1986

An Architecture for a Local Network Providing Multiple Supercomputer Access

Dan Cristian Marinescu

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

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

Report Number:
86-573

AN ARCHITECTURE FOR A LOCAL NETWORK PROVIDING MULTIPLE SUPERCOMPUTER ACCESS

Dan Cristian Marinescu
Vernon Rego
Wojciech Szpankowski

CSD-TR-573
January 1986

AN ARCHITECTURE FOR A LOCAL NETWORK PROVIDING MULTIPLE SUPERCOMPUTER ACCESS

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 (some of them supercomputers) is described in this paper. A connectionless communication model is assumed and a high speed broadcast channel is used as the interconnection network among the servers and the users of the network. 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. Then, a multi-access algorithm is 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. An analytic model of a simple scheduling protocol for a token passing bus is derived. An analysis of this model is done in order to determine the exact expected values of certain critical cycle-times. The stability analysis of this model concludes our paper.

1. OVERVIEW

The investigation of supercomputer access via local networks reveals that in a supercomputer network there is a need for a higher than normal degree of cohesiveness, transparency and integration of the communication system and computing systems. A higher degree of cohesiveness is needed because the network consists of expensive resources, fast communication channels plus specialized and very fast computing engines. A higher degree of transparency is needed due to the sophistication of the equipment connected, which makes access more difficult. Therefore it is expected that new communication protocols and different architectures will emerge.

The user of the system is most often concerned with knowledge and information processing rather than data processing. Consequently he needs an environment which provides interactive access to processors with different architectures, fast response times, parallel execution of some tasks, and, in addition, convenience, fair access and simplicity. A typical example is the use of an expert system which from a high level description of a problem, generates a large number of requests addressed to different processors, then analyzes the results in order to provide an answer to the problem.

The system considered in this paper consists of a number of user nodes, and a set of server nodes. The host of a user node is generally a workstation dedicated to a user. The computations performed can be decomposed into a set of computational tasks performed locally and a set of requests for service from remote servers. The host of a server node is a processor, for example a data base machine, a LISP machine, a scalar processor, a multiprocessor system, a vector processor, a special processor (e.g. a processor specialized in image processing, or in FFT), etc. Functionally, a server node may provide computationally intensive services of one type or another, data storage and retrieval services, communication services (may be a gateway to another network or may serve as a relay station) or other generic types of services.

The network described so far is non-homogeneous in that it has server and consumer nodes, and in addition may be highly asymmetrical since both the server nodes and the user nodes may differ drastically in the communication requirements imposed. Another aspect of this asymmetry is that in a server node the outbound traffic may be disproportionately large in comparison to the inbound one. This is expected since most numerical problems generate a vast amount of output data. This imbalance is likely to be moderate for those user nodes which need to examine only windows of the output data.

In this paper we introduce a new concept of availability driven request processing. The Availability Driven Multiple Access network architecture (ADMA) is based upon the correlation of the channel allocation strategy with server availability, by means of scheduling protocols. This architecture is not restricted to a local network providing multiple supercomputer access but to any type of non-homogeneous local network in which there are many servers. It is especially useful when the granularity of user services is relatively low and the amount of traffic between users and the servers is high. ADMA provides a dynamic load balanced scheme with a minimum of control and status information traffic.

2. AVAILABILITY DRIVEN MULTIPLE ACCESS NETWORK ARCHITECTURE

The architecture described here is similar to existing distributed systems built around a broadcast network, but carries the integration of the communication and processing systems one step further by introducing the concept of scheduling protocols in order to correlate the sharing of the communication channel and the other common resources. The objectives of this architecture are to lead to a system with the following characteristics:

- Incremental growth and graceful performance degradation,
- Distributed control,

- Transparency of system configuration and ease of use from the user's point of view,
- Adaptive load balancing,
- Minimization of control information flow through the system,
- Stability and fairness

Traditionally, systems are demand driven in the sense that a user is allowed to initiate the sequence of activities related to the use of a server. In case of a remote server, this sequence starts and ends with the acquisition of the communication channel in order to send a request and to receive the results. Since a user's request can be satisfied only by a specific server or subset of servers, control and status information must flow through the system so that a user can direct his requests for service only to those servers that are both currently connected to the system and able to carry out the necessary tasks. In order to have a load balanced system, additional status information must be available at the user's site. But increasing the amount of status information traffic has a significant impact on the performance of the system and its characteristics. The network will be forced to carry a lower level of user originating traffic and at the same time will require special procedures designed to ensure that all users have consistent status information. Moreover, since the control and status information must be packaged differently, more reliable and hence less efficient protocols must be used to transmit this type of data.

For all systems in which a user needs a remote server we recognize the following pattern: the service request will first queue at the user's site, waiting to be sent to the selected server. When the communication discipline allows for its transmission, it will queue again at the remote server's site, waiting to be processed. After being processed, it will queue at the remote server's site waiting until the communication channel becomes available, and finally, the results of individual request processing by different remote servers will reach the user and will be processed locally. The time between the generation of a request and the start of its processing at a remote server consists of a sum of waiting times in a local queue and in a remote queue. Attempting to

send the request as early as possible does not guarantee a short response time since the request will still have to wait in the remote server queue until the remote server is capable of processing it. An alternative solution is to *try to send the request as late as possible, namely, when the server capable of performing the task informs all users that it is available to perform services.* This is the basic idea behind the Availability Driven Multiple Access Architecture (ADMA). An intuitive analysis shows that this strategy will not increase the response time from the user's perspective but will have a positive impact upon system robustness since an unavailable server will not offer its services, while at the same time all the other objectives listed above, i.e., adaptive load balancing, stability, fairness, transparency etc., will be easier to achieve.

The strategy we discuss will have a significant impact upon the communication discipline and the performance of the system. In most existing local area networks the communication channel is rarely saturated and does not behave as a system bottleneck for the following reasons:

- the number of participants in the traffic is low and consists mainly of processors,
- the speed of the processors is usually not very high, and even more important, the communication protocols introduce considerable overhead and limit the actual transmission rates.

The system that we consider violates in these constraints in every respect. The number of participants in the traffic is large, since in addition to a moderate number of processors (servers), a large number of workstations (users) is connected to the network, thus generating very bursty and non-homogeneous traffic. High speed processors, efficient connectionless protocols, and applications relying upon frequent invocations of remote procedures are factors which will demand a high level of traffic through the network. It then becomes questionable whether existing techniques for sharing a communication channel are satisfactory. ADMA provides an answer to this problem by introducing the concept of scheduling protocols.

The basic concept of an Availability Driven Multiple Access Network Architecture is to relate the allocation of a common communication channel to the *scheduling mechanisms* which control access to the computing resources in a server - user community. Each server in the system has a chance to master the communication channel for a certain number of slots and during this period of time to determine whether anyone in the system needs the type of service it can provide. As opposed to a demand driven architecture, ADMA is *based upon an availability driven concept in which each server capable of providing service of a certain type examines the queue of requests at each user site*. This approach simplifies the flow of status and control information throughout the system.

In this system there is no need for centralized control, and new servers may be added to the system with a resulting improvement of performance. Similarly, servers may go down without the need to inform users, in which case users will simply see that a certain type of service is no longer provided, and in general that the system performance may degrade. The users can dynamically adjust to the new situation by reformulating their requests, whenever this is possible, so that they can use other servers.

The system enjoys an adaptive load balancing property since the heavily loaded servers will not use their available slots to gather more workload. Also the less loaded servers can adjust to this situation by offering to perform a wider class of services. For example, a SIMD machine may offer to perform scalar oriented computations or a scalar processor may end up performing FFT. The imbalance between input and output traffic in a server node is also manageable as the detailed system description will show.

The users need not receive special data packets informing them about the status of the servers. They can reformulate their needs in terms of the offerings of the exiting market so that the control information flow is minimal. The architecture has the potential of an increased system robustness due to the decrease of control and status information traffic. The actual robustness of a

scheduling protocol will depend upon the specific multi-access method used. As a side effect the system can be very fair when desired or can flexibly accommodate unfair schemes.

As pointed out earlier a typical user node consists of a workstation running an expert system. Higher level application software performs a decomposition of the computation into a set of tasks which can be performed separately by some processor in the network. This decomposition stamps on each task a certain type, selected from a known set of available types. The process of defining the set of available types and the algorithm used to decide the type to be associated with a given task are non-trivial issues that will not be discussed in this paper. Each server capable of providing a set of services sends around the network, whenever its turn to control the communication channel comes, the information that it is willing to provide service. In addition to specific services, say FFT computation, a server will provide generic types of services allowing a user to start its own application on that processor.

Throughout this paper it is assumed that the network connects a large number of such user nodes which make frequent requests for service to a much smaller number of servers via a broadcast network. The following structure of the communication can be recognized: most of the traffic occurs between the users and the servers; each server may request services from another server and these requests generate a fair amount of traffic. The communication among users themselves produces a small contribution to the overall traffic and can be taken care of by including in the network servers which relay messages from one user to another.

The ADMA architecture described in this paper can be conceived in connection with a ring or a bus network topology. Different multiple access schemes may be considered in order to share the communication channel, ranging from collision-free multiple access to limited contention, stack-type algorithms, etc. Since this is a critical design issue, we expect these schemes to give rise to different performance characteristics, and an analysis of each case must be undertaken separately in order to determine its relative merits and applicability.

3. A CONNECTIONLESS COMMUNICATION MODEL

The communication model considered here is a connectionless communication in which data are transferred from one entity to another without the prior mutual construction of a connection. In this case, fully addressed, self contained data units (datagrams) are transmitted using a transport layer which has no responsibility for error recovery, sequencing, delay or loss of data packets [1]. A connectionless data transmission is useful in case of local area networks where the error rate is low and communication efficiency is important. The basic mechanisms of the lower layers of the OSI model such as data flow control, error recovery or sequence control could greatly reduce the efficiency of communication [2], since they are repeatedly applied at different layers. In addition, since traffic is often very bursty, the overhead in establishing a connection and terminating it could be considerably larger than the data transmission time.

Typically, such a connectionless communication is initiated by a network user in order to gain access to different services offered by the network, for example, to gain access to a name server, file server and so on. In our discussion the connectionless communication is initiated by a server offering its services to the entire community. This approach is motivated by the following facts:

- Efficiency is extremely important. The computation of any user is decomposed into a series of atomic computational units which can be performed by the different available servers. A given user, say the user located in node i, issues a request vector r^i consisting of a series of individual requests $r_{i,1}$, $r_{i,2}$, ... r_{i,l_i} which can be addressed to any resource in the network. The size of the request vector is l_i and can be rather large, depending upon the granularity of decomposition of the user's computation. Consequently, the use of a name server which would try to determine the address and the path to the corresponding server for each individual request would introduce additional overhead in the communication channel. In the case of a remote name server, this would perhaps result in a bottleneck at that name

server.

- An adaptive distributed load balancing scheme is desired.
- User convenience is a high priority objective. The user should have the illusion of a pool of resources without the need to know either the names, the addresses, or access paths to these resources.

4. SCHEDULING PROTOCOLS

Consider a high speed local network connecting N servers and M strict users. The network has N+M nodes. A server node can provide services of a certain type (types) while a user node can only request services. A server node can also request services from other server nodes. In a wider sense we consider all (N+M) nodes in the network to be user nodes.

Communication between nodes occurs only in connection with a request for service. Consequently, in order to have full connectivity the set of servers will include one or more servers specialized in relaying data packets from one user node to another.

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 [3-4] 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 (see Figure 1) 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. An important objective of ADMA is to provide an adaptive load balanced scheme and in the same time to reduce the traffic due to control and status information flow through the system. As pointed out earlier, ADMA achieves these objectives by sending out a request to be processed by a remote server as late as possible, namely, whenever a remote server is available. While in a demand driven architecture the sender must rely on knowledge of an overall system status in order to determine the target of its service request, in ADMA the request is sent only when a server is available, thus minimizing the queuing delay at the remote site, the amount of status information needed by the users, and consequently the amount of control and status information traffic.

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.* When a server obtains control of the communication channel it has a

number of options:

- It can send processed requests from its output queue to the users that generated the requests and then relinquish control of the communication channel.
- It can send (broadcast) a control packet, informing all users that it is willing to provide a certain class of services and then either accept only one request or accept requests from all users which need that class of service, let these requests join its input queue and then relinquish control of the channel.
- It can perform both actions in a certain order.

Figure 2 illustrates the four periods related to the control of a communication channel by a given server: a channel acquisition period in which the server executes an algorithm to acquire the control of the communication channel, an output period in which the server sends out the processed requests, an input period when it gathers more requests for service and a period in which the server relinquishes the control of the channel.

Some of these periods may not exist depending upon the multi-access method used and upon the server's current status. For example, when all servers obtain control of the communication channel in a cyclic order, as in the case of a token passing ring, there is no time spent in acquiring the channel (except for a token-passing delay), and a server may simply pass control to the next server (its successor) without executing any of the actions listed above.

To allow the servers to share the communication channel various strategies may be used, ranging from random [5], to limited contention [6-8] and to contention-free server multi-access. Random multi-access schemes, though easier to implement, will probably lead to a longer channel acquisition period 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 the acquisition period will be small, consisting of the time it takes the token to travel from one server to its successor in the ring.

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 which may need the service provided by a certain server.* Referring to Figure 2, the input period P3 will be analyzed. 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.

5. SCHEDULING PROTOCOLS FOR A CONFLICT FREE BUS

The architecture described in this paper can be implemented for different topologies of the interconnection network and for different multi-access algorithms. We will first investigate the case of a bus network topology and contention-free multiple access based on token-passing. Our choice is motivated by the fact that this environment is probably easier to understand and to analyze than other pairs <topology, multi-access algorithm>.

In a token passing bus, whenever a station receives the token it is granted control of the communication channel for a specified time. The significant difference between a token passing bus and a token passing ring is that on a token passing bus non token using stations are allowed on the bus, and the token passing sequence follows a logical rather than a physical ring. A station in control of the bus may poll other stations or may transmit and receive data.

As shown in Figure 4, we view the system as consisting of two concentric logical rings. The internal logical ring consisting of the M servers is referred to as *the server's ring* and the token circulating through it is called a *CAP, Channel Allocation Packet.* The external logical ring consists of all N+M users of the network and is called *the user's ring*. At any given moment of time it may contain one of M possible token types, where the type of the token is determined by the server currently in control of the communication channel. If this server's ID is k, the token circulating through the user's ring is called SAP_k, *Service Availability Packet* of server k.

Since in a real case both virtual rings are collapsed into a single physical bus, we need to introduce the concept of a control packet. A token is embedded into a control packet which may be acquired only by the successor of the station transmitting the packet. Since we have two types of tokens the control packet must contain a field describing its type and it may contain additional information. For example, in order to ensure fairness, when a server sends out an SAP it also indicates the first user from where the circulation of the SAP should start; to be fair, this user should be the successor of the user who qualified as first during the last cycle of the SAP from the same server. In addition, an SAP should contain a field describing the type of service its owner is capable of and willing to provide. It is required that both the set of servers and the set of users are totally ordered sets and each station knows its predecessor and successor in the ring structure it belongs to.

Figure 4 illustrates the sequence of events happening in a contention-free scheduling protocol in a token passing bus. Server k receives the CAP from server k-1 at time t_1 and it will control the bus until t_2. During this time it sends results of previously processed requests from its output queue and then broadcasts its SAP. If user i is the successor of the last user which has sent service requests during the previous period corresponding to this server, and if user i has a request for service, stamped with a type matching the service type in the SAP, then it will send its request as shown in our diagram. To simplify matters we assume that a server gathers only one request for service each time it has control of the communication channel. In this case, after receiving one service request from user i, server k will transfer the CAP (and hence control of the channel) to server k+1. Server k+1 will then perform a similar processing but when its SAP is sent to user i+1, since this user has no need for the type of service server k+1 is willing to provide, this user passes the corresponding SAP to its own successor. Eventually user j needs the type of being service offered and sends its request to server k+1.

We are talking about a family of protocols since variations of the basic algorithm described

above may be employed. For example, in the user's ring one may consider the use of a virtual token. In the previous example, user i+1 does not have a request for service from server k+1 and consequently passed the corresponding SAP to its successor. A scheme in which its successor will automatically obtain the right to transmit a request when user i+1 does not transmit can also be considered.

6. AN ANALYTIC MODEL

In this section is presented an analytic model that focuses on a specific protocol for the ADMA architecture. Since this research is undertaken as a preliminary study with the design's feasibility as a prime concern, the analysis addresses a very simple protocol based on token-passing systems. 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. In this sense, the system can be modelled as a single server queueing system provided each queue is looked at individually.

The outline of this section is as follows. In section 6.1 is introduced the conceptual queueing model for the ADMA architecture in terms of a simple token-passing protocol. In section 6.2 we obtain the mean values of certain random variables that are necessary in determining the stability boundary of the system operating under this protocol. In section 6.3 we demonstrate how the mean values can be used to present conditions under which this protocol operates in a *stable* fashion on the ADMA network.

6.1 MODEL DESCRIPTION

Consider the diagram in Fig. 5 showing an abstract view of a typical ADMA local network. The system is made up of two kinds of stations. The stations shown on the external ring are

strictly user stations, while those on the internal ring are server 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 $S^* = S \cup U$. Several assumptions (P1 through P7) that are necessary to describe the protocol in a precise manner are stated in the following discussion.

P1. Each 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.

In order to state further assumptions it is necessary to introduce some notation. Let CAP(j) denote the event that server station j, $j \in S$, is in possession of the unique channel allocation packet, or server station j has sent the CAP to another server who has not yet received it. Also, let SAP(j, i) denote the event that user station i, $i \in S^*$, $i \neq j$, is in possession of a service availability packet of type j, $j \in S$, or user i has sent this packet to another user who has not yet received it. Note that by assumption P3 we can exclude events of type SAP (j, j), $j \in S$, from consideration.

We assume that the subset of stations $S \subset S^*$ forms a *logical ring R_s of server stations* that

is different from the *logical ring R_u formed by all the user stations in* S^*. The two different logical rings are clearly indicated in Fig. 5. 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. Similarly, it is necessary for each user station in S^* to know its predecessor user station (i.e., the station it receives an SAP from) and its successor user station (i.e., the station it must send an SAP to) for the ring R_u to be defined.

In steady-state, control of the channel is passed between server stations and user stations with the aid of the CAP and the different types of SAPs. The CAP is passed from one server station to its successor in the logical ring 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. The event CAP(j) is initiated at the instant server station j receives the CAP from its predecessor in the ring R_s. The status of the communications channel during this event is pictorially described by Fig. 6a. The following paragraphs describe the events that take place within the event CAP(j).

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 .

The second event that occurs within the event CAP(j) is denoted as CYC(j) since it involves an SAP of type j that cycles through the network, visiting *all* stations (except server station j) exactly once. The SAP of type j is made to complete one cycle of the network in the following manner. After performing its duties, server station j broadcasts an SAP of type j containing a specific ID *nit*(j) to the network. This ID denotes the station that is *next-in-turn* to receive service from server station j. Let *pred*(i) and *succ*(i) denote the predecessor and succes-

sor user stations of station i, respectively, in the logical ring R_u, for each $i \in S^*$. In defining these two functions, we make the following asumption.

P4. For the duration of event CAP(j), $j \in S$, the logical ring R_u excludes server station j. In other words, the predecessor of server station j in the logical ring R_u must compute its successor to be the successor of server station j instead of station j.

When the network is powered up, server j, $j \in S$ computes the address $nit(j)$ arbitrarily, $j \neq nit(j)$, but each new occurrence of the event CAP(j) causes the station ID variable $nit(j)$ to be reset to $succ[nit(j)]$ in the ring R_u, in the interests of fairness. If $x = nit(j)$ and $y = pred[nit(j)]$, then the event CYC(j) is comprised of nonoverlapping events that occur in the sequence SAP(j, x), SAP($j, succ[x]$),....., SAP(j, y). The intention of sequencing these events in the protocol is to give each user station a chance to send a packet to server station j provided the user station has a packet of type j queued in its buffer. When user station $nit(j)$ receives the SAP of type j, it does the following. If it does have a packet of type j queued, it sends this packet request to server station j. Next, it sends the SAP of type j to its successor in the user ring R_u. If it does not have a packet of type j queued, it simply sends the SAP of type j to its successor in R_u. Since the network operates on a broadcast channel, each act of "sending" a packet requires a time for packet transmission and a time for signal propogation. Additionally, this generally depends upon the relative position in the network of the sending and receiving stations, and the length of the packet sent. Note that by assumption P4 the event SAP(j, j) cannot occur, so as to exclude the possibility of an SAP of type j being passed to server station j . This is also consistent with assumption P3. The event CYC(j) is terminated when the event SAP (j, y) terminates.

The final assumptions concern the manner in which the protocol is made to operate in steady-state. By making small changes to these assumptions, it is possible to introduce a class of protocols based on the token-passing approach.

P5. 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.

P6. For each server station j in S , the event CAP(j) is made up of disjoint events, occurring in the order DUTY(j) and CYC(j).

P7. For the duration of the event SAP(j, i), $j \in S$, $i \in S^*$, user station i may send *at most* one packet (request) of type j to server station j.

6.2 MEAN CYCLE-TIMES AND STABILITY

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 a CAP cycle-time is described pictorially in Fig. 6b in terms 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 (see for example [9]). Though this makes the problem of describing the distribution of C very difficult, the expected value of C may be computed in a simple

manner. Denote the random durations of the events DUTY(j) and CYC(j) as random variables O_j and c_j, respectively, for each $j \in S$. We can compute the expected duration of CYC(j), for $j \in S$, as

$$E(c_j) = E(Y) + \sum_{\substack{i \in S^* \\ i \neq j}} [\lambda_{ji} E(C)] E(X_{ji}) \quad (1)$$

where X_{ji} is the random time taken by station i to send a request packet of type j to server j, $i \in S^*$, $j \in S$, $i \neq j$. The random variable Y represents the total time required for passing the CAP from one server to another in the ring R_s. That is, Y is the sum of N random variables Y_j, $j \in S$, where Y_j represents the time taken for station j to send the CAP or an SAP to a successor station in the ring R_s or R_u, respectively, for $j \in S$. For analytic convenience, we assume that the $\{Y_j\}$ are identically distributed random variables, and $E(Y_j)$ is the maximum possible time taken by any station on the bus to send an SAP (or CAP) to its successor. Additionally, these random variables are also independent, though this does not affect the following discussion. The quantity $\lambda_{ji}EC$ appearing in (1) is the probability that buffer j of station i is nonempty.

Let H_j denote the expected duration of event CAP(j), $j \in S$. Then we can write

$$E(H_j) = E(Y) + E(O_j) + \sum_{\substack{i \in S^* \\ i \neq j}} [\lambda_{ji} E(C)] E(X_{ji}) \quad (2)$$

and since $E(C) = \sum_{j \in S} E(H_j)$, we obtain ,

$$E(C) = NE(Y) + \sum_{j \in S} E(O_j) + \sum_{j \in S} \sum_{\substack{i \in S^* \\ i \neq j}} [\lambda_{ji} E(C)] E(X_{ji})$$

or

$$E(C) = \frac{NE(Y) + \sum_{j \in S} E(O_j)}{1 - \sum_{j \in S} \sum_{\substack{i \in S^* \\ i \neq j}} [\lambda_{ji} E(X_{ji})]} \quad (3)$$

One of the most important issues in the design of a network is its stability (see for example [13]). There are many different definitions of stability. In this paper we assume that the system is *stable* if the average queue lengths at all users $i \in S^*$ are finite. If average queue length is finite only for some users, then it is said the system is *partially stable*. Let Q_{jk} denote the queue length of type j packets at user $k \in S^*, k \neq j$. This queue may be modelled as an M|G|1 queue with server vacation, and we define a *modified service time* for the queue as the elapsed time between two consecutive instants at which this queue has access to the channel. Under protocol assumptions P1-P7 the modified service time is equal to the CAP cycle-time C. Note, however, that CAP cycle-times are *not* independently distributed random variables. Nevertheless, stability conditions in this case are easily available by the Foster criterion [10], a result of Loynes [11] and the Tweedie criterion [12]. In particular, the average queue length EQ_{jk} at station k is finite if the arrival rate for packets of type j at station k is smaller than the modified service rate (reciprocal of modified service time) *under the condition that the queue length Q_{jk} is not zero* , (i.e., this queue is not empty), and the variance of the modified service time is finite. Therefore, from the stability point of view it is essential to compute the average CAP cycle-times under the assumption that queue Q_{jk} is not empty. But by protocol assumption P1-P7 such an average CAP cycle-time is the same for all $j \in S$. Hence, let C_k^* denote the random time between consecutive visits of the CAP at an arbitrary reference station $j, j \in S$, under the condition that station k in S^* transmits a request packet to at least one server in S. If $H_{jk}^*, j \neq k$, denotes the CAP cycle-time conditioned on the fact that station $k, k \in S^*$, transmits a request packet of type $j, j \in S$, during this cycle, then clearly

$$C_k^* = \sum_{j \in S} H_{jk}^* \tag{4}$$

The expected conditional holding time $E(H_{jk}^*)$ may be computed, for $j \neq k$, as

$$E(H_{jk}^{*}) = E(Y) + E(O_j) + \sum_{\substack{i \in S^{*} \\ i \neq j,k}} [\lambda_{ji} E(C_k^{*})] E(X_{ji}) + E(X_{jk}) \tag{5}$$

and the expected conditional cycle-time as

$$E(C_k^{*}) = \frac{NE(Y) + \sum_{\substack{j \in S \\ j \neq k}} [E(O_j) + E(X_{jk})]}{1 - \sum_{i \in S} \sum_{\substack{j \in S^{*} \\ i \neq j,k}} E(X_{ji})] } \tag{6}$$

Now by Loynes result [11] the following condition

$$\lambda_{jk} < \frac{1}{EC_k^{*}} \tag{7}$$

for any $j \in S$ implies that queue j at station k possesses a steady-state solution. By the Tweedie result [12] and the above assumptions, this means that the average queue length at this queue is finite if (7) holds. Finally, if (7) is satisfied for all $k \in S^{*}$, then the system is stable, that is, all average queue lengths are finite.

References

[1] Gihr O. and Kuehn P.J. "Comparation of Communication Services with Connection-Oriented and Connectionless Data Transmission", Proceedings of the International Seminar on Computer Networking and Performance Evaluation, September 1985, Tokyo

[2] Meister Bernd W., Janson Philippe 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.

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

166-175.

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

[5] 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.

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

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

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

[9] Kuehn P.J., "Multiqueue systems with nonexhaustive cyclic service", B.S.T.J. vol.58, 3, 1979, pp.671-698.

[10] Foster. F.G., "On the stochastic matrices associated with certain queueing processes", Ann. Math. Stat., 24, 1953, pp.355-360.

[11] Loynes R.M., "The stability of a queue with non-independent inter-arrival and service times", Camb. Philos., 58, 3, 1962, pp.497-520.

[12] Tweedie R.L., "The existence of moments for stationary Markov chains", J. Appl. Probab., 20, 1983, pp.191-196.

[13] Szpankowski, W., Marinescu D., Rego, V., "Stability problems in local area

networks: A qualitative approach", Purdue Technical Report CDS TR 558, 1985.

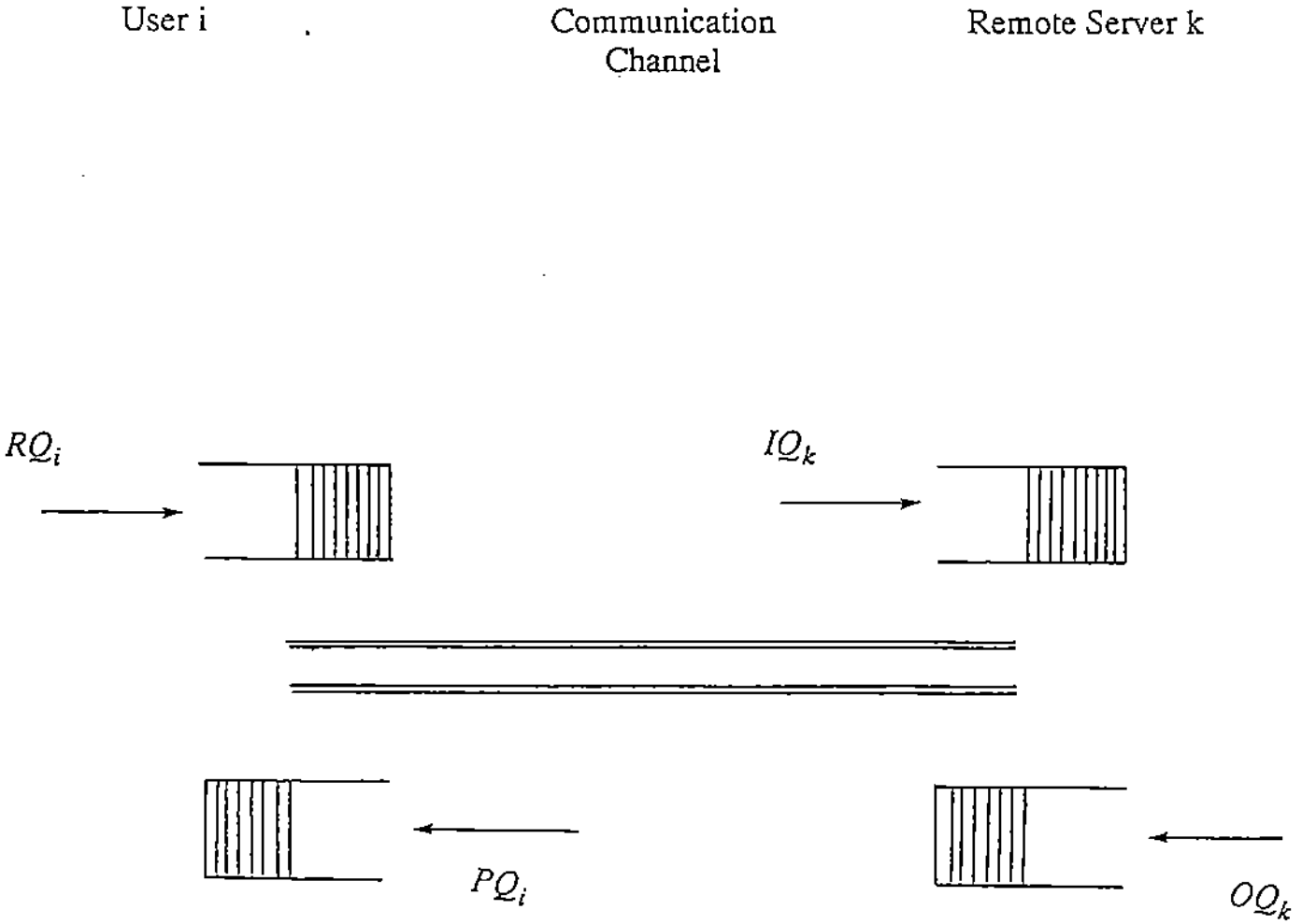
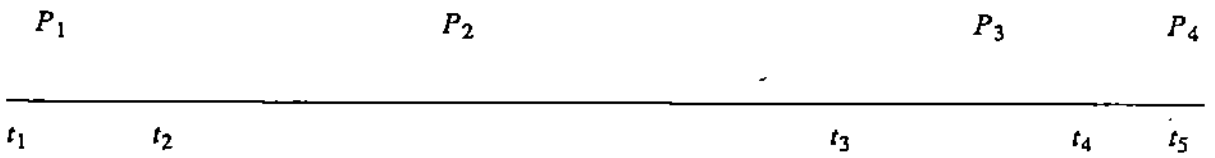


Figure 1. Queueing delays in case of a remote server access



P_1 - channel acquisition period

P_2 - output period

P_3 - input period

P_4 - channel relinquish period

Figure 2. Channel ownership periods for a given server

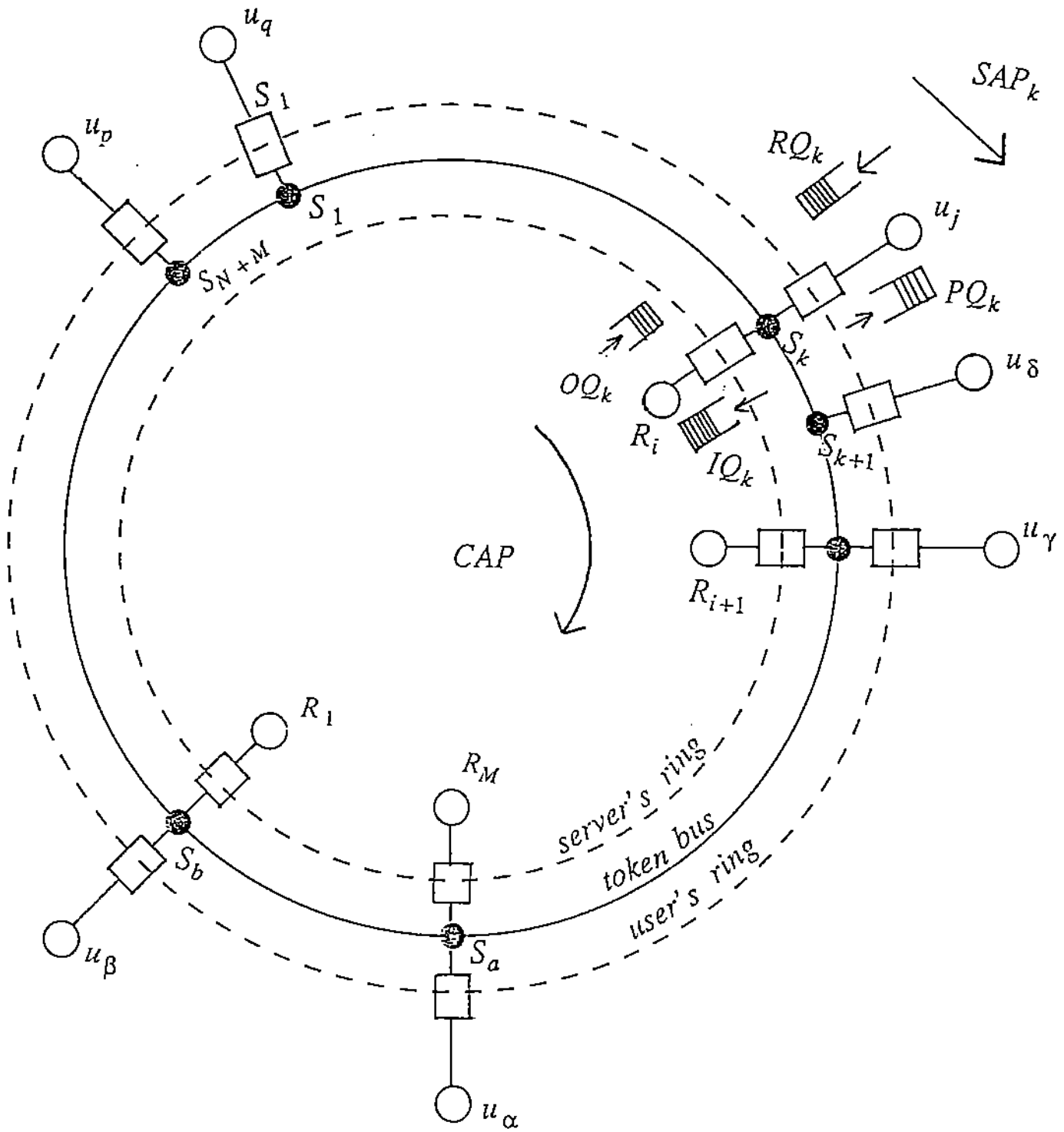


Figure 3. Contention free scheduling protocols

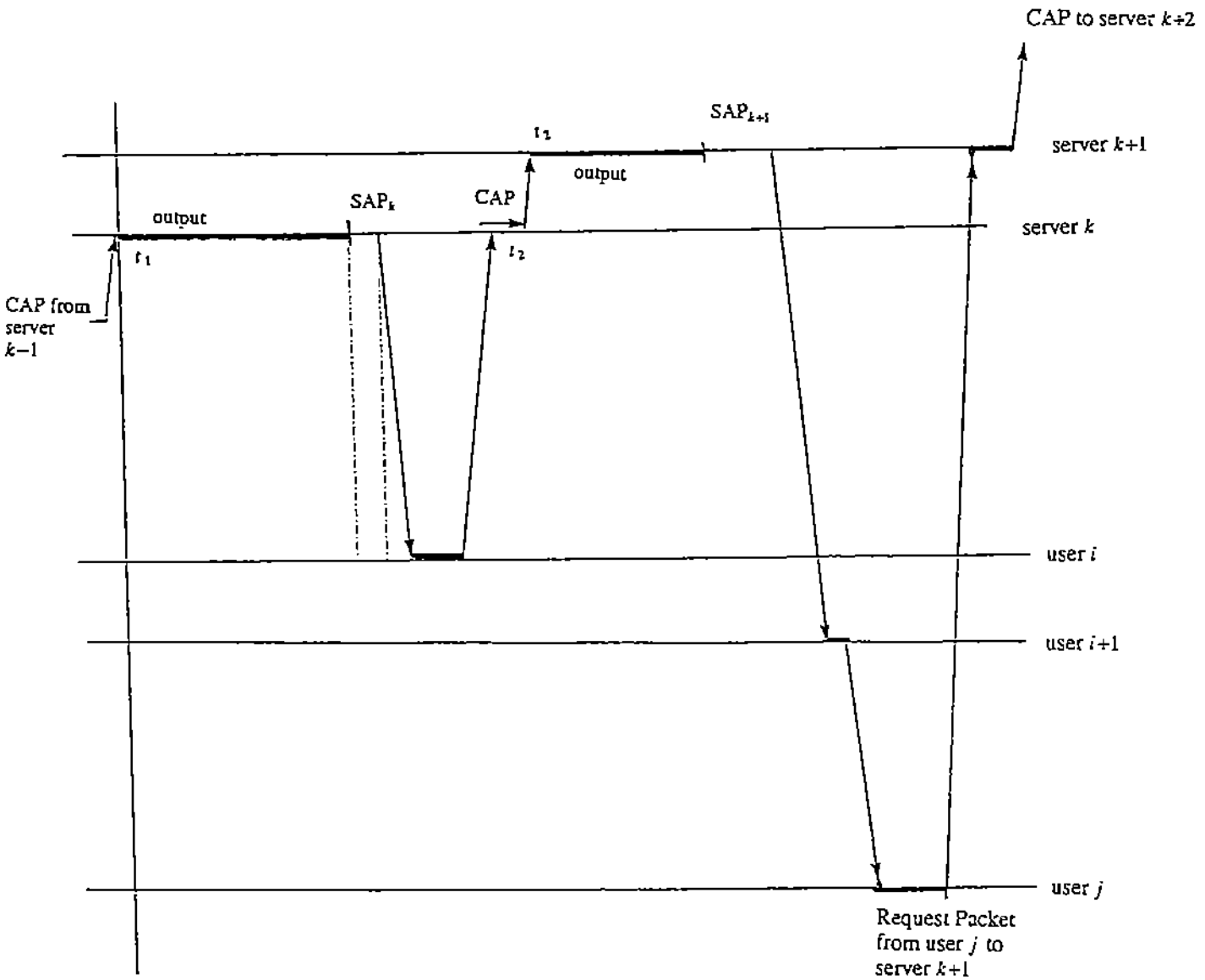


Figure 4. Timing diagram in a scheduling protocol for a token passing bus

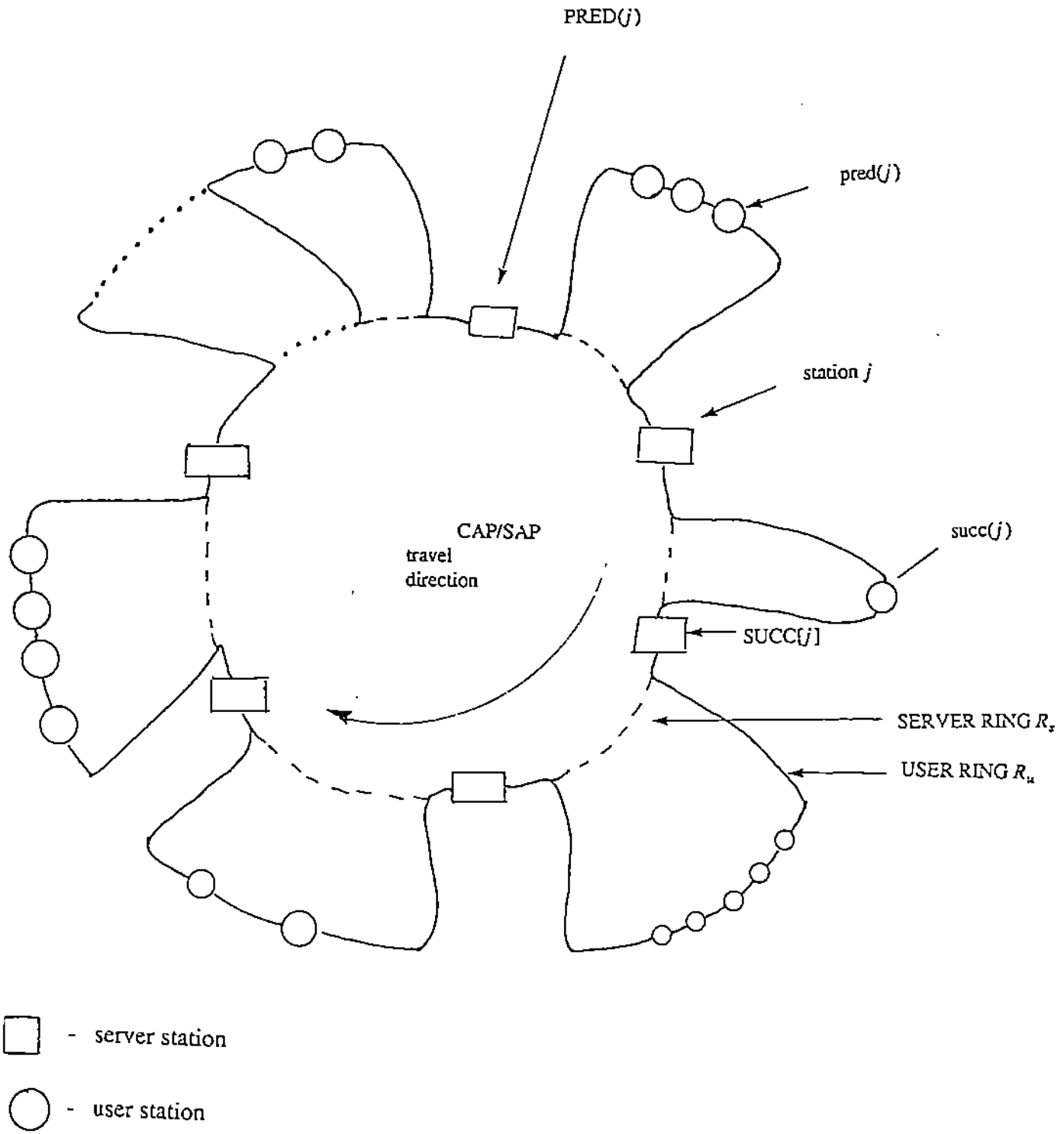
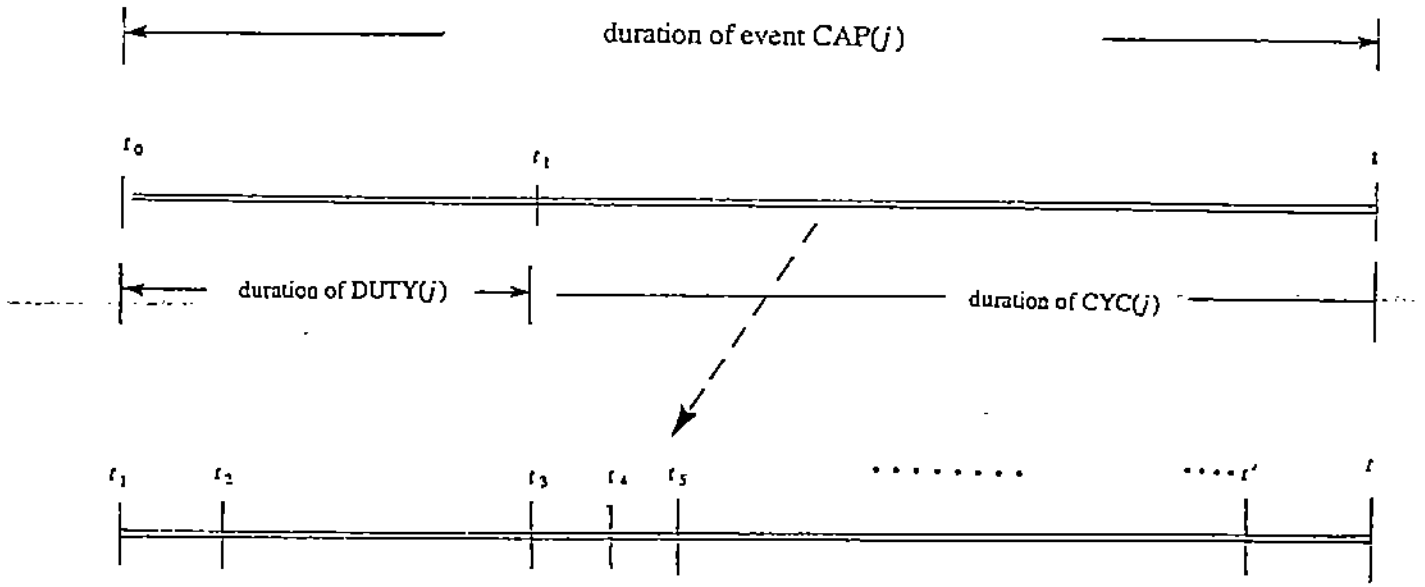


Fig. 5. Logical rings of server stations
and user stations



t_0 : Server j receives the CAP

t_1 : DUTY(j) termintes, server j transmits SAP

t_2 : succ[j] receives SAP and has a packet of type j. So transmission starts.

t_3 : Transmission ends, succ[j] sends SAP to succ[succ[j]] = k

t_4 : k receives SAP but has no packet of type j queued. So transmission of SAP to succ[k] starts.

t_5 : succ[k] receives SAP.....

.

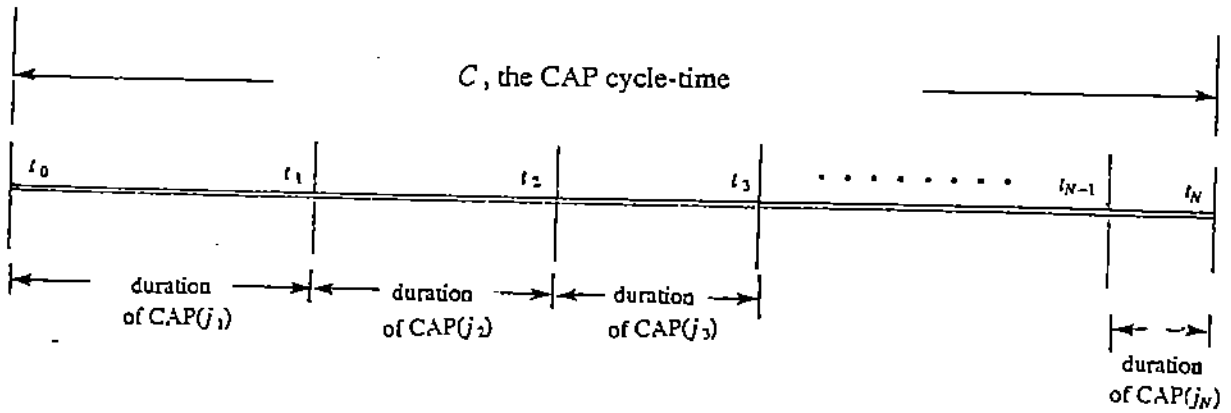
.

.

t_1 : Server j determines that SAP cycle is complete. Transmission of CAP to succ[j] begins.

t : succ[j] receives CAP, CAP[succ[j]] starts.

Figure 6a. Description of event CAP(j)



t_i : Server j_{i+1} receives CAP from PRED[j_{i+1}].

t_0 : CAP cycle begins, as seen from server j_1.

t_N : CAP cycle ends, as seen from server j_1.

Fig. 6b. CAP cycle-time