1986

Scheduling Protocols Based Upon Limited Contention Multiple Access

Dan Cristian Marinescu

Report Number:
86-571

SCHEDULING PROTOCOLS BASED UPON LIMITED CONTENTION MULTIPLE ACCESS

Dan Cristian Marinescu

CSD-TR-571
January 1986

Sheduling Protocols Based Upon Limited Contention Multiple Access

Dan Cristian Marinescu
Computer Science Department
Purdue University
West Lafayette, Indiana 47907 U.S.A.

Abstract

An architecture for non-homogeneous local networks connecting a number of servers to a set of workstations has been proposed recently [5]. The Availability Driven Multiple Access (ADMA) architecture correlates the allocation of a common communication channel with a scheduling mechanism based upon server availability. ADMA is particularly suited for the case when the workstations generate requests for remote services at a high rate. In this case the communication channel becomes the critical resource of the system and simple and efficient connectionless communication protocols are needed. The scheduling protocols described in this paper are based upon collision-resolution algorithms. The multiple access to the common communication channel has two distinct phases: initially the subset of servers willing to provide services are allowed to compete. When a server is granted the control of the communication channel, then the subset of all users who need any of the services provided by that server will compete to send their requests. Then the server will send out the results of previous processing and relinquish the control of the communication channel. The resulting system enjoys an adaptive load balancing property, is robust and fair.

1. Introduction

The current trends in scientific and engineering computing environments indicate an increased interest in networks of workstations connected to a number of various types of processors with different architectures, different computing power, and providing a wide range of services [1-2]. The workstations which are themselves powerful machines, are dedicated to specific user applications, very often expert systems, and they generate requests for service from remote servers at a high rate. Examples of such requests are related to remote operations as: file access, procedure calls, paging, virtual I/O, editing and browsing of remote files, etc. The processors may provide generic type of services and specialized ones. An example of a generic service is to start a specific application on a processor and a specialized service can be related to the use of that application.

An important characteristic of such system is the need for a high bandwidth communication channel and efficient communication protocols. This need is related to the high ratio of communication related activities to the computational ones. The communication subsystem should provide functions suitable for implementing higher level primitives, for example remote procedure calls, that will be frequently used by the applications running on the workstations.

In our model, a user generates service requests from all available servers, with a certain probability. Because any server may request service from any another server, we view the system as consisting of N servers and N+M users, where M is the number of strict users connected

to the network. A *strict user* is one which cannot provide services for the others, for example an workstation dedicated to a user. Our communication model consists of three domains: most of the traffic occurs between *strict users* and servers and this first domain is the most important one for the performance of the system. The second domain is concerned with the traffic among servers and the third domain with the traffic among *strict users*.

The network is nonhomogeneous and has nodes with different traffic requirements. The amount of traffic in and out of a server node which provides a service in high demand is certainly much higher than the traffic in a user node, especially if the server is a fast computing engine. The traffic related to a server node can be highly asymmetrical, since very often computationally intensive tasks produce a large volume of output data for a small input.

Serious consideration must be given to the efficiency of communication protocols [3]. To complete a given task, a user may need to send a number of requests to each of the available servers and a considerable overhead could be involved in establishing connections with each available server. Connectionless communication protocols are more adequate than connection-oriented ones. A recent study of the performance of connection-oriented versus connectionless protocols, [4], shows that the performance penalty in terms of throughput can be substantial for the connection-oriented protocols. The results obtained show that in case of file transfer a connection oriented protocol at the link-control level may reduce throughput by as much as 50 percent. We expect that for a typical application in the class we are investigating, a similar performance degradation can be expected.

The study of nonhomogeneous local networks connecting a large number of workstations to a considerably smaller number of servers using either a ring or a bus network topology has lead us to the investigation of an Availability Driven Multiple Access architecture, ADMA [5]. The main idea of this architecture is to correlate the allocation of the common communication channel with the server's availability. In fact ADMA is based upon the concept that the need for the communication channel occurs only in connection with the need for service from one of the available servers in the network. A more efficient use of the communication channel can be achieved when the user in need of a certain service starts competing for the use of the channel only after the server or the servers capable of providing that type of service made known to the entire user population their availability and willingness to perform that service.

Multiple access protocols are concerned with the sharing of a common communication channel among all stations connected to it, and gives to each one of them an equal chance to use the channel. Scheduling protocols are a central concept in ADMA and they combine channel allocation with sharing of available computing resources. Scheduling protocols are based upon a two level approach for the sharing of the communication channel. Initially, only the servers are allowed to compete for the communication channel. As soon as a server has control of the channel, it may take a number of different actions, one of them is to broadcast an invitation to perform a set of services. Any user station may use this chance to send a request for service.

To provide full connectivity our model assimilates the communication among users with a service provided by some servers which can relay messages from one user to another.

2. Remote Procedure Call Execution - Efficiency Considerations

The remote procedure call paradigm seems very promising for the distributed environment needed for scientific and knowledge processing applications, discussed in this paper. Using the RPCs, both data and computations can be moved to any processor in the system. Nevertheless, the question of efficiency has to be taken into consideration especially when the frequency of executing RPC is relatively high and the granularity of remote operating is fine. Ultimately, if the overhead involved in a RPC is equivalent to say 2000 machine instructions, it will be rather inefficient to execute remotely say 50 useful instructions.

To provide an answer to the question to which extent the DPA (DoD Protocol Architecture) model provides a suitable environment for RPC implementation, we have performed several measurements in the local network of our department. We have a 10 Mbps Ethernet connecting several VAX 780 and 785, one 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 is relatively low, most of the time the load is lower than 5% of its capacity. The file server is responsible for 10-30% of the total network traffic. The packet size distribution is: 20 to 30% of the packets have a length of 970 - 1150 bytes, and 60 to 70% of the packets have a size in the 60 - 241 bytes range.

In this environment we have performed some rather crude measurements in order to estimate the processing overhead related to remote procedure call execution. We have defined eight types of remote procedures 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. The remote procedure call protocol is very simple and is based upon the UDP (User Datagram Protocol). A client was running on a VAX 785 and several servers were active on:

- the client machine (VAX 785),
- a VAX 780 connected to the Ethernet and acting as a gateway,
- a dual VAX 780 connected to the gateway through a 10 Mbps token passing ring (PRONET),
- a Sequent Balance 8000 connected to the INTERNET and located in California.

The error rates were extremely low for the local machines and considerably higher for the connection through a long haul network.

The results reported here are based upon a large number of measurements points, (930 for the Sequent, 2000 for the dual VAX and over 4000 for the other two) performed over a period of one month, at different loads of the server machines.

The variance of the response time is very low for short computations performed on the local machines. This indicates that the contention for the communication channel was very low and that the queueing delay at the remote server machines was small. The situation is different for the Sequent. Here the communication delay through the long haul network represents a significant part of the response time. Some observations related to the data presented in the Table 1 are:

- The overhead due to processing time in the upper layers of the communication software (the application layer, implementing the RPC, the socket layer and the IP (Internetwork Protocol layer) is significant and accounts for roughly half of the total processor overhead in case of local network connection. Since the remote procedure call protocol was very simple, this overhead can be confidently attributed to the socket and IP layer only. This overhead was measured when the client and the server process resided on the same machine and the IP layer recognized a local address and routed back the packets.

- The total processor overhead associated with the execution of the "null remote procedure", the one denoted by *s, s, s* , is in the 24-28 msec range. Since the packet transmission time and the propagation delay have a small contribution to the response time we can state that roughly 50% of this time is spent in the lower layers (UDP and the Ethernet driver) of the communication architecture.
- The overhead is significant for "fine-grain RPCs", the ones with "small" computations, where the response time increases by a factor of three from the <s, s, s> to the <l, s, l> type, on the same server machine. In fact the response time depends upon the number of packets exchanged during the execution of the remote procedure. In case of "coarse-grain RPCs", the ones with "large" computations, the variation of the response time for different types of RPCs is significantly lower. For example in case of the Dual Vax the increase is about 15 percent.
- In case of the long haul network other factors, especially the transmission time are determining the response time and the processor overhead is less significant.

The data presented above are consistent with similar measurements performed elsewhere [18] and they suggest that in order to support remote operations efficiently, alternative networking architectures need to be investigated. The values measured for the response time are likely to increase when the traffic load will increase. This will probably occur when the number of servers, for example disk servers and workstations connected to the network will increase. At the present time the processor overhead involved in all types of communication activities, is in fact a limiting factor which keeps the total network traffic at a low level.

As a result of increased traffic through the network, the communication overhead will become a factor which cannot be neglected when determining the total overhead, especially if the distribution of the packet size will have a maximum in the low size region.

The processor overhead related to communication functions could be significantly larger in other communication architectures with a larger number of layers than DPA.

The solution proposed here is to have the remote procedure call as the building block of the networking architecture, to move most of the communication functions currently implemented in the kernel to an interface and to correlate channel allocation with the availability of the servers capable to carry on requests for service through scheduling protocols.

Remote Procedure Type	95% Confidence Interval for the Response Time (msec) for Different Server Locations			
	VAX 785 (here the client is located)	Vax 780 (acts also as a gateway)	Dual Vax 780 (connected through the gateway)	Sequent Balance 800 (low haul network)
s s s	(11,11)	(24,24)	(26,28)	(866,898)
s s l	(12,12)	(48,48)	(60,62)	(1911,1971)
l s s	(19,21)	(42,44)	(54,56)	(2185,2237)
l s l	(22,24)	(68,70)	(86,88)	(3174,3216)
s l s	(292,298)	(523,535)	(390.392)	(1426,1454)
s l l	(341,351)	(663,689)	(425,429)	(2532,2592)
l l s	(353,365)	(793,837)	(418,422)	(2728,2766)
l l l	(375,389)	(933,989)	(458,464)	(3816,3876)

Table 1. Measured Response Time For Different Types of Remote Procedures.

3. Availability Driven Multiple Access Network Architecture

The basic concepts of ADMA can be easily explained in connection to a contention-free multiple access protocol for a token bus network topology. Reference 5 presents in detail such an architecture. The system can be viewed as consisting of two concentric logical rings, one containing only the M servers and the other one containing all N+M users as shown in Figure 1. Each logical ring is a token passing ring and there are two types of tokens, the first one visiting only server nodes and the second one visiting all user nodes. When the first type of token visits one of the servers, the server may generate a token of the second type which will visit all user nodes in such a sequence to ensure fairness. More specifically, when a server is visited by the first type of token, it has the control of the channel for a number of slots. It may first send out the results associted with earlier service requests and then generate the second type of token only if it is not overloaded with work and is willing to accept more service requests. Otherwise it will simply pass control to the next server.

Since both logical token rings coexist in a token bus and data packets are broadcasted so that all stations can hear them, the tokens are embedded into control packets which define the type of the token and the station which may acquire the token. In addition, each type of control packet may contain specific information. For example a server will include in the control packet of the second type the time it intends to use the channel and a descriptor of the type of services it can provide. Central to our concept is the idea that any user node, in particular the workstations can choose among a set of available types and stamp on each request they generate, a certain service type. When any user node is visited by a service token (the token of the second type) the decision whether the station will acquire the token or not is based upon a comparison of the type of service the server may provide and the type of service stamped on each request in the queue of requests waiting to be processed.

This concise description of ADMA implementation for a token bus topology, with scheduling protocols based upon collision-free multi-access, though incomplete, allows us to explore some of the important features of an ADMA system.

The system enjoys an adaptive load balancing property since the heavily loaded servers will not use their available slots to gather more work. Also the less loaded servers may adjust to this situation by offering to perform a wider class of services whenever this is possible. For example a vector processor may perform scalar computations when the demand for vector oriented computations is low. A server with a large volume of output may use its allocated channel slots just to send the processed requests from its output queue.

A user may also adapt to the current system load and level of performance. Since it can observe how often a certain type of service is offered the user may estimate the turnaround time for that type of service and may try to adjust by changing the type of request whenever this is possible. For example sometimes may be feasible to perform a vector oriented computational task on a scalar processor or vice versa.

Another important characteristics of the system is its robustness. The flow of control information is minimized. Each node is capable of performing accurate measurements of the load of each server, its response time, the communication delays, the availability of a certain type of service, etc. Though such information may be available from servers acting as network measurement centers, each node can obtain the raw data by monitoring the channel, it has the potential to act as a measurement center. The failure of any node including a server node is not a catastrophic event, unless that server provides a unique type of service and even in this case it will affect only a subset of the users. Only the immediate neighbors of a failing node need to update their image of the system, when they observe that the node fails to act when it is visited by an incoming token. The result for the users is a graceful performance degradation.

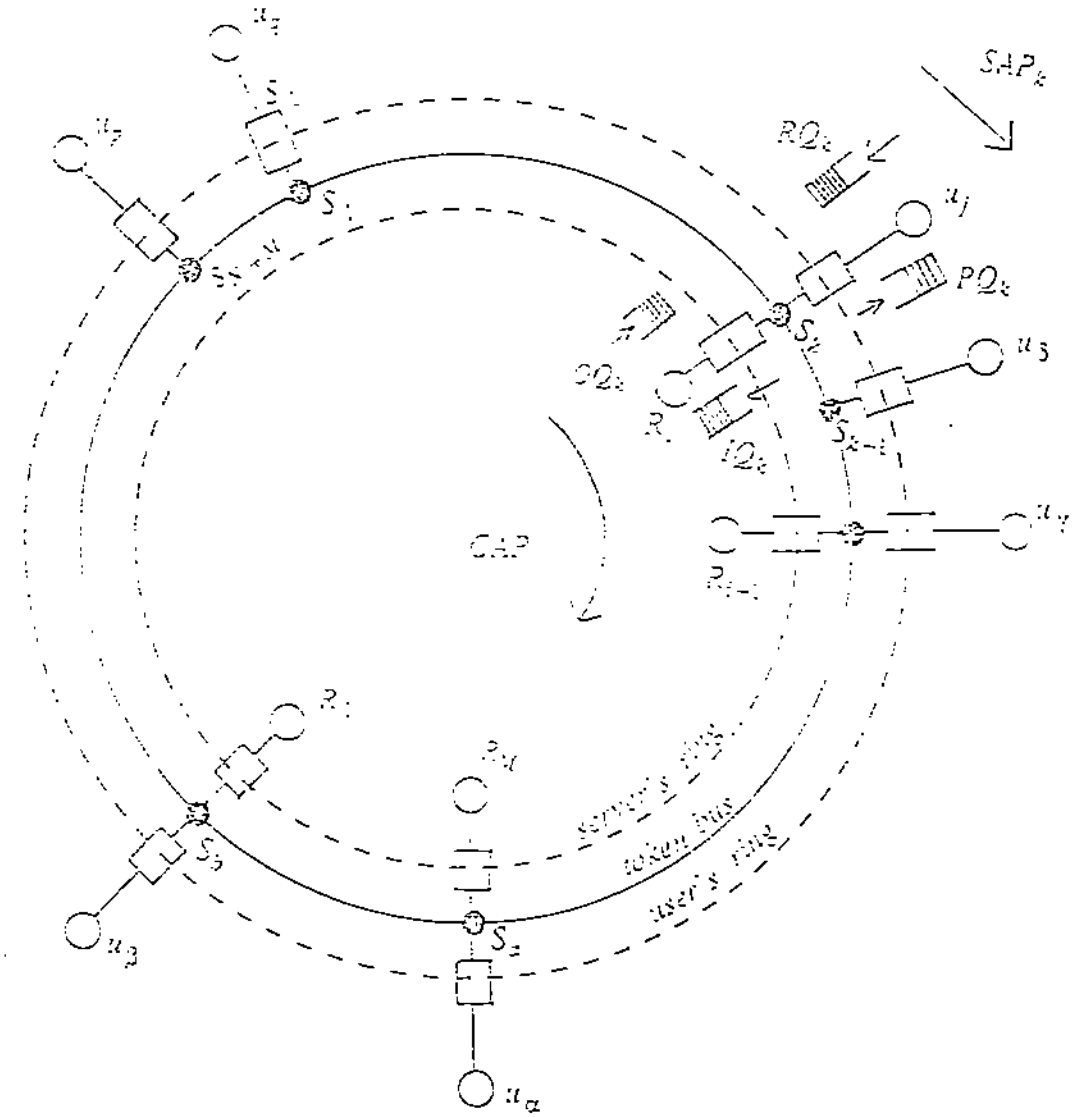

Figure 1. Contention free scheduling protocols

Another interesting property is the fairness of the system. Since a token ring is one of the most fair systems, the ADMA for a token bus, based upon contention-free multi-access scheduling protocols is also a fair system.

There are a number of issues that need to be carefully scrutinized to determine whether the characteristics of an ADMA system for a token bus and based upon a contention-free multi-access scheduling protocols can be improved.

First of all since many users address most of their requests to a subset of servers it seems only reasonable to construct a logical ring around each server and to have only the users which need service from that particular server connected to this ring, as shown in Figure 2. Obviously, a given user may be in a number of such logical rings. When a server has the control of the channel it should pass the service token only around his ring. This would eliminate unnecessary visits of the token to stations which do not need the service; as a result the mean time between visits made by a server token of a given type to a user node could decrease considerably. Unfortunately the price to pay for such an approach is rather high:

- The need to insert and delete nodes from individual server rings will increase and will vary according to the dynamics of the needs of each user. Such operations involve a large overhead and carefully designed node deletion algorithms must be incorporated to prevent a drastic performance degradation.
- The complexity of the Network Interface Unit (NIU) which connects a user node to the network will increase since each node will have to keep track to which logical rings it belongs to.
- The robustness of the system will suffer and the ability of any node to have a global picture of the system will decrease.

A second observation is: the processors connected to the network have different speeds and it seems unreasonable to have them visited by a channel allocation token (a token of the first type) at the same rate. A slow processor would probably require less frequent visits than a fast one. A related problem occurs when the number of servers is very low or when a few of them provide general services while the others provide only specialized ones. The problem of giving the control of the communication channel to different servers, at a different rate can also be solved but the price to pay is again increased overhed, more complex NIU for the server nodes and a decrease in robustness and fairness.

With this two important observations in mind it is only natural to consider the scheduling protocols based upon limited contention. In this case a subset of all nodes may compete for the communication channel, using one of the contention resolution algorithms known. A number of possible solutions can be analyzed:

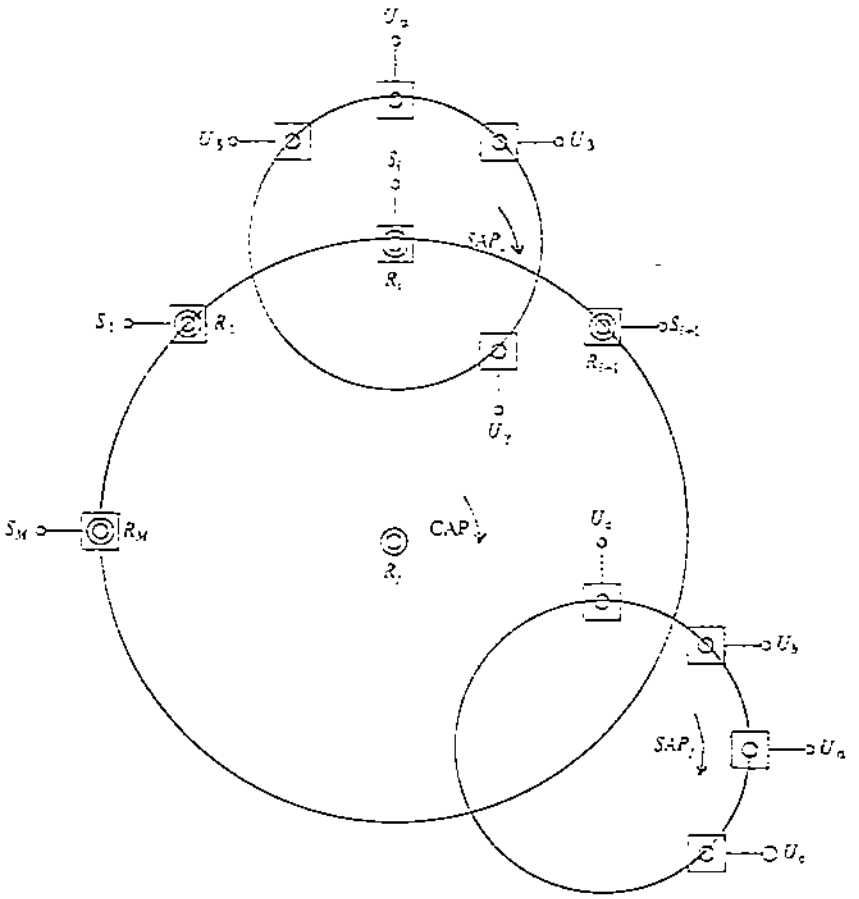

Figure 2: Contention-free scheduling protocols in a token bus with individual user rings around each server.

a. Only the servers are allowed to compete for the communication channel and when one gets the control of the channel it sends a polling to all stations allowing any user to get service from it. Such protocols will be called Type 1 scheduling protocols based upon limited contention. Such algorithms solve only the second problem discussed earlier.

b. The servers form a logical ring and they are visited by a channel allocation token. When a server has the control of the communication channel it may broadcast a special control packet indicating the type of service it is willing to perform. After receiving the control packet all users in need of the service offered are allowed to compete. The protocols in this family will be called Type 2 scheduling protocols based upon limited contention and they solve only the first type of problem discussed previously.

c. Finally, protocols which are capable of solving both problems will be considered and they will be called Type 3 scheduling protocols based upon limited contention multi-access. In this case contention will occur among the servers and then among a subset of users. First, all servers are allowed to compete for the channel. When a server has control of the channel it may broadcast a control packet inviting all users connected to the network to send requests for service. The contention among the users who send request packets is resolved by means of a collision resolution algorithm.

A bus network topology will be considered throughout this discussion but a contention ring is another alternative which needs to be explored and analyzed. In order to substantiate the use of contention resolution algorithms for multi-access, their properties will be outlined briefly.

4. Contention Resolution Algorithms

Random access over a time-slotted communication channel has been intensively investigated [6-7], in recent years. A random access scheme allows any sender to seize the entire communication channel when it has information to transmit.

In case of random access, the contention for the communication channel creates destructive interferences when more than one user may attempt to seize the channel simultaneously. Packet collision result in mutual destruction and the need to retransmit all packets involved in a collision.

Different algorithms for random multiple access have been investigated since Abramson published his paper [8], dedicated to the analysis of the ALOHA system. A major result was obtained in 1978, by Capetanakis who has proved that collision resolution algorithms exist, more specifically that tree algorithms are stable, have a good average delay properties and a maximum throughput higher than those of an ALOHA system. A collision resolution algorithm guarantees that all packets involved in a collision will be successfully transmitted during a finite time called a collision resolution interval (CRI). Capetanakis tree algorithm [9] makes use of the known past history and enforces a discipline allowing all transmitters whose packets have collided to retransmit successfully.

An entire class of splitting algorithms have been proposed recently, by Gallager [7], Hayes [10], Tsybakov and Mikhailov [11]. All such algorithms split the set of packets involved in a collision into two subsets, a transmitting subset and a non-transmitting subset. The process of splitting continues until the expected number of packets in a transmitting subset becomes of the order of one. Splitting algorithms differ in the strategy used for the splitting and they can be classified as blocking and non-blocking depending whether they require a transmitter with a new packet to wait until the collision resolution interval terminates or not.

The analysis of collision resolution algorithms is usually carried out using simplifying assumptions. A slotted channel with packets of unit length is assumed. Any packet is transmitted

in the slot (t,t+1) for some positive integer t. An infinite number of stations is assumed and each station at any time has at most one packet requiring transmission. At the end of each slot, each station knows which one of the possible events, idle slot, successful transmission or collision has occurred.

Massey presents in Reference 12 a blocking random binary tree algorithm for collision resolution and a simple and extremely interesting implementation of it attributed to Gallager. The algorithm is formulated as follows:

After a collision all transmitters involved flip a binary fair coin. Those flipping 0 retransmit in the very next slot, those flipping 1 retransmit in the next slot after the collision (if any) among those flipping 0 has been resolved. No new packets may be transmitted until after the initial collision has been resolved.

The analysis of this algorithm leads to the investigation of binary trees in which intermediate nodes correspond to slots with collisions while terminal nodes correspond to slots in which either a successful transmission has occurred or they were idle. A collision in some slot is resolved only when the algorithm has advanced to the point that the corresponding node is the root node of a completed binary subtree.

A binary rooted tree is a tree in which from every node including the root either no branches or two branches emerge. A node with no extending branches is a terminal node and a node with two extending branches is an intermediate node. A binary tree has an important property: *the number of terminal nodes is equal to the number of intermediate nodes excluding the root, plus two.*

Most of the high school students learn to prove this property in connection with a tennis tournament table in which N players are involved. The students are asked to find out how many matches must be played. Since every match eliminates one player and there is one winer, N-1 matches must be played. Therefore the number of terminal nodes (players) is equal to the number of intermediate nodes (games), including the final game (the root), plus one.

The algorithm implementation due to Gallager is based upon a reformulation of this property as follows: a collision in some slot is resolved when the number of subsequent collision free slots exceeds by two the number of subsequent slots with collision.

This implementation requires every transmitter to have two counters. The first one allows a transmitter to establish when he can transmit after being involved in a collision and the second one is used to detect when the original collision has been resolved and transmitters with new packets may start competing.

The collision resolution algorithm can be formulated as: when a transmitter is involved in a collision he flips a coin. If he flips 0 he may start transmitting in the next slot. If he flips 1 he must wait until all collision among those flipping 0 have been resolved. To detect this event he sets his first counter $C_1 = 1$ when flipping 1 after a collision in which he is involved and:

a. increments C_1 by one for each subsequent slot with collisions,

b. decrements C_1 by one for each subsequent collision-free slot,

c. when $C_1 = 0$ the transmitter may retransmit in the next slot.

If a transmitter does not have a packet to transmit he uses the second counter to establish when he may start competing in the event that a new packet arrives after the beginning of a collision resolution interval. He keeps the second counter initialized to 1 and whenever he observes a collision in a slot, the second counter is incremented. The second counter is decremented in each subsequent collision-free slot and incremented in each subsequent slot with a collision. When the second counter reaches 0, the CRI which started when the first collision was observed is terminated, and the transmitter may transmit the new packet.

An important measure of the performance of the algorithm is the conditional mean length of a CRI, defined as:

$$L_N = E(Y|X=N)$$

with X the number of packets transmitted in the first slot of a CRI and Y, the length in number of slots of the same CRI. Massey, [12], establishes very tight bounds for L_N and also an accurate approximation for it:

$$L_N = 2.9 \times N - 1$$

Though more sophisticated collision resolution algorithms exit, for example algorithms which can avoid *certain to contain a collision slots*, they can suffer deadlocks. For this reason our scheduling protocols will be based upon the simple algorithm described above.

5. Scheduling protocols based upon collision resolution algorithms

The collision resolution algorithms have unique properties among different classes of multi-access protocols:

a. Are based upon simple algorithms,

b. Exhibit good performance in terms of throughput and delay,

c. Allow a variable number of servers and users to compete for the control of the communication channel without an unacceptable level of control information exchange.

d. Can be adapted to ensure a high degree of robustness,

e. They ensure a higher degree of fairness than other multi-access protocols. More precisely, in case of a collision resolution algorithm every transmitter involved in a collision will transmit its packet in the CRI with probability 1. In case of a random access method, for example in case of an ALOHA system there is no guarantee that a transmitter with a packet will be able to transmit that packet.

These are the main reasons to select collision resolution algorithms for the scheduling protocols described in this paper.

Three distinct classes of scheduling protocols based upon random multiple access have been identified. The protocols of Type 1 belong to the first class and they solve the the first problem, allow a variable number of servers to compete for the control of the communication channel. But as soon as the server has obtained the control of the channel it polls all stations. This type of protocols could be inefficient since many users may not have any request for a given server. For this reson we will not describe and analyze this type any further but we will concentrate on the remaining two classes.

5.1. Type 2 scheduling protocols based upon collision resolution

This class of algorithms allows a variable number of users to send requests for service to each server without an excessive exchange of control information. As shown in Figure 3, the servers are organized in a logical token ring and each one of them is visited in turn by a token. The token is embedded into a control packet, called Channel Allocation Packet, (CAP), sent by the server that is giving up the control of the channel and addressed to the next server in the logical ring.

The server receiving the CAP may perform different actions depending upon the status of its input and output request queues. He can decide to gather more work, or empty his output queue of already processed requests, or both actions in this sequence or simply pass the CAP to the next server.

When server k decides to retain control of the channel, gather more work and send out processed requests. After receiving the CAP, server k immediately broadcasts its Service Availability Packet, SAP_k, that describes the type of services its owner is able to provide and then sets its counter to 1. Upon reception of this packet all users are capable to decide whether they need any of the services provided by that server.

A heavily loaded server will use channel allocation during the next cycles only to send out processed requests. Each server must adapt its behavior from cycle to cycle. The average number of users requesting service from it together with the average service time per incoming request and the average number of output packets generated for every input request are needed to determine the average behavior of server k. At each cycle the server needs to adjust these values in order to keep his input and output queue lengths within desired limits.

The functions of a Network Interface Unit for a user node are straightforward. They need to monitor the communication channel only when they have to send service requests. If the service type of the first request in the request queue of user i, is θ_j with $1 \le j \le K$, when the SAP_k, $1 \le k \le M$, the control packet broadcasted by server k is received, a very fast comparison of θ_j with σ_k will determine if the user will transmit in the next slot or not. In case of a match the user transmits, sets its internal counter to 1 in case of a collision and follows the collision resolution algorithm. After sending a packet the NIU will execute immediately the procedure *flip the coin* so that in case of a collision it is ready for the next step. A more sophisticated NIU capable of handling up to K queues of requests in parallel and, in case of a match, to send the request from the highest priority queue can be considered.

The NIU does not need to maintain a second counter to detect the beginning of CRI since a SAP_k marks it, for all users who need service from server k. The other users need not monitor the channel events until they see a CAP followed by the next SAP.

The functions of the NIUs for server nodes are similar to those described in reference 5. In addition they have to maintain a counter to detect the end of a collision resolution interval, following a collision among users requesting service from it.

The scheduling protocols discussed in this section are more robust than the collision resolution algorithms since any new user can join the transmitting set as soon as he observes an SAP_k of interest. A new user does not need to use sophisticated algorithms to detect the beginning of a collision resolution interval. Fairness is another property they exhibit. All servers get a chance to use the channel and all users in need of a service provided by someone in the network have a chance to use it.

5.2. Type 3 scheduling protocols based upon collision resolution

Let us consider now a system in which the M servers do not form a logical token ring so that each of them has a guaranteed cyclic access to the communication channel, but they have to compete for the channel. Clearly, such a system is more suited for a configuration in which high speed, general purpose servers providing services in high demand are mixed with specialized, unfrequently used servers. In addition the system behaves quite well when the number of servers is small.

At the beginning of a contention resolution interval for the servers, all servers ready to perform services will broadcast their SAPs. Each server will follow the collision resolution algorithm described earlier, with the following modification: as soon a SAP is successfully transmitted, all servers will stop following the collision resolution algorithm, by updating their two counters, until the server which has successfully transmitted its SAP will relinquish control of the channel by broadcasting a Termination Packet, TP. In the slot following a TP, the contention among servers will resume as if the successful server had used only one slot.

The successful transmission of a SAP will determine the beginning of a collision resolution interval involving the subset of the users interested in the type of service offered. The sequence of events will follow precisely the description given for Type 2 protocols but instead of passing a CAP to its successor, the server will broadcast the TP signaling that it gives up the communication channel.

The period of time starting with a collision among all servers ready to perform services and ending with slot following a slot in which the last one of them sends a TP will be called an extended collision resolution interval among servers, ECRI.

If we compare this algorithm with the Type 2 algorithm it is easy to observe that they will lead to similar implementations. In fact the NIUs for the users nodes are identical in both cases, but the NIUs for the server nodes differ. Instead of implementing a token passing procedure for the Channel Allocation Packet it must implement the collision resolution algorithm among servers.

6. Elements of Performance Analysis for a Type 2 Protocol

In this section is presented an analytic model for a very simple system using a Type 2 scheduling protocol. The system being modelled is essentially a multiqueue and multiserver system in which requests may be processed at the servers concurrently, but only one server or one user may keep control of the communication channel at any instant in time. The system is analyzed from the communications viewpoint, with the channel being treated as the chief resource.

This is a preliminary study and its main purpose is to gain insight into the system by defining precisely its parameters and behavior rather than to provide final results concerning the system's performance. Due to the complexity of the system, an exact analysis is not feasible at this stage and even an approximate analysis raises serious difficulties. The approach considered here is based on Kuehn's method of queuing network decomposition [16] and the analysis of different subsystems in isolation by assuming renewal arrival and departure processes. Simulation experiments are planned to validate different simplification assumptions considered in our approximate analysis.

In the followings we present first a model of the system and then we discuss approximations for determining the mean cycle time and delay.

6.1. Model Description

The logical interconnection of a set of server nodes and user nodes in an ADMA system using Type 2 scheduling protocols based upon limited contention multiple-access is presented in Figure 3. The stations connected to the logical ring are server stations while the ones connected to the logical bus are user stations. A server station is defined to be a computing system that can provide precisely one kind of service to any other station on the system that requires such a service. Since it is permissible for a server with ID x to require service of type y from the server with ID y, $x \neq y$, we see that a server station may also be a user station.

Let S denote the set of N server stations, U denote the set of M user stations, and $S^* = S \cup U$.

We assume that the subset of stations $S \subset S^*$ forms a *logical ring R_s of server stations*. It is necessary for each server station in S to know its predecessor server station (i.e., the station it receives a CAP from) and its successor server station (i.e., the station it must send a CAP to) for the ring R_s to be defined.

In steady-state, control of the channel is passed between server stations R_s, and the sequence of token-passes thus defined is infinite due to cyclic repetition. Since the network architecture is *availability driven*, the server stations maintain priority of channel usage over the user stations.

Let CAP(j) denote the event that server station j, $j \in S$, is in possession of the unique Channel Allocation Packet (CAP), or server station j has sent the CAP to its successor who has not yet received it. The analysis of the event CAP(j), server j is in control of the communication channel, is based on the diagram presented in Figure 4. The event CAP(j) is initiated at the instant server station j receives the CAP from its predecessor in the ring R_s. During the first part of the event CAP(j), server station j may use the channel to return already processed service requests to user stations, obtain information on network status, etc. Since these activities represent duties performed by station j, we associate the channel status during these duties with an event DUTY(j), for each j in S. When the duty cycle is terminated, server j broadcasts its SAP (Service Availability Packet) informing all stations connected to the network that it is capable to perform services of a certain type. The next event that occurs within the event CAP(j) is denoted as SERV(j) and corresponds to a collision resolution interval among all user stations which need the service provided by server j. When this collision resolution interval is terminated the CAP is passed from server j to its successor in the ring of server stations and this event is denoted by PASS(j).

The analysis of collision resolution algorithms is carried out using the simplifying assumptions discussed in section 3 of the present paper. Several additional assumptions are necessary to describe the scheduling protocol in a precise manner and to lead to a tractable analysis.

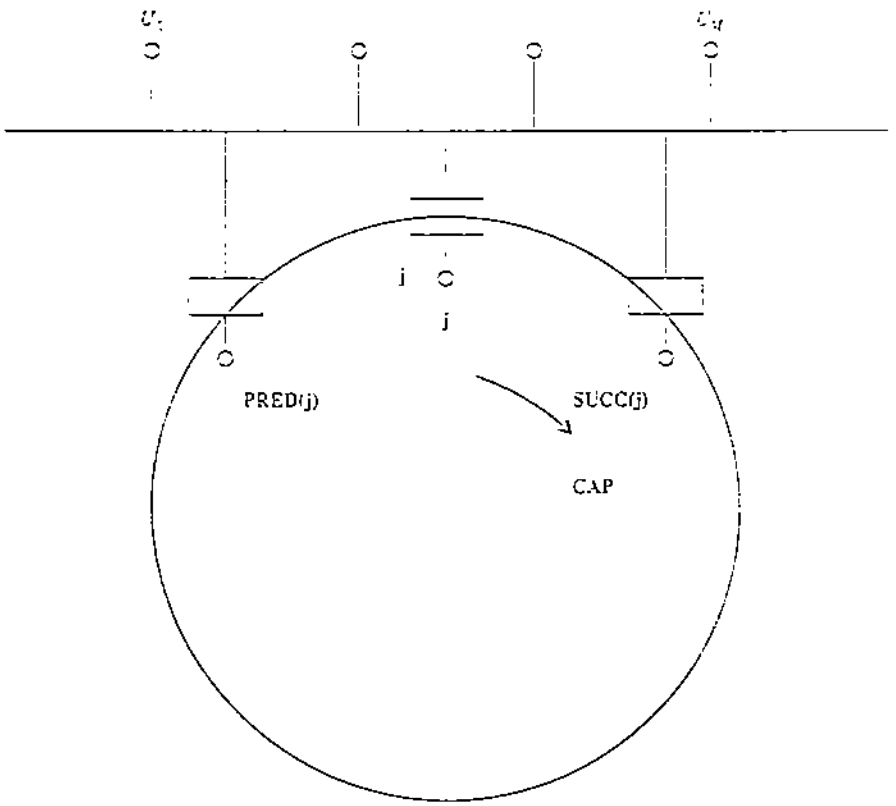

Figure 3: Logical interconnection of servers and users in an ADMA system using Type 2 scheduling protocols.

P1. Each user station in S* has N local buffers for holding N different types of queued packets. Packets of type j queued at user station i (in the jth buffer of this station) are service requests that can execute on (or obtain similar service from) server station j .

P2. Each server station in S provides only one type of service.

P3. Packets of type j queued at server station $j \in S$ (i.e., server station j 's own service requests for its own type of service) do not require to pass through the communication channel. That is, server station j handles these requests without the use of the channel, and in this sense such requests do not affect the operation of the protocol. For the duration of the event CAP(j), $j \in S$, the set of station competing for the communication channel excludes station j.

P4. The sequence of disjoint events CAP(j_1), CAP(j_2),....., CAP(j_N) where $j_{i+1} = succ[j_i]$ and the sub indices are incremented modulo N, is an infinite sequence due to cyclic repetition.

P5. For each server station j in S , the event CAP(j) is made up of disjoint events (see Figure 4), occurring in the order DUTY(j), SAP(j), SERV(j), PASS(j). The event SAP(j) corresponds to broadcasting of the Service Availability Packet. The event PASS(j), corresponds to passing of the Channel Availability Packet from one server to its successor. The total duration of both events, SAP(j) and PASS(j) is denoted by Y(j). Without restricting the scope of the analysis we can assume that the random variables Y_j are independent and identically distributed. The expected value of them is denoted by E (Y).

P7. The user station i may send *at most one packet of type* j to server station j.

6.2. Approximations for the Mean Cycle-Time and Delay

In this subsection we present a simple method to obtain expected values for certain cycle-times that are critical in determining the stability boundary for this protocol. Let server $j \in S$ be a reference server. Define the *CAP cycle-time* C to be the random time between two consecutive visits of the CAP at the reference server j. By symmetry, this random time C will have the same distribution independently of server index j, $j \in S$. The most important assumption made in this analysis is the assumption that *each server provides a unique type of service* (i.e., no two servers provide the same service).

Unless otherwise stated, we assume that all distribution functions are arbitrary, with finite first and second moments. In particular, packet arrivals of type k at station i are governed by a Poisson process with constant rate λ_{ki}, $i \in S^*$, $k \in S$. Arrivals at the different queues are mutually independent.

The random duration of the event CAP(j) is denoted by H_j and as indicated in Figure 4 it is the sum of:

a. the duration of: the duty cycle, DUTY(j), denoted by D_j,

b. the duration of the service cycle, SERV(j), denoted by L_j,

c. the duration of the communication overhead related to the transmission of control packets (SAP and CAP) and denoted by Y_j.

$$H_j = D_j + L_j + Y_j \tag{1}$$

The expected value of the CAP(j), is then:

$$E(H_j) = E(D_j) + E(L_j) + E(Y_j) \tag{2}$$

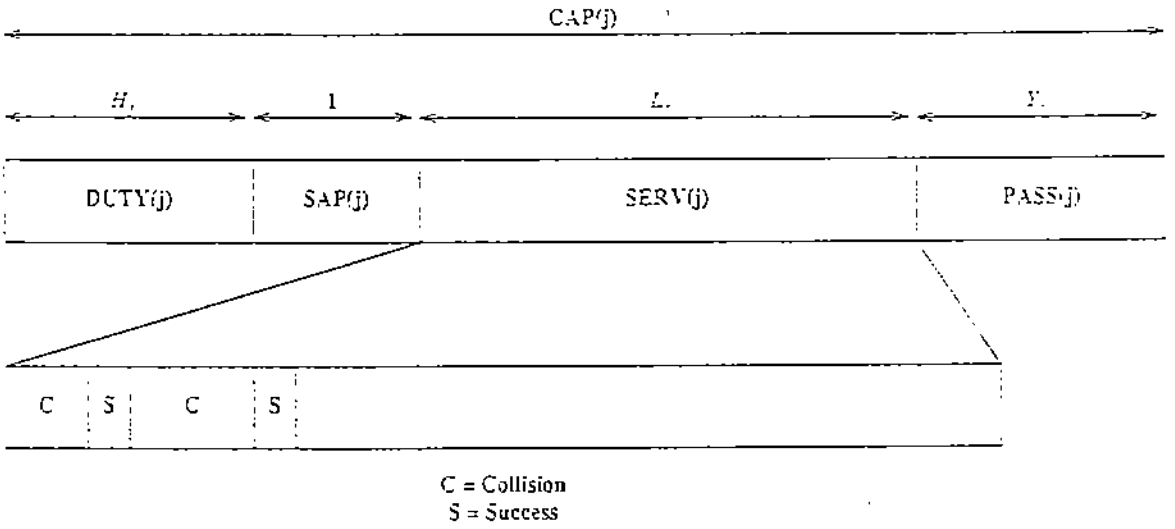

Figure 4: The event CAP(j).

As shown in Figure 5, the duration of the CAP *cycle-time* denoted by C is the sum of the duration of the events CAP(j_1), CAP(j_2),...., CAP(j_N). The expected value of C may be computed as the sum of the expected durations of each of the events. Though the events are disjoint *in time*, they are strongly dependent in general, as shown in [17]. This makes the problem of describing the distribution of C very difficult, but the expected value of C and the variance of C may be computed using approximations which will be discussed shortly. The expected value of C is:

$$E(C) = \sum_{j \in S} E(H_j) \tag{3}$$

Hence:

$$E(C) = N E(Y) + \sum_{j \in S} E(D_j) + \sum_{j \in S} E(L_j) \tag{4}$$

The computation of the expected value of the cycle time requires estimation of the expected value of the service cycle and of the expected value of the duty cycle. The expected value of the service cycle, SERV(j), is rather difficult to compute in the general case since, as we know from the study of the collision resolution algorithms, the distribution of the length of a CRI is not available.

We can express:

$$E(L_j) = \sum_{n_j=0}^{N+M-1} E(L_j \mid N_j = n_j) \times Pr[N_j = n_j] \tag{5}$$

An upper bound for $E(L_j)$ can be obtained assuming that the total number of servers and users, $N+M$ is large and then we can use the approximation presented earlier for the conditional length of a CRI (conditioned by the number of stations with ready packets at the beginning of the CRI). Then since at most $(N+M-1)$ users may have service requests for any server when he gets the CAP we can write:

$$L_{max} = 2.9 \times (N+M-1) \tag{6}$$

Let us denote by Q_{ji} the queue length of type j (containing service requests for the j-th server), at user i, $i \in S^*$, $i \neq j$. This queue can be modelled as an $M \mid G \mid 1$ queue with service vacation. We define "a modified service time" for the queue as the time elapsed between two consecutive instances at which this queue has access to the communication channel. Under the assumptions made earlier this modified service time is equal to the CAP cycle-time, C. The probability that Q_{ji} is not empty can then be expressed as:

$$p_{ji} = \lambda_{ji} E(C) \tag{7}$$

Clearly, in the general case, the length of Q_{ji} $j \in S$, $i \in S^*$, $i \neq j$ are not independent random variables, they depend upon each other through C, the CAP cycle-time. But if we focus our

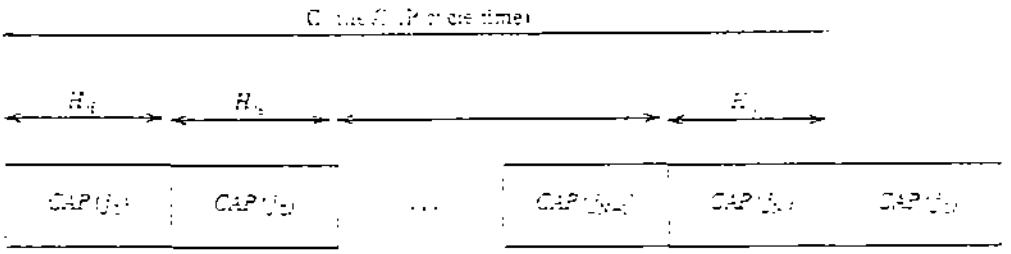

Figure 5: The CAP cycle-time.

attention only on the (N + M - 1) queues, one at each user, containing service requests for server j, there interdependency is weeker than in the case of a pure collision resolution algorithm since the expected value of C depends upon (N + N)(M + N - 1) queues.

So, a sensible approximation which has to be confirmed by simulation experiments placed for the future, is to assume that for a given j the N + M - 1 the queues Q_{ji}, $i \in S^*$ $i \neq j$ are independent. Then we can approximate $Pr(N_j = n)$ as:

$$p_0^{(j)} = Pr[N_j = 0] = \prod_{\substack{i=1 \\ i \neq j}}^{N+M} (1-\lambda_{ji} E(C)$$

$$p_1^{(j)} = Pr[N_j = 1] = \sum_{\substack{k=1 \\ k \neq j}}^{N+M} \lambda_{jk} \cdot E(C) \prod_{\substack{i=1 \\ i \neq j,k}}^{N+M} (1-\lambda_{ji} E(C))$$

.....

$$p_{N+M-2}^{(j)} = Pr[N_j = N+M-2] = \sum_{\substack{k=1 \\ k \neq j}}^{N+M} (1-\lambda_{jk} E(C)) \prod_{\substack{i=1 \\ i \neq j,k}}^{N+M} \lambda_{jk} E(C))$$

$$p_{N+M-1}^{(j)} = Pr[N_j = N+M-1] = \prod_{\substack{i=1 \\ i \neq j}}^{N+M} \lambda_{ji} E(C)$$

(8)

In general for a given server, say j, only a subset of size n_j of λ_{ji}, $i \in S^*$, $i \neq j$ will be nonzero. It follows that $p_{n_j(j)}$ will be a function of $(E(C))^{n_j}$. In particular when only $\lambda_{jk} \neq 0$ we obtain:

$$\begin{cases} p_o^{(j)} = 1-\lambda_{jk}E(C) \\ p_1^{(j)} = \lambda_{jk}E(C) \end{cases} \tag{9}$$

Then:

$$E(L_j) = \lambda_{jk}E(C) \tag{9'}$$

Let us now examine what happens at server j. Here we can recognize a [illegible] two queues, the input queue Q_j^{In} and an output queue Q_j^{Out}. The arrival process at the input queue is characterized by the density function, $A_j(t)$ of the interarrival times $T_{a,j}$, defined by:

$$A_j(t) = Pr[T_{a,j} \leq t] \tag{10}$$

The arrival pattern is the following: during a relatively short period of time of length L_j, there are n_j arrivals of requests for service, then a period with no arrivals followed by the next group (bulk) of requests which arrive after a cycle time.

So, the input system at server j can be modelled as an $GI^{[X]}|M|1$ system. The average number of requests arriving in one bulk is:

$$\overline{n}_j = \sum_{n_j=0}^{N+M-1} n_j \cdot p_{n_j}^{(j)} \tag{11}$$

The system can reach equilibrium iff:

$$\mu_j \overline{n}_j < E(C) \tag{12}$$

with μ_j the average service time at service j.

If the system reaches equilibrium we can approximate the length of the duty cycle as:

$$E(D_j) = \alpha_j \overline{n}_j \tag{13}$$

where α_j is dependent upon the type of service provided by server j. For example if the duty cycle is used by server j only to send the results of processed requests, then α_j is related to the average output size produced as a result.

Using the equations (4), (5), (8), (11) and (13) we are able to estimate the mean cycle time E(C).

Let us now examine briefly the problem of estimating the average delay of a request for service from server j, issued by user i. As shown in Figure 6 this delay has three components, the queuing delay at user i, the delay in the input queue of server j and the queuing delay in the output queue of server j. The first and the third components of the delay occur due to the sharing of the communication channel.

$$W_{ji} = W_{ji}^{Out} + W_j^{In} + W_j^{Out} \tag{14}$$

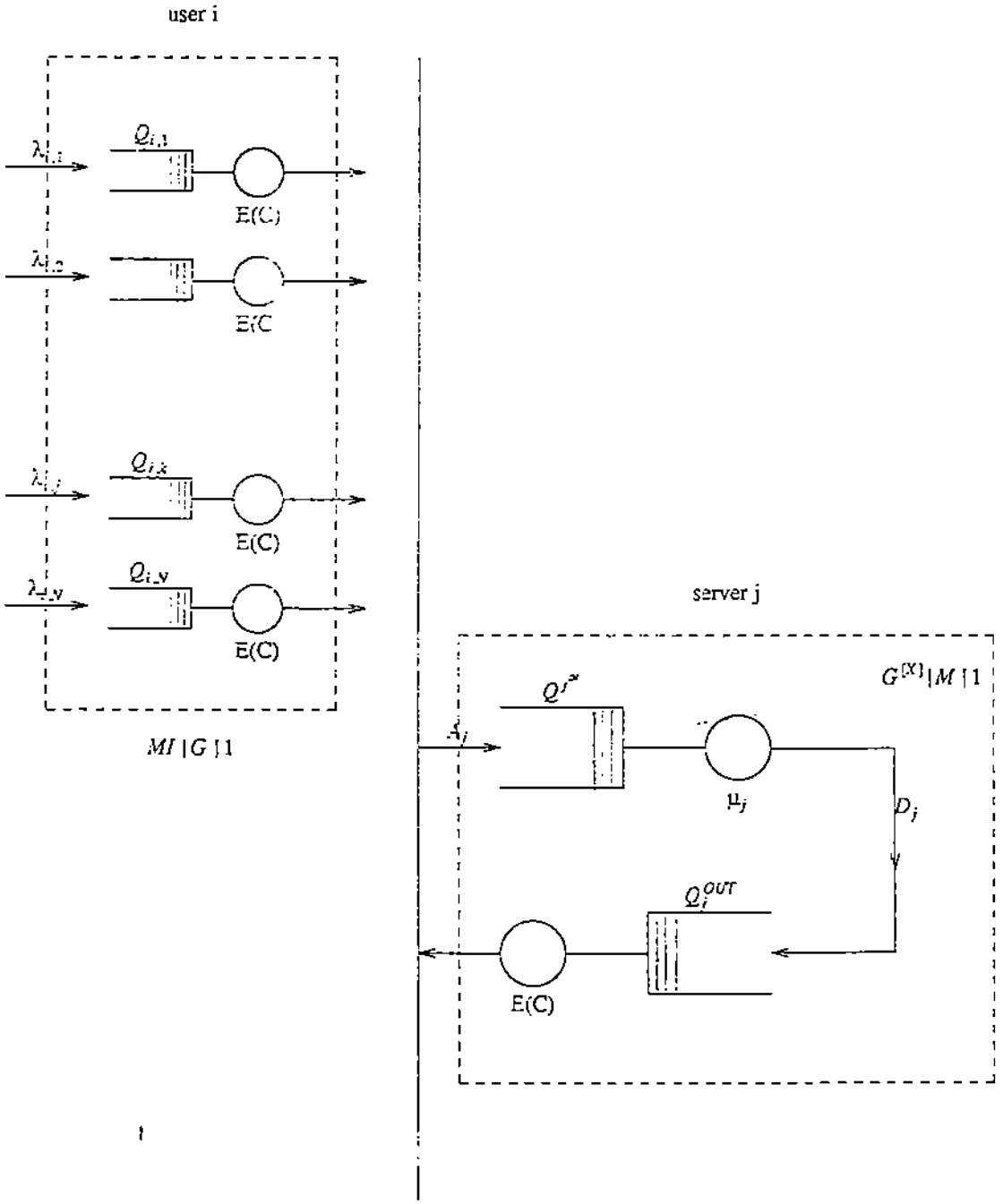

Figure 6: The networks of queues related to a Type 2 scheduling protocol.

Since Q_{ji} can be modelled as an M|G|1 system with the modified average service time E(C), we can compute the waiting time of a request for server j, queued at user i, using Pollaczek-Khintchine's formula as:

$$W_{ji}^{Out} = \frac{1}{\lambda_{ji}}(\rho_{ji} + \frac{\rho_{ji}^2 + \lambda_{ji}^2\sigma_c^2}{2\times(1-\rho_{ji})}) \tag{15}$$

with: σ_c = the variance of the cycle time, $C_c = \frac{E(C)}{\sigma_c}$, the coefficient of variance of th

cycle time and $\rho_{ji} = \frac{\cdot_{ji}}{E(C)}$.

Now let us examine the second queue. As mentioned earlier, the input queue at server j can be modelled as a $GI^{[X]}|M|1$ system. Using Kuehn's approximation, [16], the mean waiting time in Q_j^{In} is

$$W_j^{In} = \frac{1}{\mu_j}\frac{\rho_j}{2(1-\rho_j)}(C_{A_j}{}^2 + 1) \tag{16}$$

with

$$\rho_j = \frac{\lambda_j}{\mu_j} \tag{17}$$

and C_{Aj} is the coefficient of variation of the arrival process. We have estimated the mean arrival rate at server j as

$$\lambda_j = \frac{\overline{n}_j}{E(C)} + \lambda_{jj} \tag{18}$$

with λjj denoting the arrival rate of requests for service j generated by itself.

The mean waiting time in the queue Q_j^{Out} can be estimated using again the Pollaczek-Khintchine's formula since this queue can be modelled an an M|G|1 system with the modified average service time equal to E(C).

The estimation of the variance of the cycle time involves approximations of the conditional length of a CRI. The computations are too intricate and will not be presented here.

7. Conclusions

Three classes of scheduling protocols based upon collision resolution algorithms have been introduced in this paper. The qualitative analysis of their properties indicate that they will lead to a local area network connecting a large number of workstations or personal computers to a much smaller number of servers, with interesting properties. The resulting system is simple, exhibits a dynamical load balancing property, allows graceful performance degradation in case of server's failure, is fair and robust. Quantitative performance analysis of ADMA systems must be carried out in order to be able to compare them with other architectures.

ACKNOWLEDGMENTS

The author acknowledges the contributions of Wojciech Szpankowski and Vernon Rego in long discussions concerning ADMA. Andrew Rojappa has carried out the measurements discussed in this paper.

References

[1] McDonald John Alan and Pedersen Jan, "Computing Environments for Data Analysis, Part I and II", SIAM J. Sci. Stat. Comput., vol 6, No. 4, October 1985, pp 1004-1021.

[2] Becker R. A. and Chambers J. M., "Design of the S System for Data Analysis", Comm. ACM,27, pp 486-495, 1984.

[3] 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, September 1985, Tokyo

[4] Meister Bernd W., Janson Phlippe A., and Svobodova Liba, "Connection-Oriented Versus Connectionless Protocols: A Performance Study", IEEE Transactions on Computers, vol C-34, No. 12, December 1985, pp 1164-1173.

[5] Marinescu Dan C., Rego Vernon and Szpankowski Wojciech, "An Architecture for a High Speed Local Network Providing Multiple Supercomputer Access" Technical Report CS-TR-555 Computer Science Department, Purdue University, November 1985.

[6] Kleinrock Leonard, "On Queuing Problems in Random-Access Communication", IEEE Transactions on Information Theory, vol IT-31, No. 2, March 1985, pp. 166-175.

[7] Gallager Robert G., "A perspective on Multiaccess Channels", IEEE Transactions on Information Theory, vol IT-31, No. 2, March 1985, pp. 124-142.

[8] Abramson N., "The ALOHA system - Another alternative for computer communications", in Proc. 1970 Fall Joint Comput. Conf. AFIPS Conf., vol 37. Montvale NJ, pp. 281- 285.

[9] Capetanakis John I., "Tree Algorithms for Packet Broadcasting Channels", IEEE Transactions on Information Theory, vol IT-25, No. 5, September 1979, pp. 505-515.

[10] Hayes J. F., "An Adaptive Technique for Local Distributions", IEEE Transactions on Communication, vol. COM-26, August 1978, pp. 1178-1186.

[11] Tsybakov B. S., Mikhailov V. A., "Free Synchronous Packet Access in a Broadcast Channel With Feedback", Problemy Peredachi Informatsii, vol 14, no 4, October-December 1978, pp 32-59.

[12] Massey James L., "Collision Resolution Algorithms and Random-Access Communications", University of California, Los Angeles, Rep. UCLA-ENG 8016, April 1980.

[13] Szpankowski Wojciech, Marinescu Dan C., Rego Vernon, "Stability Problems in Local Area Networks: A Qualitative Approach", Purdue University West Lafayette, In., Computer Science Department, TR 558, November 1985

[14] Takagi H., Kleinrock Leonard, "Analysis of Polling Systems", JSI Research Report, TR 87, August 1985.

[15] Tobagi F. A., Nassehi M.M., Marhic M.E., "Fiber Optic Configurations for Local Area Networks", Proceedings of the International Seminar on Computer Networking and Performance Evaluation, Tokyo, September 1985.

[16] Kuehn P.J., "Approximate Analysis of General Queuing Networks by Decompositions", IEEE Transactions on Communications, vol COM-27, No 1, pp 113-126, January 1979

[17] Kuehn P.J., "Multiqueue Systems With Nonexhaustive Cyclic Service", B.S.T.J., vol 58, 3, pp 671-698, 1979.

[18] Spector A.Z., "Performing Remote Operations Efficiently on a Local Computer Network", CACM, 25(4), pp 246-259, 1982