze. In fact, control

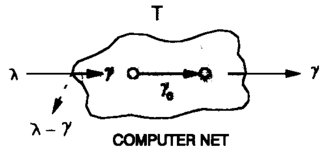

Fig. 8. Throughput, response time, and loss.

procedures in general tend to cause catastrophic failures in networks (some of which have been reported in the public newspapers over the past few years!). The reason for this is that the logic used to provide the control tends to be extremely complex, and the complexity leads to logic errors and thus to system crashes. Most large systems suffer from this problem, and the problem is especially insidious because the failures are hard to predict, and typically they lie latent in networks for years before the "right" conditions arise which trigger them. Such is the nature of any complex system.

In the window and token flow control procedures, we note that there is a fixed number of tokens flowing between source and destination. Thus one naturally looks to the theory of closed queueing networks [4], [9], [17] to provide an analytically tractable model. Fortunately, this theory has been fairly successful in allowing one to calculate the throughput for networks using such flow control. In addition, the theory of Petri Nets allows one to include timing constraints in the model, and has also been found to be effective in calculating throughput [18].

Rather than going into the details of the queueing network analysis, we prefer to focus on some of the system level tradeoffs to see how they might dictate our design philosophy. Specifically, we see that in any computer network, one is concerned with three competing performance measures: throughput, response time, and loss (or blocking). Of course, one would like to maximize the first and minimize the last two of these quantities. This is the tradeoff we wish to explore. These various quantities are shown in Fig. 8. We note that T is the mean response time of the network, and γ is the throughput carried by the network. In addition, we let λ denote the offered input traffic rate to the network. Furthermore, in the figure, we have shown the network capacity as γ_0 (messages per second).

Note that some of the offered traffic is "lost" due to the flow control scheme; this amount is simply $\lambda - \gamma$. Let us first focus on this, the flow control function $\gamma(\lambda)$; that is, how much traffic γ does the network carry when it is offered an amount λ? If throughput were our only consideration, then the ideal situation would be $\gamma = \lambda$ and we would have no lost traffic. However, since the network has a maximum capacity of γ_0, we see that the "ideal" flow control function is that shown in Fig. 9. Below the input rate of γ_0, we have no loss, and beyond that point, we have the minimum possible loss, namely, $\lambda - \gamma_0$. Also shown in the figure are other more realistic functions. For example, if we apply no flow control at all, then the "free-flow" curve is often the one that occurs. In this case, we see that for low input rates, we achieve nearly the ideal, but as

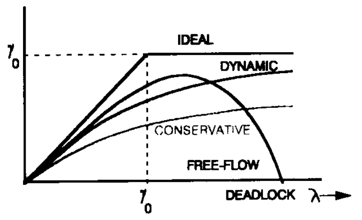

Fig. 9. The flow control function.

the load increases, we reach a peak and then begin to fall off as more load is applied; that is, as we push harder, we get less throughput! Eventually, these uncontrolled schemes can drive the system into a situation of very low throughput, or even zero throughput (also known as a deadlock). Such behavior may be found in systems other than networks; for example, thrashing in paged memory systems, traffic snarls on highways and city streets, water flowing out of a vessel, etc. Such a degradation is usually caused by the fact that some system resource is being wasted; in our case, it often is due to unnecessary retransmissions, or unnecessarily long paths for traffic, or buffering problems, etc. An alternative is to be extremely conservative in admitting traffic, in which case we usually achieve the behavior shown in the figure where we lose lots of traffic in light load, but do well as the load increases. On the other hand, if we introduce a dynamic flow control scheme (such as the throttling schemes discussed earlier), then we often achieve the behavior as shown for that case in the figure. Here we see that we do not do badly at low loads, and certainly do continue to increase throughput as we increase load, approaching the maximum network throughput fairly rapidly.

Now let us get to the basic tradeoffs. We begin with the tradeoff between throughput, $\gamma(\lambda)$ and the mean response time, $T(\gamma(\lambda))$. Clearly, we wish to have lots of throughput and small delay; unfortunately, these cannot both be achieved simultaneously, and so we wonder where is the correct "operating point" for the system. A quantity known as "power" (P) has been studied which combines these two performance variables into a single measure, and is defined as throughput divided by delay, namely

$$P = \gamma(\lambda)/T(\gamma(\lambda)). \tag{14}$$

We wish to find that optimum value of λ which maximizes P. It is easy to show [19] that this value is such that

$$T(\gamma(\lambda))/\gamma(\lambda) = dT(\gamma(\lambda))/d\gamma(\lambda). \tag{15}$$

That is, optimum power occurs when the input is such that the derivative of response time with respect to throughput is exactly equal to the value of response time divided by throughput. This situation is shown in Fig. 10 where we show an arbitrary response time versus throughput profile. In this figure, we also show that straight line out of the origin which has the smallest slope and which is tangent to the curve. The point of tangency defines the optimum operating point (i.e., it maximizes power). We note that the slope of this straight line is exactly the inverse of the

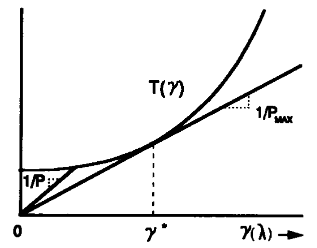

Fig. 10. Locating the optimum power point.

power; thus it is clear that we seek the minimum slope line that touches the curve. Note that this result is good for *any* flow control function $\gamma(\lambda)$ and for *any* response time function T.

If the response time profile had a sharp knee (similar to the curve shown in Fig. 3, then the optimum operating point would have been at γ^*, namely, the "knee" of the curve. This corresponds to exactly where your intuition would tell you to operate since it provides minimum response time and maximum throughput. In more general cases, the optimum power point essentially identifies the knee of the curve.

If we apply this principle of maximum power to a simple M/M/1 queueing system, we find that the optimum operating point is at $\rho = 1/2$ (i.e., $\gamma_{OPT} = \gamma_0/2$.) At this point, the response time is twice its minimum (the no-load delay) and the throughput is half its maximum. More importantly, $\overline{N}$, the average number of customers in the system at this point is exactly $\overline{N} = 1$. Moreover, whereas the optimum value of ρ changes for the M/G/1 queueing system as a function of the general service time, it remains the case that optimum power occurs when $\overline{N} = 1$! Now, let us consider a simple single-node queueing system in which you have complete control over how messages (customers) enter the system; suppose you were asked to find that input control scheme which maximizes power for the system. The obvious answer is that control policy which allows exactly one message in the node at a time, and only when that message leaves would you insert the next message. Of course, we cannot control traffic in so simple a fashion in a real system. What is really interesting is that our analytic result above says that we should operate the system at a load such that *on average* we should have one customer in the system! Thus our analytic result matches our intuitive reasoning very well.

This result carries over very nicely to the case of computer networks. It states that the optimum number of messages in flight in an n-hop path through the network should equal exactly n messages (i.e., one per hop). Moreover, if we ask what should be the optimum value of the window W in an n-hop path using token flow control, then once again, we find the answer is $W = n$.

One can also include the effect of loss in the definition of power. As shown in [20], it is still the case that the optimum operating point is exactly that point where our "deterministic reasoning" would tell us it should be. In fact, this result extends to much more general definitions of power where the only requirement is that the power be an increasing function of throughput as well as a decreasing function of both delay and loss.

V. Gigabit Networks

As we enter the 1990's we find that we are rapidly moving into a world of gigabit per second networks. We must ask ourselves if gigabits represent just another step in an evolutionary process of greater bandwidth systems, or, if gigabits are really different? In the opinion of this author, gigabits are indeed different, and the reason for this difference has to do with the effect of the latency due to the speed of light.

Let us begin by examining data communication systems of various types. It turns out that there are a few key parameters of interest in any data network system. These are:

1) C= capacity of the network (megabits per second: MBPS)
2) b = number of bits in a data packet
3) L = length of the network (miles).

It is simplest to understand these quantities if one thinks of the network simply as a communication link. One can combine these three parameters to form a single critical system parameter, commonly denoted as a, which is defined as:

$$a = 5LC/b \qquad (16)$$

This parameter is the ratio of the latency of the channel (i.e., the time it takes energy to move from one end of the link to the other) to the time it takes to pump one packet into the link. It measures how many packets can be pumped into one end of the link before the first bit appears at the other end [21]. The factor 5 appearing in the equation is simply the approximate number of microseconds it takes light to move 1 mi.[2] Now, if we calculate this ratio for some common data networks, we find the values shown in Table 1: Note the enormous range for the parameter a. At one extreme, namely, local area networks, it is as small as 0.05, while at the other extreme, namely a cross-country gigabit fiber optic link, it is as large as 15 000. This is a range of nearly six orders of magnitude for this single parameter!

We see that a grows dramatically when we introduce gigabit links. So we naturally must ask ourselves if networks made out of gigabit links are different in some fundamental way from those made out of kilobit or megabit links. There are two cases of interest to consider. First, we have the case that a large number of users are each sharing a small piece of this large bandwidth. In this case, it is fairly clear that to each of them, a gigabit network looks no different than today's networks.

However, if we have a few users each sending packets and files at gigabit speeds, then we do see a change in behavior and we do run into new problems. At these speeds,

[2]Throughout this paper we make the simplifying assumption that light propagates in a fiber optic channel as quickly as it propagates in free space.

Table 1 The Enormous Variation in the Ratio a = Propagation Delay/PKT TX Time

	Capacity C (MBPS)	PKT Length (b) (bits)	Prop Delay (τ) (μs)	Ratio "a"
Local-area net	10.00	1000	5	0.05
Wide-area net	0.05	1000	20 000	1.00
Satellite	0.05	1000	250 000	12.50
Fiber link	1000.00	1000	15 000	15 000.00

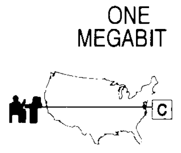

Fig. 11. Transmitting a 1-Mb file across the country.

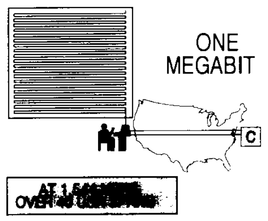

Fig. 13. Transmitting over a 1.544 MBPS T1 line.

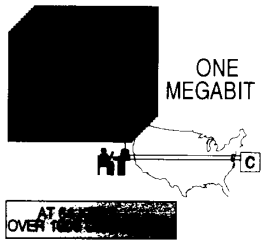

Fig. 12. Transmitting over a 64 KBPS line.

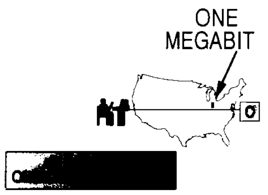

Fig. 14. Transmitting over a 1.2 GBPS OC-24 line.

a gets very large. To see the effect of this change, let us consider the following scenario. Assume we are sitting at a terminal and wish to send a 1-Mb file across the United States to some remote computer as shown in Fig. 11.

Now, if the speed of the communication channel we have available is 64 kb/s–KBPS (as in, say, an X.25 packet network), then, as shown in Fig. 12, the first bit of this transmission will arrive at the East Coast computer after approximately 1000 b have been pumped into the channel. Thus we see that the channel is buffering roughly 0.001 of the message; that is, there is 1000 times as much data stored in the terminal's buffer as there is in the channel. Clearly, if we had a higher speed channel, the time to transmit our 1-Mb file could be reduced. That is, we can benefit from more bandwidth.

Thus let us now increase the speed of the channel and use a T1 channel (1.544 MBPS). In Fig. 13 we show this new configuration. Now we find that the terminal is buffering roughly 40 times as much data as is the channel. Once again, we see that we can benefit from more bandwidth.

Let us now increase the channel speed to a gigabit channel; in particular, we will assume a 1.2-Gb/s–GBPS link (the OC-24 SONET offering). This case is shown in Fig. 14 where we see the entire 1-Mb file as a small pulse moving down the channel! Indeed, the pulse occupies roughly only 0.05 of the channel "buffer." It is now clear that more bandwidth is of no use at all in speeding up the transmission of the file; it is the *latency* of the channel that dominates the time to deliver the file!

Therein lies the fundamental change that comes about with the introduction of gigabit links into nationwide networks. Specifically, we have passed from the regime (of pre-gigabit networking) in which we were *bandwidth limited*, to the new regime of being *latency limited* in the post-gigabit world. Things do indeed change (as we shall see below). The speed of light is the fundamental limitation for file transfer in this regime! And the speed of light is a constant of nature which we have not yet been able to change!

In the considerations above, we assumed that our file was the only traffic on the link. Let us now consider the case of competing traffic with smaller packets. Indeed, let us now assume that we have the classical queueing model of a Poisson stream of arriving messages requesting transmission over a communication link, where each message has a length which is exponentially distributed with a mean of 128 bytes (i.e., an M/M/1 queueing system). If, as usual, we let ρ denote the system utilization factor, then $\rho = \lambda(1024/C)$ where λ is the arrival rate (messages per microsecond) and C is the channel capacity (MBPS). In this situation, we know that T, the mean response time (seconds) of the system (i.e., the mean time from when the

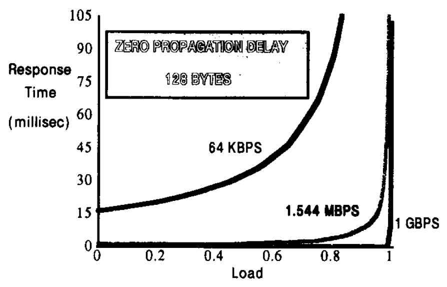

Fig. 15. Mean response time at zero propagation delay.

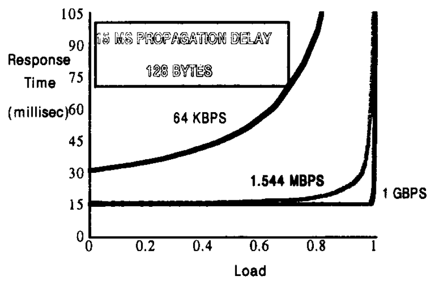

Fig. 16. Mean response time at 15-ms propagation delay.

message arrives at the tail of the transmit queue until the last bit of the message appears at the output of the channel, including any propagation delay), is given by

$$T = \frac{1024/C}{1-\rho} + \tau \tag{17}$$

where τ is the propagation delay (i.e., the channel latency) in seconds.

Let us ask ourselves if gigabit channels actually help in reducing the mean response time T. In Fig. 15, we show the mean response time (in milliseconds) versus the system load ρ for three different channel speeds. In this figure, we assume that the speed of light is infinite, and so $\tau = 0$. The channel speeds we choose are the same as those considered above, namely 64 KBPS, 1.544 MBPS, and 1.2 GBPS. We note a significant reduction in T when we increase the speed from 64 KBPS to 1.544 MBPS; thus the faster $T1$ channel helps. However, note that when we go from 1.544 MBPS to 1.2 GBPS, we see almost no improvement. (The only region in which there is an improvement with gigabits is at extremely high loads, a situation to be avoided for other reasons.) As far as response time is concerned, gigabits do not help here!

One might argue that the assumption of zero propagation delay has biased our conclusions. Not so; in Fig. 16 we show the case with a 15-ms propagation delay (i.e., the propagation delay across the USA) and we see again that gigabits do not help.

We can sharpen our treatment of this latency-versus-bandwidth discussion as follows. Let us assume that we have an M/M/1 model as above, where the messages have an average length equal to b bits. Assume we wish to transmit these files across the United States, as in the

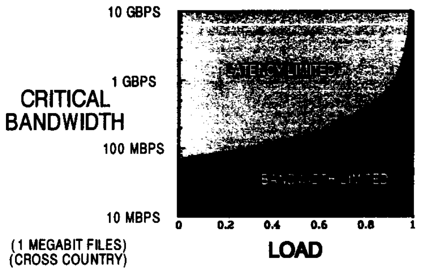

Fig. 17. The bandwidth-latency tradeoff for a 1-Mb file.

earlier figures. Now, as can be seen from (17), there are two components making up the response time; namely, the queueing-plus-transmission time delay (the first term in the equation) and the propagation delay (τ). In this paper, we have been discussing the relative size of each of these and we referred to regions of bandwidth-limited and latency-limited systems. Let us now make those concepts more precise. We choose to define a sharp boundary between these two regions. In particular, we define this boundary to be the place where the two terms in our equation are exactly equal; namely, where the propagation delay equals the queueing-plus-transmission time delay. From (17) we see that this occurs when the bandwidth of the channel takes on the following critical value:

$$C_{\text{CRIT}} = \frac{b}{(1-\rho)\tau} \tag{18}$$

In Fig. 17, we plot this critical value of bandwidth (on a log scale) versus the system load ρ; we have drawn this plot for the case of τ = 15 ms and a message length of 1 Mb. Above this boundary, the system is latency-limited which means that more bandwidth will have negligible effect in reducing the mean response time T . Below this boundary, the system is bandwidth-limited which means that it can take advantage of more bandwidth to reduce T. Note for these parameters, that the system is latency-limited over most of the load range when a gigabit channel is used; this means that for these parameters, a gigabit channel is overkill so far as reducing delay is concerned.

We repeat this plot in Fig. 18 for a number of different message sizes. Without labeling the regions, the same comments apply, namely, systems above the curve are latency-limited, and below they are bandwidth-limited. We note that gigabit channels begin to make sense for messages of size 10 Mb or more, but are not helpful for smaller file sizes. This comment about message size refers to the file size that the user application generates; the fact that ATM uses 53 byte cells has little to do with this comment.

Figures 17 and 18 apply to the case of a cross-country link (i.e., propagation delay of roughly 15 ms). For other than τ = 15 ms, the critical bandwidth which defines the boundary is given from (18).

There are a number of other issues to be addressed in gigabit nets. Consider the example from the previous section, namely, a gigabit link spanning the United States. Suppose we start transmitting a file a time t=0. Roughly 15

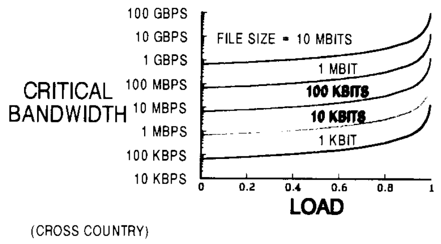

Fig. 18. The bandwidth–latency tradeoff for various size files.

ms later, the first bit will appear across the country. Now suppose that the receiving process decides immediately that it cannot accept this new flow which has begun. By the time the first bit arrives, however, there are roughly 15 million bits already in the pipe heading toward this receiving process! And, by the time a stop signal reaches the source, another 15 million bits will have been launched! It does not take too much imagination to see that we have a problem here. It is basically a congestion control and flow control problem as discussed in Section IV-B. Clearly, a closed control feedback method of flow control is too sluggish in this environment (due, once again, to latency). Some other forms of control must be incorporated. For example, one could use rate flow control in which the user is permitted to transmit at a maximum allowable rate.

Another issue has to do with the maximum attainable efficiency that one can obtain by taking advantage of statistical multiplexing of bursty sources in a gigabit environment. If we have a large number of small bursty sources, then statistical multiplexing takes exquisite advantage of the Law of Large Numbers [4], and allows one to drive these channels at very high efficiencies. However, if we have a small number of large sources, then the multiplexing does not usually lead to very high efficiencies. This is because statistical smoothing of a small number of sources is not sufficient to bring about the advantages of statistical multiplexing. Furthermore, if we have a large number of nonhomogenous sources, one must calculate the effective number of such sources in order to calculate the efficiency to be expected from multiplexing [22].

VI. Conclusions

In this paper, we have taken a tour over the analytic landscape of computer networks, flying fairly high over the terrain. The purpose was to give the reader the understanding as to the approximate state of the art in performance modeling and analysis of computer networks. The reader should be aware that currently there is a great deal of effort going into network modeling and analysis. The driving force behind this effort has been the move toward broadband networks. These networks involve parameter ranges and tradeoffs that we have not seen before. The switch has become the economic and performance bottleneck of the system. Thus we are in the process of inventing new protocols and architectures to give us access to the very high bandwidths afforded us by fiber optic channels. As a result, these new protocols and architectures require modeling, analysis and design.

We have introduced some of the basic modeling and evaluation tools for networks. We have glimpsed some of the tradeoffs that must be understood. And most of all, we have just scratched the surface of gigabit networks; they are not yet understood. There is currently a significant effort underway to develop the modeling and analytic tools to provide this understanding.

References

[1] L. Kleinrock, *Communication Nets; Stochastic Message Flow and Delay.* New York: McGraw-Hill , 1964. (Out of print. Reprinted by Dover Publications, 1972.)

[2] P. Baran, , "On distributed communications," *RAND Series Reports,* Aug. 1964.

[3] J. D. C. Little, "A proof of the queueing formula $L = \lambda W$," *Oper. Res.*, vol. 9, pp. 383–387, 1961.

[4] L. Kleinrock, *Queueing Systems, Vol I: Theory.* New York: Wiley, 1975.

[5] O. J. Boxma, "Analysis of models for tandem queues," Ph.D. dissertation, University of Utrecht, Utrecht, The Netherlands, 1977.

[6] I. Rubin,"Communication networks: Message path delays," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 738-745, 1974.

[7] 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.

[8] H. Frank and I. T. Frisch, *Communication, Transmission, and Transportation Networks.* Reading, MA: Addison-Wesley, 1971.

[9] L. Kleinrock,*Queueing Systems, Vol II: Computer Applications.* New York: Wiley, 1976.

[10] M. Gerla, "The design of store-and-forward (S/F) networks for computer communications," Univ. of California, Los Angeles, School of Eng. and Appl. Sci., Eng. Rep. UCLA-ENG-7319, 1973.

[11] M. Gerla and L. Kleinrock, "Flow control: A comparative survey," *IEEE Trans.Commun.*, vol. COM-28, no. 4, pp. 553--574, Apr. 1980. Also published in *Computer Network Architectures and Protocols,* P. Green, Ed. New York: Plenum, 1982, pp. 361–412.

[12] D. Bertsekas and R. Gallager, *Data Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

[13] S. J. Lowe, "Plugging into frame relay for speed and flexibility," *Data Commun.Mag.*, pp. 54–62, Apr. 1990.

[14] R. G. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE Trans.Commun.*, vol. COM-23, pp. 73–85, 1977.

[15] D. P. Bertsekas, E. M. Gafni, and R. G. Gallager, "Second derivative algorithms for minimum delay distributed routing in networks," *IEEE Trans.Commun.*, vol. COM-32, pp. 911–919, 1984.

[16] D. W. Davies, "The control of congestion in packet switching networks," in *Proc. 2nd ACM IEEE Symp. on the Optimization of Data Communications Systems*(Palo Alto, CA, Oct. 1971.), pp. 46–49.

[17] T. G. Robertazzi, *Computer Networks and Systems: Queueing Theory and Performance Evaluation.* New York: Springer-Verlag, 1990.

[18] M. A. Marsan, G. Balbo, and G. Conte, *Performance Models of Multiprocessor Systems.* Cambridge, MA: MIT Press, 1986.

[19] L. Kleinrock, "On flow control in computer networks" in *Conf. Rec., Proc. Int. Conf. on Communications,* vol. II (Toronto, Ont., Canada, June 1978), pp. 27.2.1–27.2.5.

[20] ——, "Power and deterministic rules of thumb for probabilistic problems in computer communications," in *Conf. Rec., Proc. Int. Conf. on Communications* (Boston, MA, June 1979), pp. 43.1.1–43.1.10

[21] ——, "Channel efficiency for LANs", *Local Area & Multiple Access Networks* , R.L. Pickholtz, Ed. Rockville, MD: Computer Sci. Press, 1986, pp. 31–41.

[22] ——, "Performance of distributed multi-access computer communication systems," in *Proc. IFIP Congr.77*, Aug. 1977, pp. 547–552.

Leonard Kleinrock (Fellow, IEEE) received the B.S. degree in electrical engineering from the City College of New York, New York, NY, in 1957 and the M.S.E.E. and Ph.D.E.E. degrees from the Massachusetts Institute of Technology, Cambridge, MA, in 1959 and 1963, respectively.

He is currently Chair and Professor of Computer Science at the University of California, Los Angeles, where he has been since 1963. His research interests focus on performance evaluation of high-speed networks and parallel as well as distributed systems. He has had over 180 papers published and is the author of five books. He is the principal investigator for the DARPA Advanced Networking and Distributed Systems grant at UCLA. He is also founder and CEO of Technology Transfer Institute, a computer communications seminar and consulting organization located in Santa Monica, CA. He has received numerous best paper and teaching awards, including the ICC 1978 Prize Winning Paper Award, the 1976 Lanchester Prize for outstanding work in Operations Research, and the Communications Society 1975 Leonard G. Abraham Prize Paper Award. In 1982 he received both the CCNY Townsend Harris Medal and was co-winner of the L. M. Ericsson Prize, presented by His Majesty King Carl Gustaf of Sweden, for his outstanding contribution in packet switching technology. In 1986, he received the 12th Marconi International Fellowship Award, presented by His Royal Highness Prince Albert, brother of King Baudoin of Belgium, for his pioneering work in the field of computer networks. Also in 1986 he received the UCLA Outstanding Teacher Award, and in 1990, the ACM SIGCOMM Award recognizing his seminal role in developing methods for analyzing packet network technology.

Dr. Kleinrock is a member of the National Academy of Engineering, is a Guggenheim Fellow, and a founding member of the Computer Science and Telecommunications Board of the National Research Council.