INTRODUCTION

In this paper we describe the features of a program designed to simulate computer networks. The networks are assumed to consist of hosts attached to one or more network nodes (communication units) which in turn communicate with one another via one or more communication channels. The program accounts for user-host, host-node, node-node, node-host, and host-user protocols, as well as network topology and hardware characteristics. As reported in a later section, the program has been primarily used to conduct simulations to ascertain the performance (in terms of message or packet throughput and delay) for a variety of node-node channel access protocols, for nodes connected to a single common communication medium (e.g. a multi-drop cable or radio channel). It is, however, possible to accommodate networks in which nodes may be connected to several separate communication channels. Further, it is possible to simulate the behavior of store-end-forward networks as they can be viewed as a network in which the nodes are connected to multiple communication links. For either class of network the program can be used to verify protocol correctness or to assess network performance.

MODEL PROGRAM FEATURES

The model program is designed (layered) to account for interactions at the user-host, host-node, node-node interface levels. The model program is constructed in a highly modular manner, so that alterations at one interaction level can easily be made without affecting the other layers. That is, the program is designed to accommodate alterations for the inclusion of additional or altered features. The program was developed along modular lines so as to make it more receptive to the inclusion of new protocols, buffering strategies, etc. Its basic organization accommodates the following characteristics.

Host-User Characteristics—Including message inter-generation and length modules, specification of the node or nodes to which a host is attached, and modules to handle the host-node message transfer mechanisms, including considerations of available buffer storage in both the host and node, and message addressing.

Node-Characteristics—Including queues and buffers for outgoing messages and message reassembly (from packets), buffers for retransmission of packets (messages) and acknowledgments.

- Additionally, node modules characterize the channel access protocol, i.e., node-node interactions (discussed later) and node-host interfaces including message addressing and reassembly issues. Finally, modules are included to accommodate channel-error characteristics (due to noise) and error-handling procedures (for errors resulting from noise or message collision, as applicable).

Measures—The model program modules currently collect data to provide estimates, where applicable, on:

1. Channel throughput.
2. Average node dependent transmission delays (message or packet).
3. The total offered channel traffic.
4. The expected length of the various node queues.
5. The number of doubly-received packets (messages).
6. The number of packet collisions.
7. Graphs for delay/throughput and delay/offered traffic.

Additionally, the modules will produce a full trace for each message that enters the network, from which additional information can be abstracted.

NODE-NODE CHANNEL ACCESS PROTOCOLS

We have stated that, on the one hand, the model program is constructed in a highly modular fashion to mitigate the process of altering one or more levels of protocol. We have, on the other hand, also attempted to make the extant modules general so that changes can be accomplished by varying module parameters rather than module structure. It is possible, for example, to characterize seemingly unrelated channel access protocols via scheduling functions and the notion of node-groupings, and to capture the essence of network topology in the connectivity or distance-control matrix.
To explain these notions we must imagine the $K$ nodes to be mapped by a grouping function $g(i)$, into $m$, $1 \leq m \leq k$, groups such that each node with index-$i$ is a member of one and only one group. For purposes of scheduling transmission times a node assumes the group identity, i.e., its identifier becomes the index of the group of which it is a member.

The LOS (line-of-sight) matrix, $X$, for a network is $K \times K$ and has entries $x_{ij}$, such that:

$$x_{ij} = \begin{cases} 1 & \text{node-i to node-j} \\ 0 & \text{node-i not to node-j} \end{cases}$$

where $\leftrightarrow$ denotes the presence of a direct point-to-point channel (not necessarily dedicated) between node-$i$ and node-$j$, and $\not\leftrightarrow$ its absence. Using $X$, the connectivity matrix DCM (distance control matrix) can be constructed. The DCM matrix has entries $d_{ij}$ such that:

$$|d_{ij}| = \text{minimum number of node-node transmissions required for a packet (message) to propagate from node-i to node-j}.$$  

The value of sign $(d_{ij})$ can be used for secondary purposes and an example will be given below. Thus a DCM can be established to represent, for example, a star or loop network, a network organized around a multiple drop cable, a network based upon a radio channel with an arbitrary dispersion of hidden nodes, or even a store and forward network. A sample network is shown in Figure 1a, and its corresponding DCM matrix is given in Figure 1b.

Using the grouping function, $g$, the connectivity matrix, DCM, and the scheduling function a wide variety of network configurations and transmission protocols are easily represented. To so demonstrate, we must first define the sched-
Algorithm A

1. Determine, $T$, the next scheduling period leading edge.
2. Compute the next potential transmission time (for the ready node with index-$i$) by $t_i = T + SF(g(i), j)$, with $j$ the group identification of the node which successfully transmitted last.
3. At time, $t_i$, if the channel is sensed idle (as determined by the LOS relationships given by the DCM) then transmit; otherwise return to Step 1.

The value of the scheduling period leading edge, $T$, is usually given by the trailing edge of a transmission (successful or not), or by the value of a counter which is zeroed and begins to increment at the termination of every transmission (used in the case where nodes approach an idle channel). Details on the determination of $T$ can be found in References 1 and 3.

The expressive power of Algorithm A together with $g(\cdot)$, $SF(\cdot)$ and the DCM is demonstrated by Figure 2, wherein networks with a wide variety of channel access protocols are characterized by various configurations of $g$, $SF$, and DCM values. Descriptions of the protocols mentioned in Figure 2 can be found in Reference 7 (ALOHA, CSMA variants, TDMA), Reference 8 (MSAP), Reference 1 (BRAM), Reference 2 (parametric BRAM), and Reference 3 (SUPBRAM).

Some are applicable to a cable or radio channel (e.g., CSMA, BRAM) or to radio channels with a network containing hidden nodes (e.g., BRAM, SUPBRAM, etc.). In the case of BRAM or SUPBRAM operating in a radio network with hidden nodes, the DCM sign is used to convey additional message routing information. A sample DCM with routine information is given in Figure 1b, and Reference 3 should be consulted for details.

Other protocols can also be captured by the structures, and to imagine doing so the reader must bear in mind that while certain protocols may not be realized by these constructs (in an actual system) they can be represented by them for simulation.

FORMAL VALIDATION OF MODELS

Typically a simulation model program is validated employing pilot runs and statistical tests. It is possible, however, to formally validate the model program’s input/output behavior against the actual system. Such validation can thus serve as a kind of ‘proof of correctness’ of the model program. The methodology employed is described in Reference 10 and in this section, for demonstration, we apply it to a simple network model which employs a slotted ALOHA transmission protocol. The reader should be assured, however, that the technique is applicable to any model program (configuration, protocol, etc.) constructed.

The methodology is based on the notions of base model, an experimental frame and a lumped model. In summary (consult Reference 10 for details), a base model is a model capable of accounting for all the input/output behavior of the actual system. As a consequence it must be faithful, detailed and consequentially complex representation of the system.

An experimental frame characterizes a limited set of circumstances under which the actual system is to be observed. Thus the experimental frame serves to define the subset of allowable input/output behaviors which are of interest.

The lumped model results from a process of abstraction by taking the base model and simplifying it (by grouping or lumping components, etc.) so that it accounts for the input/output behaviors specified by the given experimental frame, but not necessarily others.

The validation involves establishing a structural homomorphism between the base model and the lumped model as determined by the well defined experimental frame. In the network context, the base model is required to account for all network nodes, queues, protocols, etc. as encountered in the actual system.

To establish a homomorphism between the base and lumped models a formal machinery is necessary. The methodology of Reference 10 proceeds by first generating an informal description of the models in terms of system components, descriptive variables (descriptive of the components) and component interaction rules. From the informal description a formal discrete event system specification (DEVS) springs. The formal description is given in terms of:

$$a_1, a_2, \ldots, a_n$$  — The input variables,
$$\beta_1, \beta_2, \ldots, \beta_m$$  — The state variables, and
$$\delta_1, \delta_2, \ldots, \delta_k$$  — The output variables,

with the cross product of the ranges of these variables giving the set of INPUTS, STATES, and OUTPUTS of the system.
## Protocols and Configurations

<table>
<thead>
<tr>
<th>Protocol name</th>
<th>SF(i,j) =</th>
<th>g(i) =</th>
<th>LOS/DCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALOHA [7,9]</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1-persistent CSMA</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>p-persistent CSMA [7]</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>non-persistent case CSMA [7]</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>prioritized CSMA see [4]</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TDMA (time division multiple access)</td>
<td>i packet</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>[7,9]</td>
<td>transmission time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSAP see [8]</td>
<td>(i-j+k) mod k i ≠ j</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>BRAM or centralized polling [1]</td>
<td>(i-j+m) mod m i ≠ j</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>SUPBRAM [3]</td>
<td>See [3]</td>
<td>1 ≤ g(i) ≤ m arbitrary, d_{ij}</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2**—Protocols currently captured by g( ), SF(,) and DCM configurations. k = number of nodes in network; y = topology-dependent value; LOS = within line of sight and range.

To account for the discrete time points at which events (inputs, outputs, state changes) occur, the notion of a hatching time is introduced, to specify the times at which events occur. The next hatching time is specified by examination of a non-ordered set of linearly decreasing variables, T1, T2, ... , Tk, (a kind of sequencing set) such that at time t, the next event time (hatching time) is given by τ(s) = t + min{T1, T2, ... , Tk} where s refers to the state variables. The formalism is completed by specifying first:

\[ \delta \varphi: STATES \rightarrow STATES \]

which describes the internal or endogenous state transition that occurs in traversing from time t to time \( t^+ \). specifying second:

\[ \delta ex: Q \times INPUTS \rightarrow STATES \]

which characterizes the transitions brought about by externally-generated events, with Q denoting the set of pairs \((s, e)\) where s denotes system state (as before) and \( e \in \mathbb{R}_0^+, \lambda(s, e)\) (i.e., a real number in the range \((0, \lambda(s, e))\), and finally specifying:

\[ \lambda: Q \rightarrow OUTPUT \]

the output function. Taken together, the set \( \{INPUTS, STATES, OUTPUTS, \delta \varphi, \delta ex, \lambda, \tau\} \) constitutes the DEVS for the model (lumped or basic).**

To apply the methodology we must first develop a DEVS for the base model, then determine the experimental frame from which the lumped model and its DEVS can be derived.

Owing to limitations of space, we will not present the

** Additional discussion can be found in Reference 10.
DEVS for the base model, but instead, we will first informally describe how our sample lumped model results from the base model; second, we give the DEVS for the lumped model; and finally informally establish the required homomorphism, and hence the validation of the lumped model.

To proceed, an experimental frame must be specified, and for simplicity we assume the experimental frame as given by the single variable which gives channel utilization (under the assumption of a slotted ALOHA protocol as assumed earlier). The lumped model then results in:

1. Omitting in the lumped model variables, components, and interactions which account for user→host→node relationships.
2. Accounting for the omissions of (1) by replacing certain deterministic variables having to do with packet length, etc. by variates drawn from distributions determined appropriate by examination of the omissions.
3. Coarsening in the lumped model the range of certain descriptive variables in the base model (e.g., packet identification in the lumped model need only specify the associated source node, while in the base model packet identification must include source node identification, destination, node identification, sequence number within a message, message number, etc.) and eliminating several queues.

The network lumped model informal description

Under the assumptions given above the lumped model consists of components, descriptive variables and interactions as given next.

Components—Source, packet-queue, retransmission-queue, channel, with their obvious interpretations.

Descriptive Variables—For the source we have (the variable) NEW PACKET which can assume values \( \chi \), with range \( \chi \in \{0, 1, 2, \ldots, K\} \). This specification is compactly written as:

\[
\text{NEW.PACKET: } \chi \in \{0, 1, 2, \ldots, K\}
\]

for a \( K \) node network. Additionally we have,

\[
\text{PACKET.QUEUE: } y_i \in \mathbb{Z}^+
\]

where \( y_i \) gives the length of the queue at node \( i \), and

\[
\text{RETR.QUEUE: } (Z_i, T_i) \in \mathbb{Z}^+ \times \mathbb{R}
\]

with \( Z_i \) denoting the length of the retransmission queue at node \( i \) and \( T_i \) giving the scheduled retransmission time with \( T_i \) a variable drawn from a retransmission time generator with seed \( r_i \in [0, 1] \). For the channel we have

\[
\text{TRANS.TIME.LEFT}_i = \sigma_i \in [0, \text{Packet transmission time}]
\]

with \( K \), packet transmission time (PTT), and the retransmission time (SCHEDULED.TRANS.TIME) distribution established parameters.

Component Interactions—Component interactions are specified by noting that a newly-arrived packet joins PACKET.QUEUE, according to \( \chi \), the value assigned NEW.PACKET, causing \( Z_i \) to be incremented. If \( Y_i = 1 \) the node proceeds to transmit by increasing \( T_i \) to the value packet transmission time. If \( Y_i > 1 \), a transmission is scheduled for the time when \( \sigma_i \) becomes zero. If \( \sigma_i \) is reset while another \( \sigma_j, j \neq i \), is non-zero a retransmission time is drawn and \( Z_i \), and \( T_i \) are adjusted.

Formal lumped model description

The formal description constitutes the DEVS for the lumped model and contains the input variable NEW.PACKET such that:

\[
\text{INPUTS} = \{\phi, 1, 2, \ldots, K\}
\]

Further, using the state variables given in the informal description, we obtain:

\[
\text{SESTATES} = \{\text{RETR. TIME.SEED, PACKET.QUEUE}_1, \text{RETR.QUEUE}_1, \text{CHANNEL}_1, \\
\text{PACKET.QUEUE}_2, \text{RETR.QUEUE}_2, \text{CHANNEL}_2, \\
\vdots \\
\text{PACKET.QUEUE}_K, \text{RETR.QUEUE}_K, \text{CHANNEL}_K\}
\]

The time advance function becomes:

\[
\tau(s) = \min\{\sigma_1, \sigma_2, \ldots, \sigma_K, T_1, \ldots, T_K\}
\]

and for simultaneous events we choose the tie-breaking rule function:

\[
\text{SELECT (} \{\text{TRANSM. TIME.LEFT}, \text{SCHEDULED.TRANS. TIME}\}\) = \text{TRANSM. TIME.LEFT}
\]

otherwise: \( \text{SELECT (} \{X\} \) =\( X \) in the absence of simultaneous events)
The next hatching time will therefore be given by:
\[ t_{hi} = t_j + \min\{I_j, T_j\} \]

If we let \( T_i = \min T_i \) and \( \sigma_i = \min \sigma_i \), then \( \delta \phi \) is given by:

If \( T_j < \sigma i \)

and if \( \sigma i = 0 \)
\[
S = Y g \left( \begin{array}{c} I_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ T_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ \sigma_i - I_j \ , \ \gamma_1 \\ I_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ T_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ \sigma_i - T_j \\ y_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ T_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ \sigma_i - T_j \\ y_k \in \mathbb{Z} \cup \{ -\infty \} \ , \ T_k \in \mathbb{Z} \cup \{ -\infty \} \ , \ \sigma_i - T_k \\ \end{array} \right)
\]

and if \( \sigma i > 0 \)
\[
S = Y g \left( \begin{array}{c} I_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ T_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ \sigma_i - I_j \ , \ \gamma_1 \\ I_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ T_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ \sigma_i - T_j \\ y_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ T_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ \sigma_i - T_j \\ y_k \in \mathbb{Z} \cup \{ -\infty \} \ , \ T_k \in \mathbb{Z} \cup \{ -\infty \} \ , \ \sigma_i - T_k \\ \end{array} \right)
\]

where \( X = X = \{ x < 0 \} \) and \( \Gamma(r_i) \) denotes a drawing from a stream with seed \( r_i \), and initially \( \sigma_i = T_i = 0 \), and \( T_i = -\infty \) if \( t_{\text{out}} \). In the absence of external events, i.e., when \( \text{SOURCE} = \emptyset \), the next hatching time, \( t_{\text{out}} \), is given by \( t_{\text{out}} = t_j + \min(\sigma_i, T_j) \). For times, \( t \), at which \( \text{SOURCE} = m, i \leq m \leq k \), the state at time \( t + t \) is given by:

\[
\delta_{\text{ext}}(s, t, m) = \begin{cases} 
I_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ T_j \in \mathbb{Z} \cup \{ -\infty \} \ , \ \sigma_i - t \\
y_j = I_j + 1 \\
y_j = y_j \\
y_k = T_k + t, \ (\sigma_i - T_k) \\
\end{cases}
\]

Finally using the experimental frame which characterizes channel utilization, the output function becomes:
\[
\lambda(S) = \begin{cases} 
\text{YES} & \text{if } \exists m \geq \sigma_m \neq \emptyset \sum_{i=0}^{n} \sigma_i = 0 \\
\text{NO} & \text{otherwise.}
\end{cases}
\]

as for \( \sigma_i \neq 0 \) for multiple \( i \) values collision results. From \( \lambda(s) \) channel utilization is trivially obtained.

**Lumped model validation**

Recall that our purpose is to validate the lumped model for the given experimental frame by using the DEVS of the base and lumped models. This validation is done by showing the existence of a homomorphism between the base and lumped models. That is, a valid lumped model has the same input/output behavior as the base model for the given experimental frame.

The homomorphism is established by proving the existence of a mapping \( h: (\text{base model states}) \rightarrow (\text{lumped model states}) \), which preserves the time advance, transition and output functions. To formally establish the mapping requires a formal DEVS for the base model which we have not supplied. We proceed informally, therefore, by suggesting that \( h \) has the form shown in Table I, wherein grouped states of the base model correspond to states of the lumped model (assuming both models initialized identically).

As a result the states have been grouped (or lumped) from many to four. It is important to note that the mapping, \( h \), must be onto so that each component interaction in the base model is a member of only one group in the lumped model. Having suggested a viable homomorphism, \( h \), we can proceed to establish the preservation of the input/output time advance and transition functions.

**Time advance preservation**

For both base and lumped models the next hatching time for a channel event is determined by the end of a transmission or retransmission. Denoting the columns of the grouped base states by \( g_1, \ldots, g_4 \) (from Table I) and the columns of the lumped states by \( S_1, \ldots, S_4 \), then \( h(S) = \min (\sigma_i, T_i) \) for the lumped model, and \( t'(g) = \min (\sigma_i, T_i) \) for the base model. Since we have assumed identical initial conditions for the two models, we have:

\[
g = (g_1, \ldots, g_4), t'(g)
\]

\[
h: (g_1, \ldots, g_4) \rightarrow \min (\sigma_i, T_i)
\]

which guarantees the preservation of the time advance function.

**Transition function preservation**

To formally prove transition function preservation we have to show that given initial correspondence between the states of the two models, that any \( (\text{group} \rightarrow \text{group or state} \rightarrow \text{state}) \) transition brings both models into corresponding states. Specifically, we have to show that:

\[
h(\delta(g)) = \delta(h(g)), i.e., \text{for the columns } g_1, g_2, g_3, g_4 \text{ we obtain by the } h \text{ mapping the corresponding states } S_1, \ldots, S_4 \text{ for the columns of the lumped model.}
\]

Additionally, we must show that \( s_t \) represents \( g_t \) following

**Table I—Suggested Homomorphism between Grouped Base Model States and Lumped Model States**

<table>
<thead>
<tr>
<th>GROUPED BASE MODEL STATES</th>
<th>&quot;BECOME&quot; LUMPED MODEL STATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. User→host→node→packet queue</td>
<td>packet queue</td>
</tr>
<tr>
<td>2. Control unit→transmission buffer→channel</td>
<td>channel</td>
</tr>
<tr>
<td>3. Receiver buffer retransmission queue</td>
<td>retransmission queue</td>
</tr>
<tr>
<td>4. Retransmission seed</td>
<td>retransmission seed</td>
</tr>
</tbody>
</table>
all state transitions. Although our ability is limited here due to not having a DEVS for the base model, we can still show the preservation by basic observation. We begin by recalling that the lumped model states are in fact representatives of "groups" of base model states. We further note that we assume all transitions within a group of base model states to be zero time transitions that do not affect $\sigma_i, T_i$. Thus only transitions among states from different groups have to be observed—and this is done by observing the state changes in the corresponding states of the base model. Because of this special relationship between base and lumped model structures, the transition function preservation can be justified without use of DEVS.

Output function preservation

In this case preservation dictates that corresponding states (in the two models) have to provide the same output. In the lumped model a YES period is recorded for those cases where a single transmitter has a non-zero $\sigma_m$ value. In the base model a receiving controller records correct reception, i.e., a YES period if and only if the channel is busy with only one transmission. It is easily seen that the two YES periods correspond so that $\lambda_l = \lambda_b$, since in the base model we record YES if:

$$\sum_{i=1}^{n} \sigma_i = 0 \land \exists \sigma_m \not= 0.$$
Commentary on validation approach

Our purpose has been to demonstrate a formal approach to the validation of a simulation model. Our example was purposely chosen to be simple, where complexity of the lumped model is dependent upon the experimental frame and actual system. For an experimental frame addressing only channel utilization only four-state descriptions were necessary. Quite obviously, as the experimental frame complexity becomes more comprehensive, the lumped model approaches the base model in complexity, and Reference 10 should be consulted for additional details.

We find the approach helpful in structuring network model programs (configurations, protocols, etc.) even when applied with no more rigor than was used in the example. Specifically, its use promotes the same forethought and structuring exercises for simulation programs as do verification and specification techniques for programming generally.

We believe errors have been avoided and more highly structured lumped models produced in reasonable times as a result of only informal use of the methodology. In any case it serves as a guide to the abstraction process (always difficult), is consistent with the view of simulation we believe most natural, and so represents a formal approach to the art of modeling (model development), and several variants of the network model program were informally validated using the approach.

MODEL PROGRAM IMPLEMENTATION CONSIDERATIONS

Guided by the validation procedure of the fourth section, and the scheduling function, \( SF() \), grouping function, \( g() \), and distance control matrix, DCM, as given in the third section, the model program is organized according to the modular structure shown in Figure 3. The program is written in FORTRAN (for efficiency, universality, and sequencing set structural reasons) and consists (for most variants) of around 3500 lines, which compile into approximately 62000 octal words of executable code on a CYBER 74.
Sequencing set considerations

The model program structure shown in Figure 3 is designed, of course, to promote the alterability of the program. That is, for example, all issues concerned with addressing at the host–node interaction level are contained in a single module so that addressing functions can be identified and altered (replaced) without affecting the remaining program. While serving the primary function of flexibility the structure causes certain potential overheads at execution time. Specifically because of the structure, i.e., the columns in Figure 3, it becomes necessary for activities at time $t_i$ to schedule multiple other actions for time $t_i$. Moreover, to maintain the rectitude of the model the multiplicity of events scheduled for a given $t_i$ must occur in a specified order. The use of conventional sequencing set structuring techniques, e.g., the linear list, (see Reference 6), causes considerable overhead in the insertion or deletion of event notices when there are multiple events scheduled for a given event time. Furthermore, these structures do not give particular attention to our paramount requirements on the ordering of the execution of these simultaneous events. The nature of our model program suggests use of the two-dimensional sequencing structure shown in Figure 4, wherein each activity scheduling notice contains not only a primary key giving time of occurrence, $T_i$, but also a secondary key $P_{Ti,j}$ which reflects the activities assigned priority (which determines its placement among the collection of events scheduled for $T_i$). Note further that the priority value assigned a scheduled event cannot be static (i.e., cannot be assigned a priori) to the activity) but rather must be assigned dynamically when the activity is given an occurrence time, with assignment based on the state of the system (the collection of events scheduled) when the scheduling occurs. For example, the order of handling the simultaneous events "set node busy status," "set channel busy status" occur in different orders dependent upon whether the system state is node transmission or node reception.

The two-dimensional structure, Figure 4, aids the assignment of activity priorities as well as the insertion and deletion of notices by reducing the number of scans (of notices) necessary to a more acceptable level than would be possible with conventionally used sequencing set structures. The structure thus allows use of the modular structure without the penalty in model program execution speed which would be incurred with conventional sequencing sets.

Input traffic generators

A large number of network simulations are conducted assuming that message intergeneration (arrival for transmission) times at node-i obey a Poisson process with parameter $\lambda_i$. Typically a simulation program responds to this assumption by scheduling an activity "arrival" for each of the $K$ nodes in the network. For large $K$ this adds significantly to the number of event notices present in the sequencing set, and hence degrades notice insertion deletion operations. A cleaner more efficient approach is given by forming $\lambda = \sum_{i=1}^{K} \lambda_i$ and scheduling a single event with inter-event

<table>
<thead>
<tr>
<th>PROTOCOL</th>
<th>centralized star network</th>
<th>distributed network</th>
<th>node priority assignment</th>
<th>availability of slotted or unslotted model</th>
<th>centralized or distributed control</th>
<th>protocol dependent variants</th>
<th>separate Ack. channel or incorporated ack. channel</th>
<th>LOS/non LOS coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADMA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Both</td>
<td>no control</td>
<td>existence of special file transfer protocol</td>
<td>both possibilities exist</td>
<td>non-significant</td>
</tr>
<tr>
<td>CSMA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Both</td>
<td>Both</td>
<td>separate zero time</td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td>BRAM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Both</td>
<td>Both</td>
<td>separate zero time</td>
<td>Both</td>
<td></td>
</tr>
<tr>
<td>SUPBRAM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Both</td>
<td>distributed</td>
<td>simple, grouped, L-optimized</td>
<td>&quot;</td>
<td>both (but specially designed for &quot;LOS&quot;)</td>
</tr>
<tr>
<td>MSAP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Both</td>
<td>centralized</td>
<td>&quot;</td>
<td>LOS case only</td>
<td></td>
</tr>
<tr>
<td>TUPA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>non-significant</td>
<td>&quot;</td>
<td>&quot;</td>
<td>non-significant</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5—Networks simulated to date.
times drawn from the exponential distribution with parameter \( \lambda \). Each occurrence of the event signals “arrival” and the “cumulative distribution” of the \( \lambda_i \), can be used to determine the node to which the message has arrived. Specifically if a random number \( U \) is drawn and \( \sum_{i=1}^{j-1} \lambda_i < u \leq \sum_{i=1}^{j} \lambda_i \), the message is declared to have arrived at node-\( j \). The procedure works due to the ability to split a Poisson stream into multiple streams via a multibranch Bernoulli trial, and serves to insure that a single, rather than \( K=1 \), notice is in the sequencing set to signal message arrivals.

CONCLUSION

In this paper we have examined a number of techniques designed to aid our development of a modular network simulator designed to serve as a research tool. We believe the aids have contributed to our ability to easily simulate a variety of network topologies and access protocols. In Figure 5 we summarize the uses of the simulator to date. The uses shown by Figure 5 reflect our current interests rather than the limitations of the model.

REFERENCES