A Rule-Based Analyzer with Temporal Reasoning For Communication Protocols*

Mark Morsch and Robert Meyer
Electrical & Computer Engineering Dept., Clarkson University, Potsdam, NY 13676

Charles Meyer
DCLD, Rome Laboratory, Griffiss Air Force Base, NY 13441

Abstract

With data networks growing and serving less sophisticated users, there is a need for tools which facilitate or automate diagnosis of network failures. In this paper we describe a rule-based system for analyzing communication protocols. This system employs an architecture which allows the diagnostic knowledge for a particular protocol to be separate from the reasoning mechanism. The protocol-specific knowledge is held in a structure we refer to as the Protocol Diagnostic Model, or PDM, and consists of a set of associations between patterns and actions. A diagnosis of a network failure can be made by recognizing certain symptoms from observed protocol events. These symptoms are detected by matching representations of the temporal relationships between observed events against the patterns in the PDM.

1 Introduction

Well-implemented protocols allow quick transfer of data across computer networks. This requires the receiver and transmitter operate in coordinated fashion, following closely the same set of protocol specifications. When recurrent or intermittent failures occur and can not be isolated through any independent test of network components, the bit streams travelling over network links may be analyzed. This analysis involves tapping into a link with a protocol analyzer. To understand the output of the protocol analyzer and to explain a failure the technician or engineer using the equipment must possess knowledge about how diagnoses can be inferred from the observable data.

The procedure described above works well for maintaining small networks and/or networks running well specified and well understood protocols. However, for networks operating over a geographically distributed area or running protocols which are not as well specified or understood, better tools are needed to control maintenance costs. An example of a network fitting these two characteristics is the emerging Integrated Services Digital Network (ISDN). Ongoing trials have shown the hardware to be highly reliable; the difficulties have been with failure, or unexpected operation, of the protocols [3]. In this paper we describe a rule-based system using a general purpose architecture for knowledge-based protocol diagnosis. Note that future references to the 'architecture' mean the 'general purpose architecture for knowledge-based protocol diagnosis'. By 'general purpose' we mean a generic kernel operating on protocol-specific diagnostic knowledge.

Up to this time there have been two general approaches to the problem of diagnosis [9].

Diagnosis from first principles. This involves a model, usually represented by a logic system (e.g. first order logic, temporal logic, etc) describing the components of a system. If observations of the system contradict its expected behavior based on the model, one or more components of the system are suspected of failing. Examples of this type of approach are seen in [5, 9].

Diagnosis based on heuristic knowledge. This involves capturing the knowledge of technicians experienced with the operation of a system or procedure of interest. The knowledge is coded into production rules which simulate the associations made by the expert. Examples of this type of approach are the MYCIN [2] and Mud [7] systems.

There are other deep-reasoning systems which are related to the first approach. Sembugamoorthy and Chandrasekaran [10] describe a compiler which generates a diagnostic rule-based system from the functional representation of a device or system. For this architecture, we purposely choose not to make any assumptions about the implementation of the protocols. Like De Jong [4], we would like to represent the deep knowledge, the protocol specifications, along with the shallow knowledge, the heuristic associations. However, instead of representing this knowledge as rules, our method represents the protocol specifications and heuristic knowledge as a set of data elements. These data elements comprise the Protocol Diagnostic Model.

2 System Overview

This system would operate as an adjunct to a single node in a network, meaning the diagnostic reasoning is centralized to that node. Such a design could have several centralized diagnostic agents each serving a portion of the network. A node being served by a diagnostic agent, upon observing some error, should deliver a protocol report to the agent. It is then the diagnosing agent's responsibility to collect reports from the other nodes it serves. Note that the architecture we are outlining in this paper is for a centralized diagnosing agent, and it is assumed that a single diagnosing agent has access to the protocol reports of any node it needs.

The three major aspects of this architecture are:

*This work was sponsored by the United States Air Force, Air Force Systems Command, Rome Laboratory, Griffiss Air Force Base, NY 13441-5700 through the University of Dayton, Research Institute, 300 College Park, Dayton, OH 45469-0001.

CH2981-9/91/0000-0396 $1.00 © 1991 IEEE

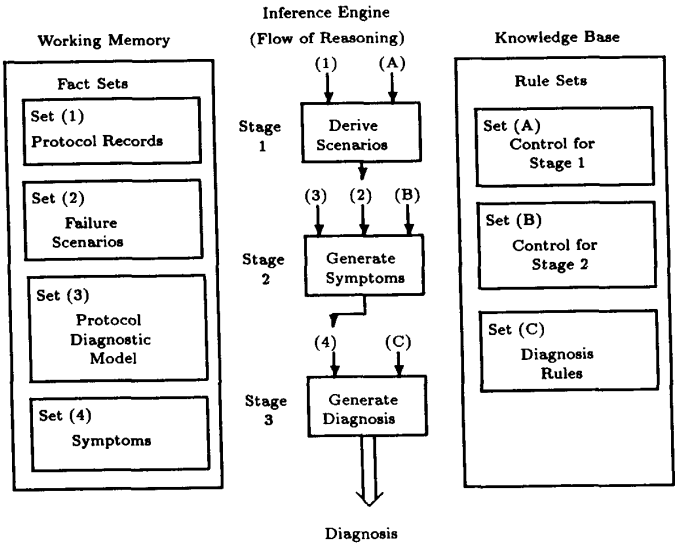

Figure 1: High Level Architecture

1. the processing of the observed data (protocol reports) which involves deriving a set of possible failure scenarios consistent with the observations;
2. the representation of the protocol-specific diagnostic information which we will refer to as the Protocol Diagnostic Model (PDM); and
3. the reasoning mechanism which involves matching each derived scenario against the PDM to generate symptoms, and the use of symptoms to support particular diagnoses.

Figure 1 shows a high-level view of the architecture. An agent running under this architecture would begin an analysis with four modules: fact set (3) and rule sets (A), (B), and (C). Protocol records would be collected, translated into event facts, and loaded into working memory by a separate local process. This gives fact set (1). In the first stage of reasoning the failure scenarios are derived from the event facts producing fact set (2). In the second stage the symptoms are generated for each failure scenario producing fact set (4). In the third stage the diagnoses are generated from the symptoms. Protocol-specific knowledge is in the PDM and in the diagnosis rules. Rule sets (A) and (B) and the inference engine are protocol-independent.

A protocol running on a node executes a sequence of events. These events can be described in terms of the manipulation of a few basic objects. The common features of protocols upon which this architecture is based are message, primitive, timer and state. The names for these objects are based on the Specification and Description Language (SDL) used in CCITT standards documents [6].

Message Messages hold the information which is passed between peer protocols at communicating nodes.

Primitive Primitives represent the exchange of information with local processes which are controlling and/or monitoring a protocol.

Timer Timers control the timing constraints under which a protocol operates.

State At any point in time a protocol is in a 'state'. As the protocol runs, it advances through a sequence of states acting as a finite state automaton.

Each of the four objects defined above has failure modes. The goal of our diagnosis is to explain a recognized fault in terms of the failure modes, placing blame on the node which originated the failure. A vital consideration is how the failure conditions are recognized. It can not be assumed that a protocol would be able to flag its own error. More likely, the failure would produce noticeable side effects at the originating node or at other nodes in the network. Diagnosis is triggered by a recognizable fault indicator which is in all likelihood a side effect of previous protocol failure(s). Fault indicators include time-outs, reception of an unexpected message, or reception of an unrecognizable message.

3 Representation of Protocol Records

In this architecture we consider a protocol report to consist of a list of records; each record describes an event which occurred while a protocol was running. Although this architecture does not depend upon a particular format for the records, we will discuss how records could be organized. A record may consist of named slots with the number of slots dependent upon the type of event being described. As part of the internal representation, events are categorized into nine types: receive-message, transmit-message, primitive-accepted, primitive-sent, state-transition, start-timer, stop-timer, time-out, and procedure-call. These nine types cover the operations on the four protocol objects: message, timer, primitive, and state.

Each record should include a time stamp, normally in the last slot of a record, indicating the time of execution

and $F_\ell^{ma} = 0$ otherwise. Then (for $X \geq 0$)

$$P(F_\ell^{ma} = 1|X) = 1 - \left(1 - \frac{1}{q}\right)^{2X}. \quad (3)$$

We now characterize the external interference process. We assume that the ambient noise is stationary, and define

P_{noise} = Probability of symbol error due to background noise.

Furthermore, we assume that noise affects each symbol in a statistically independent fashion.

We assume that a time-stationary jammer is present, and jams J of the q frequency bins. The jammer reselects which bins are jammed fast enough to prevent the user network from being able to avoid them, but slow enough to make partial symbol overlaps negligible. For example, we may assume that the jammer selects a new jam band each slot. We define

P_{jam} = Probability of symbol error due to jamming.

Since we assume that the jammer is stationary in the sense that J is constant, we have $P_{\text{jam}} = J/q$.

We also define

P_I = Probability of symbol error due to external interference,

where "external interference" is either background noise or jamming. We have that

$$P_I = 1 - (1 - P_{\text{noise}})(1 - P_{\text{jam}}). \quad (4)$$

Denote $K_\ell^I = 1$ as the event that the ℓ^{th} symbol transmitted by a user is corrupted by either ambient noise or jamming, and $K_\ell^I = 0$ otherwise. Similarly, let $F_\ell^I = 1$ denote the event that the ℓ^{th} symbol interval/frequency bin monitored by an idle user detects either ambient noise or jamming, and $F_\ell^I = 0$ otherwise. We have that

$$P(K_\ell^I = 1) = P(F_\ell^I = 1) = P_I. \quad (5)$$

The effects of either multiple access or external interference are combined as follows. Let $K_\ell = 1$ denote the event that the ℓ^{th} symbol transmitted by a user was unsuccessful, and $K_\ell = 0$ otherwise. Then

$$\begin{aligned} P(K_\ell = 1|X) &= 1 - [1 - P(K_\ell^{ma}|X)][1 - P(K_\ell^I|X)] \\ &= 1 - \left(1 - \frac{1}{q}\right)^{2(X-1)}(1 - P_I). \end{aligned} \quad (6)$$

Similarly, let $F_\ell = 1$ denote the event that the ℓ^{th} symbol interval/frequency bin monitored yielded "something," and $F_\ell = 0$ if it yielded "nothing." Then

$$\begin{aligned} P(F_\ell = 1|X) &= 1 - [1 - P(F_\ell^{ma}|X)][1 - P(F_\ell^I|X)] \\ &= 1 - \left(1 - \frac{1}{q}\right)^{2X}(1 - P_I). \end{aligned} \quad (7)$$

The assumptions implicit in the above formulas are that the monitor detects multiaccess, noise and jamming levels at exactly the same levels at which they cause interference to transmissions. The justification of these assumptions is strongly dependent on specifics of the physical implementation of the waveforms employed and receiver characteristics. Relaxation of these assumptions and sensitivity analyses are left as an area for future research. Pronios [5] has considered a more general characterization of these aspects in his work.

III. Backlog Estimation Procedure

We now proceed to derive the backlog estimation algorithm that uses the assumed feedback and an a priori estimate of the interference level. We denote

$\widehat{P_I}$ = a priori estimate of P_I.

As was indicated earlier, the system is assumed to operate in a slotted fashion. All packets consist of L symbols, so that each slot comprises L symbol intervals. Monitoring takes place whenever the user either has no packet pending or is deferring transmission in accordance with the ALOHA protocol. Every user updates his backlog estimate once per slot. Since the system operation is stochastically stationary during a given slot, the feedback samples (monitored at a symbol level) can be accumulated over each slot (a "sufficient statistic"). We denote H^{mon} as the total number of symbol interval/frequency bin samples that have "something" detected in them in a given slot, so that

$$H^{mon} = \sum_{\ell=1}^{L} F_\ell, \qquad \text{and} \quad (8)$$

$$E(H^{mon}|X) = LP(F_\ell = 1|X) = L[1 - \left(1 - \frac{1}{q}\right)^{2X}(1 - P_I)], \quad (9)$$

where the latter equality holds because the F_ℓ are identically distributed for each ℓ (in the same slot).

We begin the backlog update procedure derivation by first estimating the number of users that must have transmitted given that H^{mon} hits out of L were detected. We form our heuristic estimate of X by first equating the given empirical statistic H^{mon} with the expected value as indicated by equation (9) above, with P_I replaced by the a priori estimate $\widehat{P_I}$:

$$H^{mon} \doteq L[1 - (1 - \frac{1}{q})^{2X}(1 - \widehat{P_I})]. \quad (10)$$

The estimate for X is then obtained by solving this equation for X in terms of the given feedback H^{mon}, obtaining

$$\widehat{X} = \frac{\ln(1 - H^{mon}/L) - \ln(1 - \widehat{P_I})}{\ln(1 - \frac{1}{q})}. \quad (11)$$

Next, we estimate the number of successes out of the $\widehat{X}$ that transmitted. If one of the transmitted symbols is selected at random, its chance of failing is estimated as

$$\widehat{P_{K=1,X}} = 1 - \left(1 - \frac{1}{q}\right)^{2(\widehat{X}-1)}(1 - \widehat{P_I}). \quad (12)$$

The mean number of erasures in one of the $\widehat{X}$ transmitted packets is taken to be a binomially distributed random variable with parameters L and $\widehat{P_{K=1,X}}$. We assume that an extended Reed-Solomon code with k information symbols is used, and that perfect side information is available. Therefore, the probability that a packet succeeds is the probability that the number of erasures is no more than $L-k$. We also approximate the success probabilities of different packets in the same slot (conditioned on X) as being independent. Therefore, the expected number of successes, $\widehat{S}$, is

$$\widehat{S} = \widehat{S}(\widehat{X}) = \widehat{X}P(Y < L-k), \tag{13}$$

where Y is binomially distributed with parameters L and $\widehat{P_{K=1,X}}$. If L is large, Y can be approximated as a Gaussian r.v., so that

$$\widehat{S} = \widehat{X}\Phi\Big(\frac{L-k-\mu}{\sigma}\Big), \tag{14}$$

where $\Phi(\cdot)$ is the standard Gaussian cumulative probability distribution function, $\mu = L\widehat{P_{K=1,X}}$, and $\sigma = \sqrt{L\widehat{P_{K=1,X}}(1-\widehat{P_{K=1,X}})}$. (Since precision is not needed in the update algorithm, a simple approximation may be used for $\Phi(\cdot)$ in implementing the backlog estimation procedure.)

Finally, the backlog estimate $\widehat{B}$ is updated by adding the expected number of new packets to transmit $(N-\widehat{B})\Delta$ and subtracting the estimated number of successes:

$$\widehat{B_{t+1}} = [\widehat{B_t} + (N-\widehat{B})\Delta - \widehat{S}]^+, \tag{15}$$

where $[\cdot]^+$ denotes $\max(\cdot, 0)$.

Some minor comments for implementation of this estimation algorithm follow. First, if $X = 0$ then we would still expect to detect energy due only to the interference, causing a mean number of detections $E(H^{mon}|X=0) = LP_I$. We should therefore first test whether the feedback H^{mon} is less than $\widehat{LP_I}$, and if so, estimate $\widehat{X} = 0$ and $\widehat{S} = 0$. Also, there is small range of values of H^{mon} that can result in $0 < \widehat{X} < 1$, and this can cause problems when $\widehat{X} - 1$ is used as an exponent. In this case, we should estimate $\widehat{X} = \widehat{S} = 1$. Finally, for large values of H^{mon} it is possible to obtain overly large estimates for X; we should always ensure $\widehat{X} \leq N$. In particular, the unlikely case $H^{mon} = L$ will cause numerical problems (viz., $\ln(0)$) if not tested for.

The above procedure specifies the backlog update procedure for users while they are idle. However, the backlog estimate must also be updated when either the user is transmitting a packet or receiving a packet. In our studies, we have made the common assumption that feedback regarding the success of a transmitted packet is obtained immediately at the end of the transmission. Also, we have assumed that there is no conflict due to multiple transmitters simultaneously sending data to the same receiver or due to half-duplex operation. In essence, we have assumed that receiver resources do not present a significant bottleneck. It is intuitive that the action taken by a user that receives a packet should be similar to the action taken by a user that just transmitted a packet (and obtained immediate feedback as to its success). The simple backlog update policy that we have incorporated into our simulation is:

1. If the packet was successful, then it is likely that all X packets were successful. The value of X is estimated to be β times $\widehat{B}+(N-\widehat{B})\Delta$, where $\beta = \beta(\widehat{B})$ is computed so as to cause $X = G$ if there are sufficiently many users with packets (i.e., $\widehat{B}$ is large enough), where G is the desired number of transmissions per slot. Therefore, the backlog estimate is updated as $\widehat{B_{t+1}} = [\widehat{B_t} + (N-\widehat{B_t})\Delta - G]^+$.
2. If the packet was unsuccessful, then it is likely that all X packets were unsuccessful. Following similar logic as above, the backlog estimate is updated as $\widehat{B_{t+1}} = \widehat{B_t} + (N-\widehat{B_t})\Delta$.

This concludes the derivation of the backlog estimate algorithm. The above backlog estimate algorithm can now be coupled with the dynamic control algorithm that was developed in earlier work [2].

IV. Numerical Results

In this section we present performance results that were obtained from our simulation model. The backlog estimation algorithm derived in the previous section was modeled in the simulation, and utilized together with dynamic control procedures of [2]. Fully distributed control is employed; since each user perceives the environment differently because each user is using a different CDMA code, the users will typically have somewhat different backlog estimates.

We have fully validated the simulation model by comparing its results under "genie" or static control assumptions with those of a mathematical analytical model [2,3]. In addition, rigorous software engineering discipline was adhered to, with numerous validation tests performed with each increment in its development. In the interest of clarity and brevity, we do not present these comparison test results; the simulation essentially always agreed very well with the analytical model. In the following, we limit the simulation results to the practicable scheme as described in the previous section; these are identified as "real" results. Some analytical results are provided for the "genie" model and are identified accordingly.

We assume that the network consists of $N = 250$ users, $q = 100$ frequency bins are used, packets have a length of $L = 64$ symbols, and Delayed First Transmission (DFT) slotted ALOHA is employed.

Figure 1 presents a comparison of throughput performance for "genie" and "real" schemes. Steady state normalized information throughput s is plotted versus the arrival rate while in the thinking state per user. The normalized throughput s is the mean number of successful transmissions per slot divided by the number of frequency bins q (100) and multiplied by the code rate k/L ($L = 64$; recall k is the number of information symbols per R-S codeword). The arrival rate η is the inverse of the mean time spent in the "thinking" state by a user and equals $\eta = \Delta/(1-\Delta)$. Three dynamic control cases are considered, based upon

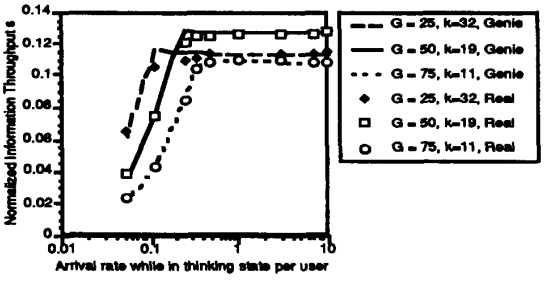

Figure 1: Steady state throughput s versus arrival rate η, "genie" and real cases.

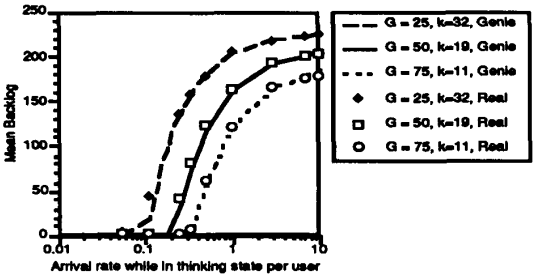

Figure 3: Figure 3. Steady state mean backlog $E(B)$ versus arrival rate η, "genie" and real cases.

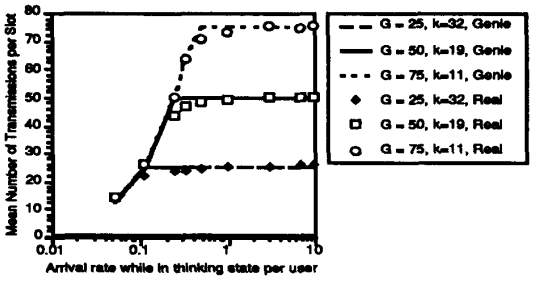

Figure 2: Steady state mean number of transmission per slot $E(X)$ versus arrival rate η, "genie" and real cases.

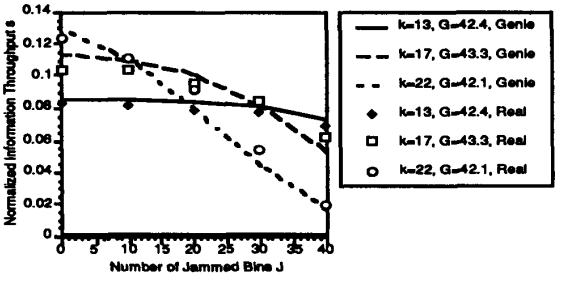

Figure 4: Throughput s versus number of jammed bins J for "genie" and "real" cases.

the simple control function given by equation (15): $G = 50$ and $k = 19$, $G = 25$ and $k = 32$, and $G = 75$ and $k = 11$. We see that the performance of the feasible ("real") control system is very close to that of the "genie" model over wide ranges of parameters. This is the first indicator that tells us that the advantages of dynamic control (stability and robustness) are truly obtainable. Also, since we cannot do better than the "genie" model, this tells us that it is not worthwhile expending effort in enhancing the backlog update procedure with greater precision.

Figures 2 and 3 provide additional steady state performance characterization for the same cases as Figure 1. Figure 2 illustrates the mean number of transmissions per slot (which should equal G when there is sufficient load on the network), while Figure 3 presents the mean backlog. Again, we see that "real" and "genie" models perform similarly.

Figure 4 illustrates steady state throughput performance in the presence of jamming. The G value is assumed to be fixed, as well as the number of information symbols per packet k. Three cases were run: $k = 13$ and $G = 42.2$ (optimal if $J = 40$), $k = 17$ and $G = 43.3$ (optimal if $J = 20$), and $k = 22$ and $G = 42.1$ (optimal if $J = 0$). Each of these was run for "genie" and "real" cases, where as usual in the "genie" case the backlog was known to all users. In the "real" cases, the backlog update was based upon feedback and the use of $\widehat{P_I}$, where $\widehat{P_I}$ was fixed at .4, .2 and 0 for the three cases respectively. Therefore, these results indicate the relative insensitivity to a priori knowledge of the interference level.

The remaining figures investigate the performance in the presence of a time-varying arrival rate. These illustrate the transient behavior when sudden changes in system characteristics occur, and also provide an indication as to the benefit of devising additional algorithms for tracking such time-varying coarse-grain parameters.

We incorporated a Markov-modulated arrival process into our simulation model. (This was also included in the analytical model but time prevented us from exercising it.) A slowly time-varying two-state Markov chain alternated between states independently of the other system behavior. The arrival rate of all users in the thinking state would take on one of two values according to the value of the background Markov chain. In the results presented below, we let the arrival rates for the two states be $\eta = 1/9$ (corresponding to $\Delta = .1$) and $\eta = \infty$ (corresponding to $\Delta = 1$), so that the arrival rates alternated between rather extreme values. These were intentionally selected so as to exaggerate the performance impacts.

The simulation model also was made to have a "semi-genie" model selectable, in which the genie would tell the users when the arrival rate changed but nothing else. In the "real" case there was no such genie, and a constant a priori estimate of the arrival rate $\widehat{\eta}$ was used.

Figure 5 illustrates the performance obtained for the "semi-genie" model. This shows the true backlog and the estimated backlog as a function of time, measured in slots. The estimated backlog values were averaged over all users for each time slot. We see that the estimate is quite good, and both the true and estimated backlog react quickly to

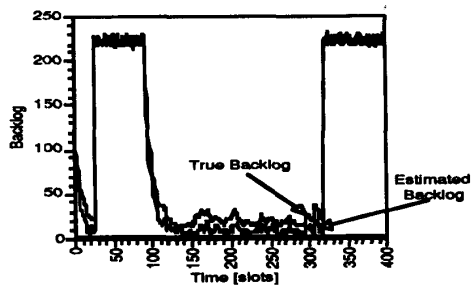

Figure 5: True and mean estimated backlog versus time,$\eta = \widehat{\eta}$ alternating between 1/9 and ∞ ("semi-genie").

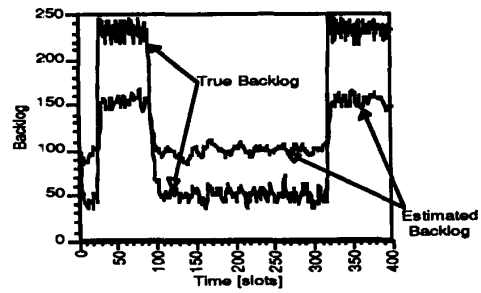

Figure 6: True and mean estimated backlog versus time, $\widehat{\eta} = .136$

transients. The normalized information throughput for this run was $s = .0558$.

Figure 6 shows a similar plot but where no genie information is provided. The a priori estimate of the arrival rate is taken to be the constant $\widehat{\eta} = .163$ ($\widehat{\Delta} = .14$). We see that for this value of $\widehat{\eta}$, the backlog is apparently over- and under-estimated roughly equally. The normalized throughput that resulted from this run was .0388, or about 70% of that achieved with the semi-genie model.

Figure 7 shows the true and estimated backlog processes for the case when $\widehat{\eta} = 1.86$ ($\widehat{\Delta} = .65$). Although the backlog is poorly estimated during the intervals of low arrival rates, the resulting normalized throughput obtained was .0415, which is somewhat better (about 74% of semi-genie) than that achieved for the case of Figure 6. We ran simulations for the entire spectrum of values for $\widehat{\eta}$, and generally found little sensitivity to the value selected. We conclude that, for the case considered, some benefit may be possible by devising algorithms for tracking the gross arrival rate, but this benefit is not major. (Recall the exaggerated arrival rate extremes.) Given that a constant value is used, it is not especially sensitive the value selected. We emphasize that these conclusions may or may not be extendable for all parameter settings, and further analysis should be made for specific network designs.

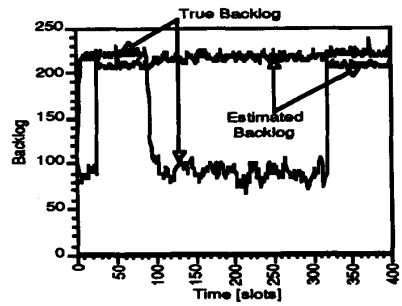

Figure 7: True and mean estimated backlog versus time, $\widehat{\eta} = 1.86$

V. Conclusions

In this paper we have developed implementable dynamic transmission control procedures for single-hop slotted ALOHA networks using frequency-hopped spread spectrum. These control procedures are based on backlog estimates, and a backlog estimation algorithm that operates in the presence of multiuser interference, ambient noise, and jamming was derived. The backlog process is estimated by sensing activity on the receiver-based code while the user is not transmitting (half-duplex operation is assumed), so that minimal (if any) additional hardware is required. Performance was derived by simulation, and numerical examples show that the feasible system operates at nearly the same level of performance as the "genie" model. These results demonstrate that considerable robustness is achievable.

VI. References

1. L. P. Clare and A. R. K. Sastry, "The effects of slotting, burstiness, and jamming in frequency-hopped random access systems," *IEEE MILCOM'89 Conf. Rec.*, Boston, MA, October 15–18, 1989, pp 154–160.
2. L. P. Clare, J. E. Baker, and A. R. K. Sastry, "A performance comparison of control policies for slotted ALOHA frequency-hopped multiple access systems," *IEEE MILCOM'90 Conf. Rec.*, Monterey, CA, September 30 – October 3, 1990, pp. 608–614.
3. L. P. Clare and J. E. Baker, "The effects of jamming on control policies for frequency-hopped slotted ALOHA," *Proc. IEEE GLOBECOM'90,* San Diego, CA, December 2–5, 1990, pp. 1132–1138.
4. B. Hajek, "Recursive retransmission control — Application to a frequency-hopped spread-spectrum system," *Proc. 16th Conf. Infor. Sciences and Systems,* Princeton University, March 1982, pp. 116–120.
5. N. Pronios, "On the stability of spread-spectrum networks, with decentralized recursive retransmission control, under jamming," *Proc. INFOCOM'90,* San Francisco, CA, June 5–7, 1990, pp. 588–594.