Availability Driven Multiple Access Network Architecture

Dan Cristian Marinescu

Vernon J. Rego
Purdue University, rego@cs.purdue.edu

Wojciech Szpankowski
Purdue University, spa@cs.purdue.edu

Report Number:
86-645

AVAILABILITY DRIVEN MULTIPLE ACCESS NETWORK ARCHITECTURE

Dan Cristian Marinescu
Vernon Rego
Wojciech Szpankowski

CSD-TR-645
December 1986

AVAILABILITY DRIVEN MULTIPLE ACCESS NETWORK ARCHITECTURE

Dan Cristian Marinescu, Vernon Rego, Wojciech Szpankowski

Department of Computer Sciences

Purdue University

West Lafayette, IN, 47907

Abstract. An architecture for a non-homogeneous local network connecting a large number of users (workstations) to a number of servers is proposed in this paper. A connectionless communication model and a high speed broadcast channel are assumed to provide an interconnection network between the servers and the users. The Availability Driven Multiple Access Architecture is based upon the correlation of the channel allocation strategy with server availability by means of scheduling protocols. A scheduling protocol defines a communication discipline in which servers capable of providing services compete for control of the communication channel by using a multiple access algorithm in one of the following classes: random, limited contention or contention free multi-access. When in control of the channel, a server performs a sequence of different activities, including a broadcast message of its willingness to provide certain types of services, thus inviting all users to send their service requests. A multi-access algorithm is then used to give each user a chance to send its request to the remote server. In addition to an inherent dynamic load balance, the system enjoys a number of other desirable properties such as fairness, robustness, stability, etc.

1. OVERVIEW

This paper presents an alternative architecture for a local network connecting a number of workstations with a set of processors. The Availability Driven Multiple Access Architecture described in the following is designed for scientific or knowledge processing environments in which each user has a dedicated workstation which may require frequent access to the set of servers connected to the network.

DPA (DoD Protocol Architecture) is a very successful computer communication architecture that has positively influenced the theoretical and practical developments in this field. The definition of communication sockets at Berkeley, the development of Remote Procedure Call Protocols and of the Network File System at SUN, are all significant contributions to DPA. Nevertheless there are few open questions related to the integration of communication functions into the computer architecture.

An important question is related to the efficiency of performing remote operations. Certainly, an architecture which splits the networking functions into a large number of layers, in order to provide a general interface for different types of applications and which attempts to treat different types of networks uniformly cannot lead to an efficient implementation. Long haul networks and local area networks differ drastically from one another and it is at least questionable whether a computer communication architecture which treats them uniformly is adequate. The two types of networks have speeds which differ by 2-3 orders of magnitude, different error rates, and different topology. In addition, local networks are generally one hop networks while the long haul networks are the store and forward kind, with a list of differences that is far from being exhausted. On the other hand, while a long haul network is primarily used to move different types of data from one location to another, in a local network it is becoming increasingly important to move computations from one location to another in order to achieve a distributed processing environment.

Functional communication is defined as communication occuring only in connection with the need to serve or to request a service [10]. If functional communication is feasible from the implementational point of view, it can lead to a simpler communication architecture since the only communication primitives are remote procedure call (RPC) like operations. In addition, by raising the semantic level of communication, improved system performance as well as increased system performability can be expected.

In case of functional communication a system can be viewed as consisting of two groups of nodes, a large group of user nodes and a much smaller group of servers. The most frequent type of communication is the one across group boundaries. Nevertheless communication between users can be performed using a communication server. Such a server is also necessary to perform internetwork communication functions like routing, address resolution (conversion from local addresses to internetwork addresses and vice-versa), to run internetwork protocols as well as standard transport protocols. Servers can pretend that they are users and request services from other users.

Functional communication is based upon functional addressing, namely each request for service is identified by a service type, unique system-wide. Each server is characterized by the dynamical list of services it provides at a given moment of time.

Communication can be initiated either by a user (client) requesting service or by a server offering a certain assortment of services. The Availability Driven Multiple Access architecture utilizes functional communication based upon a client-server paradigm in which the server initiates communication. Moreover the allocation of the communication channel (i.e., the central server of the system) is done under the control of the various servers.

The paper first presents some measurements which show that even in a DPA environment, the processor overhead associated with data or computation movement is considerable. It limits the actual transfer rates available at the user level. This explains why the communication channel is rarely the bottleneck of present systems. Based on the measurements performed we argue that a simpler architecture is necessary and we propose ADMA. Next the basic concepts behind the ADMA are presented. We next introduce scheduling protocols and present a taxonomy of scheduling protocols. Finally, we discuss performance issues. Two kinds of scheduling protocols have already been presented in [7] and [8]. The models of the ADMA systems discussed in these two papers can be used together with Kuhen's approximations [5] for a delay analysis. The behaviour of the system is sufficiently complex to suggest that an exact analysis appears not to be tractable at this point in time. Hence we use qualitative rather than quantitative arguments in our effort to show that ADMA is feasible and exhibits good performance. The results of a modeling effort based upon simulation are then presented.

2. PERFORMANCE ISSUES IN DPA

DPA seems to be a more suitable model for a computer communication architecture than OSI as far as efficiency of implementation is concerned. It has a simpler structure, with only four layers (network interface, internetworking, transport and application), gives a legitimate status to connectionless communication, allows direct communication between non-adjacent layers and provides a hierarchical approach to communication.

Given a high speed communication channel and an environment where the ratio between communication related activities and computational ones is considerably higher than in most present local networks, an interesting question is whether DPA is still suitable from the performance point of view. To provide an answer to this question we have performed several measurements in the local network of our department.

At the time when the measurements reported here were performed, our 10 Mbps Ethernet connected several VAX 780s, a VAX 785, a file server, less than a dozen SUN workstations (model 2 and 3), a multiprocessor system, FLEX 32, some graphical workstations and several other processors, all running Berkeley UNIX (TM), 4.2 BSD. The traffic carried by our Ethernet was relatively low and most of the time the channel load was considerably lower than 5% of its capacity, averaged over sixty seconds. The file server was responsible for 10-30% of the total network traffic. The packet size distribution was: 20 to 30% of the packets were 970 - 1150 bytes long, and 60 to 70% of the packets were 60 - 241 bytes long. Since then a VAX 8600 as well as several new SUNs and a new file server have been added. Now the traffic has increased and is often above 5% of the channel's capacity, and the packet length distribution is evenly divided between small and large packets.

In this environment we have performed some rather crude measurements in order to estimate the processor overhead related to file transfer protocols and to remote procedure call execution.

2.1. Measurements Related to the File Transfer Protocol

The measurements reported here were performed while transferring files between a VAX 785 running in a single user mode and a VAX 780 with a relatively light load. The goal of our measurements were to establish upper bounds for the actual data transfer rates available at the user level and to determine to which extent the implementation of DPA is a limiting factor in achieving higher data transfer rates. Clearly, the actual data rates between machines with a normal load would be considerably lower than those reported here.

Put File (Send)		Get File (Receive)	
ASCII	BINARY	ASCII	BINARY
31.24	86.38	13.12	44.79

Table 1. F.T.P. data transfer rates (kbyte/sec)

Table 1 shows the measured transfer rates which vary from 13 Kbps to 87 Kbps depending upon the type of data being transferred and the actual type of transfer, send or receive. A significant difference can be noticed depending upon the direction of the data transfer. The receiving data rate is 50 % or less of the sending data rate. No sensible explanation can be provided; the study of FTP code could provide some light. As expected, the transfer rates depend upon the type of data being transferred, it is at least twice as high for binary data than for ASCII. The load on the Ethernet due to other types of traffic, unrelated to our experiment, was very low, less than 2% of its capacity. The number of collisions observed was less than 10^3 for 10^6 packets transmitted. Consequently we are confident that the packet transmission time, the propagation delays, the queuing delays for the communication channel and packet retransmissions delays are very small and consequently do not influence the actual transmission rates.

Protocol Layer	Put File (Send)		Get File (Receive)	
	ASCII	BINARY	ASCII	BINARY
Socket	0.44	0.48	0.66	0.62
TCP	1.29	1.16	1.30	1.06
IP	0.53	0.51	0.66	0.58
Device Driver	1.22	1.40	0.89	0.96
Total	3.48	3.55	3.51	3.28

Table 2. The kernel CPU time (seconds) for networking functions when transferring 1 Mbyte of data with FTP from/to a VAX 785 in single user mode.

Table 2 shows total time spent in the layers of the communication architecture implemented in the kernel, for the transfer of 1 Mbyte of data in each of the four cases examined. The actual measurements were performed by profiling both the kernel and the FTP code on the machine working in a single user mode, where the FTP was initiated. The following observations are in order:

-a. The total kernel CPU time for networking functions is approximately equal in the four cases examined (the difference is less than 10% of the maximum value measured), and a careful implementation of FTP can probably improve its overall performance to some extent,

-b. TCP uses less than 33% of this time and between 5% and 15% of the total processor time (the total processor time ranges from 6.59 seconds per Mbyte for get-binary to 23.53 seconds per Mbyte of data for put-ASCII)

-c. The processor overhead related to execution of networking functions limits de facto the transfer rate between two VAX 785s both working in single user mode.

From our data it seems very unlikely that the use of a connectionless transport protocol, UDP, can speed up the transfer rates considerably. The window size in TCP seems to be adequate and the total overhead of TCP is not excessive. The data presented here seems to contradict the results of Meister et al [12]. Their results indicate that file transfer protocols based upon connectionless communications is about 50% more efficient than the ones based upon connection oriented transfer protocols. In our case even if UDP involves less overhead than TCP it is hard to believe that a 50% improvement in performance can be achieved. Unfortunately, no reasonable file transfer protocol based upon UDP was available in order to provide the definitive conclusion that in DPA, file transfer protocols based upon connectionless transport protocols are not considerably more efficient than protocols based upon connection oriented transport protocols.

2.2. The Performance of Programs with Remote Procedures

In order to measure the response time in case of client-server communication we have implemented a very simple Remote Procedure Call protocol based upon UDP (User Datagram Protocol) and we have activated several server processes located at different distances from the client process. In our measurements four server processes were active, one on the same machine (machine A) with the client (zero distance), one on machine B, on the same Ethernet (distance 1), the third on machine C located in another local subnet connected to the Ethernet through a 10 Mbps token passing system, (distance 2) and one on machine D accessible through INTERNET and located in California. A large number of measurement points were collected and the results of these measurements have been used to model the behavior of a program with remote procedures. The measurements were performed at different loads of the server machines and at different levels of traffic over a period of one month.

The measurements and the conclusions of our modeling are reported in [6]. These results will be summarized in the following. We have defined eight types of remote procedures (RP), classified according to:

- the size of the input (arguments) to the remote procedure,
- the amount of computations performed remotely,
- the size of the output (returned value) produced as a result of remote procedure execution.

Each of these parameters can be *s* (small) or *l* (large). For example an *<s, s, l>* type means: small input, small computation and large output. Large argument/returned value means 1 kbyte of data, and large computing is an empty loop executed 10^5 times. Small argument/returned value means 100 bytes or less and small computations are few dozen machine instructions. For the *<s, s, s> RP* type, the 95% confidence intervals of the response time are: (11,11), (24,24), (26,28) and (866,898). All times are given in milliseconds and they represent the response time of machines A, B, C, and D respectively. The corresponding values for the *<l, l, l> RP* are: (375,389), (933,989), (458,464) and (3816,3876).

Based upon this data we have been able to model the behavior of a program which uses a certain mix of the basic eight flavors or RP's. For example we have modeled paging activity as an equal mix of *<s, l, l>* and *<l, l, l> RP* 's, the first case corresponds to pages which have not been changed and the second case to updated pages. The results are shown in Figure 1. We have plotted the relationship between the so called gamma factor and the paging rate. The gamma factor is defined as the ratio between the execution time with remote paging and to the time taken when paging is done using a local disk with a fixed paging time of 25 msec/page. In case of machine D, the performance degradation is considerable even for low paging rates:127.65 for 0.31 pages/sec, 26.43 for 0.30 pages/sec, 4.10 for 0.23 pages/sec and reaches 1.31 for 0.075 pages/sec. These results are only to be expected, since paging over a long haul network does not seem a reasonable approach.

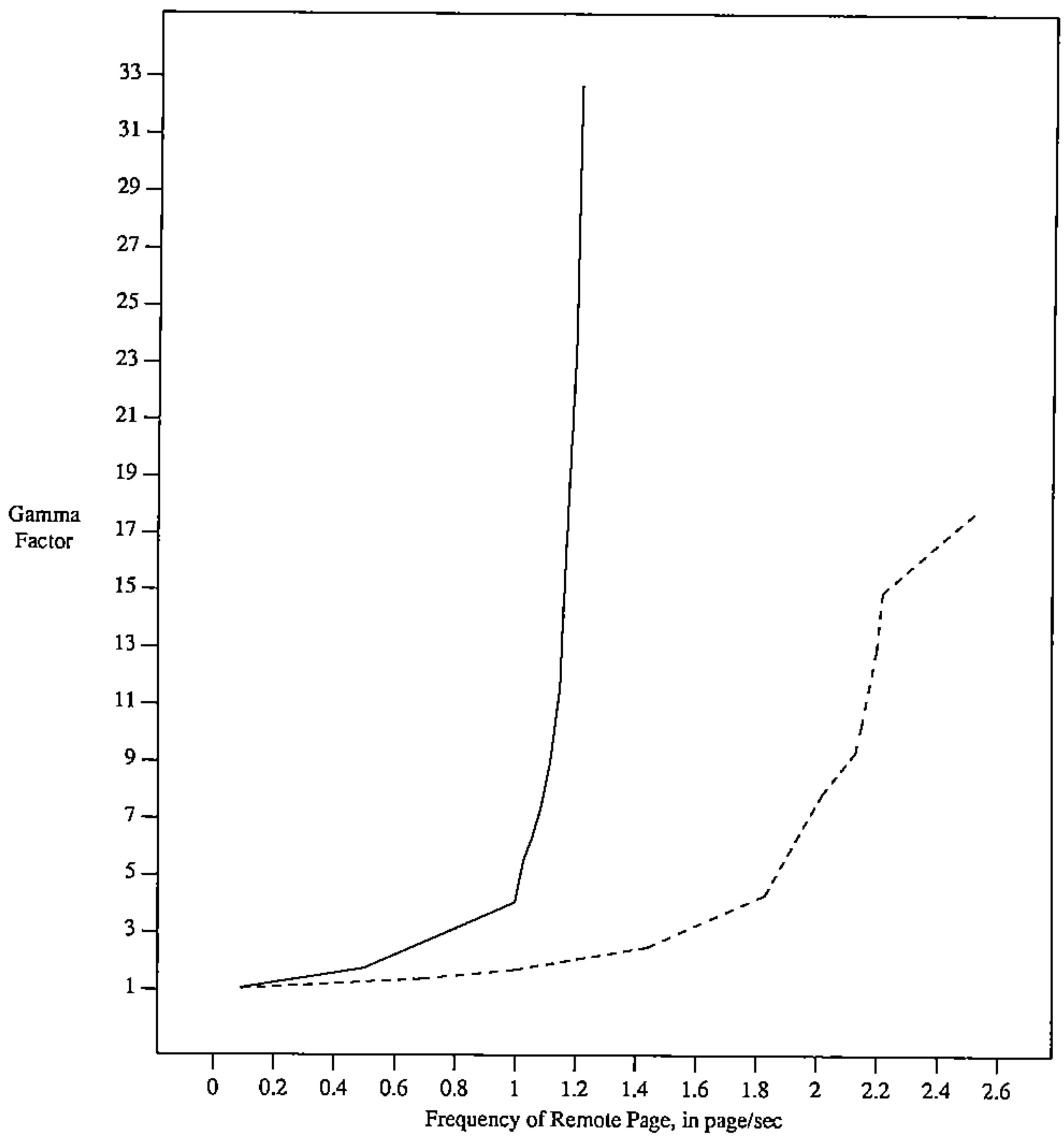

Figure 1 The gamma factor for machines B (solid) and C(dashed) function of the paging rate in pages/sec

2.3. Observations related to the measurements

The level of the traffic in most local networks very rarely exceeds 10% of the channel capacity, when averaged over a period of sixty seconds or more. There are several explanations for this level of traffic: first of all, in case of multi-user systems the networking software limits the rate at which a given host can transfer data. In addition many local networks connect only a few multi-user hosts. Another reason is that there are only a few applications with a non-neglectable ratio of communication to local computational activities. A possible explanation could be the relative unfriendliness of the communication software which does not encourage the design of distributed applications.

We have observed a bimodal distribution of packets carried over the network. Typically, we have short packets (smaller than 240 bytes), and large packets with size closed to 1000 bytes.

The data transfer rates available at application protocol level are rather modest. In case of the file transfer protocol the processing overhead at each end is considerable. For example when sending a binary file, the kernel processing time at each end amounts to 30% of the total delay. A null procedure call requires about 25 msec.

Even for relatively low load of a remote server and low traffic through the network, frequent execution of remote operations can be very costly. For example in case of remote paging, the degradation in performance is tolerable only for rates no larger than 1 page/sec. Clearly these values depend upon the speed of the remote server, its load, as well as the network traffic load. But significant performance degradation may occur for relatively small variations in the execution rate of the remote operation.

3. ADMA ARCHITECTURE

3.1. Heterogeneous systems interconnected by a broadcast channel

A tendency toward computing based upon powerful user dedicated workstations can be observed. Cost/performance considerations as well as the convenience of a distributed environment are the driving force behind this tendency. Different types of servers are integrated in such environments. File servers are the most popular ones now, but printing servers and computational servers will be likely to emerge. A computational server could be a machine with a special architecture needed, say, in scientific or knowledge processing.

The need for remote services is increasing in such systems. It has already triggered the development of convenient user interface through Remote Procedure Call protocols. Once the networking software becomes friendlier an increased number of applications will use it.

Functional communication becomes an intriguing approach to communication in such systems. It does eliminate the lowest levels of present communication architectures and leads to potentially simpler systems. At the same time the basic communication mechanisms like flow control and error control need to be exercised only at the remote computation level and hence better performance could be expected.

A detailed description of the communication primitives as well as the mechanisms to establish a functional connection are beyond the scope of this paper. We focus only on the communication aspects of distributed systems.

An important observation is: the communication channel becomes the central server of such a system and hence the algorithms to allocate this resource are now very important. The transmission of each data unit requires the execution of the channel access algorithm.

To analyze the transmission delays for a data unit one has to take into account processing and waiting times at both ends, the transmission time and the propagation delay as well as the overhead and the waiting time associated with the execution of the channel multiple access algorithm. The objectives of any communication architecture are to reduce as much as possible the components listed above. Increased processor speed and faster communication channel help decrease the total communication delay of every data unit. But we advocate an approach based upon qualitative changes in the way entities communicate rather than on quantitative ones.

3.2. Motivation for ADMA

The overhead associated with switching the communication channel is implied by the transmission of every data unit. For multiple access algorithms this overhead depends upon: the channel load and the propagation delay. As we have argued earlier, the channel load is likely to increase in systems like the ones described here. There is little hope of decreasing propagation delay. So we have to investigate new methods of sharing the communication channel.

On the other hand, faster channels will allow larger data transmission units. But unless we improve the communication paradigm it is questionable whether one could effectively use larger transmission data units. The bimodal packet distribution discussed earlier indicates that the network carries a considerable amount of control and status information delivered as short packets. We conjecture that functional communication could make better use of larger transmission data units.

Another crucial observation is that a user-server community is highly nonhomogeneous as far as the traffic is concerned. Server nodes are considerably more active than user nodes. At the same time there are much fewer server nodes than user nodes. Hence it does make sense to consider the possibility of giving priority to servers in controlling the channel. In this case only one execution of the channel access algorithm will be required to send an entire queue of processed results, when a server is in control of the channel. At this point we should observe that grouping of nodes in a broadcast system other than user and server nodes is possible, and other methods of sharing the channel may also be possible.

3.3. Functional description of an ADMA system

Each server in the system provides a set of services. The mechanisms to start a server process on a server machine are part of the establishment of a functional connection. They can be implemented as a service request sent to a known port available on each server machine, and based upon a system-wide known set of semantics.

When a server is in control of the channel it broadcasts the list of services it is capable of and willing to perform. Then it empties its output queue containing the results of previous computations during a *duty period*. When the duty period is terminated a *service period* begins. Any client which needs any of the services advertised sends its request using a multi-access algorithm. If a certain service is offered by multiple servers, each of them will receive the request and treat it according to its individual scheduling policy. The heavily loaded servers may even ignore it. If the client does not request multiple instances of execution of its request, all servers in the same functional group will have the option to discharge the request in whatever phase of processing it is as soon as they observe another server's reply.

A functional group consists of a set of servers capable of performing a certain service. When a *service period* is terminated, the server relinquishes control of the channel, and depending upon the scheduling algorithm, another server will obtain control of the channel. A large number of variations of the scheme described above can be imagined.

An ADMA system seems to enjoy a number of desirable properties as far as performability is concerned. By communicating with a functional group rather than with a particular server, the client is no longer concerned with the name or the location of the server(s). The system seems more robust, and more reliable. Stateless servers can be easily implemented.

An interesting observation is that communication is essentially carried out by parallel RPC's. The implementation of parallel RPC's is rather tricky in other systems. The client has to dynamically update the multicast list of target servers. Further, the servers tend to respond at the same time. In the case of collision based multiple access methods this leads to an excessive level of collisions. Additionally, an ADMA system can dynamically balance the the load on its servers. The drawback of an ADMA system is probably an increased response time, especially for services bound to service groups with low population.

3.4. Scheduling protocols

The concept of a scheduling protocol introduced in this paper reflects the need for a coordinated access to a communication channel and to a number of resources connected to their users through a shared communication channel.

Multiaccess protocols provide an answer only to the problem of sharing a communication channel [1], [4], [11], without any concern for why the need for communication occurs. In our communication model we consider the problem of causality, by recognizing that communication occurs only in connection with a request to use a remote resource. Hence, independently of processor and communication channel scheduling strategies, we can identify in each use of a remote server a cycle consisting of the following sequence of events:

- a user node needs to use a remote resource; as a result of this need a request packet is created at the user's site,
- the request packet joins a queue of local requests, *RQ (Request Queue),* waiting to gain access to the communication channel,
- when the channel becomes available the request is transmitted to the server, and joins a queue of system wide requests waiting to be processed by that server, *IQ (Input Queue),*
- the request is processed and results are obtained,
- the output packets containing the results are added to an *OQ (Output Queue),* at the server's site, waiting to be transmitted to the node which has originated the request,

- the result packets are transmitted; when received by the user node they join the queue of partial results of processed requests, *PR (Processed Queue)*. There, they might wait for other partial results from other servers or may be interpreted independently.

A scheduling protocol correlates channel allocation with resource allocation based upon the sequence of events described above. Scheduling protocols are based upon a two level approach to the problem of multi-access. *First of all, they enforce a communication discipline allowing only the servers to control the communication channel.*

To allow the servers to share the communication channel various strategies may be used, ranging from random, to limited contention [11] and to contention-free server multi-access. Random multi-access schemes, though easier to implement, will probably lead to a longer channel acquisition period for higher input rate and consequently to a lower channel utilization. In the case of random multiple access, instability problems are likely to occur. The resulting system will be less fair but more robust than systems based upon other multi-access methods. On the other hand, when contention-free multi-access protocols are used, such as when servers form a logical token passing ring, the system will have additional desired properties. Not only will it be fair in giving each server a chance to use the channel, but it also gives higher throughput.

The server's multi-access strategy strongly depends upon the characteristics of the set of servers, namely, their number, their relative processing speeds and their relative usefulness to the set of users. In a network with a few servers, of comparable characteristics, providing similar functions, a contention-free strategy such as a token passing scheme is more reasonable since it gives each server an equal, orderly chance to use the channel and there is no need to allocate more channel time to a heavily used server. On the other hand, when the number of servers is large and their processing speeds and functions differ widely, limited contention or even random multi-access schemes are advantageous provided that the total level of traffic is relatively low.

Let us now examine the second important aspect of a scheduling protocol, namely, *how the channel is shared among all users in need of the service provided by a certain server*. Assuming that the server which currently has control of the channel decides that its present status allows it to acquire more work, it broadcasts a control packet informing all users of its availability and a description of the type(s) of services it is willing to perform. At that moment a multi-access algorithm for the set of users needs to be employed. Random multiple access, limited contention or contention-free algorithms can then be used to provide each user (in need of the particular type of service being offered) a chance to use the channel.

The arguments discussed in connection with the servers' multi-access method are also valid in case of users' multi-access method. Random multi-access is more adequate for systems with a large number of users which spread their requests evenly onto the set of servers, and for which at any given moment of time the average number of requests waiting to be sent to the remote servers is relatively low. The channel acquisition period can become large and performance degradation can be significant when the average length of the Request Queues is large. Another factor which must be taken into consideration is whether a given server should accept all service requests or only one request, in response of its willingness to perform a certain type of service. In case all requests have to be accepted, a good solution is to use limited-contention multi-access channel acquisition algorithms.

3.5. A Taxonomy of Scheduling Protocols and ADMA Systems

A scheduling protocol is characterized by the pair consisting of the server multi-access algorithm and the user's multi-access algorithm, denoted by the pair: (S, U). Each of these algorithms could be: random (R), collision resolution (CRA), carrier sense multiple access with collision detection (CSMA-CD), or contention-free (CF).

The architecture described in this paper can be implemented for different topologies of the interconnection network. We will first investigate the case of a bus network topology, and conflict-free scheduling protocol among servers and each of the above protocols for users. Our choice is motivated by the fact that this environment is probably easier to understand and to analyze than other pairs <topology, scheduling protocol>.

The philosophy governing the ADMA is that the servers must have priority over the users for the use of the communication channel, since this is considered to be the chief resource in the system. A user may send a request for service only when a server capable of processing that request has broadcasted its willingness to perform such a service. In the systems considered here the servers pass channel control to one another in a cyclic manner, through the use of a control packet called a *CAP*, or *Channel Allocation Packet*. An important measure of the system performance is the *CAP-cycle-time*, which is the time needed by the CAP to cycle through all the servers.

The servers may certainly use other multiple access methods such as collision resolution with reservations or CSMA-CD with reservations, in which case they would compete for the control of the communication channel rather than share it peaceably. In such a case as soon as the server obtains control of the communication channel, all competition stops until the server in control terminates its own reservation period by explicitly broadcasting the CAP to another server.

Each server maintains two queues: an input queue (IQ) for gathering requests from users, and an output queue (OQ) for storing requests already processed, awaiting transmission via the channel to respective users. When a given server has received the CAP (and consequently is in channel control), it performs two types of tasks. It first empties its output queues by returning results of already processed requests to the respective request-originating users. Next, it attempts to solicit more work (if it is able) from users.

A server will announce its willingness to acquire more service requests by broadcasting a control packet called *the SAP (Service Availability Packet)*, if its current workload permits. If a server is heavily loaded it will bypass the broadcasting of its SAP, for one or more CAP cycles until its workload reaches an optimal level. This characteristic of ADMA systems makes them ideal for implementation of load balancing strategies with a minimum level of traffic dedicated to status and control information. As soon as a server has broadcasted its SAP which contains a descriptor identifying all services its owner is willing to perform, the system enters the *user's phase*. In this phase all users which have requests for service, matching the ones offered by the SAP, attempt to send these requests.

The first type of system modeled is a *token passing system*, [8], [13]. In this case all users are arranged in a logical ring, and each knows its predecessor and successor in the ring. The SAP contains the address of the first user allowed to send a request. In order to ensure fairness the first user to receive the SAP is the successor of the last user which has received service in a previous CAP cycle. When a user receives the SAP, it sends its request(s) for service if it has any, and then it passes the SAP to its successor. This procedure continues until the SAP has made a full cycle through the ring of users. After a SAP cycle, the server which has originally broadcasted the SAP, passes the control of the communication channel to its successor by sending the CAP. It should be noted that when a user has multiple requests for service, different strategies can be used, e.g., all requests can be sent (exhaustive service), only already waiting requests (gated service), at most a fixed number of requests (nonexhaustive service), or at most one request (single request strategy). A request for service may consist of one or more data packets sent through the channel.

The second type of system modeled is one using a *Collision Resolution Algorithm (CRA)* [11], [14]. In this case all users with ready service requests attempt to transmit them when they hear a SAP. If more than one user attempts to send its requests, a collision occurs and a splitting algorithm is used in order to assign to each contender its own transmission slot. The algorithm described in [7] and attributed by Massey [11] to Gallager is used. Each user involved in a collision flips a binary coin. All those who flipped a 0 transmit in the next slot while those who flipped a 1 transmit only *after* those who flipped a 0 resolve their potential collisions. The procedure continues until all the users who have initially collided in the first slot of the CRI (Collision Resolution Interval) have transmitted successfully. We use a blocking algorithm, and any user with a new request has to wait until the completion of the CRI.

The third system modeled is based upon CSMA-CD. *The Carrier Sense Multiple Access with Collision Detection* [15] requires that prior to the use of the channel, a user listens to the channel in order to determine if it is idle and if not, politely delays its attempt to transmit. If a user is involved in a collision (there is still a vulnerability period when two or more users could get the impression that the channel is idle) then the user has to wait a random amount of time before attempting to retransmit again. More precisely a user will transmit with probability $\frac{1}{L}$ in one of the next L slots. We have considered two policies concerning retransmission which can be best explained in terms of a slotted channel. The first policy is the truncated binary exponential backoff [16] which starts with $L = 2$ and doubles the value of L each time the user experiences a new collision. The second policy one we call a a fixed CSMA/CD and utilizes a fixed value of L, namely $L = 5$.

4. MODELLING AN ADMA SYSTEM

In this section we describe the queueing model of the ADMA based upon which we built a simulation model of the ADMA. Some preliminary simulation results are discussed.

4.1. Queuing Model of an ADMA System

The ADMA system consists of three components: a set of users, a set of servers and the broadcast channel. The channel is characterized by scheduling protocols as described earlier. In this subsection we present a queuing model for a server and a user. This model is used in simulation to obtain global performance measures of the system.

Let us start with a user description. We assume that an external source of information (SI) generates requests to the system (to users) according to a Poisson distribution with a rate of λ requests per slot. Henceforth, we refer to λ as the *generated request arrival rate.* Requests from SI are routed to the i-th user with branching probability p_i, $i = 1, 2, \ldots, M$. Each user contains K queues for K different type of services, as described in Sec. 2.2. The input rate to the k-th queue at the i-th user is $\lambda\, p_i\, q_k$, where q_k is branching probability for the k-th type of service. The queues at users are finite, and throughout the simulation we assume that the capacity of a user's queue is equal to five messages. The real input rate introduced into a user's queue is called the *effective arrival rate.* The message length (in packets) is geometrically distributed. We assume that only eight types of generic services are provided by servers, i.e., $K = 8$. Moreover, each user's service discipline is exhaustive (up to the limit specified by the capacity of the user's queue). We also assume that each user's request has some pre-processing time, which is set to be constant.

According to the scheduling protocols described earlier, users have access to the channel only *after* receiving an SAP from the server currently soliciting service from users. When an SAP is received, a user checks to determine if there is a match between services provided by the SAP and the actual user request. If a match occurs, the appropriate user's queue is served in an exhaustive manner. The request is transmitted through the channel to the active server, i.e., to the server which possesses the CAP packet. Note that the user's queue may be modeled as M|G|1 system with server vacation.

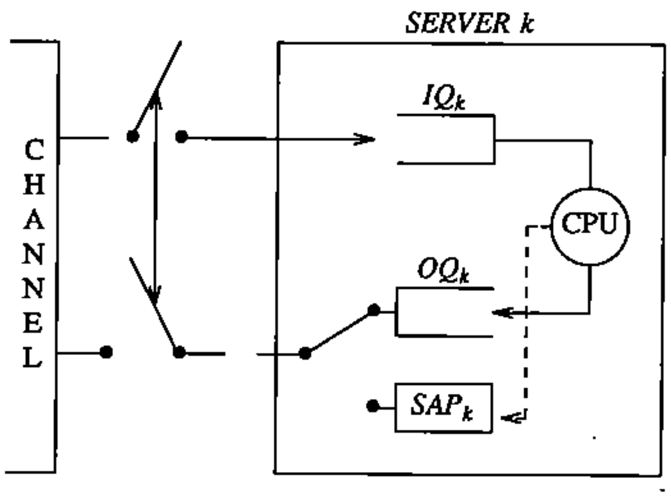

Figure 2: Queueing model of a server in ADMA Adopted for simulation.

In Figure 2 a model of the k-th server is presented. The server contains two queues: the input queue IQ_k, and the output queue, OQ_k. In addition, an SAP block is shown in the figure. A server may access the communication channel only when in possession of the system's CAP. This is represented in the figure by 'gates' labeled CAP. When a server is active, it exhaustively serves the output queue, OQ, and then based on information regarding the utilization of CPU and other resources in the server, transmits the SAP packet. After sending the SAP, the server waits for request input from users. These requests are queued in the input queue, IQ_k of the server. The active phase of a server is terminated when the server transmits the CAP to its (server) successor in the servers' logical ring. Both queues, i.e., IQ and OQ, are assumed to have an infinite capacity. Note that the input queue may be modelled as a $GI^{[X]}$ |M|1 queue (batch arrivals), and the output queue as a G|M|1 queue with server vacation.

When a server is not in possession of the CAP it processes the jobs from the input queue. The length of a job is assumed to be geometrically distributed. The average job length is either "small" or "large" as described above. The queuing discipline for IQ is FIFO. The simulation model is used to estimate some characteristics of the system. In particular, we are interested in total delay in the system and the average value of the CAP cycle time. The results and discussion of the simulation are presented in the next section.

4.2. Simulation Experiments

The simulation model has the following parameters:

(a) The number of users is is fixed at $M = 10$. The branching probabilities of generated requests are uniformly distributed into the set of users so that all users generate service requests at approximately the same rate, this making the user subsystem symmetric.

Each user has as many request queues as service types offered in the network. in our case $K = 8$ and we assume that the branching probability of a service request of being of type i is $q_i = \frac{1}{8}$ for $i = 1,...K$. Each request queue has limited buffer space where a maximum of 5 requests can be queued. If the request queue is full all incoming requests are rejected.

The maximum number of users allowed to send during a SAP cycle is 10. On the other hand only one queue per user is serviced during the SAP cycle so that each of the 10 users has the opportunity to send requests. An exhaustive service policy is implemented, so that all requests in the queue selected by the user are sent. Consequently the maximum number of service requests collected by a server during a SAP cycle is $5 \times 10 = 50$.

(b) The number of servers is fixed at $N = 4$. All servers are identical in terms of processing speed and other resources (buffer space). Each server provides two different types of services, one computationally intensive, the other involving a light computation. Service times are geometrically distributed with means of 20 for computationally intensive services and 1 for the light ones. Input and output queues at the server have a large capacity, and they can be assumed to be infinite buffer queues, so that no rejection takes place at these queues.

Whenever a server is in control of the channel it exhausts its output queue in returning processed requests to users. Here we assume a fault-free routing of results to correct users.

(c) The channel is assumed to be slotted with slot duration equal to a packet transmission time. Transmission times are geometrically distributed with means 1 for s , small, and 3 for l, large. Propagation delays are considered very small and are neglected.

The investigation of the different types of ADMA systems presented as a function of varying input request rate is a central issue of this report. The objectives of our simulation study are: delay analysis, the determination of critical cycle times, for the CAP and for the SAP, as well as the estimation of the inherent overhead related to system control. In order to make use of the simulation results in analytical modeling, we have considered *a totally symmetric system in which the behavior of all users is identical, and all servers are symmetric.*

If we denote by n_I the average number of packets required to transmit a service request from a user to a server, then in such a symmetric system each user (there are M such users) generates communication requests at the rate:

$$\lambda_{c,u} = \frac{n_I \times \lambda_{ef}}{M}$$

Here the *effective request arrival rate* is defined as:

$$\lambda_{ef} = (1 - P_{rej}) \times \lambda$$

with P_{rej} the probability of rejection and λ the generated service request arrival rate.

If we denote by n_O the average number of packets required to transmit the results of processing one request from the server to the user, and if the server subsystem is symmetric then each server generates communication requests at the rate:

$$\lambda_{c,s} = \frac{n_O \times \lambda_{ef}}{N}$$

The overall communication load on the channel is:

$$\lambda_c = M \times \lambda_{c,u} + N \times \lambda_{c,s} = (n_I + n_O)\lambda_{ef}$$

In our case $n_I = 2$ and $n_O = 2$, hence:

$$\lambda_c = 4 \times \lambda_{ef}$$

The systems under investigation were subject to service request rates in the range 0.1 to 0.4 requests/packet time. A simple reasoning has indicated that our system is communication bound.

Hence the system will saturate when $\lambda_c = 1$. It follows that the maximum service request rate for which the system is capable of processing all incoming service requests is: $\lambda_{max} = 0.25$. More precisely, if the users buffer were made infinite in capacity, then the maximum throughput would be about 0.25. Below this rate the effective request arrival rate is "almost" equal to the generated one since rejections are small enough to be negligible. Above this rate this system will start rejecting incoming service requests through the rejection mechanism built at the user level.

Clearly, the system will begin to saturate earlier since this rough estimate has not taken into account the communication overhead related to the distributed control of each multiple access method considered. In fact the difference between this value and the actual arrival rate at which the system starts rejecting input requests is a rough measure of control overhead.

We have performed *steady state simulation* experiments. To guarantee steady state the experiments were conducted for a large number of service requests arrivals namely 20,000 for $\lambda \geq 0.2$, 10,000 for $\lambda \geq 0.16$ and 7,500 for lower values of the service request arrival rate. The simulation time had varied in the range 75,000 for low arrival rates to 50,000 for large values of λ, all times being expressed in packet's time.

The simulation was performed using ASPOL. ASPOL is a very compact simulation system developed by Control Data and available on CDC mainframes and on the Cyber 205. It is suitable for simulation of computer systems. ASPOL is a process oriented simulation system based on FORTRAN. Four similar simulation environments, one for each type of ADMA system, have been constructed.

4.3. Discussion of Results

In this section we present the results of a set of intensive simulation experiments. Additionally, we attempt to justify, as best as we can, some of the trends that result from the experiments. We stress that the study is not concluded and is only a preliminary effort intended to yield some insight into the complex interactions of a client-server based ADMA network. More detailed discussion is offered in [9].

To begin with, consider the relationship between *effective* and *real* arrival rates for service requests under each of the four protocol operating modes. By definition, the effective service request arrival rate is the number of requests that have not been rejected per slot. Clearly, for a probability of rejection equal to zero (i.e., buffers of infinite capacity) the best that a system can do is offer an effective rate *equal* to the real rate. Observe that the collision-free ADMA protocol lays claim to this behavior for real loads λ such that $\lambda \leq 0.25$. It appears that $\lambda_{max} = 0.25$ causes all ADMA protocols to begin rejecting incoming requests. A heuristic explanation of why $\lambda_{max} = 0.25$ approximates the maximum throughput was given in Sec.4.2.

Figure 3 presents the total delay in the ADMA versus the effective service rate. As expected, the <CF, CF> scheduling protocol exhibits the largest delay for small values of the effective service rate. The reason for this is best explained in terms of the token ring protocol. In the latter, message delay is higher than for the CSMA/CD protocol for (comparable) low loads simply because of the *relatively* high token-passing overhead at low loads. In contrast, the CSMA/CD protocols exhibit little or no channel overhead at low loads. In the ADMA system, the effect is basically the same. Note that the trend for request delays is the same, with the largest contribution to delay (coming from a queue) varying with the λ range. It is of considerable interest to determine the value of λ for which the collision-free ADMA protocol saturates.

Finally, Fig. 4 demonstrates mean CAP cycle-time for the four protocols on the ADMA system. These quantities might be used to build a hybrid-analytical model of ADMA. It turns out that an analytical model of ADMA is available if one knows the first two moments of the CAP and SAP cycles times [17]. Unfortunately, an analysis to determine cycle-time distribution is certainly very intricate, if at all possible. A possible way out, and one we plan to use, is a moment approximation approach that utilizes the first two cycle-time moments obtained from simulation

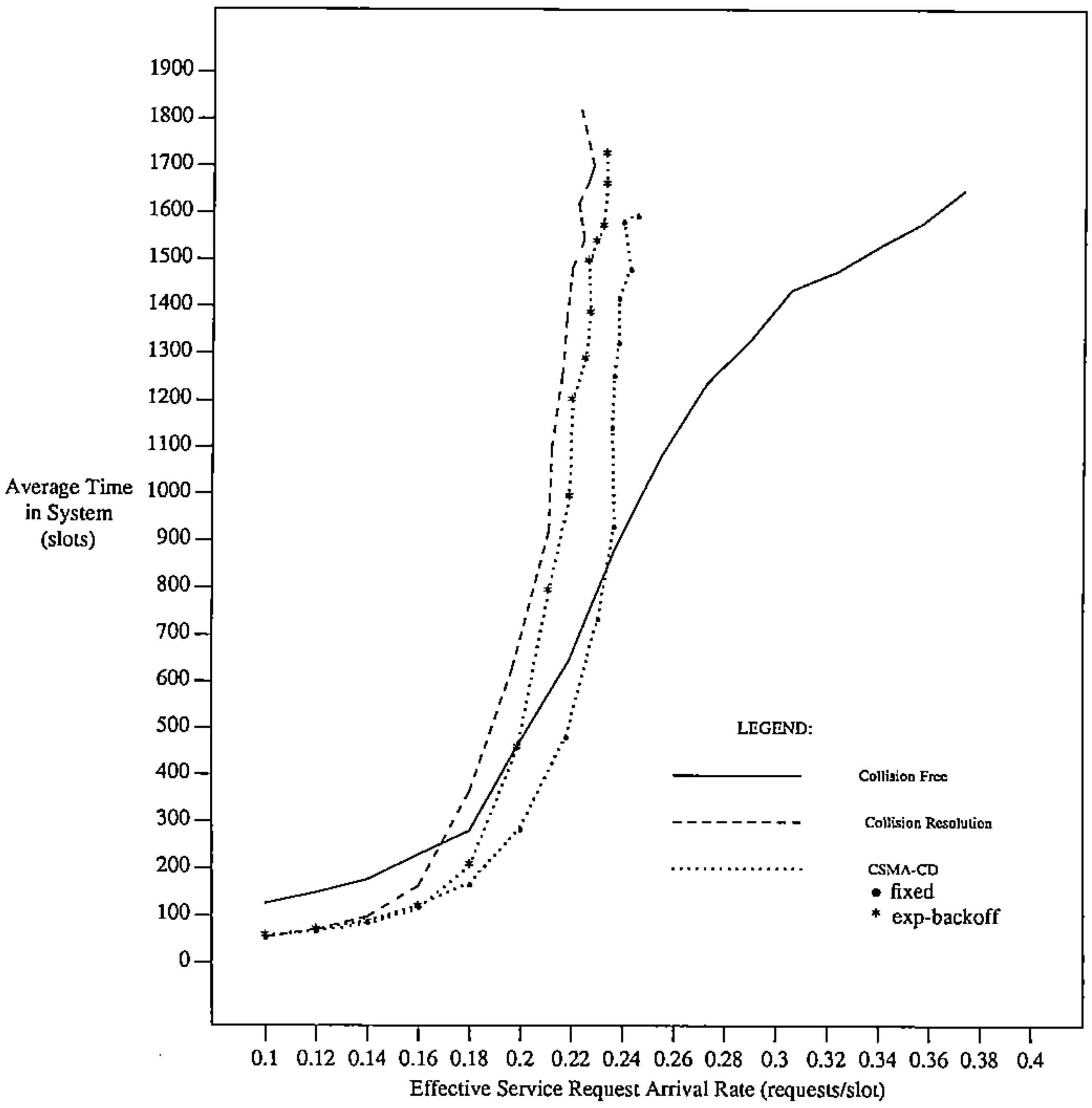

Figure 3 Average Time in System versus Effective Request Arrival Rate for Different User Multi-Access Strategies in an ADMA System

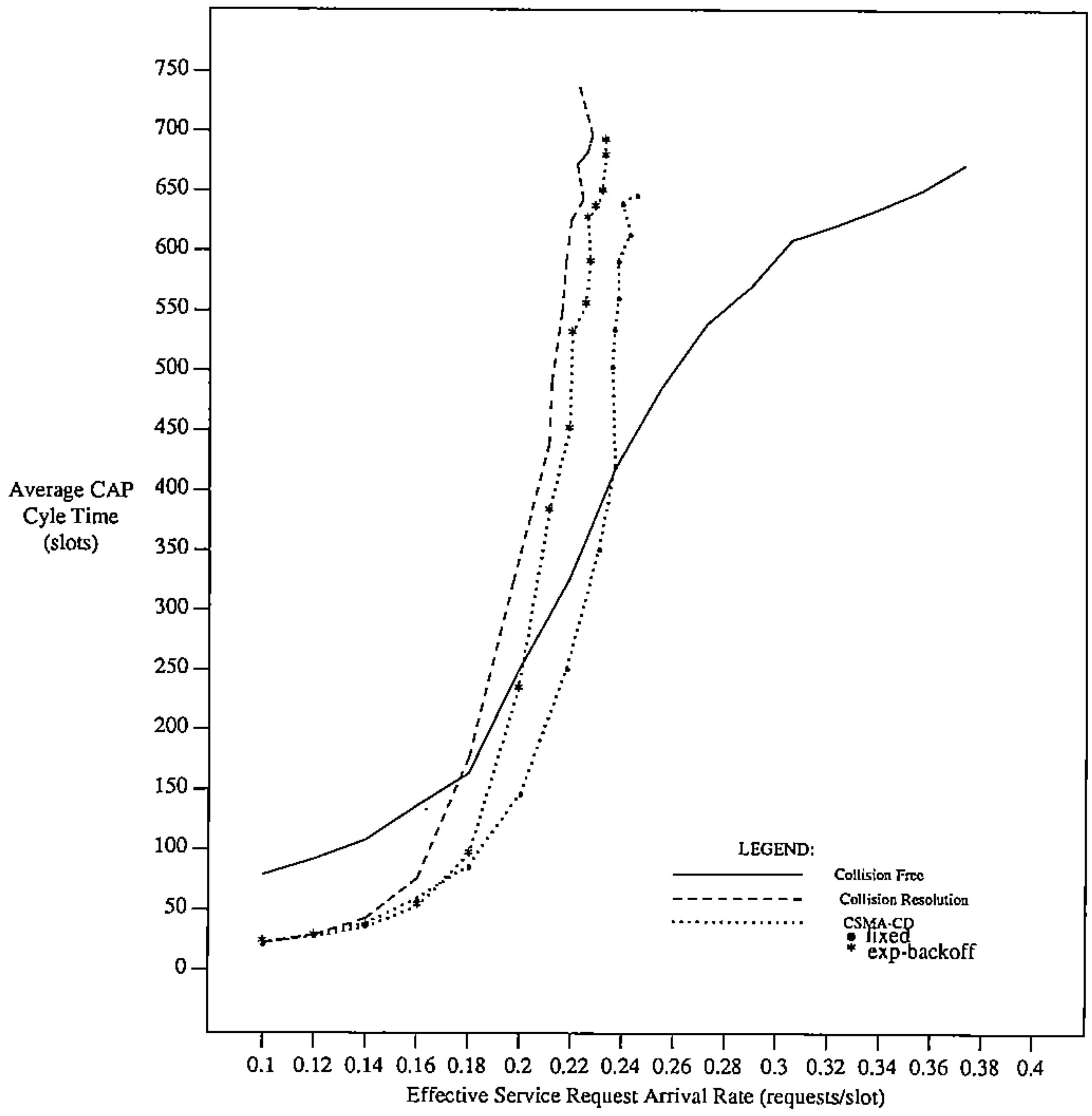

Figure 4 Average CAP Cycle Time versus Effective Request Arrival Rate for Different User Multi-Access Strategies in an ADMA System

in an analytical model of the ADMA.

ACKNOWLEDGMENTS

The authors express their thanks to Paul Albitz and Bavasch Patel who have performed measurements related to file transfer execution.

REFERENCES

1. Gallager Robert G., *A Perspective on Multiaccess Channels*, IEEE Transactions on Information Theory, vol IT-31, No. 2, pp. 124–142, March (1985).

2. Gihr O. and Kuehn P.J., *Comparison of Communication Services with Connection-Oriented and Connectionless Data Transmission*, Proceedings of the International Seminar on Computer Networking and Performance Evaluation, Tokyo, September (1985).

3. Hayes J.F., *An Adaptive Technique for Local Distributions*, IEEE Transactions on Communication, vol. COM-26, pp. 1178–1186, August (1978).

4. Kleinrock, L., *On Queuing Problems in Random-Access Communication*, IEEE Transactions on Information Theory, vol IT-31, No. 2, pp. 166–175, March (1985).

5. Kuehn P.J., *Multiqueue Systems With Nonexhaustive Cyclic Service*, B.S.T.J., vol 58, 3, pp. 671–698, (1979).

6. Marinescu, D.C., *On the Behavior of Programs With Remote Procedures*,CSD-TR-636, Computer Sciences, Purdue University, November (1986).

7. Marinescu, D.C, *Scheduling Protocols Based Upon Collision Resolution Algorithms*, IFIP Networking Symposium, Toulouse, France, November (1986).

8. Marinescu, D.C., Rego, V. and Szpankowski, W., *Heterogeneous Local Networks Supporting Scientific and Knowledge Processing Based Upon a Token Passing ADMA System*, IEEE Networking Symposium, Washington DC, November (1986).

9. Marinescu, D.C., Rego, V. and Szpankowski, W., *Modeling of an Availability Driven Multiple Access Network Architecture*, CSD-TR 626, Computer Sciences, Purdue University, submitted to SIGMETRICS 87, September (1986).

10. Marinescu, D.C., *Functional Communication in Distributed Systems*, in preparation.

11. Massey J.L., *Collision Resolution Algorithms and Random-Access Communications*, University of California, Los Angeles, Rep. UCLA-ENG 8016, April (1980).

12. Meister B.W., Phlippe, J.A. and Svobodova, L., *Connection-Oriented Versus Connectionless Protocols: A Performance Study*, IEEE Transactions on Computers, vol C-34, No. 12, pp 1164–1173, December (1985).

13. IEEE Standard 802.4, Token Passing Bus Access Method and Physical Layer Specifications, 1984.

14. Capetanakis, J., "Tree algorithms for packet broadcast channels", IEEE Trans. on Information Theory, IT-25, 1979, pp.505-515.

15. Tobagi, F., Hunt. V., "Performance analysis of Carier Sense multiple access with collision detection", Computer Networks, 4, 1980, pp. 245-259.

16 Metcalfe, R., Boggs, D., "Ethernet: Distributed packet switching for local computer networks", Commun. of ACM., 19, 7, 1976, pp. 395-404.

17. Kuenh P., "Approximate analysis of general queueing networks by decomposition", IEEE Trans. on Communications, vol. COM-27, 1, 1979, pp. 113-126.