CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Address Resolution in One Lane Automated Highway Systems

Soheila V. Bana, Pravin Varaiya
University of California, Berkeley

California PATH Research Report
UCB-ITS-PRR-99-22

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

Report for MOU 318

July 1999

ISSN 1055-1425

Address Resolution in One Lane Automated Highway Systems

University of California at Berkeley
Soheila V. Bana and Pravin Varaiya

February 1999

Contents

Abstract

Address Resolution Protocols (ARP) are used in Automated Highway Systems (AHS) to establish communication among vehicles. The purpose of the ARP is to determine the network address of neighboring vehicles. We are proposing an innovative solution that takes advantage of the automated road infrastructure For providing addresses initially and uses the communication network itself for updating the network communication addresses in a one lane automated highway system.

The protocol has been modeled using PROMELA [1], and simulated and verified by SPIN [2]. SPIN is a tool for analyzing the logical consistency of concurrent systems, specifically of data communication protocols. The verification results show that automated vehicles in a single lane remain accurately informed about the communication addresses of their neighbors despite maneuvers (dynamics) on the road.

1 Introduction

An important problem in the design of vehicle-vehicle and vehicle-roadway data communication systems for the Automated Highway Systems (AHS) is the protocol design for address resolution in the communication network.

In any communication network, a user has a logical identifier (its IP address or social security number) and a physical location. The mapping between the logical identifier of a user and its physical location is called *binding*.

An *Address Resolution Protocol* (ARP) is a distributed algorithm that defines the binding and maps the logical identifier of a user to its physical location. In AHS applications, one of the addresses is determined by relative position, e.g., the vehicle that is immediately in front of my vehicle, and the other address is determined by an "absolute" address, e.g., the vehicle's license plate number.

In an Automated Vehicle Control System(AVCS), the coordination layer is responsible for the planning and supervision of maneuvers. Address resolution is especially important for the coordination layer because maneuver coordination among automated vehicles requires a local area network for communication. The difficulty is that such a local area network of vehicles is not only mobile, i.e., vehicles are moving and their distances are changing, but is also dynamic, i.e., vehicles are performing maneuvers and changing their relative positions with respect to each other. Thus the mobile local area network is continuously changing its configuration.

We are proposing an innovative solution that takes advantage of the high-tech road infrastructure[1] for providing addresses initially and uses the communication network itself for updating the network communication addresses.

The problem of address resolution for coordination layer communication has been divided into two parts: *initialization* and *updating*. *Initialization* refers to assigning each vehicle a network communication address and informing it of its neighborhood configuration as it enters the automated road. The lane number that the vehicle is in is also given.

Knowing the address information and relevant positions of its immediate neighbors, a vehicle is initially prepared, i.e., *initialized*, to network with other vehicles.

Once a network of vehicles is formed, this network itself can be utilized for the updating process of the address resolution. Since vehicles are initially aware of their neighborhood configuration and network communication addresses, each vehicle can contact the neighboring vehicles and inform them of the changes that it makes in the local area network configuration. This is what we call the *updating* process of address resolution. Protocols are designed so that updating is done by both informing other vehicles when one maneuvers and periodically confirming the current information with relative vehicles (*confirm*). In case some information is missing or not confirmed, there are protocols (*query*) to obtain the desired information.

[1]This infrastructure has been standardized by Title 21 of California Code and is currently being implemented for road pricing and toll collection in California.

This paper is organized as follows: section 2 overviews the research background in address resolution for mobile networks, section 3 discusses the abstract model of the information set that a vehicle in the coordination layer needs, section 4 explains our approach to address resolution for a single lane automated road while section 5 discusses the protocols and notes the shortcomings of the proposed address resolution scheme for a multi-lane automated road. Section 6 details the simulation and verification tool that was used and provides verification results for the proposed protocols, and section 7 concludes with an outline of our future work.

2 Background

The rapid growth of wireless communications systems along with the widespread use of networks has created the issue of mobile networking. Mobility should be distinguished from portability. Portability means the system (computer) can operate at any of a set of points of attachments; mobility is referring to operation *during the move* as well as operating at the attachment points. One of the most fundamental problems facing mobile networking today is *location management*: How does the network know where the intended recipient of a message is currently located? Who should be responsible for determining the user's location? The need for location management arises when the binding between the logical identifier and the physical location of a user is not fixed and changes over time[3].

The issue of binding is being discussed in different communities: the Internet community and cellular telephony.

The Internet protocol (IP) that is in charge of addressing and routing assumes that there is a close relationship between a computer's IP address and its physical location, i.e., the binding is fixed over time[4]. Moreover, IP and other inter-networking protocols are designed based on a hierarchical scheme to support scalability. The hierarchy that is seen in the logical identifier of a user in general reflects the physical connection of that user to the network. For example, in IP, the network address or logical identifier of a host is divided into two levels of hierarchy: a network number identifying the network to which the host is connected, and a host number identifying the particular host within that network. Routers within the Internet know (and care) only how to route packets based on the network number of the destination address in each packet; once the packet reaches that network, it is then delivered to the correct individual host on that network.

Source routing, the insertion of routing information into a datagram by the node from which it originates, is a proposal to overcome the problem of hierarchical routing based on physical location. A source that is aware of mobility of another user will specify the mobile's current physical address each time it sends a message and will not rely on its logical address, hence the term source routing. Strict source routing specifies the route hop-by-hop while loose source routing specifies the end user.

Considerable work has been done concerning mobile IP; however, there is not a standard set as of the time of this writing. One of the fundamental problems that faces the designers is compatibility of the proposed standard with the existing infrastructure. It is important for a mobile IP proposal to support addressing and routing to mobile hosts from existing correspondent hosts that have not been modified to support mobility and seem likely to remain so for some time.

The issues facing cellular telephony are similar to the problems of mobile IP and they in fact both use similar solutions to overcome the location management problem. For example, in most mobile addressing scenarios a mobile user has a home agent that maintains current location information for the mobile user and receives packets for it and *tunnels* them to the mobile user at its current location. We can draw an analogy between cellular telephony and mobile IP. A mobile terminal has permanent association with a home location register (HLR) and uses the service of a visitor location register (VLR) as it moves. HLR and VLR are analogous to home agent and foreign agent for a mobile terminal. There is also *call forwarding* by HLR that is similar to tunneling since it is HLR that has to find the current location of the mobile terminal. Both protocols are explained in more details shortly.

There are yet differences between the two technologies. For example, real-time ubiquitous binding and bandwidth are basic issues for cellular telephony with its increasing number of users. However, the buffers in communication networks can store packets until a valid binding is achieved. For similar reasons the bandwidth is assumed to be able to accommodate all potential users and not to be an issue in the near future.

We shall briefly discuss some of the prominent projects and proposals regarding mobile IP and cellular telephony and then will explain why address resolution methodologies in mobile IP and cellular telephony cannot be applied to a mobile network of automated vehicles.

2.1 Mobile IP

Sony's virtual IP proposal considers a virtual address and a physical address (VN, PN) for every mobile user. The virtual address, indicated by the VN part of the binding, is the permanent address (IP) of the user. The physical address, the PN part of the binding, is a temporary IP address acquired by the mobile host when it moves to a new network. The two addresses are identical if the user is at home or if the physical address is unknown. Binding is updated by the mobile user whenever it obtains a new physical address and is maintained at its home gateway. The home location is responsible for redirecting the incoming messages for the mobile user. In this scheme, senders and intermediate gateways can cache binding information and send packets directly to the current address of the mobile user.

The IBM proposal [3] defines an IP address for the mobile user plus a physical location that is the address of its current Mobile Support Station (MSS). The binding is the mapping between the permanent IP address of the user and the IP address of the current MSS. Unlike Sony's proposal, there is no need to acquire a temporary IP address for a mobile user.

The Columbia project is similar to the IBM proposal in defining the physical address as the address of the current MSS. However, there is no concept of a home location directory. The location directory is maintained at several MSSs that cover the mobile subnet within a campus. If binding information is not valid, a search is used by the MSSs to find the up-to-date binding. Finally, the logical identity of the mobile user is local only to his campus and changes for inter-campus, wide area moves.

The Internet Engineering Task Force (IETF) proposal also assumes a home host for a mobile user. When a mobile user moves, either it registers with a new foreign agent or obtains a care-of address. The home host is updated by the mobile user of its care-of address and redirects the incoming messages to the mobile user at its new care-of address. For routing optimization purposes, the home host can alternatively inform the sender of the new address and the messages are able to be sent directly to the mobile user.

2.2 Cellular Telephony and Personal Communication Networks

For location management in cellular telephony there are several schemes that have been used in practice for a while. In AMPS, which is very similar to GSM, a mobile user *roams*, i.e., registers with the Mobile Telephone Switching Office (MTSO) of its area that covers several base stations. In order to find the location of the callee, base stations within the MTSO are paged for the terminal equipment number. The appropriate cell then responds, completing the connection establishment. Thus, the initial set up is based on a search that may involve potentially a large number of base stations as the mobile user moves from one cell to another and active connection is automatically handed over to the new base station.

In CDPD, the mobile unit has a unique IP address and registers with the Mobile Data Intermediate System (MDIS) of its area. When it moves to another cell, it has to re-register with the new MDIS and the appropriate forwarding of packets from the old MDIS to the new one is provided.

Both AMPS and CDPD use a general concept of a "home agent". Location management for the future PCN is expected to face complex difficulties because of the high number of users that results in a high volume of database traffic due to location updates. It is desired to use a set of databases that are distributed in a hierarchical network. A user will have two addresses similar

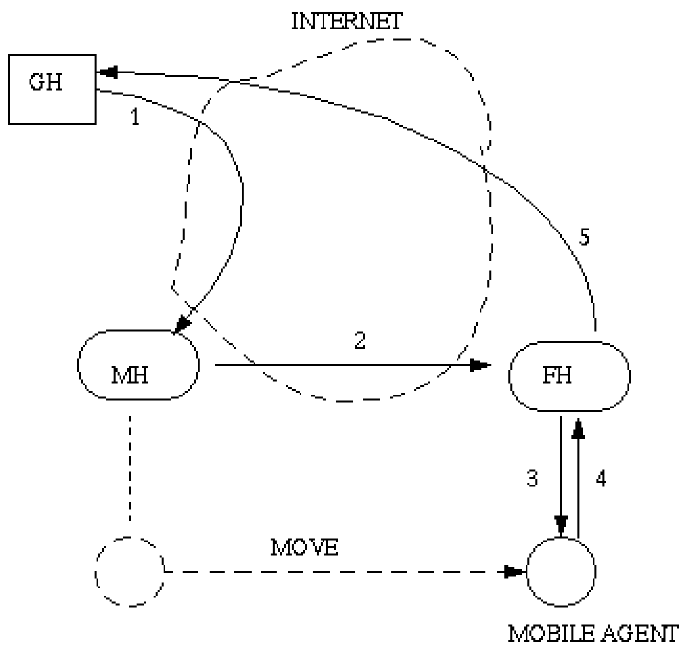

Figure 1: *A general host (GH) sends a message for the mobile agent to the mobile home (MH). The mobile home tunnels the message to the mobile agent via its foreign host (FH).*

to mobile IP proposals, and its home plays a role similar to that of the home agent in mobile IP. Instead of updating location information about its care-of address of the mobile station, the updating is deliberately limited to the region covered by several mobile stations. The imprecision is eventually eliminated by different types of locating algorithms or search strategies[3].

Hierarchical Topology and Support Stations

Mobile IP and cellular telephony have a couple of major similarities. One, they both use a hierarchical topology for network and addressing. Two, they both assume existence of at least some stationary stations such as a home agent, mobile support stations, and routers.

The above assumptions are not true for the type of mobile networks that we are interested in. In a mobile network of automated vehicles, there is no hierarchical topology of network; the network is flat. More importantly, there is no support station of any kind that its services of routing or redirecting could be assumed.

2.3 Ad-hoc Networks

An example of a flat network with no mobile support station is an *ad-hoc* network. It is the cooperative engagement of a collection of mobile hosts without the required intervention of any centralized access point[3].

From a graph theoretical point of view, an ad-hoc network is a graph, $G(N, E_{(t)})$, which is formed by denoting each mobile host by a node and drawing an edge between two nodes if they are in direct communication range of each other. The set of edges, $E_{(t)}$, so formed, is a function of time, and it keeps changing as nodes in the ad-hoc network move around. The topology defined by such a network can be very arbitrary since there are no constraints on where mobiles could be located with respect to each other.

Routing protocols for existing networks have not been designed specifically to provide the kind of self-starting behavior needed for ad-hoc networks. There have been routing methods proposed for ad-hoc networks that take an approach similar to the shortest path computation method[3]. Each node maintains a view of the network topology with a cost for each link. To keep these views consistent, each node periodically broadcasts the costs of its outgoing links to all other nodes using a multi-cast protocol such as flooding. As a node receives this information, it updates its view of the network topology and applies a shortest path algorithm to choose its next hop for each destination. Each node has routing table that lists all available destinations and the number of hops to each.

Each route table entry is tagged with a sequence number which is originated by the destination station. To maintain the consistency of routing tables in a dynamically varying topology, each station periodically transmits updates, as well as immediately transmitting updates when significant new information is available. However, there is no notion of synchronization and no assumption about the update periods between the mobile hosts. These packets indicate which stations are accessible from each station and the number of hops necessary to reach these accessible stations. The packets may be transmitted containing either MAC layer or network layer addresses. However, some of the link costs in a node's view can be incorrect because of long propagation delays, the partitioned network, etc. that might lead to formation of routing loops.

2.4 Network of Automated Vehicles

The address resolution protocol in conventional networks assumes that a computer is aware of its own binding, i.e., it knows its own network address and absolute address or identity. Hence, if one can address a computer by its absolute address, then one can ask for its network address and vice versa.

In the AHS context, each vehicle knows its absolute address, e.g., license plate number. Suppose now that vehicle A wishes to learn the network address of the vehicle in front of it. If it knows the latter's absolute address or identity (say, it is B) it can broadcast a message, "What is vehicle B's network address?" and then B could reply with its network address. Unfortunately, if all that A knows is that it wishes to send a message to the "vehicle in front of A", it cannot do so. Note that "vehicle in front of A" uniquely identifies B, but neither A nor B know that B is that vehicle. Indeed, if A broadcasts the message "who is in front of A?", all vehicles in A's neighborhood will hear this message, but will not be able to figure out which one of them is in front of A.

This is because the binding information that is important for every vehicle in the coordination layer is the mapping between the relative position of a vehicle and its logical identifier, i.e., $< relative\ position, ID >$. Focusing on one lane automated highway, the relative position means either immediately in front or immediately behind, called *front* and *behind*, respectively, for abbreviation. The ID could be the IP address or license plate number. More specifically, in a one lane automated highway a vehicle needs to know $< front, ID >$ and $< behind, ID >$ since the only other vehicles that matter to it from the coordination point of view are its *front* and *behind* vehicles. Even though a vehicle network, like an ad-hoc network, is flat and has no central support station for addressing, unlike ad-hoc networks, the amount of information for every vehicle is very limited. Besides, the binding database of one vehicle is *not* useful for another and every vehicle should have its own special database that reflects the relative position of other vehicles with respect to it.

Importance of Position and Relative Position

The most distinguishing characteristic of address resolution in automated highways is the importance of the location in binding. All other mobile schemes start with the logical identifier and address resolution is to find the binding map to the location. In an automated highway, the address resolution protocols shall start with the (relative) location of a vehicle and map it to a logical identifier to provide binding. Hence, tunneling and message forwarding are of no use in this scheme.

Puri et al.[5] considers the question of what information is needed to build an address resolution protocol for vehicles in automated highways. It is shown that inter-vehicle distances are insufficient, but relative coordinates by absolute locations are sufficient. Furthermore, it is shown that relative coordinates with local communication are insufficient. Assuming that each vehicle has the relative x- and y- coordinates of vehicles in a circle around it, and the communication is local and is not biased in any direction, it is shown that the problem of finding the vehicle in front becomes unsolvable under these conditions.

Puri et al. suggests the use of absolute position. However, we will show that in a one lane automated highway, absolute position is not necessary to resolve the relative position. An external point of reference, as we will see later, is sufficient to provide the information about relative position. The binding information can then be obtained using the network and implementing our proposed protocols.

It should be mentioned that address resolution is an important component of the coordination layer communication. Our research has benefited from the work that has already been done on the coordination layer, specifically, the design of controller by Varaiya[6], Hsu et al.[7], Eskafi[8], and Godbole[9]. The work on controller design for coordination layer, however, assumes that communication address is not a problem and builds on the assumption that address resolution protocols are available.

3 Modeling

We define some terminology for the purpose of modeling. The term *agent* refers to an automated vehicle that may be a single vehicle or a platoon leader; only single vehicles and platoon leaders are active in the coordination layer of AVCS. A follower in a platoon has to first become a leader or free agent in order to individually perform or participate in a maneuver. In the context of modeling, a follower is invisible in the coordination layer and only contributes to the "length" characteristic of its leader agent.

The implicit model of an agent in the coordination layer [8] is a system with a controller that is responsible for performing maneuvers. We add to this model an *addresser* which is in charge of address resolution for the agent.

We start with the system (agent) and its exosystem(road and other agents) and then modify the model according to some practical requirements. We show that the model we develop is necessary and sufficient for an agent to function and perform safely in the coordination layer.

3.1 Local Neighborhood of A System

An agent, say A, is a system that interacts with its exosystem which we call its neighborhood. We are interested in a specific segment of the neighborhood that is important for A in terms of the coordination layer, which we call the *local neighborhood*. The local neighborhood for A in the coordination layer consists of all the agents that *can* potentially perform a maneuver with A. Considering one-lane maneuvers only, the local neighborhood is reduced to A's *front* and *behind* agents. So a *local neighbor* of A is any neighboring agent of A who *can* potentially get involved in a maneuver with A. The set of the local neighbors make up the *local neighborhood*. Note that local neighborhoods are not the same for any two agents. Besides, a local neighborhood changes in time. Also note that the local neighborhood is all that an agent needs to be aware of in the coordination layer, i.e., having the network communication addresses of the local neighbors is necessary and sufficient for an agent to perform maneuvers in the coordination layer. It can request a local neighbor for a maneuver or can grant a request for a maneuver by a local neighbor. The addresser's job would then be defined as updating the addresses of the local neighbors. Once we see how the addresser takes care of address resolution, it becomes clear that the local neighborhood can be extended to n number of agents in front/behind if desired.

If we consider s^A the state of the local neighborhood of A, we see that the changes that happen to the network configuration over time make it a parameter of time and it becomes s_t^A. State s_t^A is specified by the local neighbors:

$$s_t^A = (Front; Behind)$$

"Front" and "Behind" are vectors that carry information about the the *front* and *behind* agents.

An exact model of the reality would include the ID of every local neighbor and its distances from A.

$$s_t^A = (Front(distance, ID); Behind(distance, ID))$$

"ID" contains the the communication address of the immediate neighbor and can either take the value of an address or *"unknown"*.

For example, knowing that at time $t = 1$ agent B is 250 meters in front of A and agent C at 130 meters behind A, then the local neighborhood of A is uniquely specified by

$$s_t^A = (Front(250m, B); Behind(130m, C)).$$

Note that there always exist some *front* and *behind* agents, even though the distance from A may be infinity.

3.2 Model Simplification

The model of the local neighborhood of A, represented by s_t^A, uniquely represents the local neighbors of A. However, there is information in this model that is not of any use for the coordination layer. The distance of the local neighbors to A is not useful information to the coordination layer, but is used by the regulation layer which has its own means of finding the accurate distance and will not be discussed here. So we can justifiably discard the distance from our model and replace it with a binary symbol that presents the *existence* of a local neighbor within a certain distance. The certain distance is determined by the hardware that is used for communication in the coordination layer and has a limited range. Based on this fact, we limit the model of the local neighborhood to the reliable range of communication hardware. This is because only the agents that can communicate can coordinate maneuvers, and, by definition of a local neighbor, if an agent is out of communication range of A, it is not considered A's local neighbor anymore.

The model then is simplified as:

$$s_t^A = (Front(Exists, ID); Behind(Exists, ID)).$$

The parameter "Exists" for "Front" and "Behind" agents takes binary value 1 when there is an agent within the communication range, and 0 otherwise. As a result, if there is an agent B behind A that is too far to communicate with A, it will not be included in the model.

Notice that this model is not continuous because it does not keep track of the actual distance of the agents. It is discrete because in practice the distance between agents is not a concern of the coordination layer, but of the regulation layer. What is important to an agent in the coordination layer is the existence and relative position of the local neighbors. So, we have simplified a continuous model of the local neighborhood into a discrete model. Next we see that this discrete model has to be valid in continuous time. A relatively continuous parameter is introduced that verifies the accuracy of the model in real time.

3.3 Model Accuracy Awareness

The proposed address resolution scheme relies on protocol exchange among agents, and communication failure is a possibility that should be seriously considered. A communication problem could result in incorrect modeling of the local neighborhood which is absolutely undesired. While we present no solution to overcome such problems, we propose a solution for awareness of such cases.

Before performing a maneuver, agents should verify their models with each other, and an agent who is aware of a possible inaccurate model of the local neighborhood by itself or a local neighbor will not attempt to perform or grant a request for a maneuver. Awareness of the accuracy of the model of the local neighborhood is an important factor that is represented by a flag. *Flag* is a binary parameter that shows whether the model is accurate or not. Since the information about the front of the vehicle is independent from the information about the rear of the agent, there are two flags which are independent of each other:

$$s_t^A = (Front(Flag, Exists, ID); Behind(Flag, Exists, ID))$$

Notice that the flag has to be continuous. This is because the prompt request by a controller for a maneuver is not synchronized with the addresser and could be at any point in the continuous time. This means that the discrete parameter of "existence" of a local neighbor has to be updated continuously. The solution that we offer is *relatively continuous* updates of the information. The coordination layer moves are mechanical motions by the agents, while the communication of protocols is electronic and propagated by electro-magnetic waves, i.e., orders of magnitudes faster than mechanical motion of the vehicles. The addresser updates the model of the local neighborhood at such a high frequency that it seems "continuous" to the coordination layer while it is actually a very high frequency discrete event; hence the term "relatively continuous".

3.4 Modeling the Transition Periods (Maneuvers)

During a maneuver, the "ID" parameter update could take as long as a maneuver takes to change. The solution to this is to reflect the maneuver process in the modeling information. So the model adopts one more binary parameter to represent the status of the local neighbors, i.e., busy or not:

$$s_t^A = (Front(Flag, Exists, ID, busy); Behind(Flag, Exists, ID, busy))$$

3.5 Addresser and Modeling

It should be mentioned that each agent supposedly has similar information about itself. An addresser must be aware of such information about its own agent in order to communicate it with local neighbors. The model that addresser must be updating then is extended to include self ID and status:

$$s_t^A = (Self(ID, busy); Front(Flag, Exists, ID, busy); Behind(Flag, Exists, ID, busy))$$

The above model represents the status of A and its local neighbors. This model is maintained by the addresser such that its parameter values reflect the real time status of the agents. Addresser of A communicates with other agents on the road and updates the above modeling parameters. In the coordination layer, only agents, leaders and single vehicles, need to have an updated model. A follower does not need to have an addresser; it would be a waste of bandwidth to have it.

The only maneuver a follower performs is split, which is in cooperation with its leader and can be done via intra-platoon communication. When the maneuver is done, it will become an agent itself and its addresser will be activated.

3.6 Updating Condition

To make the model complete, the model should include one more parameter, *leader*.

$$s_t^A = (Self(ID, leader, busy); Front(Flag, Exists, ID, busy); Behind(Flag, Exists, ID, busy)) \quad (1)$$

The binary parameter *leader* is added to the model so that when it is set to 1, it reflects the status of a vehicle as an agent (platoon leader or single vehicle). A follower could split from a platoon and become an agent, in which case its leader parameter becomes 1. Similarly, it becomes 0 after a join.

The model is only updated by addresser when $leader = 1$, i.e., the vehicle is a leader in the coordination layer and has both *access* to the communication channel and the *need* to update the model. The addresser is active for agents only; a follower does not have an addresser.

When B splits from A, the initializing of B and providing initial network configuration of its local neighborhood is done by A. As it is shown in the protocols in upcoming sections, A asks B to set its *front ID* to A, and its *behind ID* to C, while itself switches its *behind ID* from C to B, and also informs C to switch the *front ID* from A to B. The simulation Figure (9) is an illustration of this process.

4 Address Resolution

As it can be observed in modeling, the important component of the binding map is the relative position (front/behind) and address resolution is required to find the logical identifier (ID or IP address) for each relative position.

Breaking up the problem of address resolution into *initialization* and *updating* is the important principle of this design. Initialization is done externally, while updating is an internal task of the network itself. The attractive feature of this design is that after initialization, the network itself becomes responsible for maintaining its address resolution, i.e., updating protocols are designed such that each node is constantly aware of its local area network configuration and relative network addresses. Furthermore, the network itself becomes in charge of *initializing* a new node after a split. As Puri et al. proves, without an external reference (initialization) it is impossible to resolve the logical address in the binding.

In this section, address resolution algorithm, initialization and updating will be explained. In the following sections, after presenting the protocols and their simulation and verification, we will discuss theoretical generalization of the external initialization and internal updating of dynamic networks.

4.1 The Algorithm

The proposed address resolution has a very simple algorithm. It takes advantage of Zermelo's *well-ordering theorem*[10] which states that every set may be well-ordered. We choose the order of the position of agents, $\mathcal{A}$, in one lane to order the agents $(\mathcal{A}, \leq)$. We then define a one-to-one mapping between this well-ordered set and a well-ordered set of integers $(\mathcal{Z}, \leq)$ where the mapping preserves the order.

The algorithm to find the relative position of agents with respect to each other uses the ordering of their maps to compare the integers. Suppose we choose their position from front to back of the line of agents for ordering them such that an agent A in front of agent B is smaller than B, $A \leq B$. This well-ordered set is mapped into a set of integers. For example, the agents in Figure (3) will be mapped: $A \longmapsto 1$, $B \longmapsto 2$, $C \longmapsto 3$.

The integers correspond to the ID numbers of the agents. In practice, every agent is assigned a number which is greater than the number assigned to the vehicle in front of it and smaller than the number assigned to the vehicle behind it. This becomes the absolute ID of the agent on the road. It is a unique serial number that specifies where this vehicle is with respect to other agents. The algorithm is a comparison of ID numbers. Two agents compare their ID numbers, and the agent with a larger serial number is in front of the other one. Similarly, a set of agents can compare their serial numbers and each would find the same exact order of agents. This provides a simple means of picturing their positions from front to back according to their ID numbers.

Theorem 1 *Address resolution of agents through local communication is possible by well-ordering them according to their position in one lane.*

Proof:Assume that an agent, say A, wants to find out who is its *front* neighbor. It asks all its front agents for their ID numbers and receives some responses and compares them. In a non-empty, well-ordered set of its front agents it can identify its *immediate predecessor*, because "there is a least number greater than"[11] its own ID number. So it can find its *front* neighbor. Similarly, an agent can find its *behind* neighbor.

When agents A and B compare their ID numbers, they find out either $A \leq B$ or $B \leq A$. Both cases cannot be true simultaneously unless $A = B$. For every two agents there is only one way of ordering the IDs in $\leq$ relation. By induction, the models carried by single agents resemble pieces of a jigsaw puzzle that when put together provide an accurate model of their position ordering. Well-ordering of the agents and communicating the ID numbers among agents make figuring out the correct position order of agents possible for all of them. $\sharp$

It should be emphasized that assigning ID numbers to the leaders in a linear order is the most important job of the road station. It is the core of initialization. The updating protocols as we will see are capable of realizing and updating other parameter values in the state model by a simple algorithm using these ID numbers.

4.2 The Initialization Infrastructure

The infrastructure for initialization is special hardware on the side or top of the road that establishes short-range back-scattering communication with vehicles.

The short-range communication between the vehicle and the reader provides a means of reliable individual communication between the automated vehicle and the road. The actual physical device that is used for this purpose is a small card-key-size tag [12] that is installed on the front windshield and is connected to the vehicle's computer and to the *reader* that is installed on the road. The reader generates either modulated signals that provide reader-to-tag communication or un-modulated sine waves to be modulated by the tag and reflected back to the reader which becomes the tag-to-reader communication *,back-scattering.* This scheme uses CRC-16[2] for high accuracy and is economical since the vehicle does not generate any signals and does not need any energy source[3]. The tag and the reader are capable of reading and writing to each other while the vehicle is moving at freeway speed. Consequently, the reader can provide every vehicle information about its front vehicle(s) that just passed by the reader. Similarly, if one more reader is employed with appropriate distance prior to the first one, it can provide information regarding its following vehicle(s). In a similar manner, more readers can be employed in the adjacent lanes in order to provide information about the vehicles in other lanes. It is assumed that the readers are locally connected to each other and can exchange data.

4.3 The Initialization Process

The information required to initialize a vehicle consist of the parameter values for its state model as in equation (1):

$$s_t^A = (Self(ID, leader, busy); Front(Flag, Exists, ID, busy); Behind(Flag, Exists, ID, busy))$$

Before we explain how the infrastructure can assign proper values to the state model parameters, we need to explain some assumptions about the road station and how it measures the distance between vehicles. The initialization road station in addition to the reader has some sensors and a simple computer that can analyze the data received from the sensors, the reader, and other road stations.

The assumption is that during initialization, no maneuvers are allowed until vehicles find their place in the coordination layer as leaders, receive their communication IDs, and become able to network and communicate together. The vehicles are then moving at a relatively constant velocity. Located next to the vehicle-detector, there would be a speed-sensor on the road that could relay the average speed of a vehicle to the computer. The computer, equipped with a timer, knows of

[2]This is according to Title 21 of California Code that requires error detection code with a generator polynomial of $X^{16} + X^{12} + X^5 + 1$

[3]There is a small battery in the tag that lasts a few years.

the time lapse between the passage of two consecutive vehicles and can calculate their distance:

$$d = v.t$$

. The accuracy of v is not crucial to realize whether two consecutive vehicles belong to the same platoon or not.The large difference between distances in the two cases make it easy to verify whether a vehicle is following the previous one even if v is not very accurate. For example, if the sensor has a one second or more lapse between sensing two vehicles with speeds of $20m/s$ to $30m/s$, it can assert that the two vehicles do not belong to the same platoon because their distance would be more than $20m$. On the other hand, the sensor's time lapse must be shorter than 0.2 second for vehicles in the same platoon. The sensor information then enables the computer to assign the *leader* value to vehicles and to assign serial numbers to the leaders, which we assume to be their communication IDs.

The information about the *front* neighbor similarly can be given to a leader. For instance, if the passage of a leader has been within v/R seconds prior to passage of A, then that leader would be identified as A's *front* agent and its information will be given to A, where v is the average speed, and R is marginally greater than the range of communication hardware. Note that the computer could be conservative and overestimate R for leaders. Its error in any case is not crucial as long as the leaders are assigned *unique consecutive serial* numbers as their IDs.

Assuming that there are two sets of readers and sensors that are connected to the computer within v/R distance, and that by the time agent A is passing through the second set another agent is passing by the first set, the computer would inform A via the second reader about its *behind* agent. Again, this is a possibility and is not necessary.

Initialization provides the essential addressing that is necessary for the agents to figure out their relative position with respect to each other and establish communication with their local neighbors. Initialization could provide addresses of local neighbors such that agents can immediately start maneuvering. However, maneuvers change the network configuration and the communication addresses need to be updated with respect to the new positioning of vehicles after each maneuver.

Before we explain how addressers resolve the communication addressing issue using the initialization assignment, we present a simple algorithm for initialization of a follower by a platoon leader.

4.4 Initialization by Platoon Leaders

The main purpose of initialization is to provide binding information, i.e., to map an agent's position to a serial number or communication ID. Later, we will explain how the information about local neighbors is provided.

A vehicle that passes the initialization road station as a follower is not initialized, i.e., it does not have a serial number and does not know of its local neighbors. Once this vehicle decides to split from its platoon leader and become an agent itself, it needs to have a serial number that is unique and in a linear order with respect to other agents' IDs, and that reflects its position in the line of agents on the road. Furthermore, initialization of other vehicles that split from the original platoon leader must be consistent with the first one. Also, the first vehicle that splits may have followers who in turn may want to split. The serial numbers that will be assigned to these vehicles must reflect their relative positions on the road with no ambiguity. While we use integers for road initialization, we consider using real numbers for future initialization due to splits and changing lanes. The use of real numbers assures an unlimited possibility of split and change-lane maneuvers.

An example would be useful in presenting the algorithm that we suggest for initialization by platoon leaders. Consider vehicle B in Figure (2) which is about to split from agent A. A is to provide a proper serial number for B. Assume that the serial number assigned for agent A by road initialization is 5. Agent A would add nine decimal points to its own serial number and assign 5.9 to B if B is the first follower to split. To the next follower that splits, A would assign 5.8, and to next one 5.7 and so forth. This way, A can only assign 9 serial numbers, i.e., it can have a maximum of 9 splits. However, if we change the base for our numbering, we can increase it to any desired number.

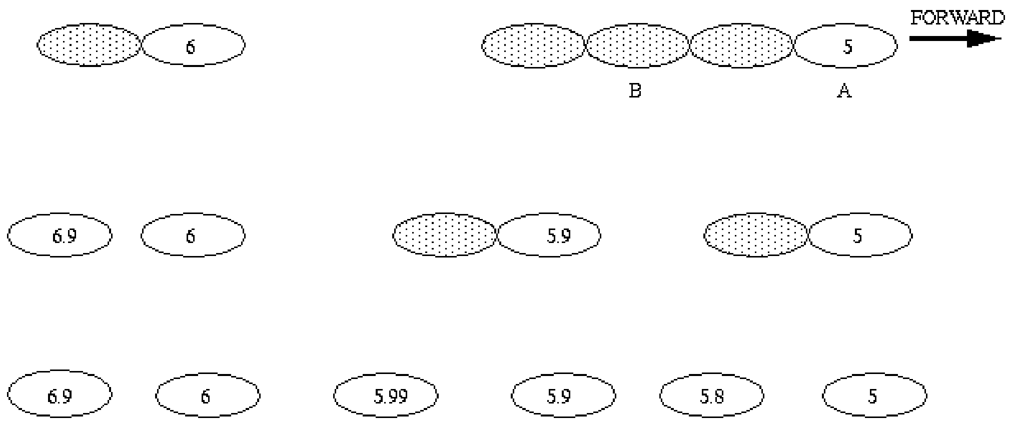

Figure 2: *Platoon leaders initialize splitting vehicles.*

If a follower is to split from B, it would be assigned 5.99 if it is the first one, 5.98 if it is the second one, and so forth. The agent number 5.99 would assign numbers from 5.999 to 5.991 to its splitting followers. Such an assignment algorithm keeps the serial number maps of the well-ordered agents appropriately unique with respect to their relative position on the road.

Note that the agent behind A would assign 6.9 to 6.1 to its splitting followers, and they in turn would be assigning numbers in the range of 6.99 to 6.11. This way, the initializations by different platoon leaders do not interfere with each other. Serial numbers remain to be maps of well-ordered agents that preserve their position order. Hence, comparison of serial numbers remains a reliable algorithm for finding agents' relative position with respect to each other.

The total number of digits that would be used in such assignments may be limited by the memory of the vehicle's computer, communication packet's length, etc. Consequently, the number of splits from an original platoon leader may be limited.

4.5 The Updating Process

We discuss the updating process with the assumption that initialization has already provided all the parameter values for the state model. It will be shown later that providing serial IDs for agents is necessary and sufficient for configuration of the network and updating process. Assuming the initial network is established, this network itself can be utilized to update the network addresses of agents.

There are three types of maneuvers: *join, split* and *lane-change*. The first two, join and split maneuvers, do not change the relative position of vehicles with respect to each other, but only change the grouping of vehicles into different platoons. In the case of these two maneuvers, there is no need to know the absolute position of vehicles, since the relative position is constant. For example, if agent A is immediately in front of B and B is in front of C and so forth, it will always be so as long as there is no lane-change maneuver performed. Enter and exit are also special cases of lane-change maneuver that will not be considered here. Address resolution updating is simply done by keeping track of the maneuvers performed by the agents as they change their platoon groupings. The designed communication protocols relay the information about the maneuvers and relative position of agents.

In every agent, the maneuvers are controlled by a *controller*[6]. We propose an *addresser* for every agent to be in charge of the address protocol exchange and address resolution. Note that while in practice controller and addresser could be combined, they are different in abstract.

The controller is in charge of controlling the maneuver, while the addresser provides the network communications addresses to the controller. The controller's communication range is limited to the *front* and *behind* agents for maneuver purposes, while the addresser can contact many agents in a wide range of communication that is limited only by hardware capability and power and interference problems. Finally, the controller is in the coordination layer, while the addresser can have a range of services that go beyond the coordination layer. The interaction between the two is important: the controller provides the addresser with information about maneuver status such as being requested, performing, done, etc., while the addresser provides the controller with the updated networking addresses. In summary, the addresser is different from the controller in its functionality and range of communication access.

The implementation of the updating process is done by two types of communications.

- Rendezvous communication between the addresser and the controller of every agent.
- Periodic communication among addressers of local neighbors in a local area network.

Rendezvous communication between the addresser and the controller can be illustrated by an example. Assume agent B is behind A and wants to join it. The maneuver is controlled by the controllers of the two agents. B's controller obtains A's address from its own addresser and requests for maneuver from A. A's controller, upon granting the request, informs its own addresser of its busy status. Similarly, B's controller, upon receiving acknowledge from A, informs its own addresser of its busy status. Both controllers update their own addressers when the maneuver is done. Meanwhile, A's and B's addressers cooperatively inform the addresser of the agent immediately behind B about the maneuver so that agent, say C, is aware of the maneuver, realizes when its *front* agent is busy, and updates its *front* address appropriately. So, when an agent is engaged in a maneuver, its immediate neighbors' addressers are aware of its busy status and update their address book accordingly.

As a result of the communication between the addresser and the controller of each agent during maneuver, all the involved agents are aware of it and update any changes to the network addresses and configuration.

5 Address Resolution Protocols

The proposed protocols work based on the assumption that there is an initialization provided by the road infrastructure to all the vehicles. The initialization mainly provides the essential basis of information for the protocols to work. It would also provide initial information regarding the network configuration to the vehicles, though later we will see that in theory it is not essential, and the agents can figure it out by protocols. Every agent is initially equipped with an accurate model of its local neighborhood in the coordination layer[4]. There is one important factor that is provided by initialization that makes the updating process possible, and that is a unique serial number that is assigned to every agent. The unique linear order of the numbers reflects the unique order of positioning of the agents on the road in a single lane. The serial numbers then resemble the absolute ID for agents. Any set of agents can compare their serial numbers and conclude the same unique ordering of their positions on the road. This is specifically used in case of *query* protocols which will be explained shortly.

The protocols are of two types:

- *confirm.*
- *query.*

A *confirm* protocol is exchanged between any *two* agents who are already aware of each other as local neighbors. To confirm that their information about each other is current, they update it periodically. Each agent performs its own *confirm* protocols, even though it responds to the

[4]Initialization could potentially provide more information for other layers as well.

confirm protocols of its local neighbor. This is double checking of information by agents. The *confirm* message would contain the following information.

$$confirm \mid receiverID \mid front \mid sender\ ID$$

and its response, *ackfirm* (or acknowledge to *confirm*), would contain:

$$ackfirm \mid receiverID \mid front \mid sender\ ID.$$

Confirm is a uni-cast protocol.

In case an agent, say A, has no local neighbor in front of it, it cannot use communication to confirm *not having* a local neighbor. Instead, it is actively searching for a local neighbor as it travels along the road. Disregarding possibilities of communication failure, if it comes close enough to an agent, i.e., within the communication range, that agent will respond and A will recognize that agent as its *front* neighbor by comparing their serial numbers. Lack of any responses would mean lack of any potential neighbors.

In case there are multiple responses to A's request, A will compare the serial numbers and find out which one is immediately in front of it. A will repeat its *query* and save the responses until there are no more responses. It waits long enough to make sure it is not missing any responses to its *query* and then goes through a serial number comparison.

The content of a *query* message would be as the following:

$$query \mid front \mid senderID$$

and a response to it would be:

$$query\ response \mid receiver\ ID \mid front \mid sender\ ID.$$

Note that if there is no communication failure, A will receive responses from all the agents in front of it and within its communication range, and there is no possibility of missing the correct answer (agent ID) for the *front* neighbor of A.

Query is a multi-cast which implies that the coordination layer has multi-cast channels available to the agents in addition to the one-to-one channels.

Having presented the format of the messages, we now view the protocols' diagrams as seen in Figures (3) to (6). As the protocols reveal, the addresser and the controller can be combined in practice for practical purposes.

The addresser updates its information about the local neighborhood periodically by *confirm* protocols. While addressers are exchanging *confirm* protocols, controllers can start performing a maneuver using the same communication channel that the addressers use for *confirm*. There could be a segment dedicated to the controller message in each period of *confirm* message.

The flow diagram for protocols of a join maneuver of agents A and B in Figure (3) is presented in Figures (4), (5), and (6). The flow diagrams for split maneuvers are similar. The protocols are designed such that at any moment any agent is aware of the communication address and status of its *front* and *behind* agents. While B is performing a join, C is aware of it and sets the status of its *front* agent to *busy*.

Once the maneuver is over, A and C will update the related information of local neighbor IDs.

The fact that after the join maneuver the agent B is only a follower of A and plays no role in the coordination layer is reflected by the binary flag *leader* that becomes zero.

5.1 Absolute Position and ARP

The lane-change maneuver, however, changes the relative ordering of the agents. The periodic updating of the relative position of agents is very important in this case. Agents cannot rely on initialization even after a very short period of time. This is because different velocities in two lanes changes the relative positioning of agents. Unlike the first two types of maneuver, a lane-change maneuver demands the knowledge of absolute positions for updating the communication addresses with respect to relevant positions. Once agents know their absolute positions, they can compare it with each other and, knowing their lane number, can find out their precise relative positioning.

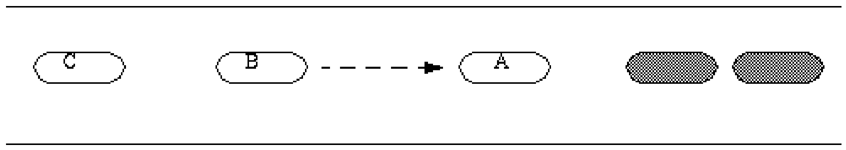

Figure 3: *Agent B wants to JOIN agent A, C is behind B, and its Front ID is affected by the JOIN maneuver.*

6 Simulation and Verification

The protocols are simulated and verified without worries about the problems regarding the communication channels; the end-to-end error-free communication has been assumed between any two addressers within the communication range.

Since the focus is on studying the structure of the protocols and verifying their completeness and logical consistency, an overall description of the protocols has been expressed in a modeling language, and some issues in protocol design, such as the message format, encoding, etc., have

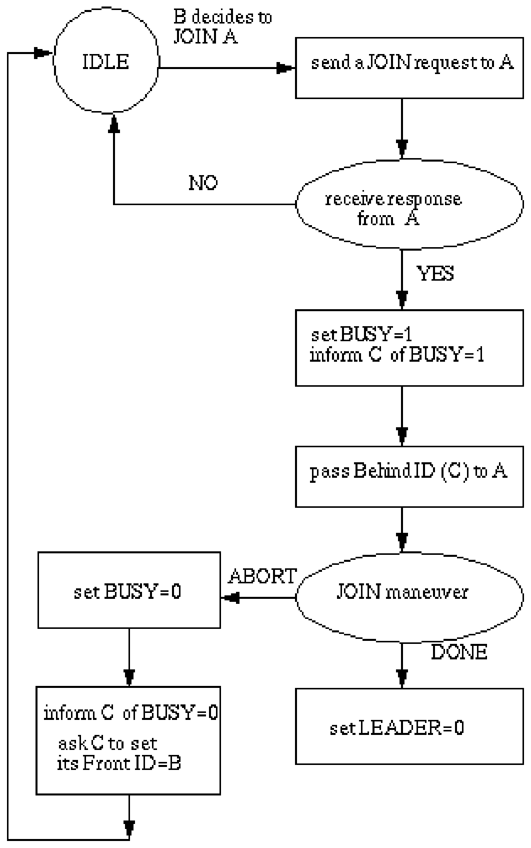

Figure 4: *Agent B wants to JOIN agent A, B requests a JOIN with A.*

been ignored. Also, simplifying assumptions have been made such as the serial numbers assigned to the agents by initialization are equivalent to their communication addresses. This allows for concentration on the design and verification of the rules that govern the interactions among addressers of different agents.

The protocols then have been simulated and verified for *safety* and *temporal claims* which will be discussed shortly.

6.1 PROMELA

PROMELA is a modeling language that is used to express the address resolution protocols. It is a set of notations for the specification and verification of procedure rules[1].

The model of the local neighborhood that was suggested for every addresser is modeled by values that are kept track of in the PROMELA model. Initialization of the addresser is modeled by setting the initial values in the PROMELA model.

The reason for using PROMELA is that it is very simple, yet sufficiently powerful to represent our communication model. It allows for concurrent processes, such as rendezvous communications between different addressers or between the addresser and the controller of the same agent.

A brief description of PROMELA is useful for understanding the protocols. There are three specific types of objects in the language[1]:

- Processes.
- Message Channels.
- State Variables.

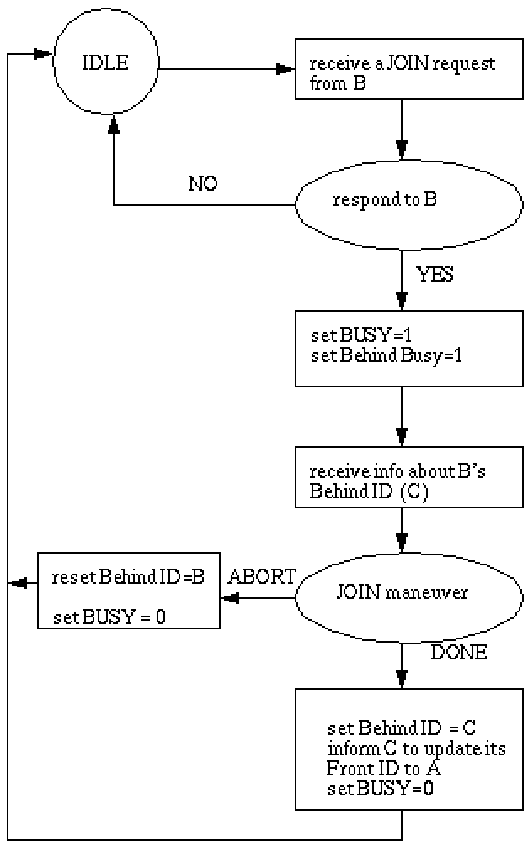

Figure 5: *Agent B wants to JOIN agent A, A responds to a JOIN request by B.*

We have used a *process* (defined by a proctype) to describe an agent's addresser. Message channels could be defined locally or globally and could store messages. In rendezvous communication that is used here, there are no stored messages in the channels. There are different types of variables and data-types from a single bit to a 32-bit integer.

What is important in PROMELA is the executability of statements. It is best defined by the PROMELA designer, G.J. Holzmann[1].

> There is no difference between conditions and statements. The execution of a statement is conditional on its *executability*. Executability is the basic means of synchronization. A process can wait for an event to happen by waiting for a statement to become executable.

There is a special command for running concurrent processes in PROMELA, the atomic command, that forces the addressers to be able to interleave and execute their statements together. For example, every addresser is modeled by a proctype and an addresser is sending a *confirm* message to its local neighbor and waiting to receive an *acknowledge*. The next executable step will be receiving the *acknowledge*, and, if it is not received, its process will not be executable and would stop if receiving the *acknowledge* is its only option to proceed. In our protocols there is another option, *query*, which, in the case of losing a local neighbor to the far distance, allows the addresser to query for its local neighbor.

Absolute time is not defined in PROMELA. Time is not an issue since the main point is the relevant timing of steps, not their absolute time of happening. Statements are executed one at a time. There is no parallel execution of separate processes and even using a counter for counting the units of time is not meaningful because, while it is counting, no other statement is being executed. For example, in the actual protocol an agent would re-transmit the *confirm* message a few times before it assumes that its neighbor is out of range and then start a *query*. This would be to rule out any channel problems. But in PROMELA it does not make sense to re-transmit a message because the focus in on the essential function of the protocols away from worries about the channel. Repeating messages are not modeled. That is why some *wait* periods that will be in the actual protocols have been omitted in the modeling by PROMELA. However, the time sequence of execution steps by different addressers can be traced as interleaved steps.

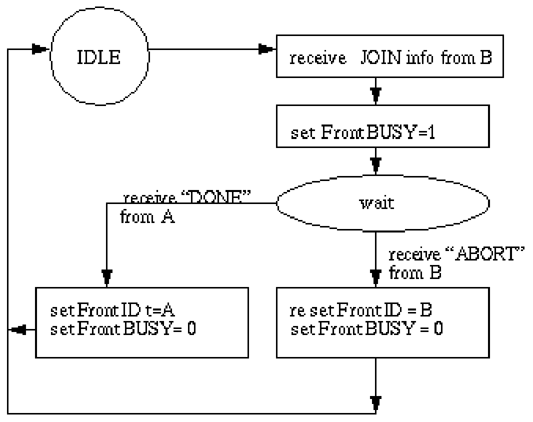

Figure 6: *Agent B wants to JOIN agent A, C is behind B, and its Front FID is affected by the JOIN maneuver.*

6.2 SPIN

Verification of the proposed address resolution protocols have been done by SPIN which is short for *simple PROMELA interpreter*[1]. Spin is a tool for analyzing the logical consistency of concurrent systems, specifically of data communication protocols[2]. It takes a model system specified in PROMELA and can either perform random simulations (Figures (7) to (10)) or verify the system's correctness properties as specified by the protocol designer. It is capable of searching for the absence of deadlocks, unspecified receptions, unexucutable codes, unreachable states, and non-progress cycles or live-locks. The verification results can be viewed in an output file, as in Figure (11) and Figure (12). Also, a state diagram of each process can be generated which shows the directed graph of finite states for each addresser. Spin can be run with different options depending on the desired type of simulation or verification. Xspin is the window-based interactive version of spin and its version 2.9.7 has been used for this project.

6.3 Simulation

Simulation has been used in preliminary debugging of the protocols. Also, the simulation output provides an illustration of the protocols function as depicted in Figure (7) to Figure (10). The address resolution protocol exchange between agent A of Figure (3) and its local neighbors is simulated as their relative positions to each other change. It should be mentioned that the protocol exchange with the *front* neighbor is independent of the communication with *behind* neighbors and vise versa. This simulation is about the address resolution protocols for A and its *behind* neighbors. In practice such ARP could similarly be applied to its front neighbors simultaneously without one ARP communication affecting another. The exception is, of course, when A is engaged in a maneuver with its *front* neighbor and, at the same time, its *behind* neighbor is requesting a maneuver. To take care of such a possibility in simulation, the *busy* flag has been modeled as a random bit that could be arbitrarily "0" or "1".

Figure (7) is a depiction of the *confirm* protocol transmitted by A to its local neighbor, B, who in turn acknowledges the *confirm* by transmitting *ackfrim*. At some point, B decides to join A and asks if A is busy. A may or may not be busy, depending on whether it has just engaged in the maneuver with its *front* agent or not, and answers B accordingly while transmitting the *confirm* protocol. Note that A would inform B about its busy status as soon as it grants a request for maneuver. The busy flag is to avoid granting two requests for maneuver from *front* and *behind* agents at the same time. Once B receives a *not-busy* response from A, it request a *join* which again may or may not be acknowledged because of the platoon size restriction.

Figure (8) illustrates the *join* process between A and B, while C is being informed of the change in its *front* neighbor's address. The neighborhood state model of addressers then is updated.

B becomes a follower after the maneuver and its *leader* binary flag turns 0. However, before it loses its access to the communication channel in the coordination layer, it informs C about its maneuver so that C's addresser can update its model of the local network configuration.

As shown in Figure (9), a follower like vehicle B can request from its leader, agent A, a *split*. If A is not busy, it would acknowledge the request and inform its *behind* agent about the maneuver to be performed. The *behind* agent, C, then updates its model of the *front* neighbor to busy status. Note that this protocol is focusing on the protocol exchange between addressers, and not the controllers'. Recall that real time makes no sense in this kind of simulation and verification; so, the sequence of the steps take equal amount of time, not reflecting the maneuver time. As soon as A acknowledges the request, it provides the communication ID of C for splitting B, and informs C about the maneuver and similarly provides the communication ID of B for C. This protocol exchange is happening while the controllers are performing the maneuver. When the maneuver is done, the controller of the agent informs its addresser, and the addresser informs other addressers. In this case, A informs B that the maneuver is done and starts exchanging *confirm* protocols with B, while B engages in a similar protocol exchange with C.

As shown in Figure (9), vehicle B as a follower is out of the picture in the coordination layer until its request for *split* is granted. It then is initialized by its leader, agent A, and its addresser

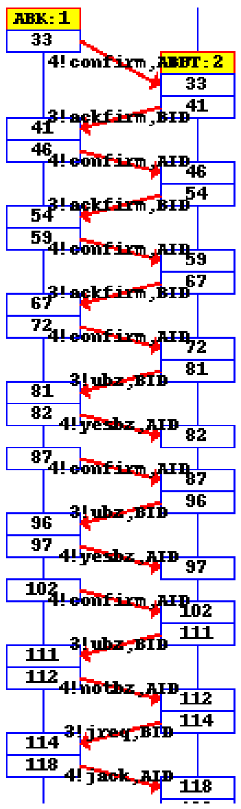

Figure 7: *Agent A, left, is exchanging* confirm *protocol with agent B, right, when B asks for* join.

is given the proper model of local neighborhood.

Note that address resolution protocol exchange is happening simultaneously with the controller's communication. In practice such simultaneous communications could refer to using different segments of the same communication packets. Since the real time is not meaningful for such simulation and verification, the time that the maneuver takes is not sensible in the addressers' communication. It is important that upon every maneuver, the involved agents update their local neighbors about the changes in the configuration. In this case, A not only initializes B, but also informs C about splitting B and provides its communication ID to C as its future *front* agent.

When an agent's model of its local neighborhood implies that it does not have a *behind* neighbor, it will transmit a *query* to find out about its current local neighbor. As in Figure (10), it will be sending its communication address along with the question about who is behind it in the lane. An agent who hears A and does not have a *front* agent would respond to the *query*. A would wait for some time to make sure it has already collected all the responses. Then it would compare the received serial numbers and find out the nearest agent. In order to make sure that A receives all the possible responses, other agents keep transmitting until A acknowledges receiving their responses. It would also notify each respondent whether that agent is its *behind* neighbor. Once it finds its *behind* neighbor, it would start a *confirm* protocol exchange with it.

6.3.1 Safety Statements

Informally, safety refers to avoiding bad states that should never happen. In other words, it refers to the correctness of the protocol. Simple propositions can make correctness claims for PROMELA models. Such propositions can make a claim about all the elements of a system state: local and global variables, control-flow points of arbitrary execution processes, and the contents of message channels. The propositions implicitly define a labeling of states with *assert* label.

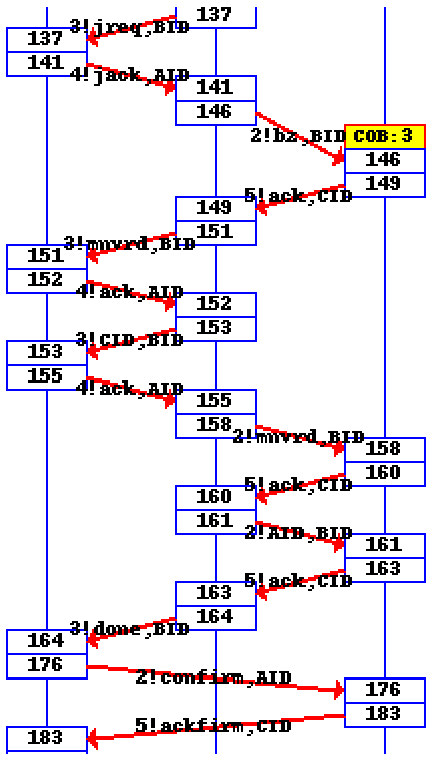

Figure 8: *Agent A, left, performs a* join *with B, center, and agent C, right, is informed by both A and B about their maneuver.*

We have used *assert* label to claim that *once a maneuver happens, the addressers of all local neighbors, including self, are aware of the change in the network configuration and reflect it in the models that they have.* Specifically, *the assertion is that a change in the network configuration is correctly reflected by the parameters in the addresser's model.*

6.3.2 Temporal Claims

Temporal claims are to specify an ordering of propositions. Within one process, the order of statements shows the order of their executions, while in concurrent processes there is no guarantee about the relative speed of different processes. However, rendezvous communication is what makes the communication among two addressers synchronous, and we can verify ordering of propositions in different addressers. Again, since we are not allowed to make any assumptions about the absolute time and relative speeds of concurrently executing processes, the only valid interpretation of the word "after" in PROMELA is *eventually* after.

We have used temporal claims to verify the functionality or *fairness* of the protocols, e.g., once an agent asks for *join*, its request will *eventually* be granted. Similarly, when a follower asks for *split*, it will eventually receive an *acknowledge*. This verification along with no error result of the safety statements shows that *changes do happen and are reflected in the models by the addressers.*

6.4 Verification

Figures (11) and (12) show the result of the *safety claim* and the temporal claim verification, respectively. The correctness assertion claims and the temporal claims can be viewed in the appendix.

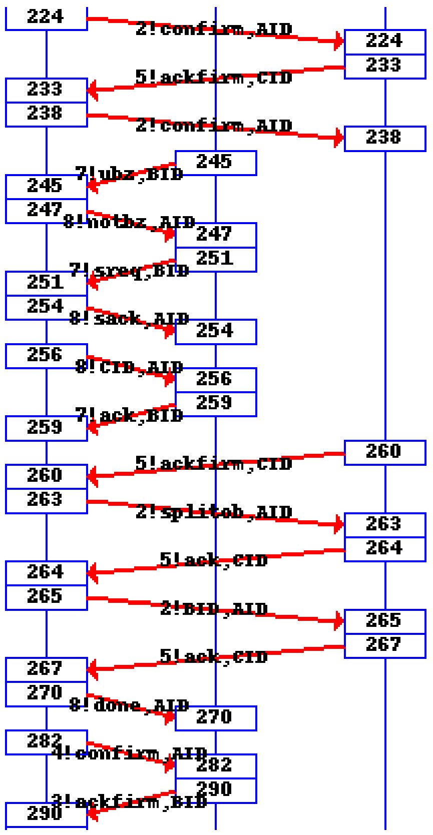

Figure 9: *B, center, asks for* split *from A, left.*

7 Conclusion and Future Work

A network of automated vehicles requires communication for coordination. Address resolution is essential for communication of automated vehicles. This specification application demands a specific binding for address resolution, i.e.,

$$< relative\ position,\ communication\ address > .$$

The particular binding along with the non-hierarchical and dynamic characteristic topology of the network distinguish this network from other mobile networks such as Internet and cellular telephony in a manner that their address resolution schemes cannot be applicable for our application.

The same characteristics of the network that distinguish it from other mobile and dynamic networks have been exploited in the design of address resolution protocols for one-lane automated highways. The proposed design takes advantage of the infrastructure of automated highways to provide an initial binding to the vehicles. Once a network of automated vehicles is initialized, the designed protocols are used to maintain and update the addresses.

In a one-lane highway, the relative position of vehicles does not change due to maneuvers. Therefore, the network that is formed by initialization allows for a correct updating process. However, this is not the case in a multi-lane automated highway where the relative position of vehicles in different lanes would change due to maneuvers. In this case, knowledge of vehicles absolute position is necessary for performing the updating process in multi-lane automated highways.

Our future work is focused on designing address resolution protocols in multi-lane automated highways. The problem of determining a vehicle's position must be resolved. Next, we plan to consider a set of design schemes for multi-lane address resolution based on bandwidth requirements, efficiency, latency, and the possible impact on the overall design of the communication architecture for automated highways.

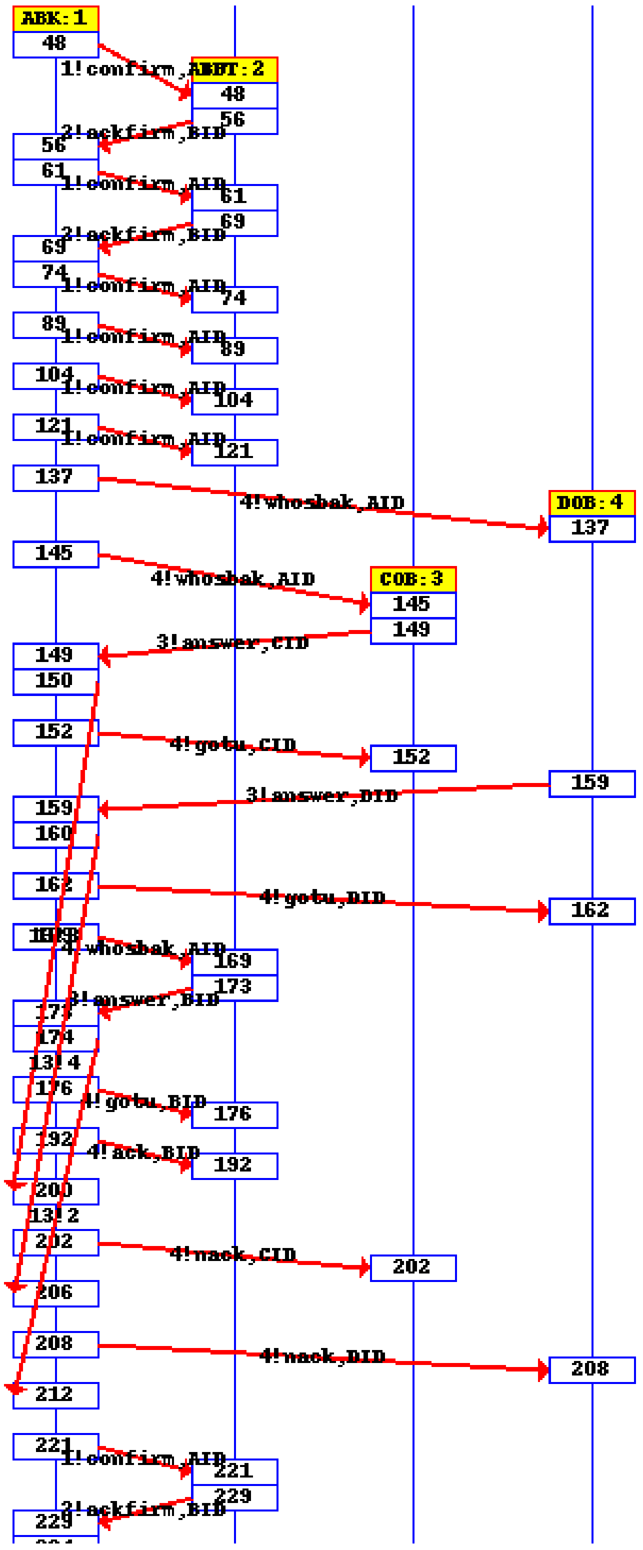

Figure 10: *Agent A, left, loses B, second from left, as its neighbor and queries around. B, C, and D, on the right, respond. A successfully finds the nearest neighbor, B.*

```
(Spin Version 2.9.7 -- 18 April 1997)
        + Partial Order Reduction

Full statespace search for:
        never-claim             - (not selected)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid endstates       +

State-vector 160 byte, depth reached 23215, errors: 0
  108332 states, stored
   25012 states, matched
  133344 transitions (= stored+matched)
       2 atomic steps
hash conflicts: 12036 (resolved)
(max size 2^23 states)

Stats on memory usage (in Megabytes):
18.200  equivalent memory usage for states (stored*(State-vector + overhead))
        compressed State-vector = 113 byte + 8 byte overhead
13.083  actual memory usage for states (compression: 71.89%)
33.554  +memory used for hash-table (-w23)
2.400   +memory used for DFS stack (-m100000)
1.133   +memory used for other data structures
50.086  =total actual memory usage
```

Figure 11: *Verification Output: Safety Claims*

```
(Spin Version 2.9.7 -- 18 April 1997)

Full statespace search for:
        never-claim             +
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness enabled)
        invalid endstates       - (disabled by never-claim)

State-vector 172 byte, depth reached 4975, errors: 0
   19475 states, stored
    7925 states, matched
   27400 transitions (= stored+matched)
       2 atomic steps
hash conflicts: 11444 (resolved)
(max size 2^19 states)

Stats on memory usage (in Megabytes):
3.506   equivalent memory usage for states (stored*(State-vector + overhead))
        compressed State-vector = 129 byte + 8 byte overhead
2.665   actual memory usage for states (compression: 76.02%)
2.097   +memory used for hash-table (-w19)
0.240   +memory used for DFS stack (-m10000)
0.284   +memory used for other data structures
5.204   =total actual memory usage
```

Figure 12: *Verification Output: Temporal Claims*

References

[1] G. J. Holzmann, *Design and Validation of Computer Protocols.* Prentice Hall Software Series, 1991.

[2] G. J. Holzmann, *Basic Spin Manual.* Bell Laboratories, Murray Hill, NJ 07974, http://cm.bell-labs.com/cm/cs/what/spin/Man/Manual.html.

[3] T. Imielnski and H. F. Korth, *Mobile Computing.* Kluwer Academic Publishers, 1996.

[4] C. E. Perkins, *Mibile IP Design Principles and Practices.* Addison-Wesley Wireless Communications Series, 1998.

[5] A. Puri and P. Varaiya, "Simple results on communication with neighbors," 1996.

[6] P. Varaiya, "Smart cars on smart roads: Problems of control," *IEEE Transactions on Automatic Control*, vol. 38, Feb 1993.

[7] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya, *Protocol Design for an Automated Highway System.* Kluwer Academic Publishers, Boston, 1993.

[8] F. Eskafi, *Hierarchical Hybrid Control of Automated Highway System s.* California PATH Research Report, UCB-ITS-PRR-95-8, 1995.

[9] D. N. Datta, *Hierarchical Hybrid Control of Automated Highway Systems.* California PATH Research Report, UCB-ITS-PRR-95-8, 1995.

[10] N. Dunford and J. Schwartz, *Linear Operators, Part I: General Theory.* John Wiley and Sons, 1988.

[11] P. Chernoff, *Berkeley Mathematics Lecture Notes, Volume 4, Lectures on Topology and Analysis.* Berkeley Mathematics, 1993.

[12] L. d. A. Tip, "Reflecting tomorrow's highways today, rf backscatter reflection in avi systems," 1996.

Appendix

```
/*******************************************************/
/*There are 3 vehicles on the road in the following order, */
/*moving to the right:                               */
/*                                            */
/*       --->             ---->           ----->           */
/*       C                B                A               */
/*                                            */
/*The protocol is for A when it does confirm with its back */
/*agent(B or C), and when it looses its back agent and does*/
/*a query (multi-cast) and B and C respond. Also when B    */
/*JOINs A and then SPLITs, it models A and C that respond  */
/*to its request and ARP update.                     */
/*                                            */
/*The vehicles have their serial numbers at initialization,*/
/*CID,BID,AID, (where the corresponding numbers are such   */
/*that (CID > BID > AID) plus ftID(front ID) and bkID      */
/*(back ID) that are also provided at the initialization.  */
/*                                            */
/*Information such as myLDR(am I a LEADER?) and their      */
/*communication channels are set. For example, at the      */
/*initialization the channels are set as Bout==Ain         */
/*(uni-cast channels), because B and A know of their       */
/*relative positions, and so forth.                        */
/*                                            */
/*The comments along the file are to explain the           */
/*protocol and the verification claims in PROMELA.         */
/*                                            */
/*                                            */
/*******************************************************/

mtype {confirm, ackfirm, gotu, nack, ack,  whosbak, /* message types */
      answer,  done, ubz,yesbz,notbz, bz, sreq,
      splitob, joinob, sack, snack, jreq, jack,
      jnack, mnvrd,null, CID, BID, AID};
int   dist,distnc;                          /* integers */
bit   distbit,dist2,Aimbz,Abkbz,Aftbz,   /*bits  that represent flags*/
            Bimbz,Bbkbz,Bftbz,   /* Aimbz == A: I'm busy, a flag of A */
            Cimbz,Cbkbz,Cftbz;     /* that shows A is busy */
                              /*similarly for B & C */


        chan Bin = [0] of {mtype, mtype};  /* communication channels */
        chan Bout = [0] of {mtype, mtype}; /* in and out of each vehicle */
        chan Qin = [0] of {mtype,mtype};   /* note that B has two sets: */
        chan Qout = [0] of {mtype,mtype};  /* one set in coordination layer*/
        chan Cin= [0] of {mtype, mtype};  /* another when B is follower */
        chan Cout= [0] of {mtype, mtype};
        chan Bfin= [0] of {mtype, mtype};
        chan Bfout= [0] of {mtype, mtype};
        chan Afin= [0] of {mtype, mtype};
        chan Afout= [0] of {mtype, mtype};

proctype ABK(     bit  myLDR)       /* only ARP for BACK of A is considered */
                          /* ABK is ARP of A with is BACK agents */
                              /*Cin,Cout,Bin,Bout are for confirm
                              communication(uni-cast),
                              while Qout, Qin is
                              for inquiry(multi-cast).  */
                              /*Bfin and Bfout are for B when
                              it becomes a follower of A*/
{     int   count;
      mtype       myID=AID,
            bkID=BID,
            ftID=null;
      chan  dummy=[5] of {byte};
      chan  inbk;
      chan  outbk;
      byte  var,adr1,var1,maybk,dumb,adr[5];
      bit   bkex;
      Aimbz=0;
      Aftbz=0;
      Abkbz=0;
      inbk=Bout;
      outbk=Bin;
START:      if
      ::bkID == 0 -> bkex =0; goto INQBK
      ::bkID != 0 ->
                  if
                  ::inbk?confirm,var -> outbk!ackfirm,myID;goto START
                  ::timeout -> goto S1BK
                  fi;
      fi;
S1BK: skip;
endA: outbk!confirm, myID;
      distbit =1- distbit ;
      if                /* randomizing maneuvers (if chooses one */
      ::distnc=1        /* option randomly)                       */
      ::distnc=2
      ::distnc=3
      ::distnc=4
      fi;
      if
      :: (dist< 4) -> dist=dist+1
      :: (dist== 4) -> dist=0
      fi;
CHKBK:      if
      ::inbk?ackfirm,var -> goto S1BK
      ::inbk?ubz,var-> if
                  ::Bin!yesbz,myID ; goto S1BK
                  ::Bin!notbz,myID ; if
                              ::Bout?jreq,var;goto JOINBK
                                    fi
                        fi;
      ::Bfout?ubz,var->if
                  ::Aimbz==1-> Bfin!yesbz,myID; goto S1BK
                  ::Aimbz==0-> Bfin!notbz,myID; goto SPLIT
                  fi
      :: timeout -> goto INQBK
      fi;
INQBK:      Qout!whosbak,myID;
      if
      ::(count<3) ->count= count+1;goto ANYBK
      ::(count==3) -> count=0; goto NEXTBK
      fi;
ANYBK:      skip;
      do
      :: Qin?answer,adr1 -> dummy!adr1;
                  Qout!gotu,adr1;
                  if
                  ::adr1>myID &&maybk==0 -> maybk= adr1 ;
                  :: adr1> myID && adr1<maybk -> maybk =adr1;
                  :: adr1> maybk && maybk!=0 -> skip
                  fi;
      ::Qout!whosbak,myID
      :: timeout -> goto NEXTBK
      od;
      skip;
NEXTBK:     if
      ::maybk==0 -> goto INQBK
      :: maybk!=0 -> bkex=1; bkID=maybk;
                  if
                  ::maybk==BID-> inbk=Bout;outbk=Bin; Qout!ack,BID;
                  ::maybk==CID-> inbk=Cout;outbk=Cin; Qout!ack,CID;
                  fi; maybk=0
      fi;
      do
      ::dummy?dumb-> if /* sending nack to wrong responders */
                  ::dumb!=bkID && dumb != 0 -> Qout!nack,dumb
                  :: dumb==bkID||dumb==0 -> skip
                  fi
      ::timeout -> break
      od;
      goto S1BK;
JOINBK:     if                      /* this is JOIN protocol */
      :: Aimbz=1;              /* A could be busy or not (random "if")*/
      Abkbz=1;           /* it also sets its flag of back-agent to busy */
            outbk!jack,myID;     /* then send acknowledge for join (jack) */
            inbk?mnvrd,var;
            outbk!ack,myID;
            inbk?var1,var; bkID=var1;
            outbk!ack,myID;
            inbk?done,var;
            Aimbz=0;
            Abkbz=0;
            outbk=Cin;
            inbk=Cout; goto S1BK
      :: outbk!jnack,myID; goto S1BK
      fi;
SPLIT:      skip;       /* similarly, this is the split protocol */
P:    Bfout?sreq,var; /* A receives B's request for split and randomly */
      if                 /*acknowledge or rejects it. */
            :: Aimbz=1;
                  Abkbz=1;
slet:             Bfin!sack,myID;
                  Bfin!CID,myID;
                  Bfout?ack,var;
                  inbk?ackfirm, var;outbk!splitob,myID;
                  inbk?ack,var;
                  outbk!BID,myID;
                  inbk?ack,var;
                  Bfin!done,myID;
                  Aimbz=0;
                  Abkbz=0;
                  outbk=Bin;inbk=Bout; goto S1BK
            :: outbk! snack,myID; goto S1BK
                  fi;
edBK: skip


}

/* B only replies to CONFIRM and INQUIRE from A, its own ARP is not modeled here
because parallel processing in PROMELA does not happen */

proctype BFT(     bit myLDR     /* this process type represents B */
{     int   num,
            count,
            m4,m5,m6,m7,m10;
      byte  var,adr;
      bit   ftex,
            bkex;
      mtype       myID=BID,
            ftID=AID,
            bkID=CID;
      chan  inft;
      chan  outft;
      chan  inbk;
      chan  outbk;
      Bimbz=0;
      Bftbz=0;
      Bbkbz=0;
      inft=Bin;
      outft=Bout;
      inbk=Cout;
      outbk=Cin;
      if
      ::bkID==0 -> bkex=0
      ::bkID!=0 -> bkex=1
      fi;
STFT: if
      ::ftID==0 -> ftex=0
      ::ftID != 0 -> ftex =1
      fi;
S1FT:   skip;
endB: if
      ::Qout?whosbak,adr-> if
                        ::ftex==1 && bkex==1 -> goto S1FT
                        ::ftex==0 /* || bkex==0 */ -> goto ANSFT
                        fi
      ::inft?confirm, var -> goto S2FT
      ::distnc==1&& myLDR==0 ->goto SPLIT
      fi;
S2FT:  if
       ::distnc==1 && dist<=1 ->
                        ftex=0;goto S1FT
       ::distnc==2 || distnc==3 /* && dist>=3*/ -> outft!ubz,myID;
                       if
```

```
                        ::inft?notbz,var -> goto JOINFT
                        ::inft?yesbz,var-> goto S1FT
                        fi
      ::outft!ackfirm,myID; goto S1FT
/* note: else really does not mean else, it means just another option! */
       fi;
ANSFT:      Qin!answer,myID;
      Qout?gotu,var;
WAITFT:           if
            ::Qout?ack,var  -> ftex=1; ftID=adr; goto S1FT
            ::Qout?nack,var -> goto S1FT
            /* num=num+1; */ /* not needed in SPIN, but needed to count
                              number of periods in real protocol. */
            /*    if
                  :: num <= 5 -> goto WAITFT
                  :: num==5 -> goto edFT
                  fi */
            fi;
JOINFT: outft!jreq,myID;
      if
      ::inft?jack,var ->
      Bimbz=1;
assert(Abkbz == 1);                 /* ASSERT statement: asserts that*/
      Bftbz=1;                /* when B is busy, A has set its flag*/
assert(Aimbz == 1);                 /*of back-agent to busy. Also, when */
                              /*B sets its front-agent flag to busy, */
                              /*A is really busy, i.e., has set its */
                              /*busy flag to 1. */
                  outbk!bz,myID;
                  inbk?ack,var;
            if
            :: bkex == 1 -> outft!mnvrd,myID;
            inft?ack,var;
            outft!bkID,myID;
            inft?ack,var;
            outbk!mnvrd,myID;
            inbk?ack,var;
            outbk!ftID,myID;
            inbk?ack,var;
            outft!done,myID;
            myLDR=0;
            goto S1FT

            :: bkex ==0 -> outft!0;
                   myLDR==0;
                   goto edFT
            fi
      ::inft?jnack,var -> goto S1FT
      fi;

SPLIT:
      Bfout!ubz,myID; if
                  ::Bfin?yesbz,var ->goto S1FT
                  ::Bfin?notbz,var ->goto ASKS
                  fi;
ASKS: Bfout!sreq,myID;
      if
      ::Bfin?nack,var -> goto S1FT
      ::Bfin?sack,var ->
                  myLDR=1;
                  Bfin?adr,var; bkID=adr;ftID=var;
                  Bfout!ack,myID;
                  Bfin?done,var;
                  ftex=1;
                  inft=Bin;
                  outft=Bout;
                  goto S1FT;
      fi;
edFT: skip;
}

/* C  OBserves the maneuver between A and B and updates its info and comes
into contact with A via B */


proctype COB (     bit  myLDR /* this process type represents C */
{     int num;
      byte var,adr;
      bit   ftex;
      mtype myID=CID,
            ftID=BID,
            bkID=null;
      chan  in, out;
      Cimbz=0;
      Cftbz=0;
      Cbkbz=0;
      in = Cin;
      out= Cout;
STFT:   skip;
endC: if
      ::in?confirm,var->
                  out!ackfirm,myID;
                  goto STFT
      ::in?mnvrd,var ->goto S1
      ::Qout?whosbak,adr -> goto S0FT
      ::in?splitob,var ->
                        out!ack,myID;
                        in?adr,var; ftID=adr;
                        out!ack,myID;
                        goto STFT;
      ::in?bz,var ->
      Cftbz=1;          /*when C sets its front-agent flag to busy,*/
assert(Bimbz==1);       /* its front agent is really busy,*/
      out!ack,myID;
      goto STFT
      fi;
S1:   out!ack,myID;
S2:   in?adr,var;
      ftID= adr;
      out!ack,myID;
      goto STFT;

S0FT:   Qin!answer,myID;
      Qout?gotu,myID;
WAITFT:
      if
      ::Qout?ack,CID -> ftex=1; ftID=adr; goto STFT
      ::Qout?nack,CID-> goto STFT
      /*:: timeout -> goto WAITFT*/ /* num=num+1;*/;
      fi;
/*    if
      :: num <= 5 -> goto WAITFT
      :: num>5 -> goto STFT
      fi; */
edFT: skip;
}
init {                        /* This command runs the processes and
                              atomic command forces them to run
                              simultaneously. */
      atomic{
      run ABK(1);
      run BFT(1 );
      run COB(1 );
            };
      }
never {                       /* Temporal claims: never is used
                              to state what is undesired and make sure
                              it does not happen.            */
      do
      ::skip
      ::Bfout?[sreq,BID]-> goto accept0   /* B as a follower asks for split */
      ::Bout?[jreq,BID]->  goto accept1   /* B asks for join */
      od;
accept0:
            do
            ::!Bfin?[sack,AID]       /* ! negates the statement==> it is
                              never the case that B asks for a maneuver
                              and its request is not acknowledges */
            od;
accept1:
            do
            ::!Bin[1]?[jack,AID]
            od
}
```