MODELS FOR ADDRESS DISTRIBUTION AND SESSION MANAGEMENT IN BROADCAST NETWORKS WITH DYNAMIC ADDRESSES

by

O. Sharon and A. Segall*

Technical Report #589
October 1989

* Work supported by the Technion V.P.R. Fund - Japan TS Research Fund
MODELS FOR ADDRESS DISTRIBUTION AND SESSION MANAGEMENT
IN BROADCAST NETWORKS WITH DYNAMIC ADDRESSES

Oran Sharon & Adrian Segall
Dept. of Computer Science
Technion, Israel Institute of Technology
Haifa, Israel 32000
segall@TECHSEL.BITNET, segall@sel.technion.ac.il
Tel. 972-4-294161

ABSTRACT

The method of dynamically assigned addresses is an alternative to the traditional method of
preassigned addresses. The dynamic approach requires protocols for assigning/unassigning
addresses, along with some kind of mechanism which distributes information about
assigned/unassigned addresses to network stations and manages sessions between stations. In
the present work we investigate the second issue in relation to broadcast networks, present
three possible models for address distribution, define the properties required for correct opera­
tion and prove that our proposed protocols posses these properties.

1. INTRODUCTION AND MOTIVATION

Broadcast networks are commonly used and are found in a large variety of communication net­
works, such as local area networks [SDRC], packed radio networks [A] and satellite networks [S]. Com­
munication between stations in such networks is performed by broadcast messages, and every station has
an address which is unique among all stations that are attached to the network at any given time. When a
station sends a message, it attaches to it the address of the destination station. Since the broadcast nature

*Work supported by the Technion V.P.R. Fund - Japan TS Research Fund
enables every station to hear every message, the destination station will get this message, recognize its own address and accept the message. Other stations will simply disregard the message.

The most commonly used method for obtaining addresses is the method of preassignment. The network owner or operator assigns an address to every station at the time it attaches to the network, making sure that this address is unique networkwise. In static networks, where attachments and detachments of stations are not common events, this method is very convenient. However, in dynamic networks, where stations join and leave the network frequently, the address assignment is more difficult to implement since it is sometimes impossible to know which and how many stations will eventually join the network. The preassignment can also result in long address fields, longer than necessary. This length is determined so as to ensure global uniqueness, and in the preassignment method all stations that may ever join the network must be considered. However, since stations join and leave the network, normally only a small number of stations is connected to the network at any given time, and uniqueness must be kept only among these stations. The length of the address field is important since every DATA message must carry the addresses of the source and destination stations.

In order to overcome these drawbacks, a different approach was suggested. This approach is to dynamically assign addresses ([R], [OS1], [SG], [JM], [GS2], [LKV]). In most of the proposed schemes, a station joining the network receives an address from a special station, which serves as an address manager or in short manager. When a station leaves the network, its address is released and the manager may reassign the address to another station. The dynamic method offers some important advantages. It removes the complexity of determining which stations may attach to the network and guarantees address uniqueness, while also ensuring that the size of the address field is kept small. In addition, the method allows the network owner or operator to impose his own addressing conventions, perhaps reflecting the current network structures or control hierarchy [JM], thereby possibly simplifying the tasks of network management and control. It also permits the assignment of two or more addresses per station, something that may be desirable when the addressed entities are logical entities such as application programs. Such programs may be dynamically created, and so in the preassignment method it is difficult to assign addresses to such programs.
The method also enables flexibility in the transition of stations among different networks. In the method of preassignment, there may be networks that use different types of addresses (hierarchy, flat, different lengths) which prevents from stations with fixed, preassigned addresses, to move from one network to another. On the other hand, in the dynamic method, a new station receives an address of the type that is used in the network it joins, and so stations can join every network with no limitations imposed by addresses.

With the dynamic method several special problems must be solved:

(A) How does the manager assign an address to a station that joins the network.

(B) How does the manager unassign the address when the station leaves the network.

(C) How to create the conditions that allow address assignment.

(D) How to inform stations about the addresses of other stations.

(E) Session maintenance in view of the dynamic assignment of addresses.

Problems (A), (B) and partially (C) were solved by the previously proposed protocols ([R], [GS1], [SG], [JM], [GS2], [LKV]). However, none of these protocols gives a comprehensive answer to problems (D) and (E) and this is the purpose of the present work.

To illustrate the importance of (D) and (E) above; consider the case where station $J$ exchanges messages with station $I$. Let $x$ be the address of $I$. If $I$ leaves the network while the session is still in progress and $x$ is assigned to a new station $K$, then three difficulties may arise:

(i) $J$ might talk with $K$, thinking it is still talking with $I$.

(ii) $K$ may accept messages from $J$ that were in fact intended for $I$.

(iii) If $J$ is allowed to store addresses in its address table, then consider the case where $J$ stores $I$ with assigned address $x$. If $J$ is not informed, before the reassignment of $x$ to $K$, that $x$ does not belong to $I$ anymore, then whenever $J$ will desire to initiate a session with $I$, it will send messages with destination address $x$. These messages will be accepted by $K$, while $J$ thinks it is starting a session with $I$. 

Notice that (i) applies to an existing session, while (iii) applies to sessions that may be initiated in the future.

The purpose of the present work is to design protocols that ensure session management while avoiding these problems. This task has a strong relation to the way addresses are distributed, and three models are considered. In the first model, stations do not maintain address tables. Thus, a station will always request from the manager the address of the other station before initiating a session with it. The motivation for this model is to save memory and processing at the stations. The task of managing address tables may require a considerable amount of processing when stations join and leave the network frequently. However, the model requires requests for the address before every session is started, even if the address does not change. This model is the simplest and requires minimal memory at the stations.

In the second model, a station $J$ requests the address of a station $I$ with which it wishes to communicate only before the first session and keeps addresses in address tables. The address of $I$ is kept until $J$ is informed by the manager that $I$ has left the network and has therefore released its address. If subsequently $J$ wants to talk with $I$ again, $J$ must ask again for the address of $I$. This must be done since $I$ may return to the network with a different address, and/or the previous address of $I$ may be redesignated to another station. In this model stations do not have to ask for the addresses of other stations every time when initiating a session. The use of the address tables also reduces network traffic and load on the manager.

In the third model, when stations join the network, they receive from the address manager a list of the addresses of all stations in the network. This list is kept in an address table and is continuously updated by the manager, without specific requests from the stations. The motivation for this model is to completely eliminate the need for requesting station addresses. However, it increases the load on the manager that must continuously update all stations.

The models we present and the properties we define subsequently for correct operation are general, and can be implemented in every network that maintains dynamic address assignment. Our work is an extension to the address assignment protocol presented in [GS2], so as to allow session management.
The paper is structured in the following manner. In section 2 we define the exact model of the communication network. Section 3 gives a short description of the address assignment protocol of [GS2] and presents its basic properties. In section 4 we define basic properties for correct session management and sections 5, 6 and 7 present the three models, propose a solution for each and give correctness proof for the proposed protocols.

2. THE MODEL

The model under consideration is the most general model for a broadcast network. Its structure is depicted in Figure 1.

![Figure 1 - The Model](image-url)
Communication between stations is performed by the transmitters and receivers at the stations and at the media. A station transmits a message by placing the message in its outgoing queue. The transmitter at the station transmits messages from the outgoing queue in FIFO order, and uses some access protocol to transfer the message to the media receiver via the outbound channel.

The message may reach the media receiver with errors, due to channel noise, collisions or any other reasons. If the message is received without errors, the media receiver places it in the media queue. The media transmitter takes messages from the media queue in FIFO order and transmits them, via the inbound channel, to the receivers of all stations attached to the network. The inbound channel may introduce errors which may be selective, causing the message to be received with errors by some station receivers and correctly by others. While we place no specific numerical bounds on the probability of noise or collision errors in the inbound or the outbound channel, we make the assumption that the probability of a message being correctly received at its destination is strictly greater than zero for any source and destination.

The destination field in a message may contain either some address or the universal address, denoted by *. The universal address is a special address known and recognized by all stations and it is used for control purposes, as shown presently. The receiver at a station places error-free messages whose address is either the station or the universal address in the incoming queue of the station. All other messages are discarded.

The delay experienced by a message in transit from one station to another consists of several queueing, processing and propagation delays. The queueing and processing delays are finite but unbounded. The propagation delays experienced in the outbound channel (from the station transmitter to the media receiver) and in the inbound channel (from the media transmitter to the station receiver) are assumed to be bounded by known upper bounds \( d_{out} \) and \( d_{in} \) respectively.

We next show that the above model corresponds to essentially all types of broadcast networks.

The first network we discuss is Ethernet. Ethernet is a local area network [SDRC], where stations are connected to a common communication channel, such as a radio channel, coaxial cable, fiber optic lines etc. The stations communicate using CSMA/CD, which is a contention based access protocol.
Attached to each station there is a transmitter used by the stations to transfer messages to the communication media, and a receiver where the stations receive messages from the media. The usage of common communication media enables every station to hear every message transmitted, and thereby enabling broadcasting. Since all stations use the same communication channel, and the access protocol is contention-based, messages may collide.

Every message is transmitted directly from the transmitter of the source station to the receiver of the destination station, without passing through any queue or transmitter-receiver in the media. Therefore, assuming that $d_{\text{out}}=0$ in the general model, that the outbound channels are collision free and the queue of the media is of size zero, we receive the Ethernet model. The communication channel in Ethernet stands for the media and the inbound channels of the general model, collisions in the communication channel stand for the collisions in the inbound channels and $d_{\text{in}}$ in Ethernet is the longest time period since the beginning of the transmission of a message until its receipt at the destination station. For this calculation the two most remote stations are considered.

Notice that in Ethernet the electrical signal carrying the messages propagates along the channel. Thus, if there are two messages in the channel at the same time, both transmitted to the same station, they will eventually collide and at least one message will not be received by the destination station correctly. Therefore, it is impossible for a message to arrive at its destination before a previously transmitted message has done so. Thus, correct messages are received in FIFO order.

Another common topology found in local area networks is the ring. The communication media in this topology is the ring, which is usually composed of fiber optic lines. Such networks have become popular lately due to their high communication rate [M]. A common example of this network is the Token Ring Network. In such network there is a special message which is called "token" [M]. This token is transmitted around the ring from station to station. A station that wishes to transmit messages, must first receive the token. It then keeps the token, i.e. does not transmit it forward, and transmits its messages. Broadcasting is achieved in the ring, since every station connected to it can hear every message transmitted. Again, the common communication media, the ring, stands for the media and the inbound channels of the
general model and $d_{in}$ is the maximal time period from a message transmission until its reception at the destination.

We now show that the ring maintains FIFO which corresponds to the FIFO in the inbound channels of the general model. What is in fact must be shown is that if two stations $I$ and $J$ transmit to the same destination station $K$, then if station $I$ transmits first (and in fact puts its message in the inbound channel of $K$), then the messages of $I$ arrives at $K$ before the messages from $J$. However, most kinds of token rings [BCKKM] implement two rules: there is only one token in the ring, and only one station can transmit messages at a time. This station is the one that holds the token. Secondly, the token is released only after the station has completed its transmission. It is clear now that $J$ must first receive the token from $I$, before transmitting to $K$. However, $I$ releases the token only when finishing to transmit its messages. Therefore, the messages from $I$ to $K$ arrive at $J$ before the token, and so $J$ transmits its messages to $K$ after transferring the messages from $I$.

Packed radio networks are used for communication over longer distances. Again, all stations use a common communication media, in this case a radio channel, and use transmitters and receivers to exchange messages through the channel.

The ALOHA system, first used at the University of Hawaii [A], was one of the first networks to use this method, which provides communication facility to a large number of users, each communicating at a slow rate, randomly. Again, since all stations are connected to the radio channel, every station can hear every message, and so broadcasting is achieved. As in Ethernet, the radio network is a special case of our general model with $d_{out}=0$, absence of collisions in these channels, and a queue of size zero in the media. The radio channel stands for the common media of the general model. A contention based access protocol, such as ALOHA, might cause collisions, and these stand for the collisions in the inbound channels of the general model. As before, $d_{in}$ is the maximal propagation time of a message.

Finally, we mention the satellite networks. These networks are used for communication between remote stations [S]. Ground stations transmit messages to the satellite, which transmits them back to earth. The
broadcast nature of the network is achieved by the capability of every station to hear every message transmitted from the satellite. The communication channels from the stations to the satellite represent the outbound channels. The satellite stands for the media, since every transmitted message traverses it. The satellite receives messages from the stations via a receiver on the satellite, and transmits the messages back to earth via a transmitter. Satellites with capability to store messages (on board storing) are characterized in the general model by the queue in the media. Satellites without this capability are characterized by a queue of size zero. The communication channels from the satellite stand for the inbound channels.

Satellite networks are characterized by high propagations delays in the inbound and outbound channels, i.e., \( d_{in} \) and \( d_{out} \) are large. The FIFO in these networks is achieved by assuming that the satellite transmits messages in FIFO order.

Up to this point we have described the correspondence between existing networks and the general model. We return now to describe some further attributes of the model. The task of assigning addresses is performed by a special station, which serves as an address manager, or for short manager. The manager is assumed to experience no failures and to remain connected to the network at all times. Its address \( m \) is known to all stations. Other stations can enter or leave the network in an arbitrary fashion. The manager controls the assignment/unassignment process by exchanging control messages with the stations via the shared media. Both the manager and the stations run address assignment algorithms which perform the assignment/unassignment procedures.

A station can be in one of two states. In alive state, its transmitter can send messages and its receiver accepts messages. In dead state, its transmitter and receiver do not act upon messages. The process of a dead station becoming alive or vice versa is based on external factors which are outside the control of the address assignment/unassignment procedure. However, if a dead station becomes alive, it is assumed to have cleared its inbound and outbound queues and to have no memory of any address that it may have been assigned in a previous alive state. It will, however, recognize the universal address.

We further assume that every station has some kind of unique preassigned identity. This identity is used only by the first messages of the address assignment protocol. Therefore, this identity may be relatively long, and uniqueness is necessary only during the process of assigning an address to it.
and the incoming queue of \( I \). Similarly, the \( M-M \) chain is defined as the path taken by a message sent by the manager and received by itself, the \( I-M \) chain is defined at the path from station \( I \) to the manager and the \( M-I-M \) chain will be the concatenation of the \( M-I \) chain and the \( I-M \) chain.

3.2 Main properties of the address assignment protocol.

The address assignment protocol of [GS2] fulfills several main properties, which guarantee correct address assignment:

Property 1: Uniqueness:

No two stations have the same assigned address, i.e., the mapping \( SA : S \rightarrow \mathcal{A} \) is one-to-one.

Property 2: Assignment:

If there exists a station \( I \) that is alive but without an address, then \( I \) will eventually be assigned an address, unless it dies or there are no available undesignated addresses. In formal notation, if

\[ I \in S \land \exists S' \land \forall \mathcal{A} \in \mathcal{A}_{\text{comp}} \]

then eventually,

\[ I \in S \lor I \in S' \lor \mathcal{A} = \mathcal{A}_{\text{comp}} \]

Property 3: Consistency:

a) \( \mathcal{A}_n \subseteq \mathcal{A}_m \); \( S \subseteq S_m \)

b) \( MA(I) = SA(I) \) for all \( I \in S \)

c) No address can be in \( \mathcal{A}_m \cdot \mathcal{A} \) forever.

A fourth property states that a station keeps its assigned address until it dies, provided the channel remains "good". The property is difficult to specify formally, as it involves a specification of what it means for the channel to be "good". Intuitively, the property ensures that the protocol requires a station to relinquish its assigned address only when this is unavoidable [GS2].
3.3 Description of the address assignment protocol

The protocol consists of the address assignment algorithm that runs at the manager, of the station algorithm and of the messages they send and receive. The manager has a separate instance of the algorithm for every address and all instances run in parallel. A station has a single instance of the algorithm.

The algorithms are composed of three types of modes: Probing mode - represented by a rectangle, Response mode - represented by an ellipse and Wait mode - represented by a square.

Probing and Response modes are depicted in Figure 3.1.

When entering into either a Probing or a Response mode, operation ACTION is performed (if such exists) and a timer is set to expire <\textit{T}_\text{timer}> seconds afterwards. In probing mode, MSG is repeatedly sent at arbitrary intervals, while in Response mode MSG is sent only when MSG" is received. In both modes, if MSG' is received, no more messages are sent. MSG' is referred to as the appropriate response for the mode.

There are two types of transitions out of a Probing or a Response mode. The F-transition is taken if the appropriate response is not received before the timer expires. If the appropriate response is received before the timer expires, the other transition out of the mode is taken: S' indicates that the transition is taken immediately upon receipt of the appropriate response, while S indicates that the transition occurs...
when the timer expires. MODE_LABEL is used to label the mode.

Wait mode is depicted in Figure 3.2.

![Figure 3.2 - Wait mode](image)

In Wait mode, the protocol waits \(<T_{\text{timer}}\) seconds and then leaves. Again, MODE_LABEL is the name of the mode.

The address assignment algorithms for the stations and for the manager are depicted in Figure 3.3.

Station \(I\) initiates the protocol by repeatedly sending \(RQST\) messages that contain its own unique identity \(I\). The destination address of the \(RQST\) messages is the universal address. Upon receiving the first \(RQST\), the address manager selects an undesignated address and designates it to \(I\). It then informs the station of the selected address by repeatedly sending \(ASSN\) messages with the universal address and containing the designated address and the identity \(I\). Upon receiving the first \(ASSN\) containing identity \(I\), the station accepts the address contained in it as its assigned address and, for this \(ASSN\) and any subsequent \(ASSN\)'s, sends back acknowledgments \(AACK\) to the address manager. Any \(ASSN\)'s that do not contain the identity \(I\) are discarded. When the address manager receives an \(AACK\), the assignment phase of the protocol, which is essentially a three-way handshake, is complete.

A new phase, the operational phase, now begins, wherein the manager determines whether station \(I\) is still operational. This is done by the exchange of \(POLL\) and \(PACK\) messages between the manager
and station $I$ every $T$ seconds. The manager initiates the process by sending $POLL$ messages until receiving the first $PACK$ message from $I$. It then remains quiet until the start of the next $T$-seconds period. The station responds to $POLL$ messages by sending $PACK$ messages to the manager.

There are two types of $POLL$ messages - $POLL0$ and $POLL1$, which are sent intermittently - an interval of $POLL0$ messages and an interval of $POLL1$ messages. $PACK$ messages are also of two types - $PACK0$ and $PACK1$. The station responds to $POLL0$ messages by sending $PACK0$ messages, and similarly responds to $POLL1$ messages by sending $PACK1$ messages. The two types guarantee that a $PACK$ message received in a particular interval, was indeed sent as a response to a $POLL$ message of the same interval.

The operational phase ends, and the unassignment phase begins if the address manager does not receive an appropriate $PACK$ before the next $T$-second interval begins. The manager assumes that $I$ is dead, and if $T$ is chosen properly, with high probability the station will in fact be dead.

There is, however, a chance that the station may still be alive, and to prevent the address from being redesignated in this case, the manager does not redesignate the station’s address until another $3T$ seconds
have elapsed. This extra wait is obtained by the WAIT_C mode. It is proved in [GS2] that when the
manager leaves this mode, the algorithm at I is in either DEAD_STATE or REQUEST mode. When the
algorithm at I is in one of these modes, station I does not have an address and is allowed to seek one.

The rest of the modes are used when the address assignment fails. These modes are not important to
our discussion, and a full description about their task appears in [GS2].

Recall that we use the terms assign and unassign to describe the stations' actions and the terms
designate and undesignate to indicate the manager's actions. The manager indicates to itself that x is
designated to I by setting $MA(I) = x$ when algorithm$^x$ leaves the UNASSIGN mode. In addition, x is added
to the table $A_m$ of designated addresses and I is added to the table $S_m$ of stations with designated
addresses. At this time we say that x was designated to I. When algorithm$^x$ returns to the UNASSIGN
mode, $x$ is taken out of $\mathcal{A}_n$, station $I$ is taken out of $\mathcal{S}_n$ and the connection $MA(I)=x$ is cancelled ($x$ is undesignated).

Station $I$ indicates to itself that its assigned address is $x$ by setting $SA(I)=x$. This is done when its algorithm leaves the REQUEST mode after receiving an ASSN message. At this time we say that $I$ has adopted address $x$. When the algorithm of $I$ enters REQUEST mode, the station does not have an assigned address, and at this time $SA(I)=\text{nil}$.

The size of $T$ seconds is defined as a time long enough to allow a station, with adequate probability, to send a message and receive an acknowledgement. We do not specify exactly what is an adequate probability. The probability that a station will receive acknowledgement within $T$ increases as $T$ increases. On the other hand, the longer $T$ is, the longer it takes to detect station disconnections and to release its address. The network operators will decide what is an adequate probability and will set $T$ accordingly, referring to the expected delays in the various queues, propagations delays, probability of collision and noise in the channels.

The protocol we described so far designates/undesignates addresses. However, as shown in the next section, it does not create all the necessary conditions to allow addresses to be redesignated, and does not include session management. The next section defines basic properties for correct session management and sections 5, 6, and 7 present three models for address distribution and session management.

4. SESSION MANAGEMENT IN A DYNAMIC ADDRESS ENVIRONMENT

In order to allow stations to maintain sessions properly, i.e. to resolve problems (i)-(iii) of Sec. 1, it is necessary to compound the protocol of [GS2]. In this section we state the additional properties that the compounded protocols must possess, in order to ensure proper session management.

First, it is important to distinguish between different designations of a given address $x$. To do this, whenever address $x$ enters $\mathcal{A}_n$ for the $n^{\text{th}}$ time, we shall say that address $x_n$ is designated. Algorithm $^x$ will denote the algorithm that designates/undesignates $x_n$. All modes of algorithm $^x$ while algorithm $^x$ is active,
contain superscript $x_n$. We now also divide the possible messages in the network into three categories:

1) The messages that are sent by the address assignment algorithms of [GS2] in the manager and in the stations, as described in Sec. 3. These messages are \textit{RQST}, \textit{ASSN}, \textit{AACK}, \textit{POLLO}, \textit{PACKO}, \textit{POLLI}, \textit{PACK1}, \textit{CLR0} and \textit{CLR1}.

2) This category contains several new kinds of messages used by our three models: \textit{FIND}, \textit{ADDR}, \textit{NO_ADDR}, \textit{CLEAN}, \textit{CACK}, \textit{ADD}, \textit{ADACK}, \textit{ADDRL} and \textit{ADLACK}. The \textit{FIND}, \textit{ADDR}, and \textit{NO_ADDR} messages are used in all the three models we present later. The second model (Sec 6) also uses the \textit{CLEAN} and \textit{CACK} messages and the last model uses all the above types. These messages accomplish the address distribution described in the various models. The \textit{ADDR}, \textit{NO_ADDR}, \textit{CLEAN}, \textit{ADD} and \textit{ADDRL} messages are sent by the manager to the stations. When such messages are sent carrying $x_n$ as destination address, corresponding to the $n^{th}$ designation of $x$, we denote these messages by $\textit{DSTR}_n$.

3) Messages that stations send to each other while maintaining a session between them. These messages will be referred to as \textit{DATA} messages. In particular, suppose that stations $I$ and $J$, with addresses $x$ and $y$ respectively, maintain a session. All messages that $J$ sends to $I$ while maintaining the session are denoted by $\textit{DATA}_x$, i.e. the subscript indicates the address of the station the \textit{DATA} messages are intended to. Similarly, the messages that $I$ sends to $J$ are denoted by $\textit{DATA}_y$ messages. A session in which messages $\textit{DATA}_x$ are sent is denoted by $\textit{session}^x$. Notice that the above session is also a $\textit{session}^x$.

With these notations, we can proceed to define three properties that summarize correct session management. Let $T^x_n$ be the time when $x$ is included in $A_n$ for the $n^{th}$ time (see Fig. 4,1) and suppose $x_n$ is designated to station $I$ ($\textit{MA}(I)=x_n$). Let $T^x_n$ be the first time after $T^x_n$ when $x$ is taken out of $A_n$, i.e. it is undesignated. With this notation and previous definitions, messages $\textit{DATA}_x$ and $\textit{DSTR}_x$ are intended to $I$. The first property states that those $\textit{DATA}_x$ and $\textit{DSTR}_x$ messages will indeed be received only by $I$, and not by stations that will later have $x$ as their assigned address. This property essentially says that messages are received only by the station they are intended to. Recall that by definition, a message is received
by a station when the latter processes the message. In particular messages arriving at the stations' incoming queue and discarded because the station does not have an assigned address or because the address does not match, are not considered as received by the station.

Thus, the first property is:

**Property (A):** $DATA_{x_n}$ and $DSTR_{x_n}$ messages are received only by station $I$ and only when $SA(I) = x_n$.

However, requiring property (A) is not sufficient. Assume that address $x_{n+1}$ is designated a long time after $T^n_1$ and suppose stations store $x_n$ as the designated address of $I$ after $T^n_1$. Since after $T^n_1$ and until $x_{n+1}$ is designated there is no station with designated address $x$, property (A) does not prevent the other stations from continuing to try to initiate sessions with $I$ by sending $DATA_{x_n}$ messages, and does not prevent the manager to send $DSTR_{x_n}$ messages. This is because these messages are in fact received by no station, and property (A) still holds. These messages load the network unnecessarily, and to prevent this situation, we also require property (B):

**Property (B):** No $DATA_{x_n}$ and $DSTR_{x_n}$ messages are sent after $T^n_1$.

Finally, note that property (A) can be trivially enforced by prohibiting redesignation of $x$. However, since the set $\mathcal{A}_{comp}$ of possible addresses is finite, this will cause the manager to run out of available addresses thereby blocking the network to new stations. Therefore, the following property is also required:

**Property (C):** Address $x_{n+1}$ becomes available for designation within finite time after $T^n_1$. 
It can be easily seen that properties (A)-(C) prevent problems (i)-(iii) of section 1. Therefore, our goal is to define protocols that have these properties in addition to the properties of the original protocol [GS2]. The protocols we suggest are expansions of the original protocol. All additions are directed to design the timing of the following events:

(T1) After address \( x \) is designated to station \( I \), what is the correct time to distribute this information to other stations, thereby enabling them to use destination address \( x \) in messages intended for \( I \).

(T2) When is station \( I \) allowed to receive a message containing destination address \( x \), i.e. when can it be sure that the message is intended for \( I \), rather than for a station that previously had \( x \) as its designated address.

(T3) After \( x \) is undesignated, how do other stations become aware that \( x \) does not belong to \( I \) anymore.

(T4) After \( x \) is undesignated, when can \( x \) be redesignated.

As shown presently, correct decisions on those issues are essential to ensure properties (A)-(C).

5. MODEL A - ADDRESS DISTRIBUTION WITHOUT MEMORY

5.1 The model

In model A stations do not maintain address tables. Thus, a station will always ask for the address of the other station before initiating a session with it. In this model the stations do not have to manage address tables, a task that may require a considerable amount of processing when stations join and leave the network frequently. However, the model requires that a station requests an address every time a session is started, even if the address does not change. In this model only problems (i) and (ii) of Sec. 1 may occur, since stations do not store addresses of other stations.

5.2 The Protocol

Before introducing a solution, we must describe the manner in which stations request addresses of other stations and manage sessions. When station \( I \) wants to find the address of station \( J \) say, it sends
messages \textit{FIND}(J) to the manager, containing the identity \textit{J}. Upon receiving a \textit{FIND}(J) message, the manager checks, in a manner to be described later, whether \textit{J} has a designated address and whether that address can be given out. If so, it sends \textit{ADDR} messages, containing the identity \textit{J} and the address. Otherwise, it sends \textit{NO_ADDR}(J) messages. If \textit{I} receives an address for \textit{J}, it begins to send messages to \textit{J} using the received address. Otherwise, \textit{I} may repeat the question immediately, delay the question to a later time or give up the attempt to talk with \textit{J}. The \textit{FIND}, \textit{ADDR} and \textit{NO_ADDR} messages are piggybacked to the \textit{(POLLPACK)} messages so as to ensure that an answer is received by \textit{I} within finite time, unless it becomes inactive as described in Sec. 3. Station \textit{I} manages the session with \textit{J} via a timeout mechanism, using a timer that expires after \textit{C} seconds. The timer is reset at the time when \textit{I} sends the first message to \textit{J} and every time when \textit{I} receives a \textit{DATA} message from \textit{J} prior to the expiration of the timer. If the timer expires, i.e. if \textit{I} receives no message for \textit{C} seconds after the timer is reset, \textit{I} terminates the session. Station \textit{J} manages the session similarly, setting its timer for the first time when it receives the first \textit{DATA} message from \textit{I}.

A station may start a session with or without involvement of the manager. In the scenario described above, \textit{I} receives the address of \textit{J} from the manager so that \textit{I} initiates the session with the help of the manager. On the other hand, the session at \textit{J} is initiated when the latter receives the first \textit{DATA} message from \textit{I}, without any communication between \textit{J} and the manager. A session can start without involving the manager at all. For example suppose \textit{I} needs an application program, whose location is unknown to it. Station \textit{I} will send a message to all stations (with the universal address *), containing its own address, informing them that it seeks the program. If the program is located at station \textit{J} say, the latter will reply to \textit{I} using the received address. In this case there is a session between \textit{I} and \textit{J} without the involvement of the manager.

Notice that the method above may also be used by \textit{I} to find out the address of \textit{J} by asking \textit{J} to send its address if it is attached to the network. However, since messages may get lost, if \textit{I} does not receive from \textit{J} an answer for its search messages, it does not know whether \textit{J} is in the network but its messages are lost, or whether \textit{J} is not in the network. The involvement of the manager using the \textit{(POLLPACK)} messages guarantees a correct answer within a known, finite period of time.
The session between $I$ and $J$ terminates when one of the following two events happens:

1. No message is received by one of the stations within $C$ seconds after receiving the previous one.
2. Exchange of appropriate termination messages.

Remarks:

1. The manner described above requires every station to ensure that the station it communicates with receives a message every $C$ seconds, as long as it wants the connection to continue. If there is no data to send for a while, but the connection is still needed, short messages with no data should be sent.
2. We do not specify the size of $C$ and will not make any assumption about it. It is a parameter of the network and must be determined as a function of the expected loads, propagation delays and delays in the various queues.

After describing the address distribution and session management, we can now proceed to suggest a protocol for the model, that will ensure properties (A-(C) of Sec. 4. The protocol is depicted in Figure 5.1, where the additions to the original protocol [GS2] are emphasized in boldface. Observe that address distribution and session management is a separate protocol for the address assignment protocol, so that the $FIND, ADDR$ and $NO_ADDR$ messages do not appear in Fig. 5.1.

To summarize the additions, every station that is alive may be in one of two states, active or inactive (see Fig. 5.1a). A boolean variable, $OP(I)$, distinguishes between the times at which station $I$ is active, ($OP(I)=\text{TRUE}$), and inactive ($OP(I)=\text{FALSE}$). When active, a station is allowed to request the address of any other station, to initiate sessions and to receive $\text{DATA}_{\text{addr}}(I)$ messages. When inactive, a station is only allowed to send or receive messages except those that belong to the address assignment protocol (in order to adopt an address). In addition, the PACK0 mode of the station algorithm is separated into two modes, PACK0 and INIT_PACK0. This distinguishes between the first $\text{POLL}1$ message received by the station after adopting an address and the later ones. The first is received by the station in
INIT_PACK0 mode, while the following ones are received in PACK0 mode. After receiving the first POLL1, a station becomes active.

There are some changes in the manager algorithm as well. A new table, \(O_m\), is maintained by the manager. Only addresses of stations appearing in this table may be distributed in ADDR messages, and replies to FIND messages (ADDR or NO_ADDR) are sent only to stations appearing in this table. A station with designated address \(x\) is included into \(O_m\) when \(algorithm^x\) makes the transition from the INIT_POLLO mode to the POLL1 mode, and is taken out of \(O_m\) when \(algorithm^x\) leaves the WAIT_C mode.

In addition, two new modes are added to \(algorithm^x\): STOP_GENERATE and CLEAN (see Figure 5.1b). The STOP_GENERATE mode accomplishes Property (B) of Sec. 4: \(algorithm^x\) stays \(d_{out}+\max(2T+C,5T)\) seconds in STOP_GENERATE, a time period after which it is guaranteed that no
Figure 5.1b - Algorithm at the address manager

more $DATA_x$ or $DSTR_x$ messages are generated.

The CLEAN$^*$ mode guarantees Property (A) of Sec 4 for $DATA_x$ messages. After the time when CLEAN$^*$ is exited, the outgoing queues of all stations are cleared of $DATA_x$ messages. Clearly, only
stations that are active at the time when CLEAN\textsuperscript{x} is entered can contain \textit{DATA\textsubscript{x}} messages at the time when CLEAN\textsuperscript{x} is exited. As we prove later, every active station appears in \textit{o\textsubscript{x}}. Therefore, when entering CLEAN\textsuperscript{x}, \textit{algorithm}\textsuperscript{x} copies \textit{o\textsubscript{x}} into a special table \textit{V}. Then, \textit{algorithm}\textsuperscript{x} will leave the mode when it is sure, in a manner to be described later, that every station in \textit{V} has cleared its outgoing queue of \textit{DATA\textsubscript{x}} messages.

We next explain why CLEAN\textsuperscript{x} is sufficient to ensure Property (A) for \textit{DATA\textsubscript{x}} messages. The only way for this property not to hold is if a station that adopts address \textit{x} after \textit{I} releases it, receives \textit{DATA\textsubscript{x}} messages. After the time when CLEAN\textsuperscript{x} is exited, \textit{DATA\textsubscript{x}} messages can be only in the outbound channels, media queue, inbound channels and incoming queues of the stations. Assume now that \textit{x\textsubscript{n+1}} is designated to a station \textit{K}. The following discussion is true for \textit{K} and every station that will adopt \textit{x} after \textit{K}. Station \textit{K} becomes active, and so receives \textit{DATA} messages, only after receiving the first \textit{POLL1\textsuperscript{x}} message from \textit{algorithm}\textsuperscript{x+1}. \textit{Algorithm}\textsuperscript{x+1} sends \textit{POLL1\textsuperscript{x}} messages only from the POLLO\textsuperscript{x+1} mode. It enters into POLLO\textsuperscript{x+1} for the first time after designating \textit{x\textsubscript{n+1}} to \textit{K}, at least \textit{T} seconds after \textit{algorithm}\textsuperscript{x} has left the CLEAN\textsuperscript{x} mode and has entered into the UNASSIGN mode. This is because \textit{algorithm}\textsuperscript{x+1} first stays \textit{T} seconds in the INIT_POLLO\textsuperscript{x+1} mode. Recall that the maximal propagation delay in the outbound channels is \textit{d\textsubscript{out}} seconds, and \textit{T} is obviously larger than \textit{d\textsubscript{out}} (\textit{T} must be set at least as the time it takes a message and its acknowledgement to travel from the manager to one station and back, and it obviously contains the maximal delays in the channels). Therefore, at the time when \textit{algorithm}\textsuperscript{x+1} enters POLLO\textsuperscript{x+1} for the first time, the outbound channels are also clear of \textit{DATA\textsubscript{x}} messages, and so \textit{DATA\textsubscript{x}} messages can be only in the media, inbound channels and incoming queues. Recall that the channels and queues maintain FIFO discipline. Thus, the first \textit{POLL1\textsuperscript{x}} message that \textit{algorithm}\textsuperscript{x+1} sends to \textit{K}, clears M-K, and in fact the entire system, of all the \textit{DATA\textsubscript{x}} messages that can possibly arrive at \textit{K}. Consequently, all \textit{DATA\textsubscript{x}} messages that arrive at \textit{K}, arrive when \textit{K} is still inactive and so are not being received. We prove later that no \textit{DSTR\textsubscript{x}} messages are received by \textit{K} either.
5.3 Correctness proof

In this section we prove that the proposed protocol has properties (A)-(C) of section 4. The proof relies on five lemmas: Lemma 5.1 shows that the additions to the original protocol do not affect its four main properties, mentioned in section 3.2. Lemma 5.2 quotes some other basic properties of the original protocol. Lemma 5.3 shows that an active station always appears in \( o_m \). Lemma 5.4 shows that after the time when \( \text{STOP GENERATE}^x \) is exited, no more \( DATA_a \) messages are generated. Finally, lemma 5.5 shows that after the time when \( \text{CLEAN}^x \) is exited, the outgoing queues of all stations do not contain \( DATA_a \) messages. After proving the five lemmas, we shall prove properties (A)-(C).

Lemma 5.1: The main properties - Uniqueness, Assignment and Consistency of the original protocol [GS2], hold for the new protocol.

Proof: The operation of the new protocol can be viewed as a possible operation of the original protocol. The \( \text{STOP GENERATE} \) and \( \text{CLEAN} \) modes only delay the transition of the algorithm at the manager from the \( \text{WAIT}_C \) mode to the \( \text{UNASSIGN} \) mode. However, since the original protocol may stay in \( \text{UNASSIGN} \) mode for an arbitrarily long time period, this delay can be viewed as extending the stay of the original protocol in \( \text{UNASSIGN} \) mode. Separating the \( \text{PACK0} \) mode into two modes, \( \text{INIT PACK0} \) and \( \text{PACK0} \), does not affect the basic properties since it merely distinguishes between the first received \( \text{POLL1} \) message and the following ones. The tables \( o_m, V \) and the variable \( OP \) also do not affect the operation of the original protocol.

In [GS1] it is shown that the original protocol possesses a number of additional, basic properties. Since the new protocol can be viewed as a possible operation of the original one, these properties hold for the new protocol as well and are stated in Lemma 5.2. These properties will be extensively used in the proof of lemmas 5.3, 5.4 and 5.5, as well as in the next models. For a precise definition of the properties, recall that the definition of \( M-I \) is the path taken by a message sent by the manager and received by \( I \), i.e. the concatenation of the manager's outgoing queue and transmitter, the inbound channel, the media.
1. Station \( I \) is either dead or in REQUEST mode (i.e. \( SA(I) = nil \)).

2. There are no \( ASSN(I) \) messages for any address \( y \) in the M-I-M chain.

3. There are no \( ASSN(J) \) or \( AACK_{\text{a}} \) messages anywhere in the system.

4. There are no \( CLR_{\text{a}} \) messages in the M-M chain.

5. No station \( J \) has \( SA(J) = x \).

\( e: \)

Let \( T_R \) be some time when \( SA(I) = x_a \) and \( I \) receives a \( POLL_{1;\text{a}} \) message in PACK0 mode. Let \( T_S \) be the time when the \( POLL_{1;\text{a}} \) is sent by \( \text{algorithm}^a \). Let \( T_E \) be the first time before \( T_S \) when \( \text{algorithm}^a \) enters POLL1, i.e. \( \text{algorithm}^a \) is in POLL1 during the entire interval \([T_E, T_S]\).

1. At \( T_E \) holds \( MA(I) = x_a \) and \( SA(I) = x_a \) (i.e. \( x \) is not undesignated in \([T_E, T_R]\)).

2. \( I \) is in PACK0 during the entire interval \([T_E, T_R]\).

3. According to the protocol, when station \( I \) receives the \( POLL_{1;\text{a}} \) messages at \( T_R \), it moves to the PACK1 mode. If \( I \) does not send a \( PACK_{1;\text{a}} \) message (i.e. no \( PACK_{1;\text{a}} \) message is leaving the outgoing queue of \( I \)) when staying in PACK1, than at \( T_R + 2T \) station \( I \) makes the transition to DEAD_STATE or REQUEST modes.

The above properties also hold when the \( POLL_{1;\text{a}} \) message is received when \( I \) is in INIT_PACK0 mode, and also with \( POLL_{0;\text{a}} \) replacing \( POLL_{1;\text{a}} \), when it is received in AACK or PACK1 modes.

Property 5.2e is not stated and is not proved in [GS1]. However, since we use it in our proofs below, we state this property here and prove it in the appendix.

Properties 5.2a - 5.2e are now used to show that the new modes fulfill their task:
1) No $DATA_{x_n}$ messages are generated after the time when STOP\_GENERATE is exited.

2) All the outgoing queues are clear of $DATA_{x_n}$ messages after the time when CLEAN is exited.

First, we show that when active, a station appears in the table $O_m$.

**Lemma 5.3:** Station $I$ in active state ($OP(I)=\text{TRUE}$) appears in $O_m$.

**Proof:** The situation is summarized in Figure 5.2.

![Algorithm](image)

**Figure 5.2 - When $OP(I)=\text{TRUE}$, $I \in O_m$**

Suppose station $I$ is active, i.e. $OP(I)=\text{TRUE}$ during $[T_3,T_4]$. We show that in $[T_3,T_4]$ station $I$ is also in $O_m$. Let $x$ be the address of $I$ during $[T_3,T_4]$ and suppose this is the $n^{th}$ assignment of address $x$.

As described in Sections 3 and 4, we denote this by $SA(I)=x_n$.

First, we show that at $T_3$ station $I$ is included in $O_m$. Station $I$ sets $OP(I)\leftarrow\text{TRUE}$ when its address algorithm makes the transition from INIT\_PACK0 to PACK1. This transition occurs when $I$ receives the first $POLL_{1^2}$ message after setting $SA(I)=x_n$. Messages $POLL_{1^2}$ are sent only from POLL1 mode at
algorithm\textsuperscript{*}. Let $T_2$ be the time when algorithm\textsuperscript{*} sent the POLL\textsubscript{1} that is received at $T_3$, and let $T_1$ be the first time before $T_2$ when algorithm\textsuperscript{*} enters POLL1, i.e. algorithm\textsuperscript{*} is in POLL1 mode over the entire interval $[T_1,T_2]$.

At $T_3$ the POLL\textsubscript{1} message is received by $I$ in INIT\_PACKO. Lemma 5.2e implies that $I$ is in INIT\_PACKO over the entire interval $[T_1,T_3]$ and that $MA(I)=x_n$ at $T_1$. Since the stay in INIT\_PACKO is at most for $2T$ seconds, holds $T_3-T_1<2T$. On the other hand, after entering $o_n$, a station $I$ stays there for at least $4T$ seconds. At $T_1$ station $I$ enters $o_n$ since algorithm\textsuperscript{*} enters POLL1 with $MA(I)=x_n$ and hence at $T_3$ station $I$ is still in $o_n$.

We now show that at $T_4$ station $I$ is still in $o_n$. Let $T_5$ be the next time after $T_1$ when $I$ is taken out of $o_n$, i.e. when algorithm\textsuperscript{*} leaves WAIT\_C. Lemma 5.2c implies that at $T_5$ the address algorithm of $I$ is either in REQUEST or DEAD\_STATE and thus $OP(I)=$FALSE. Consequently $T_5$ cannot be before $T_4$ since in $[T_3,T_4]$ holds $OP(I)=$TRUE. Hence $T_5>T_4$ and the interval $[T_3,T_4]$ is included in the interval $[T_2,T_3]$.

\[ \square \]

Lemma 5.4 proves that the STOP\_GENERATE\textsuperscript{z} mode performs its task regarding DATA\textsubscript{z} messages.

Let $T_0$(STOP\_GENERATE\textsuperscript{z}) be the time when STOP\_GENERATE\textsuperscript{z} is exited.

\[ \text{Lemma 5.4: No DATA}_{\text{z}}\text{ messages are generated after }T_0(\text{STOP\_GENERATE}^{z}). \]

\[ \text{Proof: DATA}_{\text{z}}\text{ messages are generated only in sessions}^{z}. \text{ Therefore, it is sufficient to show that there are no sessions}^{z} \text{ after }T_0(\text{STOP\_GENERATE}^{z}). \]

Let $T_E$(STOP\_GENERATE\textsuperscript{z}) be the time when STOP\_GENERATE\textsuperscript{z} is entered, and let $J$ be an arbitrary station. We separate the discussion into two cases: If after $T_E$(STOP\_GENERATE\textsuperscript{z}) station $J$ receives neither DATA messages from $I$ nor ADDR($J,x_n$) messages, then $J$ will terminate every session\textsuperscript{z} it possibly maintains within $C$ seconds after $T_E$(STOP\_GENERATE\textsuperscript{z}), i.e. until $T_E$(STOP\_GENERATE\textsuperscript{z})+$C$. 
That is clear since only DATA messages from I, or ADDR(I,x_n) messages, can cause J to initiate or maintain sessions_x^*, and the timer mechanism ensures the termination of all sessions_x^* in J.

Therefore, assume that after T_E(STOP_GENERATE^*) station J receives either DATA messages from I or ADDR(I,x_n) messages. Such messages are not generated after T_E(STOP_GENERATE^*), because after T_E(STOP_GENERATE^*) station I is not in o_m with MA(I)=x_n anymore. (I is taken out of o_m at T_E(STOP_GENERATE^*) and x_n is assigned only once). Thus, using lemma 5.3, it is clear that after T_E(STOP_GENERATE^*) station I is not active again with SA(I)=x_n, and thus will not generate DATA messages containing x_n. Also, since I is not in o_m, the manager will never generate ADDR(I,x_n) messages again.

Let L be the last message of the above two types, that is received by J. Message L arrives at M-J at the latest at T_E(STOP_GENERATE^*)+d_wait. This is the case when L is a DATA message sent by I, since station I is inactive at T_E(STOP_GENERATE^*), its outgoing queue is clear of DATA messages at this time and it will take d_wait seconds for the last DATA message that leaves the outgoing queue of I to arrive at the media. Notice that if L is an ADDR(I,x_n) message, then L is in M-J immediately after being generated, because only the manager generates and sends ADDR messages.

If L stays in M-J less than 2T seconds, then every session_x^* in J is terminated until T_E(STOP_GENERATE^*)+d_wait+(2T+C). That is because until T_E(STOP_GENERATE^*)+d_wait+2T, message L is received by J. Lemma 5.2b implies that while in STOP_GENERATE^*, there is no station K such that SA(K)=x. Thus, within C seconds after T_E(STOP_GENERATE^*)+d_wait+2T, station J does not receive any DATA message from a station with assigned address x, and therefore terminates all sessions_x^*.

The last case we must consider is when L stays in M-J for 2T seconds or more. Let T_1 be the time when L arrives at M-J. Recall that T_1\leq T_E(STOP_GENERATE^*)+d_wait. We assume that L is received by J at time T_3 say, and so J is active at T_3. Notice that SA(J)=nil at T_3, and suppose SA(J)=y. We will prove later that at T_1, algorithm^2 is in POLLO, POLL1 or WAIT_C. Using this result, J terminates all sessions_x^* until T_1+d_wait+5T. To see why this is true, we first assume that algorithm^2 is in WAIT_C at T_1. Within at
most 3T seconds after \( T_1 \), at time \( T_2 \) say, \( \text{algorithm}^y \) leaves \( \text{WAIT}_C \) and moves to \( \text{UNASSIGN} \) mode. Lemma 5.2b implies that at \( T_2 \) there is no station in the network with designated address \( y \) and so \( J \) is not active with \( SA(J)=y \) at \( T_2 \) either. We now argue that \( L \) is received by \( J \) before \( T_2 \).

In order to become active with \( SA(J)=y \) after \( T_2 \), and receive \( L \), station \( J \) must first adopt \( y \) by receiving \( \text{ASSN}(J) \) message from \( \text{algorithm}^y \). At \( T_2 \) there are no such messages in the network (lemma 5.2c). Thus, such an \( \text{ASSN} \) message must be sent after \( T_2 \), and so it will clear \( M-J \) of \( L \) and clearly \( L \) will not be received by \( J \) when \( SA(J)=y \). Since we assume that \( L \) is received by \( J \) when \( SA(J)=y \), then it must have been received before \( T_2 \). At \( T_2 \), station \( J \) is inactive, after receiving \( L \). Thus, \( J \) terminates all \( \text{sessions}^x \) until \( T_2 \), and recall that \( T_2 \) is before \( T_1+3T \), i.e. before \( T_E(\text{STOP}\_\text{GENERATE}^x)+d_{\text{out}}+3T \).

Suppose now that \( \text{algorithm}^y \) is in \( \text{POLLO} \) at \( T_1 \). The case in which \( J \) is in \( \text{POLLI} \) is symmetric. If \( \text{algorithm}^y \) does not receive a \( \text{PACK}_0^y \) message in \( \text{POLLO} \), it leaves the mode within \( T \) seconds, i.e., until \( T_1+T \). It then moves to \( \text{WAIT}_C \) mode for \( 3T \) seconds, and leaves \( \text{WAIT}_C \) at \( T_1+4T \), at the latest. Using similar arguments as before, at \( T_1+4T \), station \( J \) had already terminated every \( \text{session}^x \). Notice that \( T_1+4T\leq T_E(\text{STOP}\_\text{GENERATE}^x)+d_{\text{out}}+4T \). If \( \text{algorithm}^y \) receives a \( \text{PACK}_0^y \) in \( \text{POLLO} \), it moves to \( \text{POLLI} \) at time \( T_2 \) say, where \( T_2 \) is between \( T_1 \) and \( T_1+T \). Notice that \([T_2,T_2+T]\), which is the time period in which \( \text{algorithm}^y \) stays in \( \text{POLLI} \), is included in \([T_1,T_1+2T]\), and during this period \( L \) is in \( M-J \) . Lemma 5.2c implies that at \( T_2 \), there are no \( \text{PACK}_1^y \) messages in the network. While in \( \text{POLLI} \), station \( J \) sends \( \text{POLL}_1^y \) messages to \( J \), but those will not be received by \( J \) while \( \text{algorithm}^y \) is in \( \text{POLLI} \) in \([T_2,T_2+T]\), since message \( L \) is in \( M-J \) during \([T_1,T_1+2T]\) and so it blocks \( L \) from being received by \( J \). Thus, at \( T_2+T \) \( \text{algorithm}^y \) will leave \( \text{POLLI} \) and will move to \( \text{WAIT}_C \) mode. At \( T_2+4T \), \( \text{algorithm}^y \) leaves \( \text{WAIT}_C \), a time at which \( J \) had already terminated every \( \text{session}^x \) for the same reasons as before. Since \( T_2 \) is before \( T_1+T \) we have \( T_2+4T\leq T_1+5T\leq T_E(\text{STOP}\_\text{GENERATE}^x)+d_{\text{out}}+5T \).

We can now conclude that if after \( T_E(\text{STOP}\_\text{GENERATE}^x) \) station \( J \) receives either a \( \text{DATA} \) message from \( I \) or an \( \text{ADDR}(I,x_n) \) message, then it terminates every \( \text{session}^x \) until \( T_E(\text{STOP}\_\text{GENERATE}^x)+d_{\text{out}}+\text{MAX}(2T+C,5T) \). This time is longer then \( T_E(\text{STOP}\_\text{GENERATE}^x)+C \) which is sufficient in the case when \( J \) does not receive such messages after \( T_E(\text{STOP}\_\text{GENERATE}^x) \), and
so it is the appropriate stay in STOP_GENERATE.

We now explain why $algorithm'$ is either in POLL0, POLL1 or WAIT_C at $T_1$. Assume that $J$ is not in one of these modes. Thus, at $T_1$ station $J$ is not included in $\omega_a$ and lemmas 5.2a and 5.3 imply that $J$ cannot be at $T_1$ active with $SA(J)=y$. However, we assume that at $T_3$, when $J$ receives $L$, it is active with $SA(J)=y$. Thus, there is a time $T_2$ in $[T_1,T_3]$ when $algorithm'$ moves from INIT_POLL0 to POLL1 and includes $J$ in $\omega_a$. Lemma 5.2c implies that at time $T_2$, station $J$ is in DEAD_STATE, REQUEST or INIT_PACK0 modes, i.e. $J$ is inactive and there are no $POLL_1$ messages in M-J. In order to be active at $T_3$ with $SA(J)=y$, $J$ must first receive a $POLL_1$ message which was sent after $T_2$. Notice that $L$ arrives at M-J before this $POLL_1$ message. Consequently, since the $POLL_1$ is sent from the manager, FIFO guarantees that $L$ is received by $J$ before the $POLL_1$ message, i.e. when $J$ is not active with $SA(J)=y$. This contradicts our assumption, and so proves the claim.

$\Box$

Remark: The stay in STOP_GENERATE cannot be less than $d_{out}+\text{MAX}(2T+C,5T)$ seconds. Lemma 5.2 implies that $I$ can still be active until $T_E(STOP\_GENERATE^{\times})$, and therefore $L$ can indeed arrive at M-J until $T_E(STOP\_GENERATE^{\times})+d_{out}$. The $2T+C$ seconds interval is necessary, since it is possible that $L$ stays less than $2T$ seconds in M-J before being received by $J$. (It is easy to show a situation in which $L$ stays less than $2T$ seconds in M-J, then being received by $J$ and $J$ indeed maintains a session for $C$ seconds, without becoming inactive). The $5T$ seconds are also necessary since without any assumption about the size of $C$, we must take in account the case that $L$ stays in M-J more than $2T$ seconds, and that $J$ maintains a session until $5T$ seconds since the arrival of $L$ to M-J. Therefore, it is impossible to reduce the term $d_{out}+\text{MAX}(2T+C,5T)$.

We now proceed to prove lemma 5.5. In the model under consideration, messages can be in the outgoing queues, outbound channels, the media, the inbound channels and the incoming queues. Lemma 5.5
proves that at the time when CLEAN\textsuperscript{x} mode is exited, there are no DATA\textsubscript{x} messages in the outgoing queues of the stations. Let \( T_O(\text{CLEAN}^{x}) \) be the time when CLEAN\textsuperscript{x} mode is exited.

**Lemma 5.5:** At time \( T_O(\text{CLEAN}^{x}) \) and later, there are no \( \text{DATA}_{x} \) messages in the outgoing queues of any station.

**Proof:** Let \( T_E(\text{CLEAN}^{x}) \) be the time when CLEAN\textsuperscript{x} is entered. Only stations that are active at \( T_E(\text{CLEAN}^{x}) \) can possibly store \( \text{DATA}_{x} \) messages in their outgoing queues at \( T_E(\text{CLEAN}^{x}) \) and later. This is clear since inactive stations do not store any DATA messages in their outgoing queues, and no more \( \text{DATA}_{x} \) messages are ever generated after \( T_E(\text{CLEAN}^{x}) \). Thus, it is sufficient to concentrate only on stations that are active at \( T_E(\text{CLEAN}^{x}) \).

Lemma 5.3 implies that the identities of all stations that are active at \( T_E(\text{CLEAN}^{x}) \) are included in \( O_m \). Therefore, algorithm\textsuperscript{x} copies \( O_m \) into a special table \( V \) at \( T_E(\text{CLEAN}^{x}) \), and in \text{CLEAN}\textsuperscript{x} concentrates only on stations in \( V \).

Let \( J \) be a station in \( V \). Algorithm\textsuperscript{x} can conclude that \( J \) has cleared its outgoing queue of \( \text{DATA}_{x} \) messages by two possible ways: The first is either by observing that \( J \) is taken out of \( O_m \), and so \( J \) is inactive and has cleared its queues, or that a pair of \( (\text{POLL,PACK}) \) messages was exchanged between \( J \) and the manager, and so the \textit{PACK} message has cleared the outgoing queue of \( J \). The second way is to wait \( 4T \) seconds, a time period after which, as proved later, it is guaranteed that every \( J \) in \( V \) has cleared its queues.

The first way is obvious: If \( J \) becomes inactive, it has obviously cleared its queues of any DATA messages, including \( \text{DATA}_{x} \). If \( J \) remains active, algorithm\textsuperscript{x} can learn that \( J \) has cleared its outgoing queue of \( \text{DATA}_{x} \) messages by monitoring the \( (\text{POLL,PACK}) \) exchange between \( J \) and the manager. Recall that when \( J \) is active, it periodically exchanges \( (\text{POLL,PACK}) \) messages with the manager. A \textit{PACK} message sent from \( J \) clears its outgoing queue of messages generated before the \textit{PACK} message due to the FIFO discipline in the queues.
The other way is to wait 4T seconds. Notice that if \( J \) is active at \( T_E(CLEAN^x) \), then it is in PACK0 or PACK1. Assume \( J \) is in PACK0. The other possibility is symmetric. If while in PACK0, and after \( T_E(CLEAN^x) \), station \( J \) generates and sends a PACK0 message, then this message clears its outgoing queue of \( DATA_x \) messages. This happens within at most 2T seconds after \( T_E(CLEAN^x) \). However, if this is not the case, then it is not guaranteed that the outgoing queue of \( J \) does not contain \( DATA_x \) messages, and it is necessary to follow the later operation of \( J \). If \( J \) leaves PACK0 and moves to DEAD_STATE or REQUEST modes within 2T seconds after \( T_E(CLEAN^x) \), then it again clears its queues. The other possibility is that \( J \) moves to PACK1. If \( J \) generates and sends a PACK1 message when in PACK1, then again \( J \) has cleared its outgoing queue of \( DATA_x \) messages, this time within at most 4T seconds after \( T_E(CLEAN^x) \). If \( J \) does not send a PACK1 message when in PACK1, then lemma 5.2e implies that it leaves PACK1 after 2T seconds in the mode, and moves to DEAD_STATE or REQUEST modes. Again, \( J \) clears its queues within at most 4T seconds since \( T_E(CLEAN^x) \).

\[ \square \]

Remarks:

1. It is possible that \textit{algorithm} \( x \) will stay in \( CLEAN^x \) less than 4T seconds. If \( (POLL,PACK) \) messages are exchanged with every station \( J \) in \( V \), then \textit{algorithm} \( x \) will stay in \( CLEAN^x \) only 2T seconds at the most. In the worst case, \textit{algorithm} \( x \) enters \( CLEAN^x \) just after a successful exchange of \( (POLL,PACK) \) between any station \( J \) in \( V \) and the manager. \textit{Algorithm} \( x \) then waits for the next interval of \( (POLL,PACK) \) exchange and at its end leaves the mode. The stay in this case is at most 2T seconds.

2. It is possible that it will take for \textit{algorithm} \( x \) more than 4T seconds to have observed that every station \( J \) in \( V \) has left \( O_a \) or has sent a PACK message. An example for this case is when \textit{algorithm} \( x \) enters \( CLEAN^x \) immediately after \( J \) and the manager have exchanged \( (POLL.1PACK1) \) messages say. Assume that \( MA(J)=\gamma \). \textit{Algorithm} \( x \) will not send additional \( POLL.1\gamma \) messages until the T
seconds interval, in which it stays in POLLO mode, is elapsed. Then, algorithm moves to the POLLO mode and sends POLLO messages to J. If algorithm does not receive a PACK message in POLLO, it will move to WAIT_C mode, and when leaving WAIT_C, it will delete J from o. It now takes 5T seconds for algorithm to detect that J is taken out from o: T seconds stands for the stay of algorithm in the POLL1 mode, another T seconds stand for the stay in the POLLO mode and 3T seconds stand for the stay in the WAIT_C mode.

Remarks 1 and 2 explain the reason that the stay in CLEAN mode is taken as the minimum between between the two possibilities.

Proof of properties (A)-(C):

The proof that the protocol has properties (A)-(C) relies on lemmas 5.1-5.5. Let T be the time when algorithm leaves CLEAN. Observe that this is consistent with the notation in Sec. 4, since x leaves s at the time when algorithm leaves CLEAN. For convenience, we restate the properties:

Property (B): No DATA and DSTR messages are sent after T.

Proof: Lemma 5.4 implies that after T there are no DATA messages in the outgoing queues, and the property holds for DATA messages. Concerning the DSTR messages, notice that the only possible DSTR messages in this model are ADDR and NO_ADDR. Such messages are sent as a response to FIND messages, which can be sent by I only while I is active and SA(I)=x. At T (STOP_GENERATE), holds SA(I)≠x since lemma 5.2c implies that at this time I is in DEAD_STATE or REQUEST modes, and therefore had already released x. Thus, any FIND messages which was sent from I arrives at the media at T(STOP_GENERATE)+d at the latest. Notice that T(STOP_GENERATE)+d < T.
After \( T_1^n \), the manager sends an answer to \( \text{FIND} \) messages containing \( x \) as the address of the source station only if it has a station \( K \) in \( o_n \) with \( MA(K) = x \). In order to be included in \( o_n \) with \( MA(K) = x \), such a station \( K \) must exchange after \( T_1^n \) a pair of \((\text{ASSN}(K), \text{AACK}_n)\) messages with the manager. This pair of messages is generated and sent after \( T_1^n \) since at \( T_1^n \) there are no such messages in the system (lemma 5.2c). Thus, the pair of such messages will clear \( M-K \) of all the remaining \( \text{FIND} \) messages that were sent by \( I \), and these \( \text{FIND} \) messages will be received by the manager before the time when \( K \) is in \( o_n \) with \( MA(K) = x \). Thus, these \( \text{FIND} \) messages will not cause the manager to generate and send any \( \text{DSTR}_n \) messages, and the proof of the property is completed.

\[ \square \]

Property (A): \( \text{DATA}_s \) and \( \text{DSTR}_s \) messages are received only by station \( I \) and only when \( SA(I) = x_n \).

Proof: First we prove the property for the \( \text{DATA}_s \) messages. Only a station \( K \) that has \( SA(K) = x \) after \( T_1^n \) may receive \( \text{DATA}_s \) messages. A station \( K \) receives \( \text{DATA} \) messages after having received the first \( \text{POLL} \_I; \) message after setting \( SA(K) = x \). We next show that all \( \text{DATA}_s \) messages left in the system at \( T_1^n \), arrive at any such station \( K \) before the first \( \text{POLL} \_I; \) message, and thus are not received. These are the only problematic messages since after \( T_1^n \) no new \( \text{DATA}_s \) messages are generated.

Lemma 5.5 implies that at \( T_1^n \) and later, \( \text{DATA}_s \) messages may be only in the outbound channels, in the media, in the inbound channels and in the incoming queues. Before the time when \( K \) starts to receive \( \text{DATA} \) messages, it must receive a \( \text{POLL} \_I; \) message. This message is sent at least \( T \) seconds after the time when \( \text{algorithm}^{x_s} \) had entered the UNASSIGN mode, since \( \text{algorithm}^{x_{s+1}} \) stays \( T \) seconds in the \text{INIT_POLLO} mode. The maximal propagation delay in the outbound channels is \( d_{\text{out}} \) seconds, and \( T \) must be selected longer than that. Thus, at the time when the first \( \text{POLL} \_I; \) is sent, all \( \text{DATA}_s \) messages that were left in the outbound channels have already reached the media. It now follows from the FIFO discipline in the queues and the channels that all remaining \( \text{DATA}_s \) messages in the network arrive at \( K \) before the first \( \text{POLL} \_I; \) message and thus are not received.
Concerning the $DSTR_n^x$ messages, such messages that can possibly be received by $K$, can be only in $M-K$ at $T^n$. The $ASSN(K)$ message, by which the manager designates $x$ to $K$, is received by $K$ after clearing the media of all such $DSTR_n^x$ messages. Thus, these messages arrive at $K$ when $K$ is inactive and therefore are not received.

Finally, the proof of property (C) is immediate:

Property (C): Address $x_{n+1}$ becomes available for designation within finite time after $T^n$.

Proof: The only question is whether $algorithm^n x$ returns to the UNASSIGN mode. However, since the stay in all new modes is for a finite period, $algorithm^n x$ will indeed return to the UNASSIGN mode and $x$ is available for redesignation.

5.4 Other versions

The protocol of Sec. 5.2 is based on three main features:

1. Stations can be active and inactive. When active, a station is allowed to maintain sessions with other stations, while when inactive a station is allowed to communicate only with the manager.

2. Only addresses of stations that appear in a special table $O_n$, at the manager, are distributed to other stations. The protocol guarantees that active stations are included in $O_n$.

3. Addresses are redesignated only after guaranteeing that properties (A)-(C) are fulfilled.

In the protocol of Sec. 5.2, a station $I$ with designated address $x_n$ is included in $O_n$ when $algorithm^n x$ makes the transition from the INIT_POLLO mode to the POLLI mode, and is taken out of $O_n$ when $algorithm^n x$ leaves the WAIT_C mode. To ensure that $I$ is active when in $O_n$, station $I$ becomes active when its address algorithm makes the transition from the INIT_PACK0 mode to the PACK1 mode, and becomes inactive when the protocol returns to the DEAD_STATE mode. In the following we propose
four other possible versions, with no proof of correctness. The versions differ from each other, and from the protocol of Sec 5.2, in terms of the timing when \( I \) enters and leaves \( O_\alpha \) and when \( I \) becomes active. However, all versions are still based on the above three features.

Version 1:

The version of Sec. 5.2 ensures that when \( I \) is taken out of \( O_\alpha \), i.e. when \( \text{algorithm}^x \) leaves \( \text{WAIT}_C \), station \( I \) is already inactive and had unassigned \( x_n \). However, station \( I \) can still be active when \( \text{algorithm}^x \) enters \( \text{WAIT}_C \). Notice that every station that asks for the address of \( I \) after \( \text{algorithm}^x \) enters \( \text{WAIT}_C \), cannot maintain a \( \text{session}^x \) with \( I \) for longer than \( 3T \) seconds, until \( \text{algorithm}^x \) leaves \( \text{WAIT}_C \). Therefore, it may make sense to stop the distribution of \( x_n \) at the time when \( \text{algorithm}^x \) enters \( \text{WAIT}_C \), thereby disallowing initiation of such \( \text{sessions}^x \). This can be achieved by taking \( I \) out of \( O_\alpha \) when \( \text{algorithm}^x \) enters \( \text{WAIT}_C \) (instead of when \( \text{algorithm}^x \) leaves \( \text{WAIT}_C \)).

However, such a solution leads to the possibility that an active station does not appear in \( O_\alpha \), and therefore it is possible that \( V \) will not contain all stations that are active at the time when an address assignment algorithm enters into CLEAN mode. This in turn implies that when CLEAN is exited, not all active stations have necessarily cleared their outgoing queues of \( \text{DATA}^x \) messages. This in turn may cause property (A) not to hold.

The solution for this problem is simply to require \( \text{algorithm}^x \) to stay in CLEAN mode for \( 4T \) seconds. With this change, \( V \) is in fact unnecessary, and lemma 5.4 and therefore property (A) still hold.

Version 2:

The protocol of this version is depicted in Fig. 5.3. The station algorithm is similar to the original one \([GS2]\), with station \( I \) becoming active with the transition of its address algorithm from the AACK mode to the PACK0 mode, and it becomes inactive when the algorithm returns to the DEAD_STATE mode (see Fig. 5.3a). On the other hand, there are some changes in the manager algorithm (see Fig. 5.3b), comparing to the original one \([GS2]\). The CLEAR1 mode is separated into two modes, CLEAR1_A and
CLEAR1_B. In addition, there are five new modes: WAIT_D, CLEAR2, STOP_GENERATE, CLEAN and WAIT_E. When algorithm leaves WAIT_C, it enters into WAIT_D, where it waits \( d_{out} \) seconds and then it moves to CLEAR2. In CLEAR2 algorithm sends CLR2 messages until it receives one message back. These messages clear the media similarly to the CLR0 and CLR1 messages [GS2]. Then, and also after leaving CLEAR1_B, algorithm moves to STOP_GENERATE mode, where it stays \( \text{MAX}(2T+C, 5T+d_{in}) \) seconds. The STOP_GENERATE mode has the same purpose as in the version of Sec. 5.2: after STOP_GENERATE is exited, no more \( DATA_n \) messages are generated.

Later, algorithm enters into CLEAN mode, which is similar and has the same purpose as in the version of Sec. 5.2: after CLEAN is exited, there are no \( DATA_n \) messages in the outgoing queues. Later, before returning to UNASSIGN, algorithm enters into WAIT_E where it stays \( d_{out} \) seconds. When WAIT_E is exited, the outbound channels are also clear of \( DATA_n \) messages.

Finally, \( I \) is included in \( \alpha_n \) when algorithm makes the transition from the PENDING mode to INIT_POLL0 mode, and it is taken out when algorithm either leaves WAIT_B or WAIT_C modes. With these changes it can be shown that when \( I \) is active, it is in \( \alpha_n \) as in the version of Sec. 5.2.
Figure 5.3b - Algorithm\textsuperscript{a} at the address manager
We now explain the changes. As in the version of Sec. 5.2, we must ensure that after $x_n$ is undesignated, no more $DATA_{x_n}$ messages are generated. Therefore, we need the STOP_GENERATE mode so that after the time when it is exited, no more $DATA_{x_n}$ messages are generated. However, in this protocol we cannot allow \textit{algorithm} to enter STOP_GENERATE immediately after leaving \textit{WAIT}_C, as in the case in the version of Sec. 5.2, because of the following reason: Let $T_0(WAIT_C)$ be the time when \textit{algorithm} leaves \textit{WAIT}_C, and consider at this time a station $J$ with $MA(J)=y$. We do not prove it here, but a basic property of the original protocol implies that at $T_0(WAIT_C)$ it is possible that there are $ASSN(J)$ messages in the media, followed by $POLL_0$ messages that are intended to $J$. It is also possible that these two kinds of messages are followed, in the media queue, by a $DATA$ message from $I$, which was sent when $SA(I)=x_n$ held. These messages can stay in the media queue for unlimited time period. Also, at some time after $T_0(WAIT_C)$, station $J$ can enter REQUEST mode, receive the $ASSN(J)$ message and move to the AACK mode. Then $J$ can receive the $POLL_0$ message and move to PACK0 while becoming active. This scenario is depicted in Figure 5.4.

Figure 5.4 - \textit{ASSN} and \textit{POLL0} messages stay in the media for unbounded time period
It is therefore possible that $J$ will receive the DATA message from $I$ and will initiate a session$^*$. Algorithm$^*$ must wait in STOP_GENERATE for the termination of this session, but it is impossible to limit the time when $J$ becomes active in the above scenario.

However, in order to offer a solution, we use another property of the original protocol, again with no proof. Notice that in the above scenario, the ASSN1($J$) messages are followed by POLL0$^*_1$ messages. The only message exchange between $J$ and algorithm$^*$ that can lead to such a situation is as follows: algorithm$^*$ has sent ASSN1($J$) messages to $J$, which were received by $J$ while staying in REQUEST mode. Station $J$ has then sent AACK$^*_2$ messages, that were received by algorithm$^*$ in the PENDING mode and has caused its transition to INIT_POLLO. Later, algorithm$^*$ has sent POLL0$^*_2$ messages to $J$ from the INIT_POLLO mode, but it had not received a PACK0$^*_2$ message while in INIT_POLLO. Consequently, algorithm$^*$ has moved to INIT_CLEAR0. At this stage we use the property that algorithm$^*$ will not return to UNASSIGN mode before clearing the M-J of all the ASSN1($J$) messages, and so algorithm$^*$ does not send POLL1$^*_2$ messages until returning to UNASSIGN either. Consequently, even if $J$ receives in the above scenario the ASSN1($J$) and the POLL0$^*_2$ messages in REQUEST and AACK modes respectively, and it moves to PACK0, it will not receive a POLL1$^*_2$ message while in PACK0, and will return to DEAD_STATE. This will cause $J$ to terminate every session$^*$ it maintains, and again with no proof, it is guaranteed that $J$ will never initiate sessions$^*$ again. Notice that using this property, within $5T+d_{in}$ seconds after the time when the last DATA message from $I$ leaves the media, every session$^*$ in the above scenario must be terminated by $J$. The $d_{in}$ seconds stand for the longest time it can take the DATA message to arrive at $J$ from the media, and the $5T$ seconds stand for the total time period that it can take $J$ since entering the REQUEST mode and until it becomes inactive. The $5T$ seconds contain a stay of $T$ seconds in REQUEST mode together with $2T$ seconds in each of the AACK and PACK0 modes.

Our solution first suggests to clear the media of all the DATA messages sent by $I$ when $SA(l)=x_n$ held. To achieve this task it is first necessary to guarantee that the outbound channel from $I$ is also clear of such messages, since the DATA messages from $I$ arrives at the media from the outbound channel from $I$. Using again a basic property of the original protocol, it can be shown that after the time when algorithm$^*$ leaves the WAIT_C mode, or after staying for $5T+d_{in}$ seconds in WAIT_B, station $I$ is
already not active with $SA(I)=\tau_n$. Thus, $d_{\text{out}}$ seconds later, the outbound channel of $I$ is clear of $DATA$ messages. These $d_{\text{out}}$ seconds are either contained in the $\text{WAIT}_B$ mode (in addition to the $5T+d_{\text{in}}$ seconds) or in the $\text{WAIT}_D$ mode.

We now clear the media of all $DATA$ messages from $I$. If $algorithm^{*}$ enters $\text{STOP\_GENERATE}$ from $\text{CLEAR}_1\_B$, then this clearing is achieved by the $CLR_1$ messages of the $\text{CLEAR}_1\_B$ mode (see [GS2]). However, if $algorithm^{*}$ enters into $\text{STOP\_GENERATE}$ after being in $\text{WAIT}_C$, we must perform the clearing by special messages. Therefore, an additional mode, $\text{CLEAR}_2$, is introduced after $\text{WAIT}_D$ and before $\text{STOP\_GENERATE}$, in which $algorithm^{*}$ sends $CLR_2$ messages, until one is received back, thereby clearing the media.

After completing the clearing of the media, $algorithm^{*}$ can enter $\text{STOP\_GENERATE}$. Recall that at this stage it is guaranteed that the $session^{*}$ in the scenario above is terminated within $5T+d_{\text{in}}$ seconds. However, $5T+d_{\text{in}}$ seconds are not sufficient for the stay in $\text{STOP\_GENERATE}$. It can be proved that $algorithm^{*}$ must also wait $\text{MAX}(2T+C,ST)$ seconds, in order to ensure the termination of $sessions^{*}$ as in the version of Sec. 5.2. Thus, $algorithm^{*}$ must wait in $\text{STOP\_GENERATE}$ a time period of $\text{MAX}(2T+C,ST+d_{\text{in}})$ seconds, which are $\text{MAX}(2T+C,ST+d_{\text{in}})$ seconds.

After ensuring that no more $DATA_{x_n}$ messages are generated, $algorithm^{*}$ must ensure that the outgoing queues are clear of $DATA_{x_n}$ messages. This can be achieved by a $\text{CLEAN}$ mode similar to the one of the version of Sec. 5.2.

At this stage there is one problem left: $algorithm^{*}$ must also ensure that before $\tau_{n+1}$ is designated, all outbound channels are also clear of $DATA_{x_n}$ messages. It can be proved that a station $K$ that will adopt $\tau_{n+1}$, can become active within less than $d_{\text{out}}$ seconds after $algorithm^{*+1}$ leaves the $\text{UNASSIGN}$ mode. Thus, if at the time when $\text{UNASSIGN}$ is exited, there are still $DATA_{x_n}$ messages in the outbound channels, these messages can arrive at the media when $K$ is active, and so can be received by $K$. Therefore, after leaving $\text{CLEAN}^{*}$, but before entering $\text{UNASSIGN}$, $algorithm^{*}$ enters into $\text{WAIT}_E$ where it stays $d_{\text{out}}$ seconds, thereby clearing the outbound channels of $DATA_{x_n}$ messages. At this stage $algorithm^{*}$ enters
into UNASSIGN, while ensuring properties (A)-(C). The proof of these properties is now similar to the proof in the original version.

Notice that if algorithm leaves WAIT_A, it moves to the CLEAR1_A mode and returns to UNASSIGN without traversing any of the new modes. It can be proved that if algorithm reaches WAIT_A, station I never becomes active with SA(I)=x_n, and also x_n is never distributed as the address of I. Thus, no sessions are ever initiated, and there is no need for the new modes.

Version 3:

The protocol of this version is depicted in Fig. 5.5.

Figure 5.5a - The algorithm of station I

In this version station I becomes active when its address protocol enters AACK, and becomes inactive when the protocol returns to DEAD_STATE (see Figure 5.5a).

The manager algorithm is similar to version 2, except that the CLEAR1 mode is not split, and remains as in [GS2]. This ensures that algorithm always traverse STOP_GENERATE and CLEAN before undesignating x_n. This is necessary since station I is now included into O_n when algorithm enters PENDING, and so it is possible that x_n will be distributed as the address of I while algorithm is in
Figure 5.5b - Algorithm at the address manager
PENDING, CLEAR0 or WAIT_A. Sessions can therefore be initiated when algorithm is in one of these modes. Finally, in order to ensure that every active station is in \( O_n \), station \( I \) is taken out from \( O_n \) when algorithm leaves WAIT_C, or when algorithm enters CLEAR1 (see Figure 5.5b).

In version 2 we have described a scenario where a station \( J \) may initiate a session unlimited time after the transition of algorithm out of WAIT_C. This was possible due to ASSN(J) messages, followed by POLLO messages and DATA messages from \( I \), that can stay in the media queue for an unlimited time period before being received by \( J \). We mentioned in version 2 that \( ST+d \) seconds after the time when the last DATA message from \( I \) leaves the media, it is guaranteed that the session in the scenario is terminated. This scenario is also possible here, but here a session may be initiated by station \( J \) even if only \( ASSN(J) \) messages, followed by DATA messages from \( I \), are in the media. This is because in this version a station becomes active after receiving an \( ASSN \) message. It can be proved that in such a scenario it takes at most \( 3T+d \) seconds from the time when the last DATA message from \( I \) leaves the media until the session is terminated at \( J \). Since we have to handle both scenarios, the stay in STOP_GENERATE is determined according to the first one, namely \( ST+d \). Hence the protocol has in fact the same new modes as in version 2.

Version 4:

In last version we leave the protocol of [GS2] unchanged, and use a completely different idea in order to ensure properties (A)-(C). In this version we divide \( A_{\text{temp}} \), which is the set of all addresses at the manager, into groups of \( N \). As shown later, the value \( N \) is selected as a function of \( d \), \( d_n \), \( T \) and \( C \), but at this time we only mention that \( N \geq 5 \). Consider the addresses in a given group and denote them by \( x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(N)} \). Notice that here \( x^{(1)}, x^{(2)} \) etc. do not denote different assignments of address \( x \) but denote different addresses in \( A_{\text{temp}} \). Then we decide that \( x^{(1)} \) is designated first (\( x^{(1)} \) is included in \( A_n \)) and while \( x^{(1)} \) is in \( A_n \), addresses \( x^{(2)}, x^{(3)}, \ldots, x^{(N)} \) cannot be designated. After \( x^{(1)} \) is taken out of \( A_n \), address \( x^{(2)} \) is designated and the others, i.e. \( x^{(1)}, x^{(3)}, x^{(4)}, \ldots, x^{(N)} \) cannot be designated. Always, when one of the addresses in a group is designated, the other \( N-1 \) cannot be designated, and the order of
designation remains the same.

Assume that \( x^{(1)} \) is designated to \( I \). Notice that the protocol of [GS2] is equivalent to the following:

Station \( I \) joins \( o_m \) when \( x^{(1)} \) is designated to \( I \), i.e. when \( \text{algorithm}^{(1)} \) makes the transition from UNASSIGN to the PENDING mode, station \( I \) leaves \( o_m \) when \( x^{(1)} \) is undesignated, i.e. at the time when \( \text{algorithm}^{(1)} \) returns to UNASSIGN, station \( I \) becomes active when its address algorithm enters AACK and becomes inactive when the algorithm returns to DEAD_STATE.

This situation is similar to version 3. However, in version 3 we need the WAIT_D, CLEAR2, STOP_GENERATE, CLEAN and WAIT_E modes in order to ensure properties (A)-(C). Here we leave the protocol of [GS2] unchanged and accomplish the tasks of those modes by segmenting \( \mathcal{A}_{comp} \).

Notice that the additions in version 3 require \( \text{algorithm}^{(1)} \) to delay the redesignation of \( x^{(1)} \) as follows: the WAIT_D mode enforces a delay of \( d_{\text{out}} \) seconds, later, the media is cleared by CLR2 messages, and then \( \text{algorithm}^{(1)} \) stays MAX\((2T+C,5T+d_{\text{in}})\) seconds in STOP_GENERATE mode, 4T seconds in CLEAN mode and \( d_{\text{out}} \) seconds in the WAIT_E mode (notice that the CLEAN mode can also be exited after every station \( J \) in \( V \) has exchanged \( \text{POLL} \),PACK messages with the manager or left \( o_m \). We drop this possibility in the discussion here, since it complicates the implementation).

We now show why the division into groups of \( N \) accomplishes the same tasks. First we mention that after an address assignment algorithm is initiated, the algorithm is exited in no less than \( 4T+d_{\text{in}}+d_{\text{out}} \) seconds: T seconds in PENDING mode and \( 3T+d_{\text{in}}+d_{\text{out}} \) seconds in WAIT_A mode.

Assume now that \( x^{(1)} \) is undesignated. The initial necessary delay of \( d_{\text{out}} \) seconds is achieved by \( \text{algorithm}^{(2)} \) (in fact a much longer delay, since as shown before, the stay in \( \text{algorithm}^{(2)} \) is at least \( 4T+d_{\text{in}}+d_{\text{out}} \)). The clearing of the media is achieved by \( \text{algorithm}^{(3)} \): if \( \text{algorithm}^{(3)} \) exchanges \( \text{ASSN} \),AACK) messages with a station, then these messages clear the media. Otherwise, \( \text{algorithm}^{(3)} \) moves to the CLEAR0 mode and the CLR0 messages clear the media. Notice that in fact \( \text{algorithm}^{(2)} \) also clears the media in the same fashion. However, it can be shown that it is possible that this clearing takes place during the first \( d_{\text{out}} \) seconds after the undesignation of \( x^{(1)} \), and it is therefore not useful.
The final delay must be of at least \(\max(2T+C,5T+d_{in}) + 4T + d_{out}\) seconds. A time period of \(3T + d_{out} + d_{in}\) seconds is obtained from the WAIT_A mode of \(\text{algorithm}^{(3)}\), if it traverse this mode. If it does not, it can be shown that the time period we obtain from \(\text{algorithm}^{(3)}\) is even longer. Another \((N-3)(4T + d_{out} + d_{in})\) seconds are obtained from \(\text{algorithm}^{(4)}\), \ldots, \(\text{algorithm}^{(N)}\), and therefore we get a total time period of \(3T + d_{out} + d_{in} + (N-3)(4T + d_{out} + d_{in})\) seconds. We now must ensure that \(3T + d_{out} + d_{in} + (N-3)(4T + d_{out} + d_{in}) \geq \max(2T+C,5T+d_{in}) + 4T + d_{out}\). If \(\max(2T+C,5T+d_{in}) = 5T+d_{in}\) then it can be easily seen that \(N=5\) is sufficient. Otherwise, \(N\) is selected to satisfy \(3T + d_{out} + d_{in} + (N-3)(4T + d_{out} + d_{in}) \geq 6T + C + d_{out}\). It is worthwhile to set \(N\) as the minimum value that fulfills the inequality, since although we do not change the original protocol, we reduce the number of available addresses in the manager to be \(\frac{1}{N}\) of the size of \(A_{comp}\), and this reduces the maximal number of stations that can be connected to the network at any given time.

6. MODEL B - ADDRESS DISTRIBUTION WITH MEMORY

6.1 The model

In model B stations maintain address tables, where they store addresses of other stations after session termination. However, a station is still required to ask the manager for the address of another station in order to be able to initiate the first session with it. In this model there is no need to request the address of a station every time a session with it is initiated. Here the address is requested and stored before initiating the first session, and if the address does not change, the stored address is used when initiating subsequent sessions. The use of address tables therefore reduces network traffic and load on the manager and on the stations. On the other hand, whenever a station leaves the network, the manager must update the address tables of all active stations. This ensures that stations do not appear in address tables of other stations with addresses that do not belong to them any more.

In this model, stations ask for the addresses of other stations in the same fashion as in model A. The stations initiate sessions either by using received DATA or ADDR messages, or by consulting the address tables. Stations manage sessions using a timeout mechanism as in model A.
With this model we must address the three problems (i)-(iii) of Sec. 1, and in addition must make sure that the address tables are properly updated.

6.2 The Protocol

The protocol for this model is depicted in Figure 6.1, where the additions to the original protocol [GS2] are emphasized in boldface. The protocol is similar to the one of model A. However, it requires some changes.

![Figure 6.1a - The address assignment algorithm of station I](image)

The algorithm for the stations (see Figure 6.1a) is similar to the algorithm of model A, except that when active, a station keeps addresses in an address table. When inactive, the address table is empty.

The algorithm at the manager is similar to the one of model A. However, the STOP_GENERATE mode is replaced by WAIT_D and the CLEAN mode is modified (see Figure 6.1b). These changes are
necessary because of the existence of address tables. In model A it was guaranteed that after the STOP_GENERATE* mode is exited, no more DATA* messages are generated since no more sessions* are maintained. The ideas behind the STOP_GENERATE mode were to use the timeout mechanism,
which is used for managing sessions, and the fact that a station is required to ask or receive the address of any other station, everytime before initiating a session with it (see proof of lemma 5.4).

However, in this model stations may also initiate sessions after consulting their address tables, and are not required to ask for an address every time before session initiation. Thus, in order to terminate all sessions, it is required to clear all address tables of \( x_n \). This task must be performed by specific messages which algorithm sends to all active stations. We denote these messages by \( CLEAN(x_n) \). The \( CLEAN(x_n) \) messages are sent by algorithm from the \( CLEAN^* \) mode (see Figure 6.1b) and we later specify how these messages are sent and being acknowledged. When a station receives a \( CLEAN(x_n) \) message, it clears its address table of \( x_n \) and terminates every \( session^* \).

However, clearing the address tables of \( x_n \) is not sufficient. It must be guaranteed that stations will not include \( x_n \) at their tables, after receiving \( CLEAN(x_n) \) messages. Stations can include \( x_n \) at the address tables either after receiving \( ADDR(I,x_n) \) messages from the manager, or after receiving \( DATA \) messages from station \( I \) itself. After the time when the \( WAIT_C^* \) mode is exited, no more \( ADDR(I,x_n) \) messages are generated, since \( I \) is not included in \( O_n \) with \( MA(I)=x_n \) any more. Thus, every \( CLEAN(x_n) \) message that is sent to a station \( J \), is sent after all the \( ADDR(I,x_n) \) messages.

The \( WAIT_D^* \) mode ensures this property for the \( DATA \) messages sent from \( I \) when \( SA(I)=x_n \) holds. We prove later that after the time when the \( WAIT_D^* \) mode is entered, \( I \) is not active with \( SA(I)=x_n \) anymore. Thus, \( d_{out} \) seconds later all \( DATA \) messages it could have sent when \( SA(I)=x_n \), will arrive at the media. Therefore, before sending \( CLEAN(x_n) \) messages, algorithm waits \( d_{out} \) seconds in \( WAIT_D^* \) mode. Then algorithm moves to the \( CLEAN^* \) mode and sends \( CLEAN(x_n) \) messages. These messages will arrive at the stations after all \( DATA \) messages from \( I \), due to the FIFO discipline in the queues and channels.

The \( CLEAN^* \) mode guarantees that after the time when it is exited, no more \( DATA_{x_n} \) messages are generated and the outgoing queues at the stations are clear of such messages. This situation guarantees properties (A) and (B) as in model A.
Let \( T_E(\text{CLEAN}^z) \) be the time when \( \text{CLEAN}^z \) is entered. At \( T_E(\text{CLEAN}^z) \), address \( x_n \) can be included only in the address tables of active stations, and only such stations can have sessions \(^z\). As we prove later, it is enough to send \( \text{CLEAN}(x_n) \) messages only to the stations that are active at \( T_E(\text{CLEAN}^z) \), in order to guarantee Properties (A) and (B). To detect which stations are active at \( T_E(\text{CLEAN}^z) \), \textit{algorithm} \(^z\) uses table \( \mathcal{O}_n \) again, as in model A. The table \( \mathcal{O}_n \) is used in this protocol exactly as in model A, and it contains the identities of all active stations. Therefore, \textit{algorithm} \(^z\) samples \( \mathcal{O}_n \) into a special table \( V \) at \( T_E(\text{CLEAN}^z) \), and concentrates only on stations in \( V \). \textit{Algorithm} \(^z\) sends \( \text{CLEAN}(x_n) \) messages to every station \( J \) in \( V \) and if \( J \) receives a \( \text{CLEAN}(x_n) \) message, it terminates any sessions \(^z\) it is participating in, clears its address table of \( x_n \) and acknowledges the reception of the \( \text{CLEAN}(x_n) \) message by sending \( \text{CACK}(x_n) \) messages to the manager.

If \( J \) does not receive \( \text{CLEAN}(x_n) \) messages, or the manager does not receive a \( \text{CACK}(x) \) message from \( J \), then as we prove later, \( J \) becomes inactive and is deleted from \( \mathcal{O}_n \). In this case, as in the case where \( J \) is not included in \( V \) at \( T_E(\text{CLEAN}^z) \), it is also guaranteed that \( J \) will never initiate sessions \(^z\) after \( \text{CLEAN}^z \) is exited. Notice that in order to become active again and initiate sessions, \( J \) must first receive a \( \text{POLL}1 \) message, which will be sent after the time when \( J \) is included again in \( \mathcal{O}_n \), i.e. after \( T_E(\text{CLEAN}^z) \). This \( \text{POLL}1 \) message will clear M-J of all \( \text{DATA} \) messages from \( I \) (which were sent when \( \text{SA}(I)=x_n \) holds) and of \( \text{ADDR}(I,x_n) \) messages. Thus, these messages will arrive at \( J \) when inactive, and therefore will not be received. Therefore, \( J \) will never store \( x_n \) in its address table and will never initiate sessions \(^z\) again.

Notice that after the time when \( \text{CLEAN}^z \) is exited, all outgoing queues are clear of \( \text{DATA}_x \) messages. If a station \( J \) in \( V \) sends a \( \text{CACK}(x_n) \) message, then this message clears the outgoing queue. If \( J \) becomes inactive, or if it already inactive at \( T_E(\text{CLEAN}^z) \), then its outgoing queue is clear of \( \text{DATA}_x \) messages, a situation that will not change since \( J \) will not initiate sessions \(^z\).

We now explain why the \( \text{CLEAN} \) and \( \text{CACK} \) messages are a necessary part of the manager algorithm, but do not appear in the station algorithm. In our model, the manager has two tasks: first it
designates addresses to the stations, and second it is responsible to create in the system the necessary conditions to enable redesignation of addresses. For instance, in [GS1] it is proved that in order to ensure the main properties of the address assignment protocol, it is first necessary, before redesignating an address $x$, to clear the entire system of $ASSN^2$ messages. Therefore, the original manager algorithm [GS2] is basically composed of two kinds of modes and messages: the first type includes the modes and messages by which an address is designated. This type includes the PENDING, INIT_POLLO, POLLO and POLLI modes, together with the $ASSN$, $POLLO$ and $POLLI$ messages. The second type includes the modes and messages that accomplish the condition we stated above. This type includes the CLEAR0, INIT_CLEAR0, WAIT_A, WAIT_B, CLEAR1 and the WAIT_C modes, together with the $CLR0$ and the $CLR1$ messages. Notice that although the second type is not involved directly in the designation process, it is necessary for the redesignation of addresses and so it is a part of the manager algorithm.

On the other hand, notice that stations in the model do not have any system control tasks. Therefore, the station algorithm is designed only to enable stations to adopt and keep addresses. This is the reason why the $CLR0$ and the $CLR1$ messages, for instance, are not a part of the station algorithm.

The same situation holds for the $(CLEAN,CACK)$ messages. We explained before that in order to ensure properties (A)-(C), it is first necessary, before redesignating $x$, to clear the address tables of $x_n$. Therefore, the CLEAN mode appears in the manager algorithm. On the other hand, the stations have no responsibility for this task. The CLEAN messages only affect the address tables, and separate from the station algorithm that only enables stations to adopt and keep addresses. Therefore, as in the case of the $CLR0$ and the $CLR1$ messages before, the CLEAN and the CACK messages do not appear in the stations algorithm.

The CLEAN and CACK messages are attached to the POLL and PACK messages that the stations exchange with the manager. As shown later, this guarantees that either the CLEAN messages are received by a station $J$ in $V$ within finite, bounded time; or that $J$ becomes inactive. However, we also allow additional CLEAN and CACK messages, if at the time they are intended for sending, no $(POLL,PACK)$ is sent by the address assignment protocol. If they arrive, such messages accomplish the clearing of the tables faster.
6.3 Correctness proof

We now prove that the protocol has properties (A)-(C). The proof relies on seven lemmas: Lemma 6.1 shows that the additions to the original protocol do not affect its four main properties. Thus, the address assignment is correct. Lemma 6.2 shows that an active station always appears in $\mathcal{O}_n$. Lemma 6.3 shows that when algorithm $^x$ leaves WAIT_D $^x$ mode, the outbound channel from $l$ is clear of DATA and CLEAN messages.

Lemma 6.4 shows that when in CLEAN $^x$, algorithm $^x$ does not receive CACK($x$) messages from previous clearings of $x$ and lemma 6.5 shows that algorithm $^x$ stays for at most ST seconds in the CLEAN $^x$ mode. Finally, lemmas 6.6 and 6.7 show that after algorithm $^x$ leaves the CLEAN $^x$ mode no sessions $^x$ are ever initiated, $x_n$ is not included in the address table of any station $J$ and the outgoing queues are clear of DATA $^x$ messages. Using the seven lemmas, we later show that the protocol has properties (A)-(C).

Lemma 6.1: The main properties - Uniqness, Assignment and Consistency of the original protocol, hold for the protocol for model B.

Proof: The proof is similar to the one of model A. Notice that the (CLEAN,CACK) messages only update the address tables and do not affect the process of address designation.

As before, lemma 6.1 is important to guarantee correct address assignment. Also, lemma 6.1 implies that the protocol for model B has the basic properties of lemma 5.2.

Lemma 6.2: Station $l$ in active state (OP($l$)=TRUE) appears in $\mathcal{O}_n$.

Proof: Similar to model A.

Lemmas 6.3 - 6.7 prove that the WAIT_D $^x$ and CLEAN $^x$ modes fulfill their task.

Lemma 6.3: When algorithm $^x$ leaves the WAIT_D $^x$ mode, the outbound channel from $l$ is clear of
DATA and CLEAN messages.

Proof: From lemma 6.2 follows that the last time when \( I \) can still send DATA messages while active with \( SA(I) = x_n \), is when \( \text{algorithm}^* \) leaves the WAIT_C mode. This is because after this time, \( I \) is not included in \( \mathcal{O}_n \) with \( MA(I) = x_n \). The maximal propagation delay in the outbound channels is \( d_{out} \). Thus, when WAIT_D \( \) is exited, the lemma holds for DATA messages. Notice that since inactive stations not receive CLEAN messages, the lemma also holds for CACK messages.

\[ \square \]

Lemmas 6.4 and 6.5 define two properties of the CLEAN* mode. The lemmas show that no \( CACK(x_l) \) messages, for any \( l < n \) are received as an acknowledgement to \( CLEAN(x_n) \) messages, and that the stay in the CLEAN* mode is bounded.

Let \( T_E(CLEAN^*) \) and \( T_O(CLEAN^*) \) be the times when \( \text{algorithm}^* \) enters and leaves the CLEAN* mode respectively. Assume that station \( J \) is included in \( \mathcal{O}_n \) at \( T_E(CLEAN^*) \). Since \( J \) is included in \( \mathcal{O}_n \), holds \( MA(J) \neq \text{nil} \). Assume \( MA(J) = y \). According to the protocol, \( \text{algorithm}^* \) sends \( CLEAN_j(x_n) \) messages to \( J \), and waits for the first of the two events: either a \( CACK_k^*(x) \) message is received, or \( J \) is taken out of \( \mathcal{O}_n \). Assume that a \( CACK_k^*(x) \) message is received, i.e. \( J \) is still included in \( \mathcal{O}_n \) at the time of reception.

Lemma 6.4: \( \text{Algorithm}^* \) will not receive a \( CACK_k^*(x_l) \) message with \( l < n \) as an acknowledgement to a \( CLEAN_j(x_n) \) message.

Notice: The lemma says that if \( \text{algorithm}^* \) receives a \( CACK_k^*(x) \) message, then this message has been sent as a response to a \( CLEAN_j(x_n) \) message, i.e., it is guaranteed that \( J \) has cleared its address table of \( x_n \) and that \( \text{algorithm}^* \) does not receive a \( CACK_k^*(x) \) message that is left in the network from previous assignments of \( x \).

Proof: Let \( T_E(\text{PENDING}) \) be the time when \( y \) is designated to \( J \). At \( T_E(\text{PENDING}) \), all outbound channels are clear of \( CACK_k^*(x) \) messages. This is clear since \( y \) is designated to \( J \) after it was undesignated from
every station to which it was designated before. The WAIT_D mode, which is also added to algorithm\textsuperscript{y}, ensures that after every undesignation of \( y \), the outbound channel from the station to which \( y \) was designated is clear of CACK\( _y \) messages (lemma 6.3).

While \( y \) is designated to \( J \), i.e. \( MA(J)=y \), \( J \) is the only station that has SA\( (J)=y \) (lemma 5.2a). Thus, any new CACK\( _y \)\( (x) \) messages that arrives at M-J after \( T_E(PENDING) \) and while \( MA(J)=y \) are generated and sent by \( J \).

We assume that algorithm\textsuperscript{x} sends CLEAN\( _y \)(\( x_a \)) messages to \( J \), and that it receives a CACK\( _y \)(\( x \)) message while \( J \) is still included in \( \mathcal{O}_n \) with \( MA(J)=y \). Thus, if we can prove that at \( T_E(CLEAN\textsuperscript{x}) \) the M-J-M is clear of CLEAN\( _y \)(\( x \)) and CACK\( _y \)(\( x \)) messages, then we also prove the lemma. This is because it is clear that the received CACK\( _y \)(\( x \)) message has been generated by \( J \), and since the only CLEAN\( (x_a) \) messages with destination station \( y \) that can be in M-J after \( T_E(CLEAN\textsuperscript{x}) \) and until the CACK\( _y \)(\( x \)) is received are CLEAN\( _y \)(\( x_a \)) messages, then it is clear that the CACK\( _y \)(\( x \)) message is a CACK\( _y \)(\( x_a \)) message.

Notice that after \( T_E(PENDING) \) and until \( J \) is included in \( \mathcal{O}_n \), station \( J \) and algorithm\textsuperscript{y} exchange (POLL\( 0,PACK\,0 \)) messages. When no station is included in \( \mathcal{O}_n \) with designated address \( y \), no CLEAN\( _y \)(\( x \)) messages are sent by algorithm\textsuperscript{x}. Therefore, the POLL\( 0 \) and the PACK\( 0 \) messages clear M-J-M of all previous CLEAN\( _y \)(\( x \)) and CACK\( _y \)(\( x \)) messages. Thus, if no CLEAN\( _y \)(\( x \)) messages are sent to \( J \) before \( T_E(CLEAN\textsuperscript{x}) \), the claim obviously holds. Therefore, let \( x_p \) be the last address for which CLEAN\( _y \)(\( x \)) messages were sent to \( J \). Let \( T_E(CLEAN\textsuperscript{p}) \) and \( T_O(CLEAN\textsuperscript{p}) \) be the times when algorithm\textsuperscript{p} entered and left CLEAN\textsuperscript{p} respectively, as depicted in Figure 6.2.

Notice that \([T_O(CLEAN\textsuperscript{p}),T_E(CLEAN\textsuperscript{x})]\) is at least 5T seconds, since before entering into CLEAN\textsuperscript{x}, algorithm\textsuperscript{x} must have stayed T seconds in INIT_POLL\( 0\textsuperscript{x} \), T seconds in POLL\( 1\textsuperscript{x} \) and 3T seconds in WAIT\_C\textsuperscript{x}. Station \( J \) is in \( \mathcal{O}_n \) during the entire interval \([T_O(CLEAN\textsuperscript{p}),T_E(CLEAN\textsuperscript{x})]\), i.e. it stays either in POLL\( 0 \), POLL\( 1 \) or WAIT\_C modes. The stay in the WAIT\_C mode is only 3T seconds. Thus, it is clear that algorithm\textsuperscript{y} has stayed in \([T_O(CLEAN\textsuperscript{p}),T_E(CLEAN\textsuperscript{x})]\) at least 2T seconds in the POLL\( 0 \) and POLL\( 1 \) modes. The stay in each of these modes is for T seconds. Therefore, algorithm\textsuperscript{p} and \( J \) have exchanged a pair of (POLL, PACK) messages in \([T_O(CLEAN\textsuperscript{p}),T_E(CLEAN\textsuperscript{x})]\).
At least 5T seconds.

Figure 6.2 - Entrances of algorithm* into CLEAN mode

Recall that J is the only station with designated address y in \([T_E(\text{CLEAN}^y), T_E(\text{CLEAN}^x)])]. Thus, it is the only station that could have generated \(\text{CACK}^y(x_p)\) messages. After \(T_O(\text{CLEAN}^y)\), no more \(\text{CLEAN}_j(x_p)\) messages are generated, and therefore the pair of \((\text{POLL}, \text{PACK})\) messages has cleared M-J-M of all \(\text{CLEAN}_j(x_p)\) and \(\text{CACK}^y(x_p)\) messages.

\[ \square \]

**Lemma 6.5:** Algorithm* stays for at most 5T seconds in the CLEAN* mode.

**Proof:** Consider station J again, which is sampled into \(V\) with \(\text{MA}(J)=y\). The \(\text{CLEAN}(x_a)\) and \(\text{CACK}(x_a)\) messages are attached to the \((\text{POLL}_j, \text{PACK}_j)\) messages exchanged between the manager and station J.

The proof relies on this fact.

At \(T_E(\text{CLEAN}^x)\), algorithm* is either in POLL0, POLL1 or WAIT_C modes, since only in these modes station J can be in \(O_a\) with \(\text{MA}(J)=y\). If algorithm* is in WAIT_C, then within 3T seconds after \(T_E(\text{CLEAN}^x)\), station J leaves \(O_a\).

If algorithm* is in POLL0, then if algorithm* and J exchange \((\text{POLL}0^j, \text{PACK}0^j)\) messages after \(T_E(\text{CLEAN}^x)\), the \(\text{CLEAN}_j(x_a)\) and \(\text{CACK}^y(x_a)\) messages are attached to the \(\text{POLL}0^j\) and the \(\text{PACK}0^j\) messages respectively, and a \(\text{CACK}(x_a)\) message is received from J within T seconds after \(T_E(\text{CLEAN}^x)\).
If such a \((\text{POLL}0, \text{PACK}0)\) exchange does not happen, then if \(\text{algorithm}^x\) leaves \(\text{POLLO}\) and moves to \(\text{WAIT}_C\), station \(J\) is taken out of \(O_a\) within 4\(T\) seconds after \(T_E(\text{CLEAN}^x)\). If \(\text{algorithm}^x\) moves to \(\text{POLL}1\), then there are two possibilities: if \(\text{algorithm}^x\) and \(J\) exchange \((\text{POLL}1, \text{PACK}1)\) messages, then \(\text{CLEAN}^y(x_a)\) and \(\text{CACK}^y(x_a)\) messages are attached to the \(\text{POLL}1\) and the \(\text{PACK}1\) messages respectively, and a \(\text{CACK}(x_a)\) message is received from \(J\) within 2\(T\) seconds after \(T_E(\text{CLEAN}^x)\). Otherwise, \(\text{algorithm}^x\) leaves the \(\text{POLL}1\) mode after \(T\) seconds and moves to \(\text{WAIT}_C\). In this case \(J\) is taken out of \(O_a\) within 5\(T\) seconds after \(T_E(\text{CLEAN}^x)\) (2\(T\) seconds in the \(\text{POLLO}\) and the \(\text{POLL}1\) modes, and 3\(T\) seconds in \(\text{WAIT}_C\)).

The case in which \(\text{algorithm}^x\) is in \(\text{POLL}1\) at \(T_E(\text{CLEAN}^x)\) is similar, and we can conclude that within 5\(T\) seconds, \(J\) is either taken out of \(O_a\) or a \(\text{CACK}(x_a)\) message is received from it.

\(\Box\)

The two following lemmas, 6.6 and 6.7, show that the \(\text{CLEAN}^x\) mode accomplishes its task.

Let \(T_O(\text{CLEAN}^x)\) be the time when \(\text{algorithm}^x\) leaves the \(\text{CLEAN}^x\) mode.

Lemma 6.6: No \(\text{DATA}_a\) messages are generated after \(T_O(\text{CLEAN}^x)\).

Proof: \(\text{DATA}_a\) messages are generated only in \(\text{sessions}^x\). Therefore, we now show that at \(T_O(\text{CLEAN}^x)\) and later, there are no \(\text{sessions}^x\).

Let \(T_E(\text{CLEAN}^x)\) be the time when \(\text{algorithm}^x\) enters \(\text{CLEAN}^x\) mode, and let \(J\) be an arbitrary station. After \(T_E(\text{CLEAN}^x)\) station \(J\) may initiate or maintain \(\text{sessions}^x\) if it receives a \(\text{DATA}\) message from \(J\) or if it receives an \(\text{ADDR}(I, x_a)\) message or if it consults its address table and the latter contains \(x_n\).

After \(T_E(\text{CLEAN}^x)\) no more \(\text{DATA}\) messages are generated by \(I\) while \(\text{SA}(I) = x_n\). This is because after \(T_E(\text{CLEAN}^x)\), station \(I\) is not active with \(\text{SA}(I) = x_n\) anymore. No \(\text{ADDR}(I, x_a)\) are generated either, since \(I\) is not in \(O_a\) with \(\text{MA}(I) = x_n\) after \(T_E(\text{CLEAN}^x)\). Thus, at \(T_E(\text{CLEAN}^x)\) the outbound channel of \(I\)
is clear of $DATA$ messages (lemma 6.3) and therefore the $DATA$ messages from $I$ and the $ADDR(I, x_n)$ messages that can possibly arrive at $J$, can be only in $M-J$.

If $J$ is in $O_m$ at $T_E(CLEAN^k)$, then within $5T$ seconds it either receives a $CLEAN(x_n)$ message, or it becomes inactive. If $J$ receives a $CLEAN(x_n)$ message, then this message clears $M-J$ of all $DATA$ messages from $I$ and of $ADDR(I, x_n)$ messages (lemma 6.5). Thus the $CLEAN(x_n)$ message terminates every session * in $J$ and clears the address table at $J$ of $x_n$ (lemma 6.5). Notice that $x_n$ will not be included in the address table of $J$ again, since $J$ will not receive anymore $DATA$ messages sent by $I$ when $SA(I) = x_n$ or $ADDR(I, x_n)$ messages.

If $J$ becomes inactive or if it is not in $O_m$ at $T_E(CLEAN^k)$ then in order to become active again and initiate sessions, $J$ must first receive a $POU_1$ message after adopting an address. This $POU_1$ message is sent after $J$ is in $O_m$, and thus after $T_E(CLEAN^k)$. Therefore, this $POU_1$ message will clear $M-J$ of all $DATA$ messages from $I$, and of all $ADDR(I, x_n)$ messages. These messages will arrive at $J$ when inactive, and therefore will not be received. Thus, $x_n$ will not be included in the address table of $J$ again, and hence station $J$ will not initiate or maintain sessions * anymore.

$\blacksquare$

Lemma 6.7: At $T_O(CLEAN^k)$ and later there are no $DATA_{x_n}$ messages in the outgoing queues.

Proof: Let $T_E(CLEAN^k)$ be the time when $algorithm^*$ enters into the $CLEAN^k$ mode, and let $J$ be an arbitrary station. If $J$ is in $O_m$ at $T_E(CLEAN^k)$, then until $T_O(CLEAN^k)$ either $J$ will send a $CACK(x_n)$ message, or will be taken out of $O_m$.

We proved in lemma 6.6 that after either of these events, $J$ does not initiate sessions *. Thus, the $CACK(x_n)$ clears the outgoing queue of $J$ of the last $DATA_{x_n}$ message it could have sent, and if $J$ is taken out of $O_m$, i.e., becomes inactive, then it clears its queues of all $DATA$ messages.

If $J$ is not included in $O_m$ at $T_E(CLEAN^k)$, then it is inactive and its queues are clear of any $DATA$ messages. We proved in lemma 6.6 that $J$ will never initiate sessions * afterwards and thus the lemma
holds for $J$. 

**Proof of Properties (A)-(C):**

The proof of properties (A)-(C) relies on lemmas 6.1-6.7. We again define $T_1^a$ to be the time when the algorithm leaves the $CLEAN^a$ mode and enters the UNASSIGN mode. We begin by proving Property (B):

**Property (B): No $DATA_x^a$ and $DSTR_x^a$ messages are generated after $T_1^a$.**

**Proof:** Lemma 6.7 proves the property for $DATA_x^a$ messages. In the present model, the $DSTR_x^a$ messages are $ADDR$, $NO\_ADDR$ and $CLEAN$. After $T_1^a$, no $ADDR$ and $NO\_ADDR$ messages, containing $x_n$ as destination address, are generated, for similar reasons as in model A (see the proof of Property (B) in model A). No $CLEAN$ messages, containing $x_n$ as destination address, are generated, since after $T_1^a$ no station $K$ is in $O_a$ while $MA(K)=x_n$ ($x_n$ is only designated once), and $CLEAN$ messages containing $x_n$ as destination address are sent only to stations that are included in $O_a$ with designated address $x_n$.

**Property (A): $DATA_x^a$ and $DSTR_x^a$ messages are received only by station $I$ and only when $SA(I)=x_n$.**

**Proof:** The proof is similar to the one of model A, concerning both $DATA_x^a$ and $DSTR_x^a$ messages of types $ADDR$ and $NO\_ADDR$. Regarding $CLEAN$ messages, those are generated by the manager, and therefore $CLEAN$ messages containing $x_n$ as destination address, can be at $T_1^a$ only in $M-K$, for any station $K$ in the network. Thus, any station $K$ that will adopt $x$ after $T_1^a$ must first receive an $ASSN\_i(K)$ message. This message will clear $M-K$ of all remaining $CLEAN$ messages, which contain $x_n$ as destination address, and thus these messages will arrive at $K$ when inactive and will not be received.
Property (C): Address \( x_{n+1} \) becomes available to be designated within finite time after \( T_1^n \).

Proof: Property (A) guarantees that it is possible to assign \( x_{n+1} \), but the question is again whether \( algorithm \) reenters into the UNASSIGN mode. However, the stay in the two new modes is for a finite time: \( d_{out} \) seconds in the WAIT_D and at most 5T seconds in CLEAN mode, as shown in lemma 6.5. Thus, \( algorithm \) will return to UNASSIGN mode in finite time.

\[ \square \]

6.4 Other versions

In this section, as in Sec. 5.4 of model A, we present some other versions that differ from each other mainly in the times when station \( I \) is included or is taken out from \( O_n \), and when it becomes active or inactive.

Version 1:

The protocol of this version is similar to the one in Sec. 6.2, with two changes. First, station \( I \) is taken out of \( O_n \) at the time when \( algorithm \) enters into WAIT_C mode, instead of at the time when WAIT_C is exited. Recall that when \( algorithm \) leaves WAIT_C, station \( I \) is already inactive. Thus, every station that receives \( x_n \) as the address of \( I \) when \( algorithm \) is in WAIT_C, can manage a session with \( I \) for at most 3T seconds. It therefore makes sense, as in version 1 of Sec. 5.4, to stop the distribution of \( x_n \) when \( algorithm \) enters into the WAIT_C mode.

With the above change it may happen that \( O_n \) does not include all the active stations and therefore it is possible that \( V \) will not contain all stations that are active at the time when CLEAN^x is entered. Thus, CLEAN^x will not guarantee that at the time it is exited, there are no sessions^x and the outgoing queues are clear of DATA^x messages. This can happen since \( algorithm \) does not send CLEAN(\( x_n \)) messages to all active stations, and even if an active station \( J \) is sampled into \( V \), then receiving the information that \( J \) is taken out of \( O_n \) (which is one of the possibilities in CLEAN mode) is not sufficient, because \( J \) can still remain active and maintain sessions^x.
The solution we suggest for this problem leads to the second change. A new mode, WAIT_E, is added after the CLEAN mode. In WAIT_E algorithm\(^*\) waits 3T seconds, and the previous problem is solved in the following way: assume that a station \(J\) is active at the time when algorithm\(^*\) enters into CLEAN\(^*\), and that MA(\(J\))\(=y\). If \(J\) is not sampled into \(V\), then it is only because algorithm\(^*\) is in the WAIT_C mode at the time when \(O_\alpha\) is sampled. The stay in WAIT_C and WAIT_E is 3T seconds, and thus when the WAIT_E mode is exited, algorithm\(^*\) has definitely left WAIT_C and \(J\) is inactive.

Let us assume now that \(J\) is sampled into \(V\), and algorithm\(^*\) is informed in CLEAN\(^*\) that \(J\) is taken out of \(O_\alpha\). Station \(J\) is taken out of \(O_\alpha\) because algorithm\(^*\) has entered WAIT_C, but this information does not guarantee that \(J\) is inactive. However, when WAIT_E is exited, \(J\) is already inactive since algorithm\(^*\) has definitely left WAIT_C.

Thus, every station that is active at the time when CLEAN\(^*\) is entered, either exchanges a pair of (CLEAN,CACK) messages with algorithm\(^*\), or becomes inactive before the designation of \(x_{n+1}\). This ensures properties (A)-(C) as we have proved in Sec. 6.3.

**Version 2:**

The protocol of this version is depicted in Fig. 6.3.

The station algorithm is similar to the original one [GS2], with station \(I\) becoming active in the transition of its address algorithm from the AACK mode to the PACK0 mode, and it becomes inactive when the algorithm returns to DEAD_STATE (see Figure 6.3a). On the other hand, there are some changes in the manager algorithm (see Figure 6.3b), comparing to the original one [GS2]. The CLEAR1 mode is separated into two modes, CLEAR1_A and CLEAR1_B. In addition, there are three new modes: WAIT_D, CLEAN and WAIT_E.

The WAIT_D and the CLEAN modes are the same as in the version of Sec. 6.2, and also perform the same tasks. In the CLEAN mode however, there is an addition, as we explain later. In WAIT_E mode, algorithm\(^*\) stays \(d_{\text{out}}\) seconds. When WAIT_E\(^*\) is exited, the outbound channels are also clear of DATA\(_1\), messages. We explain later why this is necessary.
Finally, \( I \) is included in \( O^* \) when \textit{algorithm} makes the transition from the PENDING mode to the INIT POLL0 mode, and is taken out when \textit{algorithm} either leaves WAIT_B or WAIT_C modes. With these changes it can be shown that when \( I \) is active, it is in \( O^* \) as in the version of Sec. 6.2.

We now explain the changes. We only give a short description, since the problems in this version are similar to those described for version 2 in Sec. 5.4 and we adopt the same ideas for their solution. However, the implementation is different since we use the \((\text{CLEAN,CACK})\) messages of the CLEAN mode, instead of the \textit{CLR}2 messages of version 2 in Sec. 5.4.

In the description of version 2 in Sec. 5.4, we have described a scenario in which a \textit{session} can be initiated by a station \( J \) unlimited time after the transition of \textit{algorithm} out of WAIT_C. This scenario is also possible in this version. The first stages of the solution for this problem in version 2 of Sec. 5.4 were first to wait \( d_{out} \) seconds until all the \textit{DATA} messages sent from \( I \) when \( SA(I)=x_n \) held, arrive at the media. Then, the media is clear of these messages by the \textit{CLR}1 or \textit{CLR}2 messages (see description of version 2 in Sec. 5.4). We adopt this solution in this version as well, and the implementation is as follows: It is guaranteed that after the time when \textit{algorithm} leaves the WAIT_C mode, or after a stay of \( 5T+d_{in} \) seconds in the WAIT_B mode, station \( I \) is not active with \( SA(I)=x_n \) anymore. Thus, \( d_{out} \) seconds
Figure 6.3b - Algorithm at the address manager
later, the outbound channel from $I$ is clear of $DATA$ messages. These $d_{out}$ seconds are contained in the stay in the WAIT_B mode (in addition to the $5T+d_{in}$ seconds) or in the WAIT_D mode, which also appears in the version of Sec. 6.2 with the same task.

The clearing of the media is now achieved in two ways: If $algorithm^x$ enters CLEAR1_B mode, then the CLR1 messages clear the media. However, when $algorithm^x$ leaves the WAIT_D mode, it must clear the media with special messages. Recall that $algorithm^x$ sends $CLEAN(x_a)$ messages in CLEAN$^x$ mode. We use these messages to also clear the media in the following way: let $J$ be a station which is sampled into $V$ when $algorithm^x$ enters into CLEAN$^x$, and assume $MA(J)=y$. If a pair of $(CLEAN(x_a),CACK(x_a))$ messages is exchanged with $J$, or $J$ is taken out of $O_m$, then it is guaranteed that $J$ will never initiate sessions$^x$ again, and its outgoing queue is clear of $DATA_a$ messages. This is similar to the version of Sec. 6.2 (see proofs of lemmas 6.6 and 6.7). However, if within $5T$ seconds neither of these two possibilities happens (see proof of lemma 6.5 why $5T$ seconds are the correct time period), then it can be proved that the two possibilities do not happen since $algorithm^x$ has entered into the INIT_CLEAR0 or WAIT_B modes, and therefore it has not exchanged $(POLL,PACK)$ messages with $J$, and it has not taken $J$ out from $O_m$. In this case it is possible that the scenario of version 2 in Sec. 5.4 will occur, and $algorithm^x$ must ensure that the media is clear of $DATA$ messages from $I$ and later to wait $5T+d_{in}$ seconds to ensure that $J$ will never initiate sessions$^x$ again (see version 2 in Sec. 5.4).

The clearing of the media can be detected by $algorithm^x$ by checking if after the time when it has entered into CLEAN$^x$, a pair of $(CLEAN,CACK)$ messages has been exchanged with any other station $K$ in $V$, or whether a $CLEAN(x_a)$ message has returned from the media. If either of these events has occurred, then the media has been cleared of the $DATA$ messages from $I$. Otherwise, it is not guaranteed that the media is clear, and $algorithm^x$ continues to send $CLEAN(x_a)$ messages and waits for one message to return back.

After ensuring that the media is clear, $algorithm^x$ waits $5T+d_{in}$ seconds (see version 2 in Sec. 5.4) and then nothing else needs to be done in CLEAN$^x$ concerning $J$, since no more $DATA_a$ messages are ever generated by $J$ and its outgoing queue is clear of $DATA_a$ messages. Recall that if the CLEAN$^x$ is
entered from the CLEAR1_B, then the media is already clear of DATA messages from I.

Again, as in version 2 in Sec. 5.4, we need the WAIT_E mode where algorithm*R waits $d_{out}$ seconds before entering into the UNASSIGN mode and undesignating $x_n$. This mode is necessary to ensure the clearing of the outbound channels of DATA$_n$ messages, in order to ensure properties (A)-(C) as in version 2 in Sec. 5.4.

Notice that if algorithm*R leaves WAIT_A, it moves to the CLEAR1_A mode and returns to UNASSIGN without traversing any of the new modes. It can be proved that if algorithm*R reaches WAIT_A, station I never becomes active with $SA(I)=x_n$, and also $x_n$ is never distributed as the address of I. Thus, no sessions*R are ever initiated, and there is no need for the new modes.

Version 3:

The protocol of this version is depicted in Fig. 6.4.

![Diagram](image)

**Figure 6.4a - The algorithm of station I**

In this version station I becomes active when its address algorithm enters AACK, and becomes inactive when the algorithm returns to DEAD_STATE (see Figure 6.4a).
Figure 6.4b - Algorithm at the address manager
The manager algorithm is similar to the one of version 2, except that the CLEAR1 mode is not split, and remains as in [GS2]. This ensures that algorithm is always traverses CLEAN before undesignating x_n.

This is necessary since station I is now included into O_m when algorithm enters PENDING, and it is possible that x_n will be distributed as the address of I while algorithm is in PENDING, CLEAR0 or WAIT_A. Sessions can therefore be initiated when algorithm is in one of these modes. Finally, in order to ensure that every active station is in O_m, station I is taken out from O_m when algorithm leaves WAIT_C, or when algorithm enters CLEAR1 (see Figure 6b).

In version 2 we have mentioned a scenario where a Station J may initiate a session unlimited time after the transition of algorithm out of WAIT_C. This scenario is also possible here, and so the CLEAN mode is the same as in version 2. The other new modes, namely WAIT_D and WAIT_E, are similar and have the same task as in version 2.

Finally, we remind that in version 4 in Sec. 5.4 we have suggested a solution which leaves the original protocol [GS2] unchanged. However, we cannot implement such a solution in this model, since here it is necessary to update the address tables by special CLEAN messages.

7. MODEL C - AUTOMATIC ADDRESS DISTRIBUTION

7.1 The model

In model C when a station becomes active, it receives from the manager a list of the addresses of all active stations and keeps this list in the address table. Afterwards, this list is updated by using messages from the manager. Updates occur whenever an active station leaves the network or a station joins the network. The motivation for this model is to completely eliminate the need for requesting station addresses. This task is accomplished by automatically distributing addresses. The disadvantage is the additional load on the manager that must continuously update all stations.
The model faces the same difficulties as model B. However, automatic address distribution requires some additional tasks. When station $I$ becomes active and $MA(I) = x$, all other active stations must be informed that $I$ has joined with assigned address $x$. Also, $I$ should receive the addresses of all active stations. While $I$ is active, it should be updated regarding the address of any new active station and about stations leaving the network. Finally, when $I$ becomes inactive, all remaining active stations must be informed so they will clear their address tables of the address of $I$.

To summarize, model C must have properties (A)-(C) of Sec. 4, as well as a fourth property:

**Property (D):** Any active station $I$ knows the address of any other active station $J$ within finite time, unless $I$ or $J$ becomes inactive.

As mentioned, stations receive in this model the list of addresses of all active stations. However, we also allow sessions to be initiated directly by $DATA$ messages, without consulting the address table, i.e., a station initiates a session using the address it receives in a $DATA$ message. Sessions may be managed as in model A, i.e., using the timeout mechanism and in addition, as in model B, sessions are terminated also by $CLEAN$ messages. Thus, the termination of sessions is guaranteed by specific messages and it is possible to give up the timeout mechanism.

### 7.2 The Protocol

The station algorithm for model C is the same one as for model B (see Figure 6.1a). However, there are some changes in the manager algorithm. The manager algorithm is depicted in Figure 7.1, where the additions to the original protocol [GS2] are again emphasized in boldface.

The protocol is similar to the one of model B, except that the stay in WAIT_D is for $T$ seconds. However, the address distribution process is new here. As it is separated from the address designation process, its messages are not depicted in the address assignment protocol. We next describe the process.

When station $I$ enters $\alpha_n$, $\text{algorithm}{^x}$ begins to send to $I$ the list of addresses of all stations in $\alpha_n$. The address list is sent by a special message, $ADDRL$ (ADDRes List), which is
piggybacked to the \textit{POLL} 1 messages. Thus, the first \textit{POLL} 1 that arrives at \textit{I} after setting \textit{SA}(I) = x_n, also carries the address list and when \textit{I} becomes active, it can build its address table. Also, when \textit{I} receives the \textit{ADDRL} message, it sends \textit{ADLACK} (Address List ACK) message to the manager, acknowledging the reception of \textit{ADDRL}. The \textit{ADLACK} are attached to the \textit{PACK1} messages. Attachment \textit{ADDRL} and \textit{ADLACK} to \textit{POLL} 1 and \textit{PACK} 1 guarantees that either \textit{I} receives and acknowledges the reception of
ADDRL within T seconds, or it becomes inactive.

In addition, when \( I \) enters \( O_a \), algorithm\(^x\) also distributes the information that \( I \) has entered \( O_a \) with designated address \( x_n \), to each station \( J \) in \( O_a \). This information is sent by \( ADD(I,x_n) \) messages. When \( J \) receives an \( ADD(I,x_n) \) message, it includes \( I \) in its address table with designated address \( x_n \) and acknowledges it by sending \( ADACK(x_n) \) message to the manager.

However, assume that \( MA(J) = y \) when \( I \) enters \( O_a \). If algorithm\(^y\) is in WAIT_C mode at that time, then no \( ADD(I,x_n) \) messages are sent to \( J \) although \( J \) is in \( O_a \). That is because \( J \) will become inactive within 3T seconds. Otherwise, the \( ADD \) and \( ADACK \) are piggybacked to the \( (POLL,PACK) \) messages. This guarantees that \( J \) will either receive the \( ADD(I,x_n) \) message within 2T seconds or it will become inactive within 3T seconds later: in the worst case algorithm\(^x\) will start to send the \( ADD(I,x_n) \) messages immediately after the time when \( J \) and algorithm\(^y\) had exchanged \( (POLL,PACK) \) messages say. According to the protocol, algorithm\(^x\) does not send additional \( POLL \) messages until it moves to POLLO mode, after completing a stay of T seconds in the POLL1 mode. Therefore, only when algorithm\(^y\) starts to send \( POLL \) messages, algorithm\(^x\) also starts to send \( ADD(I,x_n) \) messages to \( J \), attached to the \( POLL \) messages. If algorithm\(^y\) and \( J \) will exchange \( (POLL,PACK) \) messages, than \( J \) has definitely received an \( ADD(I,x_n) \) message. Otherwise, algorithm\(^y\) will enter into WAIT_C after staying T seconds in the POLLO mode, and when it will leave WAIT_C, it is guaranteed that \( J \) is inactive. Also, algorithm\(^y\) stops to send \( ADD(I,x_n) \) messages when it enters into WAIT_C\(^x\) mode. This is because in this case \( I \) will become inactive within 3T seconds.

Finally, similarly to the \( (CLEAN,CACK) \) messages (see Sec. 6.2) we also allow that the \( ADD \) and \( ADACK \) messages will also be sent separately, i.e. as independent messages, and will only be attached to the \( (POLL,PACK) \) if the latter are sent by the address assignment protocol. We also recall that the \( ADD \) messages are received only by active stations. Recall that inactive stations are allowed to receive only those messages that are sent by the original address assignment protocol (messages of category 1 in Sec. 4) and only active stations are allowed to maintain sessions and to be informed of other station addresses.
It is important to include $I$ in $O_n$ before sending the $ADD(I,x_n)$ messages to the stations and the $ADDRL$ messages to $I$. This order guarantees that any station in $O_n$ knows the address of any other station in $O_n$: any station $J$ that is in $O_n$ when $I$ enters $O_n$, is informed about $I$ by the $ADD(I,x_n)$ messages, while $I$ is informed about $J$ by the $ADDRL$ messages. It can be easily shown that any other order, e.g., sending either $ADD$ or $ADDRL$ before including a station in $O_n$, does not guarantee the above.

Also, in the above scheme, $J$ may receive the information about the a new active station $I$ in both ways, via $ADDRL$ and $ADD$ messages. For example, assume that at time $T_1$ station $I$ enters $O_n$ and $MA(I)=x$; Later, at time $T_2$, station $J$ enters $O_n$ and $MA(J)=y$; at $T_3$ algorithm $\text{v}$ begins to send $ADDRL$ messages to $J$ and at $T_4$ algorithm $\text{k}$ sends $ADD(I,x)$ messages to $J$. In this scenario, $J$ is informed about the address of $I$ by both the $ADDRL$ and $ADD$ messages. However, this possibility is legal, since address distribution is achieved.

As shown later, the address distribution process described so far guarantees Property (D). However, it has a side effect: the stay in $\text{WAIT}_D$ is extended from $d_{\text{out}}$ seconds to $T$ seconds. To understand why $T$ seconds are necessary, assume that at time $T_2$, station $I$ is taken out of $O_n$. Assume that at time $T_1$, where $T_1<T$, a station $J$ enters $O_n$ and $MA(J)=y$. At $T_1$ algorithm $\text{v}$ begins to send $ADDRL$ messages to $J$, containing $I$ with assigned address $x_n$. At $T_2$, algorithm $\text{k}$ begins to send $CLEAN(x_n)$ messages to $J$. Since $T_2-T_1<T$ then at $T_2$ algorithm $\text{v}$ is still in the $\text{POLL}\_1$ mode. Thus, it may happen that algorithm $\text{k}$ will attach $CLEAN(x_n)$ messages to the $\text{POLL}\_1$ messages, while these messages are still being sent with the $ADDRL$ messages. It can now happen that $J$ will receive a $\text{POLL}\_1$ messages, become active, process the $CLEAN(x_n)$ message first and then the $ADDRL$ message. Consequently, $J$ will include $I$ with address $x_n$ in its address table although $I$ was taken out of $O_n$. To avoid this confusion, the stay in the $\text{WAIT}_D$ mode is extended to $T$ seconds. In this way, algorithm $\text{k}$ will send $CLEAN(x_n)$ messages only after every algorithm that had been sending $ADDRL$ messages containing $x_n$ has already stopped sending those messages.
7.3 Correctness proof

We now prove that the protocol has properties (A)-(D). Again, the proofs rely on several lemmas: Lemma 7.1 shows that the additions to the original protocol do not affect its main properties, and so the address designation is correct. Lemma 7.2 shows that an active station always appears in \( O \). Lemma 7.3 shows that when algorithm \( x \) leaves the WAIT\(_D\) mode, the outbound channel from \( I \) is clear of DATA, ADLACK, ADACK and CACK messages. Lemma 7.4 shows that when in CLEAN\( x \), algorithm \( x \) does not receive CACK\( (x) \) messages from previous assignments of \( x \), and lemma 7.5 shows that algorithm \( x \) stays at most \( 5T \) seconds in the CLEAN\( x \) mode. Lemmas 7.6 and 7.7 show that the (ADD, ADACK) and (ADDRL, ADLACK) messages are exchanged 'correctly'. We explain the term 'correctly' when the lemmas are stated. Finally, lemmas 7.8 and 7.9 prove that no DATA\( x \) messages are generated after the time when algorithm \( x \) leaves the CLEAN\( x \) mode, and also that the outgoing queues are clear of DATA\( x \) messages at this time and later.

**Lemma 7.1 - The main properties - Uniqueness, Assignment and Consistency of the original protocol, hold for the protocol for model C.**

**Proof:** The additions to the original protocol [GS2] are similar to those of the protocol of model B, with the addition of ADDRL, ADLACK, ADD and ADACK. However, the new messages only update the address tables, and do not affect the address designation process.

\[ \square \]

As before, lemma 7.1 is important to guarantee a correct address assignment. Lemma 7.1 also implies that the protocol for model C has the basic properties of lemma 5.2.

Lemmas 7.2 - 7.5 are similar to lemmas 6.2 - 6.5 and so are their proofs. For convenience, we restate the lemmas:
Lemma 7.2 - Station $I$ in active state ($OP(I)=$TRUE) appears in $O_m$.

Lemma 7.3 - When $algorithm^x$ leaves the WAIT_D$^x$ mode, the outbound channel from $I$ is clear of
$DATA, ADLACK, ADACK$ and $CACK$ messages.

Let $T_E(CLEAN^x)$ and $T_D(CLEAN^x)$ be the times when $algorithm^x$ enters and leaves the CLEAN$^x$ mode respectively. Assume that a station $J$ is in $O_m$ at $T_E(CLEAN^x)$. Since $J$ is in $O_m$, holds $MA(J)$=nil.

Assume that $MA(J)$=y and thus, according to the protocol, $algorithm^x$ sends $CLEAN_J(x_n)$ messages to $J$ and waits for the earlier of the two events: a $CACK_J^x(x)$ message is received, or $J$ is taken out from $O_m$.

Assume that a $CACK_J^y(x)$ message is received, i.e. $J$ is still in $O_m$ at the time of reception.

Lemma 7.4 - Algorithm$^x$ will not receive a $CACK_J^x(x_l)$ message with $l<n$ as an acknowledgement to a $CLEAN_J(x_n)$ message.

Lemma 7.5 - Algorithm$^x$ stays in CLEAN$^x$ at most 5$T$ seconds.

Notice that changing the stay in WAIT_D mode from $d_{out}$ seconds to $T$ seconds does not affect the correctness of lemma 7.3 (see proof of lemma 6.3 in model B). Also, since inactive stations do not receive ADDRL, $ADD$ and CLEAN messages, and so do not send ADLACK, ADACK and CACK messages, lemma 7.3 holds for these messages as well.

Let $T_n$ be the time when $I$ enters $O_m$ and $MA(I)=x_n$. Assume that a station $J$ is in $O_m$ at $T_n$ and that $MA(J)=y$. Since $J$ is in $O_m$, then immediately after $T_n$ algorithm$^x$ sends $ADD_J(I,x_n)$ messages to $J$.

Lemma 7.6 - Algorithm$^x$ will not receive an $ADD_J^x(x_l)$ message with $l<n$ as an acknowledgement to an $ADD_J(I,x_n)$ message.

Notice: The lemma says that if $algorithm^x$ receives an $ADD_J^x(x)$ message, then this message has been sent as a response to an $ADD_J(I,x_n)$ message, i.e., it is guaranteed that $J$ has received an $ADD_J(I,x_n)$
message and that \textit{algorithm} \textsuperscript{\$i\$} does not receive an \textit{ADACK} \textsubscript{\textsuperscript{2}}(x) message that is left in the network from previous assignments of x.

\textbf{Proof:} Let \( x_1, 1 < n \), be the last address before \( x_n \), for which \( ADD_y(K, x_1) \) messages were sent, where \( K \) is the station for which \( MA(K) = x_1 \). It is clear from the protocol that after \textit{algorithm} \textsuperscript{\$i\$} enters into \textit{WAIT} \textsubscript{\textsuperscript{C}} \textsuperscript{\textsuperscript{\$i\$}} mode, no more \( ADD_y(K, x_1) \) messages are sent. Therefore, the time interval between the first time after which no more \( ADD_y(K, x_1) \) messages are sent and the first time when \textit{algorithm} \textsuperscript{\$i\$} begins to send \( ADD_y(l, x_n) \) messages is at least \( 5T \) seconds: \( 4T \) seconds during which \textit{algorithm} \textsuperscript{\$i\$} is in \textit{WAIT} \textsubscript{\textsuperscript{C}} \textsuperscript{\textsuperscript{\$i\$}} and \textit{WAIT} \textsubscript{\textsuperscript{D}} \textsuperscript{\textsuperscript{\$i\$}}, and \( T \) seconds during which \textit{algorithm} \textsuperscript{\$i\$} stays in \textit{INIT}-\textit{POLLO} \textsuperscript{\$i\$}. The period \( 5T \) is also the minimum time interval between successive entrances into \textit{CLEAN} mode. The rest of the proof is identical to that of lemma 6.4, with the \textit{ADD} and \textit{ADACK} replacing \textit{CLEAN} and \textit{CACK}.

\( \square \)

Let station \( K \) be a station for which \( MA(K) = x_1 \), \( 1 < n \). When \( K \) is included in \( O_n \), \textit{algorithm} \textsuperscript{\$i\$} sends \textit{ADDR} \textit{L} messages to it.

\textbf{Lemma 7.7 - } \textit{Algorithm} \textsuperscript{\$i\$} will not receive an \textit{ADLACK} message sent by \( K \), as an acknowledgement to an \textit{ADDR} \textit{L} message sent to \( l \).

The lemma ensures that \( l \) either receives the \textit{ADDR} \textit{L} message, or becomes inactive.

\textbf{Proof:} The basic properties of the protocol (lemma 5.2) ensure that \textit{algorithm} \textsuperscript{\$i\$} cannot receive \textit{PACK} \textsubscript{\textsuperscript{1}} \textsuperscript{\$n\$} messages, sent by the station to which \( x_p \) was designated. Since the \textit{ADDR} \textit{L} and \textit{ADLACK} are attached to the \((\textit{POLLO} \textsubscript{1}, \textit{PACK} \textsubscript{1})\) messages, the lemma follows.

\( \square \)
Let $T_0(CLEAN^{x})$ be the time when algorithm$^{x}$ leaves the CLEAN$^{x}$ mode.

Lemma 7.8: No $DATA_x$ messages are generated after $T_0(CLEAN^{x})$.

Proof: Let $T_E(CLEAN^{x})$ be the time when algorithm$^{x}$ enters CLEAN$^{x}$, and let $J$ be an arbitrary station. Station $J$ may initiate or maintain sessions$^{x}$ after $T_E(CLEAN^{x})$ either after receiving $DATA$ messages from $I$, or by consulting its address table, which contains $x_n$, due to a previous reception of an $ADD(I,x_n)$ or an $ADDRL$ containing $x_n$.

After $T_E(CLEAN^{x})$ no more $DATA$ messages are generated by $I$ while $SA(I)=x_n$. This is because after $T_E(CLEAN^{x})$, station $I$ is not active with $SA(I)=x_n$ anymore. No more $ADD(I,x_n)$ messages are generated, because after $T_E(CLEAN^{x})$, station $I$ is not in $o_w$ with $MA(I)=x_n$ anymore. Also, no more $ADDRL$ messages, containing $x_n$, are generated and sent because of the following reason: recall that algorithm$^{x}$ stays $T$ seconds in WAIT$_D^{x}$ mode, after taking $I$ out of $o_w$. Let $T_E(WAITD^{x})$ be the time when algorithm$^{x}$ enters into WAIT$_D^{x}$ mode. Consider algorithm$^{y}$ say. If algorithm$^{y}$ starts to send $ADDRL$ messages after $T_E(WAITD^{x})$, then it will not include $x_n$ in the $ADDRL$ messages since $I$ is not in $o_w$ with $MA(I)=x_n$ anymore. If algorithm$^{y}$ started to send $ADDRL$ messages containing $x_n$ before $T_E(WAITD^{x})$, then it had definitely stopped sending these messages before $T_E(CLEAN^{x})$. This is because these messages are sent for $T$ seconds, while algorithm$^{y}$ stays in the POLL1 mode for the first time after designating $y$.

Notice now that lemma 7.3 implies that at $T_E(CLEAN^{x})$ there are no $DATA$ messages in the outbound channel from $I$. Therefore, at $T_E(CLEAN^{x})$ any $DATA$ messages that were sent from $I$ when $SA(I)=x_n$ holds, or $ADD(I,x_n)$ messages or $ADDRL$ messages, which contain $x_n$, that can possibly arrive at $J$, are in $M-J$ only. The proof is now similar to the proof of lemma 6.6.
Lemma 7.9: At $T_0(\text{CLEAN}^*)$ and later, there are no $DATA_{x_n}$ messages in the outgoing queues at the stations.

Proof: Similar to the proof of lemma 6.7, relying on lemma 7.8.

Proof of properties (A)-(D):

The proof of properties (A)-(D) relies on lemmas 7.1-7.9. We again define $T_1^i$ to be the time when algorithm$^*$ leaves the CLEAN$^*$ mode and enters into the UNASSIGN mode. We begin by proving Property (B):

Property (B): No $DATA_{x_n}$ and $DSTR_{x_n}$ messages are generated after $T_1^i$.

Proof: The proof is similar to the one in model B. Notice that in this model the $DSTR_{x_n}$ messages contain ADDRL and ADD messages also. However, these messages, containing $x_n$ as destination address, are not generated after $T_1^i$, since $I$ is not included in $O_n$ with $MA(I)=x_n$ after $T_1^i$, and $x_n$ is designated only once.

Property (A): $DATA_{x_n}$ and $DSTR_{x_n}$ messages are received only by station $I$ and only when $SA(I)=x_n$.

Proof: Similar to the one in model B, including the new ADDRL and ADD messages.

Property (C): Address $x_{n+1}$ becomes available to be designated within finite time after $T_1^i$.

Proof: Similar to model B.

Property (D): Any active station $I$ knows the address of any other active station $J$, within finite time,
unless \( I \) or \( J \) become inactive.

Proof: Since by lemma 7.3 any active station is in \( O_n \), it is sufficient to prove the property for stations in \( O_n \).

Assume \( I \) enters \( O_n \) at time \( T^n \). Assume that a station \( J \) enters \( O_n \) after \( T^n \) and \( MA(J) = y \). If when \( J \) enters \( O_n \) station \( I \) is still in \( O_n \) and \( MA(I) = x_n \), then \( algorithm^x \) sends \( ADD(J,y) \) messages to \( I \). Lemma 7.6 and the fact that \( ADD \) messages are attached to \( POLL \) messages guarantee that \( I \) will either receive the message within finite time or will become inactive. If \( J \) is already included in \( O_n \) at \( T^n \), then \( I \) receives \( y \) as the address of \( J \) by receiving an \( ADDRL \) message that \( algorithm^x \) sends to it. Lemma 7.7 and the fact that \( ADDRL \) messages are attached to \( POLL \) messages guarantee that \( I \) will either receive the \( ADDRL \) message within \( T \) seconds or it will become inactive within finite time.

\[ \square \]

7.4 Other versions

In this section, as in Sections 5.4 and 6.4, we present some other versions that mainly differ from each other in the times when station \( I \) enters or is taken out of \( O_n \), and so in the times when the \( ADDRL \) and \( ADD \) messages are sent, and when \( I \) becomes active or inactive.

Version 1:

In versions 1 of Sections 5.4 and 6.4 we have suggested to take \( I \) out of \( O_n \) at the time when \( algorithm^x \) enters the \( WAIT_C \) mode, instead of at the time when \( WAIT_C \) is exited. This change was suggested since at the time when \( algorithm^x \) exits \( WAIT_C \), station \( I \) is inactive, and so any station that receives \( x_n \) while \( algorithm^x \) is in \( WAIT_C \), can maintain a session with \( I \) for not more than 3T seconds. Therefore it was reasonable to stop the distribution of \( x_n \) when \( WAIT_C \) is entered.

However, in the version of Sec. 7.2 we have decided that no more \( ADD(I,x_n) \) messages are sent after \( algorithm^x \) enters \( WAIT_C \), for the same reason as before, and therefore version 1 in this case is actually implemented in the version of Sec. 7.2. 
Version 2:

The protocol of this version is depicted in Figure 7.2.

The station algorithm is similar to the original one [GS2], with station \( I \) becoming active in the transition of its address algorithm from the AACK mode to the PACK0 mode, and becoming inactive when the algorithm returns to DEAD_STATE (see Figure 7.2a). On the other hand, there are some changes in the manager algorithm (see Figure 7.2b), comparing to the original one [GS2]. The CLEAR1 mode is separated into two modes, CLEAR1_A and CLEAR1_B. In addition, there are three new modes: WAIT_D, CLEAN and WAIT_E.

The WAIT_D and CLEAN modes are the same as in the version of Sec. 7.2, and also perform the same tasks. In the CLEAN mode however, there is an addition, as we explain latter. In WAIT_E mode, \( \text{algorithm} \) stays \( d_{\text{sw}} \) seconds. When WAIT_E is exited, the outbound channels are also clear of \( \text{DATA}_n \) messages. We explain why this is necessary later.

Finally, station \( I \) enters \( \alpha_n \) at the time when \( \text{algorithm} \) makes the transition from PENDING mode to INIT_POLLO mode, and it is taken out from \( \alpha_n \) at the time when \( \text{algorithm} \) exits either the WAIT_B or the WAIT_C mode. Station \( I \) becomes active at the time when its address algorithm makes the
Figure 7.2b - Algorithm\textsuperscript{x} at the address manager
transition from the AACK mode to the PACK0 mode, and becomes inactive when the algorithm returns to DEAD_STATE. In this protocol, the ADDRL messages are attached to the POLL0 messages of the INIT_POLL0 mode, and algorithm\textsuperscript{xn} begins to send ADD(I,xn) messages at the time when it enters INIT_POLL0. Algorithm\textsuperscript{xn} stops to send these messages when it enters either WAIT_B or WAIT_C.

We now explain the changes. In the description of version 2 in Sec. 5.4, we have described a scenario in which a session\textsuperscript{xn} can be initiated by a station J unlimited time after the transition of algorithm\textsuperscript{xn} out of WAIT_C. This scenario is possible since ASSN and POLL0 messages, on their way to J, can stay in the media for an unlimited time period. If these messages are also followed by a DATA message which was sent from I at the time when SA(I)=xn held, then after receiving the ASSN and POLL0 messages, J can receive the DATA message and initiate a session\textsuperscript{xn} (see description of version 2 in Sec. 5.4 and Figure 5.3). It is necessary to wait until such session\textsuperscript{xn} is terminated, before undesignating x_n. For this purpose it was suggested in version 2 in Sec. 5.4 to first clear the outbound channel from I, and later the media, of all the DATA messages from I, and later, within at most 5T+d_in seconds, it is guaranteed that station J becomes inactive and does not initiate sessions\textsuperscript{xn} anymore.

However, except for the above tasks, it is also necessary in this model to clear the media of all the POLL0 messages that are sent to J in the above scenario. It can be proved that these POLL0 messages can be sent to J with ADDRL messages that contain x_n. Thus, these messages can also cause the initiation of sessions\textsuperscript{xn}.

In this version we adopt the solutions in versions 2 of Sections 5.4 and 6.4 in order to solve the first problem above. The WAIT_D mode and the \textit{d_out} seconds that algorithm\textsuperscript{xn} stays in WAIT_B mode (in addition to the 5T+d_in seconds), stand for the necessary \textit{d_out} seconds to ensure the clearing of the outbound channel from I of any DATA messages. Notice, however, that the stay in the WAIT_D mode is T seconds and not \textit{d_out} seconds. That is necessary to ensure that no more POLL0 messages, carrying ADDRL messages which contain x_n, are sent after sending CLEAN(x_n) messages (similar to the version of Sec. 7.2). Thus, this mode also ensures that no more POLL0 messages, intended for J and carrying ADDRL messages with x_n, are ever sent after entering CLEAN\textsuperscript{xn}. Then, algorithm\textsuperscript{xn} enters into the CLEAN\textsuperscript{xn} mode,
which is similar to the one of version 2 in Sec. 6.4 (see description there), and after clearing the media of DATA messages from I and the POLL0 messages sent to J, the algorithm stays \( d_{out} \) seconds in WAIT_E (for the same reasons as explained in version 2 of Sec. 6.4), and returns to the UNASSIGN mode while ensuring properties (A)-(C).

Notice that if \( algorithm^* \) leaves WAIT_A, it moves to the CLEAR1_A mode and returns to UNASSIGN without traversing any of the new modes. It can be proved that if \( algorithm^* \) reaches WAIT_A, station I never becomes active with \( SA(I) = x_n \), and also \( x_n \) is never distributed as the address of I. Thus, no sessions \( ^* \) are ever distributed, and there is no need for the new modes.

**Version 3:**

The protocol of this version is depicted in Fig. 7.3.

![Diagram](image)

**Figure 7.3a - The algorithm of station I**

In this version station I becomes active when its address algorithm enters AACK, and it becomes inactive when the algorithm returns to DEAD_STATE (see Figure 7.3a).

The manager algorithm is similar to version 2, except that the CLEAR1 mode is not split, and remains as in [GS2]. This ensures that \( algorithm^* \) always traverses the CLEAN and WAIT_D modes.
Figure 7.3b - Algorithm at the address manager
before undesignating $x_n$. This is necessary since station $I$ is now included into $O_n$ when algorithm $^*$ enters PENDING, and it is possible that $x_n$ will be distributed as the address of $I$ while algorithm $^*$ is in PENDING, CLEAR0 and WAIT_A. Sessions $^*$ can therefore be initiated when algorithm $^*$ is in one of these modes (see Figure 7.3b). In this version, the ADDR messages are attached to the ASSN messages, and algorithm $^*$ begins to send $ADD (I, x_n)$ messages at the time when it enters PENDING. Algorithm $^*$ stops sending these messages when it enters WAIT_A, WAIT_B or WAIT_C. Finally, in order to ensure that every active station is in $O_n$, station $I$ is taken out from $O_n$ when algorithm $^*$ leaves WAIT_C, or when algorithm $^*$ enters CLEAR1.

In version 2 we have mentioned a scenario where a station $J$ may initiate a session $^*$ unlimited time after the transition of algorithm $^*$ out of WAIT_C. This scenario is also possible here, and here it is also necessary to clear the media of the ASSN messages intended for $J$ in the scenario, since in this version these are the messages that carry the ADDR messages (see description of version 2). However, the WAIT_D mode guarantees this clearing as in version 2 (see description of version 2) and the CLEAN and WAIT_E modes perform the same tasks as in version 2.

**SUMMARY**

This report presents three models for address distribution and session management, and suggests a protocol for each of the models.

The second and third models require some kind of communication between different modules in the manager. This connection appears for instance in the form of CLEAN messages of one address assignment algorithm being attached to POLL messages of another algorithm. The first model, on the other hand, relies only upon appropriate delays in the STOP GENERATE and CLEAN modes and does not require communication between different modules in the manager.

In addition, the second and third models are more suitable for networks where stations attach or leave the network rarely. This is because every such event requires additional communication and load on
the manager. On the other hand, in the first model the release of an address is only expressed by the additional delay experienced by the address protocol when making the transition from the WAIT_C mode to the UNASSIGN mode (the STOP_GENERATE and CLEAN modes). Therefore, the first model is more suitable for networks where attachments and detachments of stations are common events.

An interesting problem that results from the third model is whether it can be used in order to get rid of the assumption that there is a single, fail free address manager. Since in the third model every station knows the address of all other stations, it seems that every such station can become an address manager. The difficulty is mainly how to select and agree upon a new manager in case of failure of the old one and how to transfer to it the responsibility for address management. Another problem that seems to require additional research is how to implement the three models presented, together with their properties, in point to point networks with arbitrary topology.

REFERENCES:


1986, pp. 133-146

[M] J.F. McCool "FDDI: Getting to know the inside of the ring", Data Communications, March 1988, pp. 185-192


APPENDIX

We now prove lemma 5.2e, which is a basic property of the original address assignment protocol.

Lemma 5.2e:

Let $T_R$ be some time when $SA(I) = x_n$ and $I$ receives a $POLL I^I_x$ message in PACK0 mode. Let $T_S$ be the time when the $POLL I^I_x$ is sent by algorithm. Let $T_E$ be the first time before $T_S$ when algorithm enters POLL1, i.e. algorithm is in POLL1 during the entire interval $[T_E, T_S]$.

1. At $T_E$ holds $MA(I) = x_n$ and $SA(I) = x_n$ (i.e. $x$ is not undesignated in $[T_E, T_R]$).
2. $I$ is in PACK0 during the entire interval $[T_E, T_R]$.
3. According to the protocol, when station $I$ receives the $POLL I^I_x$ messages at $T_R$, it moves to the PACK1 mode. If $I$ does not send a $PACK I^I_x$ message (i.e. no $PACK I^I_x$ message leaves the outgoing queue of $I$) when in PACK1, then at $T_R + 2T$ station $I$ makes the transition to DEAD_STATE or REQUEST modes.

The above properties also hold when the $POLL I^I_x$ message is received when $I$ is in INIT_PACK0 mode. They also with $POLL 0^I_x$ replacing $POLL I^I_x$ when it is received in AACK or PACK1 modes.

Proof: We prove lemma 5.2e for the case where $I$ receives a $POLL I^I_x$ message in PACK0 mode. The other cases are proved similarly. First we prove sections 1 and 2, and the situation is depicted in Figure 1.

The proof is separated into the following claims:

Claim 1: At $T_E$ holds $MA(I) = x_n$.

Proof: Assume that $MA(I) \neq x_n$ at $T_E$. The possible situations are: $MA(I) = x_p$, $p \neq n$, or $MA(I) = y$ with $y \neq x$, or $I$ has no designated address.

It is impossible that at $T_E$ holds $MA(I) = x_p$, $p < n$, due to the following reason: Address $x_n$ is designated only after $x_p$ had been undesigned. If at $T_E$ holds $MA(I) = x_p$, then $x_n$ is undesigned at this time, and thus, since at $T_R$ holds $SA(I) = x_n$, there is a time $T$ in $[T_E, T_R]$ when algorithm enters PENDING
Staying in POLL1 mode

Station I

$T_E \quad T_S$

$POLL_{1_s}^*$

Station I receives a $POLL_{1_s}^*$ message in PACK0 mode

Algorithm* $T_E \quad T_S$

Claim 2: At $[T_E, T_R]$ holds $SA(I)=x_n$.

Proof: If $SA(I)\neq x_n$ at some time in $[T_E, T_R]$, then there must be a later time $T$ in $[T_E, T_R]$ when $I$ receives an $ASSN^n(I)$ message and performs $SA(I)=x_n$, which is the value of $SA(I)$ at $T_R$. However, at $T_E$ there are no $ASSN^n$ messages in M-I (lemma 5.2c) and it is impossible that $algorithm^*$ will send and $I$ will receive...
such messages in \([T_E, T_R]\), as shown in Claim 1.

Claim 3: Station \(I\) is in PACK0 during the entire interval \([T_E, T_R]\).

Proof: Algorithm \(^*\) enters POLL1 mode at \(T_E\). It can enter POLL1 either from INIT_POLLO or from POLLO. First assume that algorithm \(^*\) enters POLL1 from POLLO. Lemma 5.3c implies that in this case station \(I\) is either in DEAD_STATE, REQUEST or PACK0 modes at \(T_E\). In the present situation, \(I\) can not be in DEAD_STATE or REQUEST since while in these modes holds \(SA(I) = \text{nil}\), contradicting claim 2. The only possibility left is that at \(T_E\) station \(I\) is in PACK0.

Station \(I\) can leave PACK0 either to PACK1 after receiving a \(POLL1^*_I\) message or to DEAD_STATE if it releases its address. Station \(I\) cannot enter DEAD_STATE in \([T_E, T_R]\) since \(SA(I) = \text{nil}\) when in DEAD_STATE, again contradicting claim 2. Thus, during \([T_E, T_R]\) station \(I\) can only move to PACK1 after receiving a \(POLL1^*_I\) message.

Assume therefore that \(I\) moves from PACK0 to PACK1 after \(T_E\) but before \(T_R\). At \(T_E\) there are no \(POLL1^*_I\) messages in M-I (lemma 5.2c). Thus, it is clear that the \(POLL1^*_I\) message that \(I\) has received in PACK0, has been sent after \(T_E\). Notice now that station \(I\) is again in PACK0 at \(T_R\), and this can only happen after \(I\) has left PACK1 due to receiving a \(POLL0^*_I\) message in PACK1. Also notice that the FIFO discipline in the queues and channels implies that this \(POLL0^*_I\) message was sent after the \(POLL1^*_I\) message that caused the transition from PACK0 to PACK1. However, it is impossible that station \(I\) will receive such \(POLL0^*_I\) message in \([T_E, T_R]\), since the \(POLL1^*_I\), received in PACK0 has cleared M-I of all previous \(POLL1^*_I\) messages. Next, in \([T_E, T_R]\) algorithm \(^*\) sends only \(POLL1^*_I\) messages and finally the \(POLL1^*_I\) message, received in \(T_R\), disallows the possibility of \(I\) receiving a \(POLL0^*_I\) in PACK1.

In order to complete the proof, we still have to handle the case where algorithm \(^*\) enters POLL1 from INIT_POLLO at \(T_E\). Lemma 5.3c implies that in this case station \(I\) is either in DEAD_STATE, REQUEST or INIT_PACK0 modes at \(T_E\). Using similar arguments as before, the only possibility is that \(I\) is in INIT_PACK0. However, notice again that at \(T_R\) station \(I\) is in PACK0. The protocol of station \(I\) implies that \(I\) can move from INIT_PACK0 to PACK0 only after traversing PACK1. Again, using
similar arguments as before, it is easy to show that this is not possible during \([T_E,T_R]\), contradicting the fact that station \(I\) is in PACK0 at \(T_E\). Thus, in our scenario, the case where \(algorithm^*\) enters into POLL1 from INIT\_POLL0 is in fact impossible.

Claims 1 and 2 prove section 1 and claim 3 proves section 2.

We now prove section 3. The situation is depicted in Figure 2.

At \(T_E\) \(algorithm^*\) enters POLL1 mode, sends POLL1 messages and waits for PACK1 messages. At \(T_E\) there are no PACK1 messages in the system (lemma 5.2c). Lemma 5.2 implies that no two stations can have the same designated address \(x\) at the same time, and claim 2 implies that \(SA(I)=x_r\) in the entire interval \([T_E,T_R+2T]\). Thus, \(I\) is the only station with designated address \(x\) in \([T_E,T_R+2T]\). However, \(I\) is in PACK0 during the entire interval \([T_E,T_R]\) and so does not send PACK1 messages during this time. Also \(I\) does not send PACK1 messages in \([T_E,T_R+2T]\) where \(T_R+2T-T_E>T\). Thus, \(algorithm^*\) does not receive a PACK1 message in \([T_E,T_R+T]\) while staying in POLL1. Consequently, \(algorithm^*\) moves to WAIT\_C mode at \(T_E+T\) and does not send POLL0 messages at least until leaving WAIT\_C at \(T_E+4T\). The time interval when \(I\) is staying in PACK1, i.e. the interval \([T_R,T_R+2T]\) is included in \([T_E,T_E+4T]\) (notice that
$T_R - T_E < 2T)$. The $POLL_1$ message, which is received at $T_R$, has cleared $M-1$ of all previous $POLL_0$. Thus, it is clear that $l$ does not receive a $POLL_0$ message when in PACK1 and consequently will make the transition to the DEAD_STATE or REQUEST modes at $T_R + 2T$. 

\[\square\]