Group Membership and Communication in Highly Mobile Ad Hoc Networks

Dissertation

Linda Briesemeister

Group Membership and Communication in Highly Mobile Ad Hoc Networks

vorgelegt von
Diplom-Informatikerin
Linda Briesemeister

Von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktorin der Ingenieurwissenschaften
– Dr. Ing. –
genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr.-Ing. Adam Wolisz
Berichter:	Prof. Dr.-Ing. Günter Hommel
	Prof. Dr. Reinhard von Hanxleden

Tag der wissenschaftlichen Aussprache: 5. November 2001

Berlin 2001
D 83

Danksagung

An erster Stelle möchte ich meinem Professor Günter Hommel danken für die ständige Unterstützung und Beratung während der letzten Jahre. Er hat auch in schwierigen Zeiten Vertrauen in diese Promotion investiert, das ich hoffentlich nicht enttäuscht habe. Zweitens gilt mein Dank dem Zweitgutachter Prof. Reinhard von Hanxleden, der dieser Promotion zusätzliche Impulse gab.

Diese Forschungsarbeit entstand größtenteils während meiner Zeit bei der DaimlerChrysler AG. Dort gebührt mein Dank allen ehemaligen Kollegen – insbesondere Günter Heiner, Prof. Wolfgang Merker und Stefan Böttcher – für die unzähligen Arten der Hilfe bei meinem Promotionsvorhaben. Als ehemaliger Kollege, aber auch als ein gestandener Promovierter beriet mich Dr. Ralph Sasse auf unermüdlichste Art. Ihm danke ich für die unbezahlbaren Ratschläge und Motivation, wenn das Ziel mal in weite Ferne gerückt war.

Nicht unerheblichen Anteil an dem Gelingen des Promotionsvorhabens hatten natürlich meine engsten Freunde und meine Familie. Vor allem Greg Nemet hat sich immer und kurzfristig um alle Fragen zur englischen Sprache gekümmert. Mein Bruder Malte hatte immer schnell die Lösung zu diversen Computerproblemen zur Hand. Schließlich danke ich auch meinen Eltern für die gelungene Balance zwischen Freiraum und Ratschlägen, die mich bis heute begleitet hat.

November 2001, Linda Briesemeister

Zusammenfassung

In dieser Arbeit wird eine neue Art der Nachrichtenvermittlung für die Kommunikation in hochmobilen Ad-hoc-Netzen vorgestellt. Ad-hoc-Netze arbeiten drahtlos und verwenden keine ausgezeichneten Stationen oder feste Infrastruktur. Aufgrund der Mobilität der Knoten wechselt die Netztopologie ständig und ist nicht vorhersagbar.

Diese neuartige Nachrichtenvermittlung wird hier im Kontext einer direkten Kommunikation zwischen Fahrzeugen erforscht. In solchen hochmobilen Ad-hoc-Netzen kennen die Knoten üblicherweise nicht von vornherein die Identität ihrer Kommunikationspartner. Außerdem erschweren die dynamische Netztopologie und die knappe Bandbreite der Kommunikationswege den regelmäßigen Austausch von Informationen über den Netzzustand. Dies macht ein neues Paradigma der Nachrichtenvermittlung erforderlich, in welchem die Nachrichten implizit an ihre Empfänger adressiert werden und der Empfang von der momentanen Situation abhängt.

Dabei erreicht der Urheber mit einer Nachricht durch gezieltes und kontrolliertes Fluten des Netzes (*flooding*) die Bestimmungsorte. Die Empfänger einer solchen Nachricht verwenden ihr Wissen über ihr Umfeld, um zu entscheiden, ob die Nachricht für sie bestimmt ist. Weiterhin wird der Algorithmus zur Nachrichtenvermittlung derart angepasst, dass das Problem zeitweise getrennter Netzkomponenten besser bewältigt wird.

Der hier vorgeschlagene Algorithmus zur Nachrichtenvermittlung erfordert den Zusammenschluss mehrerer Knoten zu einer dynamischen Gruppe. Daher wird ein neuartiger, lokaler Gruppenbildungsdienst spezifiziert, der sich an den schwierigen Verhältnisse in Ad-hoc-Netzen orientiert. Bei dieser lokalen Gruppenbildung verfolgen die Knoten lediglich die Mitgliedschaft ihrer unmittelbaren Nachbarn.

Um die entwickelten Algorithmen zu bewerten, wird die Anwendung des Ad-hoc-Netzes in einem Autobahnstauszenario simuliert. Es werden verschiedene Metriken eingeführt, die die Leistungsfähigkeit der implemen-

tierten Protokollschichten messen. Die Ergebnisse der Simulationen zeigen die beeindruckende Wirkung vergleichsweise einfacher, verteilter Protokolle, die über ein hochmobiles Ad-hoc-Netz kommunizieren und in einer realistischen Umgebung interagieren.

Abstract

This thesis proposes the use of a new routing paradigm to enable communication in highly mobile, ad hoc networks, which operate wirelessly in the absence of dedicated master stations or fixed infrastructure. Due to the mobility of the nodes, the network topology changes frequently and unpredictably.

We explore the new routing paradigm in the context of inter-vehicle communication. In such highly mobile ad hoc networks, the nodes commonly do not know the identity of their communication partners in advance. Rapid topology changes and scarce bandwidth prevent the nodes from exchanging updates regularly throughout the network. Therefore, we advocate a new routing paradigm that implicitly addresses message destinations based on the current situation of the network.

The originator of a message uses scoped and controlled flooding to reach the destinations. The receivers of the flooded message use their knowledge of the local environment to decide whether they match the intended destination of the message. Furthermore, we tailor the routing algorithm to overcome the problem of fragmentation in sparsely connected networks.

Our routing algorithm requires the mobile nodes to aggregate into a dynamic group. Being aware of the severe conditions inherent to ad hoc networks, we suggest a new, localized group membership service in which nodes track the membership only of adjacent nodes.

To evaluate our communication system, we simulate the mobile ad hoc network applied to a highway traffic jam scenario. We introduce several metrics to measure the performance of each implementation layer. The results of the simulations demonstrate the powerful effect of rather simple, distributed protocols communicating through a mobile ad hoc network and interacting in a realistic environment.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile computing is proliferating as devices are becoming smaller, cheaper, and more powerful. By combining mobile devices with wireless communication abilities, the vision of being connected anytime anywhere will soon be a reality. New applications arise from mobile entities interacting and collaborating towards a common goal. With cellular phones already widely employed, and the mobile Internet emerging into the market place, concepts of dynamic wireless networks that do not depend on expensive infrastructure draw attention to the rapidly expanding research area of so called ad hoc networks. The miniaturization of both computers and wireless communication devices, plus the availability of sophisticated global positioning techniques enable a novel approach in networking: Mobile devices constitute a mobile ad hoc network when they directly and wirelessly communicate with other devices nearby without any fixed infrastructure. The mobile nodes move and thus the network topology changes dynamically and frequently. The absence of any hierarchy, established infrastructure, or centralized administration forces the nodes to control the network on their own. We quote the definition of a mobile ad hoc network from the charter of the corresponding Internet Engineering Task Force (IETF):

> "A 'mobile ad hoc network' (MANET) is an autonomous system of mobile routers (and associated hosts) connected by wireless links—the union of which forms an arbitrary graph. The routers are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a standalone fashion, or may be connected to the larger Internet." [IET]

The benefits of ad hoc networks appeal to applications like conferences, meetings, disaster relief, rescue missions, and battlefield operation. Such scenarios typically lack a central administration or wired infrastructure. We identify another area in which to deploy the concept of mobile ad hoc networks—direct wireless communication between vehicles in road traffic.

In so-called inter-vehicle communication, vehicles are equipped with computer controlled radio modems allowing them to contact other equipped vehicles in their vicinity. By exchanging information, vehicles build knowledge about the local traffic situation which can improve comfort and safety in driving. In contrast to applications of inter-vehicle communication to cooperative driving [KK92, AF96], platooning and automated highways [SV93, Li95, GF97, OKFF98], we avoid the requirements of high bandwidth, expensive equipment and infrastructure, and 100% deployment. The topic of inter-vehicle communication has been widely addressed in vehicular technology. However, the term ad hoc networks is seldomly used in conjunction with inter-vehicle communication as e.g. in [BSH00, MJK+00].

As an example of collaboration through inter-vehicle communication, we study the detection of traffic jams on a highway. Once an equipped vehicle slows down significantly, it suspects that it is inside a traffic jam. Then, it starts communicating with equipped vehicles nearby to share information on the driving situation. Vehicles inside the traffic jam create a dynamic group and try to establish common knowledge about the size, the beginning and the end of the congestion. Equipped vehicles that approach the traffic jam can then be warned and alert the driver. Also, the up-to-date information on the traffic jam can trigger dynamic navigation systems in other vehicles to find a detour around the congested road. Thus, we intend to help the driver cope with a potentially dangerous or inconvenient situation.

The idea of traffic jam detection highlights a new routing problem in highly mobile ad hoc networks. Frequent topology changes, scarce bandwidth, and large-scale coverage prevent the hosts from exchanging position and routing table updates throughout the whole network. Instead, the hosts maintain network information only about the local environment. The mobile hosts commonly do not know the identities of their communication partners in advance. Rather, the vehicles direct messages to geographic regions or to other vehicles with certain constraints on velocity, relative positions in the current situation, or similar vehicular parameters. Recent approaches in geographic routing where the destination of data packets is a location rather

Figure 1.1: Traffic jam detection via inter-vehicle communication

than a fixed address, show the significance of such a new routing paradigm.

We extend the idea of implicitly addressing messages. Basically, the originator of a message uses scoped and controlled flooding to reach the destination. The receivers of the flooded message use their knowledge of the local environment to decide whether they match the intended destination of the message. We call this mechanism *situation based routing* (SBR).

We envision inter-vehicle communication to be deployed gradually. The success of the system in the market place depends on the system functioning and producing visible results even if only a few vehicles are equipped with this system. Thus, we gear the routing algorithm to small deployment of the system when the network possibly is disconnected.

1.1 Thesis Contributions

The contributions of this thesis are three-fold. First, we advocate a new routing paradigm for highly mobile ad hoc networks. In order to cope with the anonymity of hosts, the absence of hierarchy and fixed infrastructure, and the rapidly changing network topology, we shift the routing of messages from explicitly naming the identities of destinations to an implicit addressing of messages. Simple but controlled flooding distributes messages

throughout the network. By this, we avoid the prerequisites of routing tables and knowledge on the network topology. Due to the high mobility, keeping this information up-to-date would cause a large overhead both in control messages to be sent and in memory to store the routing tables.

We use intelligent and scoped flooding to fulfill the new routing requirements. We extend the relatively new ideas of controlling and aiding the message dissemination with geographic positioning. Employing omnidirectional antennas, we take advantage of the broadcasting nature of radio waves; a packet sent by one host can reach multiple receivers simultaneously. Recipients of the flooded message forward it after a waiting time that depends on the individual distance from the receiver to the sender of the message that we calculate from geographic positions. There, receivers far away need to wait for a shorter time and spread the message faster than closer receivers.

In the early stages of deployment of the inter-vehicle communication system, the resulting ad hoc network is likely to suffer from temporary disconnections. Hence, we tailor the routing algorithm to overcome the problem of fragmentation in sparsely connected networks by taking advantage of the high velocity and the mobility pattern of hosts in inter-vehicle communication. There, vehicles driving in the opposite direction transport the message backwards on the road and close gaps in the network topology which possibly occur due to sparse deployment of the system.

Second, we investigate the problem of group formation in the context of mobile ad hoc networks. Our routing algorithm requires the mobile nodes to aggregate into a dynamic group. Regarding our application to detect a traffic jam, the group includes all vehicles that slow down in the same driving direction on the highway. A prominent impossibility result in asynchronous distributed systems with crash failures [CHTCB96] is still fueling the research community in the area of partitionable group membership services. Being aware of the theoretical obstacles and of the severe conditions inherent to ad hoc networks, we suggest reducing the group membership service to the local environment of a node. Having an application in ad hoc networks at hand, we also demonstrate the usefulness of our localized membership service approach. Reasoning about the formal model illustrates the possible pitfalls to be avoided in our implementation.

Third, we integrate the routing mechanism with the localized group membership service into an inter-vehicle communication system. We implement the communication system using the Specification and Description Language

(SDL). A simulation demonstrates the effect such a system has in a realistic highway traffic situation. Thus, this thesis also outlines a prototype of an inter-vehicle communication system ready to be built.

1.2 Outline

This dissertation is organized as follows. First, we describe in Chapter 2 the new routing paradigm which is used to detect traffic jams. The section on related work in routing with geographic or spatial constraints gives the context for our proposal of a new routing method in highly mobile ad hoc networks. As indicated in this chapter, the unicast problem requires a membership service that maintains a dynamic group of mobile nodes.

Therefore, Chapter 3 covers the group membership problem applied to mobile ad hoc networks. We again discuss related work, now in the area of partitionable group communication in asynchronous distributed systems. Then, we derive a specification for a new localized group membership service that supports our routing problem.

In Chapter 4, we present an integrated vehicle communication system that implements the ideas developed in the previous chapters, namely the localized group membership service and the routing mechanism. Finally, in Chapters 5 and 6, we discuss the network simulation of this communication system embedded into a realistic highway traffic scenario. We develop metrics to measure our proposed protocols and analyze the simulation results. We conclude this thesis and suggest future directions in Chapter 7.

The Appendix A repeats the formal definition of the temporal logic operators from Manna and Pnueli [MP92] that we use for defining the group membership service. The Appendices B and C explain the simulator modules.

Chapter 2

Routing in Mobile Ad Hoc Networks

In inter-vehicle communication, the vehicles form a mobile ad hoc network which consists of mobile hosts that communicate via wireless links. Due to mobility, the topology of the network changes continuously and wireless links establish and break down frequently. Moreover, the ad hoc network operates in the absence of fixed infrastructure, forcing the hosts to organize the exchange of information decentrally. The research area of mobile ad hoc networks has recently gained importance and established itself as a standalone direction within the field of wireless communication and networking. In August 2000, the first workshop on "Mobile Ad Hoc Networking & Computing (MobiHOC)" [Mob00] took place in conjunction with the sixth annual conference on "Mobile Computing and Networking." In response to the increasing interest, the organizers decided to hold the next MobiHOC as an independent symposium on an annual schedule [Mob01].

Research in the area of mobile ad hoc networking mainly focuses on routing issues [BMJ+98, RT99, CGZ98, HOTV99, BMSU99, CL00]. In routing, we distinguish the unicast and the multicast problem. For unicasting, the network finds a communication path from a sender to one receiver. In the multicast problem, a message targets multiple receivers either by a list of individual addresses or by a single group address. In the latter case, a group communication service commonly maintains the group membership and provides a broadcast to all group members.

When applied to wireless communication between vehicles in road traffic scenarios, the problem of routing a message to a known receiver shifts towards

routing a message to unknown receivers with certain constraints on the role the addressée plays in the current situation. We call this anonymous addressing because the receivers decide on their own, based on current values of the vehicle's parameters and on the contents of the message, whether they are the intended destination. Mostly the position and velocity of vehicles determine whether they are the destination of a message. For detecting the size of a traffic jam, the leading vehicle inside the congestion sends a message to the last vehicle and vice versa. The identity of the receiver is a priori unknown to the initiator of the message. Hence, the geographic location of vehicles in a given traffic situation affects the destination of a message rather than the identity of hosts. Novel studies [BH00, IN99, KK00, SR00] in geographic and location based routing reflect the need for an alternative addressing scheme in mobile ad hoc networks.

2.1 Related Work

An overview on the topic of routing in mobile ad hoc networks is given in [BMJ+98, RT99]. The authors compare the performance of several routing protocols to find a communication path from a sender to a known receiver. Solutions for this classic definition of the routing problem are comparable. However, as mentioned above, we need a different approach to the routing problem. Hence, in the following we concentrate on work about routing with geographic or spatial constraints and reliability issues.

2.1.1 Reliable Broadcast

Pagani and Rossi [PR97, PR99] address the problem of reliable broadcast in mobile ad hoc networks. The semantics of a reliable broadcast in the finite, fully mobile network is that all mobile hosts deliver the same set of messages to the upper layer. By using the term "fully mobile," the authors imply the absence of fixed infrastructure. Other characteristics of the network model include arbitrary mobility of hosts and point-to-multipoint communication primitives as found in radio networks. The protocol tolerates communication failures and host mobility. A liveness property reflects the assumption that in case of temporary disconnections, the network is eventually repaired and remains connected long enough for message and acknowledgment exchange. The protocol works on top of a medium access control (MAC) layer and an underlying clustering layer that structures the network into cluster-heads,

gateways, and regular hosts. The failure model only allows transient failures that do not cause the loss of states. The authors prove the correctness and the termination of the protocol based on the liveness property of the network. The protocol is composed of two phases: In the scattering phase, the message is diffused to all receiver members, and in the gathering phase, the acknowledgments are collected from the receivers. A forwarding tree is constructed implicitly during the scattering phase and will be destroyed after message stabilization when the protocol terminates.

The authors presuppose that the network is already structured into clusters. The clustering algorithm uses neighborhood information that a host derives from "I-am-alive" messages. We will employ a similar heartbeat mechanism, too. The authors state the properties of the clustering algorithm and cite two known approaches in the literature [EWB87, GT95]. However, we feel that the overhead of communication caused by providing such a powerful structuring cannot be neglected and should be integrated into performance measures. In our model, we thus will investigate all layers that need to exchange messages. Finally, we allow the network to be fragmented and in turn we only achieve a weaker reliability in communication.

2.1.2 Epidemiological Protocols

Lin et al. [LMM99] study two protocols for reliable, fault-tolerant broadcast. The authors call their approach epidemiological. A simple gossiping[1] protocol is compared with a deterministic flooding[2] mechanism. The assumptions on the network are:

(1) the network is fully connected (i.e. the communication graph is a clique)

(2) each processor knows the other processors

(3) links are point-to-point connections

(4) links and nodes can fail; the model assumes for each failure type the same and independent probability.

[1] In a gossiping protocol, a receiver forwards a message to a randomly chosen subset of its neighbors.

[2] In a flooding protocol, a receiver forwards a message to all neighbors.

Further on, no processor plays a specific role (e.g. the role of a master station). The protocols work without acknowledgments and achieve fault-tolerance instead by redundant deliveries of the message. The gossiping protocol is a probabilistic flooding method where the processor chooses its communication partners for forwarding the message randomly. In deterministic flooding, the communication partners are chosen according to a superimposed Harary graph. Harary gave an algorithm for constructing a graph in $H_{n,t}$ for any value of n and $t < n$ [Har62].

Although the assumptions (1)–(4) are not valid in our application, the idea of avoiding acknowledgments in a flat network without hierarchy suits ad hoc networks. We think that a flooding algorithm achieves better results than selecting a subset of neighbors in gossiping with radio communication that inherently broadcasts messages in transmission range. In dynamic ad hoc networks, we cannot construct a Harary graph because we lack the topology information that is either stale or not available at all.

2.1.3 Sensor Networks

Intanagonwiwat et al. [IGE00] investigate communication in wireless sensor networks which are quite similar to ad hoc networks. The topology of sensor networks is not predefined and may change dynamically. As in ad hoc networks, no fixed infrastructure supports the communication. Even more important than in ad hoc networks is the large scale on which sensor networks operate. The authors present a data-centric dissemination paradigm called "directed diffusion" in sensor networks. All interactions in the network are localized and based on message exchange between neighbors or nodes in some vicinity.

A human operator queries the sensor network about data in a specified region. The network then diffuses the interest towards nodes in the named region. If a node inside the region receives an interest, it collects the sensor data and returns the information on the reverse path of interest propagation. Intermediate nodes might aggregate the data on its way back. The authors propose flooding for disseminating the interest request. Thus, the request reaches its destination possibly via multiple paths. Reinforcement-based adaption to the empirically best path then establishes the way for the data to return. By aggregating and caching data, the diffusion process is further optimized towards power consumption.

The authors simulate a static and randomly deployed sensor network with the *ns-2* simulator [ns2]. A comparison with an omniscient multicast scheme that transmits events along a shortest-path multicast tree shows that directed diffusion outperforms the traditional routing scheme. To investigate the network under dynamic changes, randomly chosen nodes fail and recover intermittently. The diffusion mechanism is stable under the conditions studied in this paper.

The data-centric approach is close to our scoped flooding scheme. Even though sensor networks also operate on a large scale, they do not exhibit high mobility like vehicle-to-vehicle communication.

2.1.4 Geocasting and Location Based Routing

In this section, we present the work specific to routing combined with geographic positions in chronological order.

At Rutgers University

GeoCast

Imielinski and Navas [NI97, IN99] coined the term "GeoCast" for multicasting a message to a certain geographical region. The intended recipients of a geocast are determined implicitly by their geographical location. The authors extend the capabilities of the current network layer of the Internet protocol (IP). A prototype system introduces three new types of components—GeoHosts that are capable of sending and receiving geographic messages, GeoNodes that maintain a subnet or wireless cell, and GeoRouters that connect to the Internet and are basically routers which are geographically aware.

We also apply geographic addressing enriched with task specific information to determine the intended recipient of a message. While Imielinski and Navas focus on integrating geographic messaging into the existing IP framework and impose a hierarchical network structure, we cannot directly transfer the routing method to the flat and decentrally organized type of mobile ad hoc networks.

At Texas A&M University

Ko and Vaidya [KV98, KV00] propose using location information to reduce the message overhead of routing protocols in mobile ad hoc networks. In

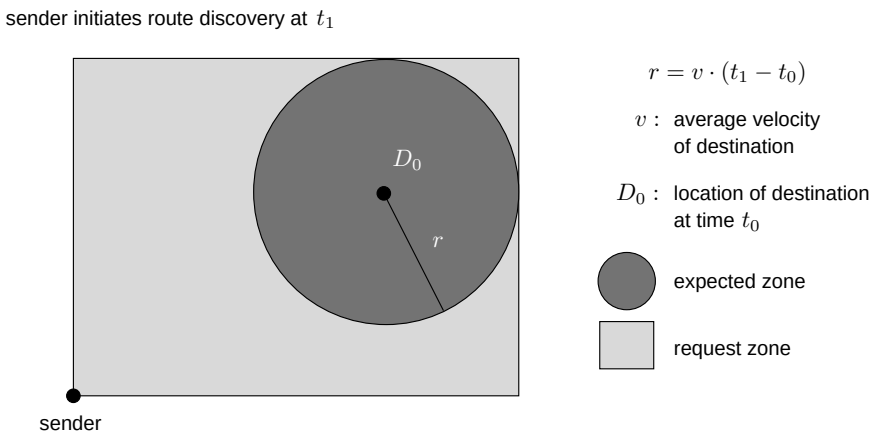


Figure 2.1: Zones in LAR by Ko and Vaidya from [KV98]

LAR

Location-Aided Routing (LAR), the destination of a data packet is composed of an identifier and an estimation on the destination node's current position. For example, the sender received position information about the destination node earlier. The sender then defines a circular region around this previous location called the "expected zone." The older the position information, the greater the radius of the expected zone. For a LAR unicast, the sender initiates a route discovery to the expected zone where it suspects the destination node to reside. The so-called "request zone" of the message connects the expected zone with the location of the sender spatially. For the route discovery, the message floods the network limited to the request zone. An example on expected and request zone is depicted in Figure 2.1. If the sender has no information in the location of the destination node, the algorithm behaves like unlimited flooding and spreads the message into all directions to find the destination node. After the request has reached the destination, the discovered route is sent back to the initiator. In a second phase, the originator sends the data along the discovered path to the destination.

LBM

The Location-Based Multicast (LBM) in [KV99] extends the idea of LAR towards multicasting. Herein, Ko and Vaidya adopt the term "geocast" for addressing a multicast to nodes within a geographical area from [NI97]. The sender defines the "multicast region" as the destination of a message. Identifiers or group addresses are not necessary. Similar to the request zone in LAR, a "forwarding zone" covers the sender and includes the multicast region. LBM uses flooding that is limited to the forwarding zone akin to LAR does for discovering a route to the destination node. In contrast to

LAR, LBM already delivers the data packet through this limited flooding process and thus the second phase is omitted.

Ko and Vaidya simulated both algorithm LAR and LBM using MaRS (Maryland Routing Simulator). The nodes move randomly with an average speed inside a 1000 unit $\times$ 1000 unit square plane. The average velocities vary between 1.5 to 32.5 units per second. The transmission range is set to 200, 300, 400, and 500 units. The units are not related to distances in meters. Also, the authors do not consider packet loss nor delays in accessing the wireless channel and transmitting packets.

We also utilize flooding in routing a data packet from the beginning of the traffic jam to the end and vice versa. In contrast, we neither know the identity of the destination nor an estimated location. Different to LAR which uses two phases, we include the data already in the flooded message i.e. no second phase after route discovery is needed.

At Federal University of Minas Gerais, Brazil

GPSAL

Câmara and Loureiro [CL00] present the GPSAL routing algorithm, which is based on the Global Positioning System (GPS) and mobile software agents modeled on ants. Again, the destination of a packet is a physical location. Ants are used to collect and disseminate information about the location of nodes. The ants update the routing tables of hosts that maintain positions and previous positions of other hosts. Upon detecting a new neighbor, hosts exchange possibly their entire routing tables. Also, each host periodically broadcasts the changes that occurred in its routing table since the last update.

The simulation model consists of up to 50 nodes. No transmission errors or delays occur. The authors measure the routing overhead as the number of generated packets per data packet. The results for GPSAL are compared to an implementation of the LAR [KV98] routing algorithm. In all scenarios, GPSAL produces less packets per data packet and hence operates with lower overhead.

In our application, we may encounter many more than 50 hosts. Also, due to high mobility, the location information on the network becomes stale and can only reflect a current snapshot if the nodes massively transmit their information. Routing tables may be long and thus the data transmission involves delays because of limited bandwidth. Finally, we assume that packets

are lost or delayed in a wireless network where the channel is used to a high extent.

At University of Ottawa and Carleton University, Canada

Bose, Morin, Stojmenovic, and Urrutia [BMSU99] present an algorithm that guarantees delivery of packets to a known destination location. They model an ad hoc wireless network as a unit graph in which nodes are points in the plane, and in which edges between two points exist if the nodes are closer than a fixed transmission range. The graph is assumed to be connected and static during the operation. As a prerequisite, each node knows its own position, its neighbors and their positions as well as the position of the destination node. The proposed algorithm FACE requires no duplication of packets or memory at the nodes. It first establishes a Gabriel graph which is a connected, planar subgraph of the underlying unit graph. The authors sketch a distributed algorithm to construct a Gabriel graph by deleting certain edges but they omit to explain how two nodes agree on deleting a link. Nor do the authors address the issue of collectively starting a distributed algorithm at individual hosts.

FACE

The edges in the Gabriel graph bound polygonals that partition the plane into faces. In the next phase, the packet is routed along those faces that are intersected by the straight line between the source node and the destination location. In Figure 2.2, we depict an example of routing with the FACE-2 algorithm. As a metric, the authors measure the "average dilation" of a routing algorithm which is the sum for all possible pairs of nodes of the length in hops of a path found divided by the length of the shortest path. The sum is then normalized through division by the number of pairs. Two versions of the FACE algorithm are compared with the algorithm for "geographic distance routing" (GEDIR) proposed by [SL99] and with a hybrid form of both algorithms. The GEDIR algorithm is a greedy algorithm that always moves the packet to the neighbor closest to the destination. In contrast to FACE, each node or the packet has to store the history information about the routing process. An extension of the memoryless FACE algorithm towards broadcasting and geocasting—i.e. multicasting inside a destination region—is also discussed.

GEDIR

Stojmenovic et al. [SL00, SR00] present further research on location based routing. The authors adopt the term "localized algorithms" from [EGHK99] as distributed algorithms where simple node behavior based on decisions

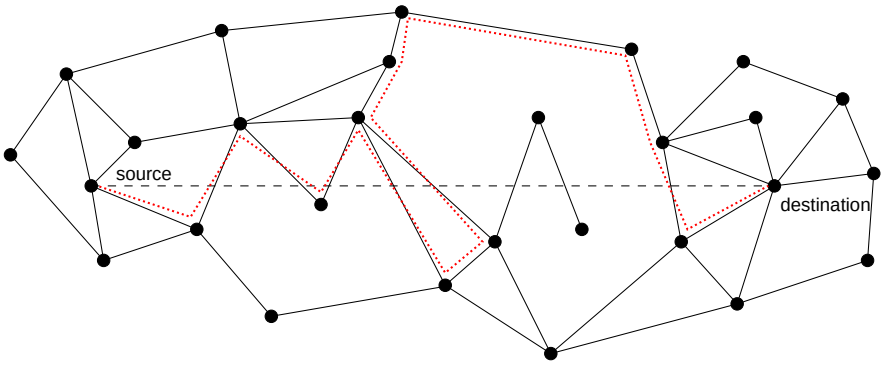


Figure 2.2: Routing using FACE-2 from [BMSU99]

made by local knowledge achieves desired global objective. Applied to the problem of location based routing, the presented localized routing algorithms only requires nodes to know their own position, their neighbors and their neighbors' position as well as the approximate location of the destination node. Stojmenovic and Russell [SR00] propose a localized routing algorithm based on depth-first search (DFS). The extension to depth-first search in graphs is that the neighboring nodes are sorted by their distance to the destination. Then, the closest neighbor to the destination is chosen for forwarding the message first. The authors also incorporate quality-of-service (QoS) constraints into DFS such that a path fulfills certain requirements on bandwidth and connection time. A node estimates the latter by taking into account knowledge about position, velocity and direction of the moving neighbors. When simulated on random static and connected networks, the DFS algorithm produces QoS paths that are on average 1 to 1.34 times longer than the shortest path.

The approach of localized algorithms is attractive in our context with highly mobile hosts. However, we cannot assume that the network is always connected and remains static during routing. Therefore, the result on guaranteed delivery is not transferable to our model. We also reduce the knowledge of the position of neighboring nodes to those inside the traffic jam.

At Massachusetts Institute of Technology

Li et al. [LJC+00] address the question of providing a location service for geographic routing in large scale, mobile ad hoc networks. The authors sketch a simple yet efficient and scalable routing mechanism, which forwards data

packets along those nodes that are closer to the physical location of the destination of a message. This requires first that each node maintains a table of its current neighbors along with their locations to decide upon forwarding which neighbor is closest to the destination. Every node announces its presence, position and velocity to its neighbors within radio range by broadcasting HELLO packets periodically.

Second, the network must provide a location service where senders query the current location of the message's destination node. If only a single and global server provides the location service, a number of disadvantages occur. These include a single point of failure, less scalability for a large number of mobile nodes and possibly long distances to a distant server querying the location of a node nearby. Thus, the authors propose a distributed grid location service (GLS) that uses a hierarchical and global partition of the world into squares. In GLS, a node maintains its current location in a number of location servers distributed throughout the network. Each node acts also as a location server for other nodes.

GLS

The authors simulated GLS and data traffic using the *ns* simulator with CMU's wireless extensions. The simulator incorporates the full IEEE 802.11 medium access control. The transmission range is a disc with a 250 m radius. The bandwidth is set to 1 and 2 Megabit per second. The nodes are placed uniformly in the square world up to 2900 m side length. The nodes move according to a random waypoint model with an average velocity between 0 and 10 meters per second. For the metric of the GLS query success rate, the maximum speed also takes values up to 50 m/s which is 180 km/h and about 110 miles per hour.

Combining GLS with geographic forwarding creates a traditional network layer: any node can send packets to any other node in the network. The mobility is hidden from the user. Still, this approach assumes that the host sending a packet knows the identity or address of the destination host. In our model, we do not know the name of the vehicles at the border of the traffic jam. Moreover, we advocate vehicles to not periodically broadcast their location and their velocity. Such information may cause privacy concerns and also extends the length of packets which are periodically broadcast. Our neighborhood service reduces the HELLO packets to the identity of the host to keep the usage of the wireless channel at a minimum.

In [MJK+00], the authors design the CarNet wireless network system around their work on GLS and geographic forwarding. As the name implies, Car-

Net is an application of wireless ad hoc networks to vehicles in road traffic. The vision of all cars forming a network is to support new applications like cooperative highway congestion monitoring, fleet tracking, and discovery of nearby points of interest. As an important goal, the authors mention IP connectivity and Internet access and thus require a fixed gateway. However, this is not the scope of our work because it violates the assumption of operation without infrastructure.

At Harvard University

Karp and Kung [KK00] propose the Greedy Perimeter Stateless Routing (GPSR) protocol for wireless networks with location information in the intermediate routers and on the packet's destination. In GPSR, routers decide upon forwarding a packet using the information about immediate neighbors. By keeping state only about the local topology, GPSR scales well with the number of network nodes and a frequently changing network topology. The GPSR algorithm solves the task of geographic routing i.e. finding a communication path to a destination's location. Hence, it requires a location registration and lookup service that maps nodes addresses of locations like previously discussed for GLS in [LJC+00]. GPSR

Periodically, each node transmits a beacon with its own address and position. To avoid synchronization of neighbors due to a fixed beacon interval B, the beacon rate is randomly chosen between $[0.5B, 1.5B]$. After a timeout interval $T = 4.5B$ not receiving a beacon, a node deletes that neighbor from its table.

The GPSR algorithm consists of two methods for forwarding packets: "greedy forwarding" and "perimeter forwarding," which is used when greedy forwarding does not succeed. First, a router uses greedy forwarding and sends the packet to the neighbor closest to the destination. In case the router itself is already the closest neighbor as seen in Figure 2.3, the algorithm switches to perimeter forwarding and saves the current distance L to the destination. Then, GPSR uses a planarized graph—either a relative neighborhood graph or a Gabriel graph—and forwards the packet counterclockwise along the face intersected by the line connecting the router with the destination. In the example of Figure 2.3, the router r would choose node s in perimeter forwarding. Upon receiving a packet in perimeter mode, the node compares its own distance to the destination with the saved length L when the packet

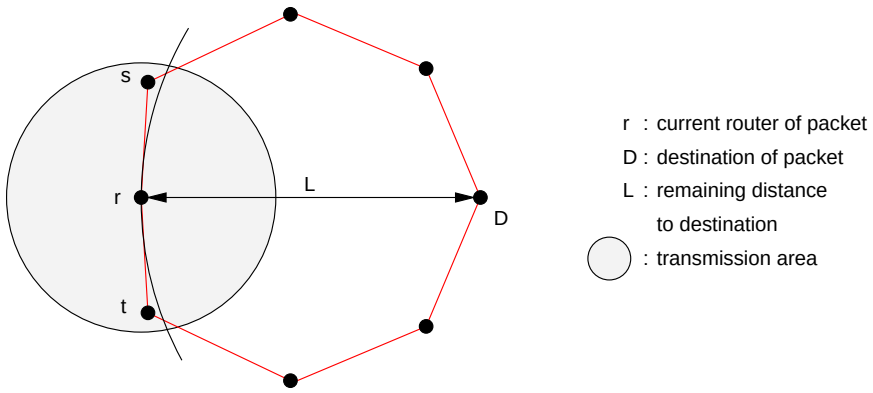


Figure 2.3: Greedy forwarding failure in Karp and Kung's routing algorithm from [KK00]

enters perimeter forwarding. If this distance is smaller than L, the router switches back to greedy forwarding.

Karp and Kung simulate GPSR in densely deployed wireless networks using the *ns-2* simulator with CMU's wireless extensions. As the medium access control, the IEEE 802.11 standard is implemented with a 250 m radio range. The nodes move according to the random waypoint model with 20 m/s as the maximum velocity. GPSR delivers upwards of 94% of data packets successfully and causes less overhead than Dynamic Source Routing (DSR) [BJM98] in networks with 112 and 200 nodes.

Karp and Kung's approach shows that scaling in both number of nodes and mobility rate is important and achievable. However, in inter-vehicle communication, extensively disseminating and updating knowledge on the neighbors locations is problematic because of scarce bandwidth. We use a similar beacon mechanism that only broadcasts the identity of the vehicle. Later, when vehicles slow down in traffic jams and collaborate towards classifying the traffic jam, single messages may contain the position of the sender.

At University of California, Berkeley

GAL

Jain et al. [JPS99] investigate a geographical routing algorithm (GAL) in wireless ad hoc networks with partial information. The authors assume that all nodes know their geographical position and the position of the packet's destination. All nodes build up a routing table that initially contains the positions of their neighbors. If a node needs to route a packet to a given

destination, it sends the packet to the neighbor that is closest to the destination.

If a node is closer to the destination than all its neighbors, the message is stuck and a route discovery procedure is initiated. Via breadth-first or depth-first search, a path is found to the destination. All nodes on the this path update their routing tables and the node where the packet got stuck sends it to the destination. If the discovered path is acyclic and the updates of routing tables happen in reverse order, the authors can prove the correctness of GAL. However, this result applies solely to statically connected networks without node or link failures.

An advantage of GAL is that it operates on small routing tables which are not required to have an entry for every node in the network. Instead of needing full information on the topology, GAL works with only partial knowledge about the ad hoc network. In order to cope with dynamics when nodes and links fail, and with the mobility of nodes, the authors propose an extension to GAL called the tear down protocol. Every node learns about its neighbors through "hello" messages. These hello messages also contain the current routing table of the sending node.

The authors simulated GAL and compared it to the Destination-Sequenced Distance-Vector (DSDV) [PB94] algorithm. The number of nodes varies between 10 and 10^3. The routing table size in GAL remains below 15 entries whereas the routing table of DSDV grows proportionally with the network size.

We agree on the approach of using partial or local information for routing in mobile ad hoc networks. In our model, the nodes can be highly mobile and the assumption on an always connected network is not necessarily valid. Thus, we further reduce control messages for routing to short beacons that contain only the identity of a neighbor. In contrast to GAL, we omit the route discovery phase and use scoped flooding to deliver a message to a destination. Hence, we do not cause overhead by establishing a route that could soon be outdated.

2.1.5 Content Based Multicast

Zhou and Singh propose in [ZS00] a new Content Based Multicast (CBM) scheme in ad hoc networks. In CBM, the content of the data being multicast together with the mobility of the receivers determine the multicast

CBM

set. The authors focus on application in the battlefield but mention also the possibility of use in disaster relief. The CBM protocol is based on the idea of "sensor-push" and "receiver-pull." Sensors detecting threats push the information out into the network to some distance and direction. Individual receivers then pull threat warnings from nodes that lie in the direction of their travel. The protocol assumes the area of operation to be mapped and divided into regions. Every node has location capabilities by employing GPS.

Also, a leader per region maintains a list of all threat warnings received via push packets. Nodes pulling these threat warnings send a query to the leader of the region that they travel to. When a leader leaves its block, the responsibility for maintaining threat warnings passes on to a new leader. For routing messages to regions, the Most Forward within transmission Range (MFR) protocol [TK84] is used. Essentially, this protocol forwards packets to the neighbor closest to the destination similar to the greedy forwarding mode [KK00] and [JPS99] described in the previous subsection. If no neighbor exists, the packet is dropped.

The authors simulate the algorithms in a 50 km by 50 km square area with a battlefield scenario. However, it is unclear to which level they simulated the ad hoc network. The topic of medium access protocols is not discussed. In future work, they plan to include packet loss. This implies that no packet loss was simulated if nodes are within transmission range. As a metric, the authors observe the message overhead in their simulation. Unfortunately, it is unclear how the message overhead is defined. From the unit the message overhead is noted in, we conclude that it resembles all messages received by a node in one second. The leader election requires detailed knowledge on the network topology within a region. We cannot follow the meager description of the algorithm and thus are not able to evaluate the idea for application in inter-vehicle communication.

2.1.6 Disconnected Ad Hoc Networks

At Dartmouth College

Li and Rus [LR00] address the problem of mobile users that are disconnected in ad hoc networks. In contrast to letting the mobile host wait passively for reconnection, the mobile hosts actively modify their trajectories to minimize transmission delay of messages. Two flavors of their approach distinguish

whether the movements of all hosts in the system are known or not known. The system is intended for applications like field operations or disaster relief that require urgent message delivery and involve cars or robots.

The authors argue that this approach is useful in ad hoc networks when most of the network is connected and the distance between two hosts is only slightly larger than the transmission range. However, we cannot force this approach in inter-vehicle communication. We will also study disconnected operation of the ad hoc network. We take advantage of the mobility pattern in road traffic where vehicles move in opposite directions. Then, a simple extension of waiting time allows us to transport messages actively towards their destination.

At Duke University

Vahdat and Becker [VB00] propose an epidemic routing protocol for disconnected networks. The routing mechanism is derived from epidemic algorithms that provide eventual consistency in replicated databases without requiring any particular replica to be available at a given time. Epidemic routing relies upon carriers of messages coming into contact with another component of the network through node mobility. At this point, nodes exchange pair-wise messages that the other node has not seen yet. Even if there never exists a path in the momentary snapshot of the network, the transitive transmission of data eventually causes a message to reach its destination.

The authors simulate epidemic routing using the *ns-2* simulator with 50 nodes in a plane of 1500 m $\times$ 30 m moving according to the random waypoint model with a maximum velocity of 20 meter per second. The authors varied the transmission range of radio communication from 250 m down to only 10 m where the network is disconnected most of the time. Still, the message delivery rate was always 100% for a transmission range greater than 25 m and dropped to 89.9% in case of only 10 m transmission range.

The idea of epidemic routing is very similar to our approach to letting nodes move while waiting to forward a message until new communication partners come into vicinity. However, we aim to avoid pair-wise communication and achieve the spread of messages through flooding i.e. receivers forward a pending message once.

Our Work

In [BH00], we present our studies on fragmented ad hoc networks in inter-vehicle communication targeted to hazard warning in a road traffic scenario. The paper focuses on a multicast to a region where vehicles reside to which the warning message is relevant. We use a flooding-based multicast similar to LBM [KV99] but force the hosts to wait longer to forward the message if no new neighbors are in sight. With this relatively simple approach we achieve high delivery rates of over 85% with only 1% of the vehicles on the road being equipped.

2.2 Routing for Inter-Vehicle Communication

In this section, we present our approach to routing in highly mobile ad hoc networks targeted to inter-vehicle communication. As mentioned in the discussion of related work, we combine several aspects of known techniques into our routing paradigm. These include localized operation knowing only the direct neighbors, usage of geographic constraints, and multihopping via scoped flooding extended towards coping with disconnected networks. Chapter 4 contains the detailed specification of our proposed algorithms.

An impediment in mapping inter-vehicle communication to routing in ad hoc networks is the inherent anonymity of participating hosts. As an essential requirement in networking, every host vehicle possesses a unique identifier. But the set of existing identifiers can easily exceed a practical size of fixed host or server tables. Plus, newly manufactured vehicles equipped with the system join the set of existing identifiers whereas the identifiers of crashed or scrapped vehicles leave the set. In contrast to the huge set of possible identifiers, an ad hoc network formed in reality on the road will only connect a small subset of identifiers. In applications of inter-vehicle communication, a vehicle often needs to address other nearby vehicles whose identities are a priori unknown rather than sending a packet to a specificly identified vehicle.

In our sample application, we use inter-vehicle communication to detect the current size and position of traffic jams on highways. Once an equipped vehicle reduces speed significantly it suspects itself to be inside a traffic jam. Then, it starts communicating with equipped vehicles nearby to share information on their driving situation. Slow vehicles create a dynamic group and exchange their global positions. We determine the current size of the

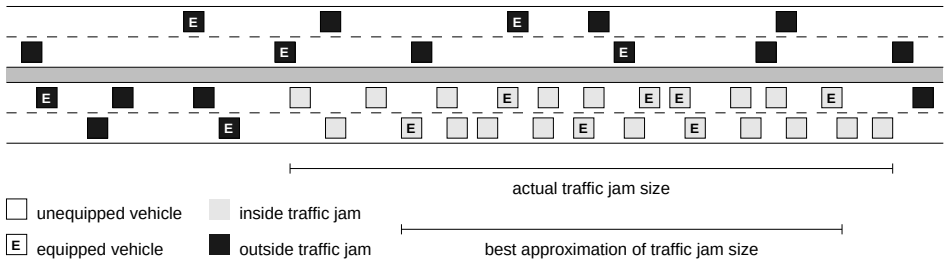


Figure 2.4: Example of detecting the traffic jam size

congestion by the distance between the two equipped vehicles at the beginning and at the end of the traffic jam. Unless all vehicles on the road are equipped, the result is always an approximation because unequipped vehicles may extend the size of the congestion but cannot participate in the wireless communication. Figure 2.4 provides an example of the above described situation.

As a requirement for building useful applications in inter-vehicle communication that improve comfort and safety in driving, vehicles need to be aware of their locations. Many vehicles do or will soon utilize navigation systems like the Global Positioning System (GPS). Although todays GPS receivers are accurate to within 10 m, we still expect improvement during the next several years. Future navigation systems will use differential correction or integrate inertial sensors to enhance the accuracy of positioning down to one meter or better [WRW98]. Assuming that equipped vehicles know their location more or less accurately, they can enrich their messages with position information that helps the system to inform the driver in a sophisticated way. Using a digital road map, the vehicle is aware of the road it drives on. Thus, each equipped vehicle knows its position and whether it travels on a highway, so that the precondition for our traffic jam detection is feasible.

As a straightforward strategy, the group inside the traffic jam would elect the two outermost vehicles in the congestion as leaders and calculate the distance between them. However, we cannot rely on wireless radio communication. Moreover, the network topology and the outermost vehicles change frequently. We require an efficient and timely algorithm whereas the classic group formation and leadership election would create a large overhead. Also, we cannot guarantee that the sub-topology covering the group members is connected all the time. Even in a connected network, the vehicles operate asynchronously and thus perfect agreement on all members is impossible to

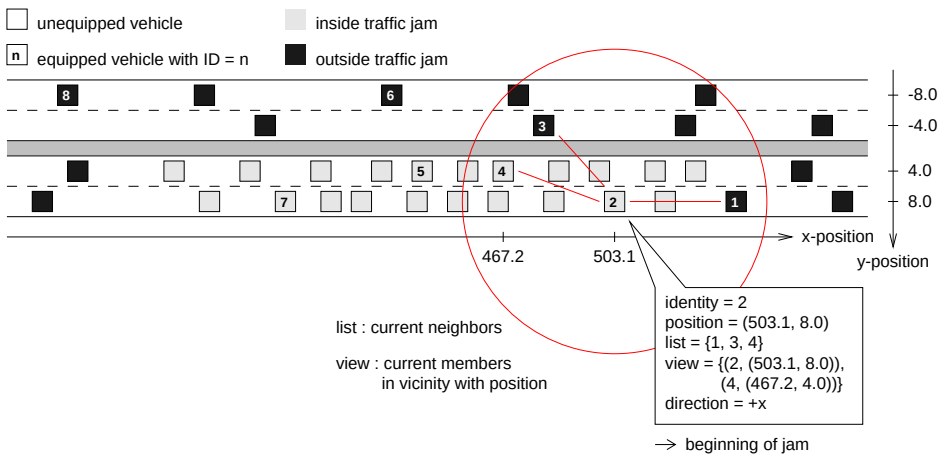


Figure 2.5: Example of classification of vehicle at beginning of jam

achieve when nodes and links are prone to failure.

We propose therefore a simpler and more efficient approach that reduces the group formation to the local environment of an equipped vehicle. Therein, only direct neighbors that are in radio communication range of each other exchange group membership and position information. Each vehicle inside the group compares its own position with the location of other jammed vehicles nearby. Then, every vehicle decides whether it is at the beginning, in the middle, or at the end of the traffic jam.

The example in Figure 2.5 illustrates the decision of an equipped vehicle at the beginning of the jam. There, vehicle # 2 learns about its neighbor # 4 which is also jammed. After exchanging their positions, vehicle # 2 concludes to be at the beginning of the jam because the only neighbor who is jammed has a smaller position.

Next, the two vehicles which have classified themselves as being at the border of the congestion send messages to each other. The ad hoc network routes these messages in an anonymous fashion—possibly over intermediate hosts—to its destination. The destination of the message is not an explicit address, but rather is determined by the sending direction and the constraints on the velocity, position, and driving direction of equipped vehicles on the highway. When a message reaches its destination, the receiver estimates the current size and the position of the traffic jam.

The routing protocol simply floods the network. Every node receiving a message forwards it to its neighbors. Flooding a network acts like a chain

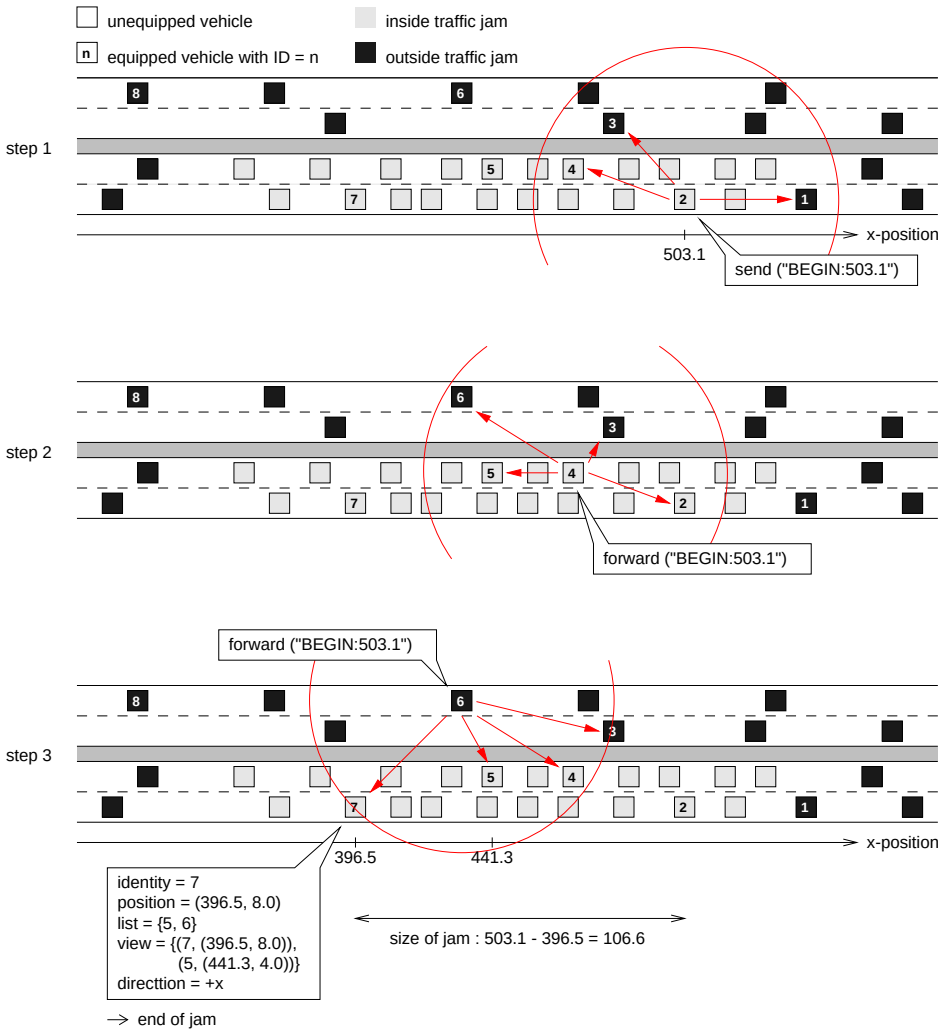


Figure 2.6: Example of routing towards end of jam

reaction that can result in exponential growth. Nonetheless, it has the advantage of working with no knowledge about the underlying network topology. We limit flooding to reach the destination of the message within a certain number of hops and within a given life time of the message. We also take advantage of the broadcasting nature of radio waves; a packet sent by one host can reach multiple receivers simultaneously. Thus, the number of sending activities only increases linearly with the number of hosts although the number of packets received still grows exponentially.

Consider the example in Figure 2.6. When equipped vehicle # 2 identifies itself to be at the beginning of the jam, it originates a message with its own

position. In step one, this message possibly reaches the three neighbors of the originator.

Knowing its own position, every receiver regardless of whether or not it is a member of the group, calculates its distance to the sender. We assume that the message contains the position of its sender. Then, the receiver determines a waiting time depending on the distance to the sender such that the waiting time is shorter for more distant receivers.

In our example, equipped vehicle # 4 is the farthest receiver and thus—in step two—forwards the message after the shortest waiting time. Receiver # 3 and the originator # 2 get the message for the second time and discard it. Again, the recipients # 5 and # 6 set an individual waiting time which expires first at equipped vehicle # 6.

In the third step the broadcast of vehicle # 6 actually reaches the equipped vehicle # 7. This vehicle # 7 has already classified itself as being at the end of the jam similarly to vehicle # 2 in Figure 2.5. Hence, when vehicle # 7 receives the forwarded message, the anonymous routing is successful and the vehicle # 7 calculates the size of the traffic jam as the distance to the originator.

The careful reader may already see that in the flooding based scheme messages are still waiting to be forwarded. Using the waiting time just caused the message to take the biggest steps towards its destination. This resembles the idea of greedy forwarding and geographic routing to the neighbor closest to the destination in [JPS99, KK00, SR00, ZS00] although in our approach hosts do not know the locations of their neighbors. As a simple optimization, nodes receiving a duplicate while waiting to forward a message could remove the message from their list of pending messages. Thus, the network load decreases but at the cost of redundancy inherent to plain flooding schemes. Refer to Figure 2.7 for an example of the advantage that redundancy provides in spreading a message. Trying to reach node d, node a sends out a message in step (1) being received by nodes b and c. The distance between b and the sender is greater than the distance of c to a. Hence, b forwards the message while c is still waiting. In step (2), b reaches nodes c and a. If we employed optimized flooding and deleted the pending message at node c upon receiving the duplicate from node b, then the forwarding process would halt without node d ever receiving the message. Otherwise, using plain flooding, node c forwards its pending message in step (3) delivering the message to node d. Also, in the presence of link failures the

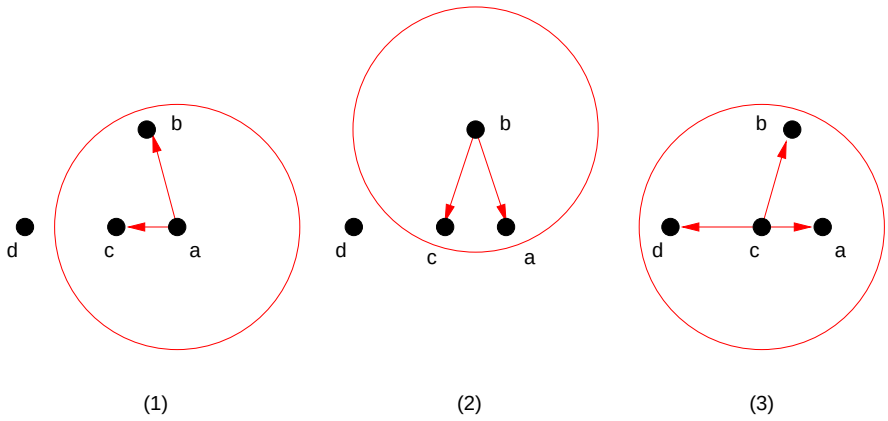


Figure 2.7: Example of redundancy in flooding

redundancy of flooding provides a higher probability of successful delivery.

To avoid messages from being circled endlessly in the network, a receiver stops forwarding the message if a certain number of hops is reached or if a given life time of the message expires.

When the network is connected, the multihopping of the message takes only a few seconds depending on the length of the traffic jam. Thus, it is feasible to assume a static topology as indicated in Figure 2.6. Nevertheless, our simulations take host mobility into account and utilize a realistic, microscopic traffic model.

In previous studies on routing in ad hoc networks [BSH00], we encountered the problem of a fragmented network due to a small number of equipped vehicles on the road. The success of the system in the market place depends on the system functioning and producing visible results with only a few vehicles being equipped. Furthermore, our simulations in [BSH00] proved that reaching the maximum of addressed vehicles a region stretching up to 5 km only takes one second. Realizing that fast delivery is not a crucial factor, we extend our routing algorithm and allow nodes to wait and not forward the message until new receivers move into their vicinity. By this simple means, we overcome fragmentation of the network due to sparse deployment and limited communication range.

Figure 2.8 sketches a traffic jam scenario with less equipped vehicles than in Figure 2.6. In step one, the message from vehicle # 2 at the beginning of the jam spreads to the jammed neighbor # 4 and the equipped vehicle # 3 traveling in the opposite direction. Observing the situation in step one,

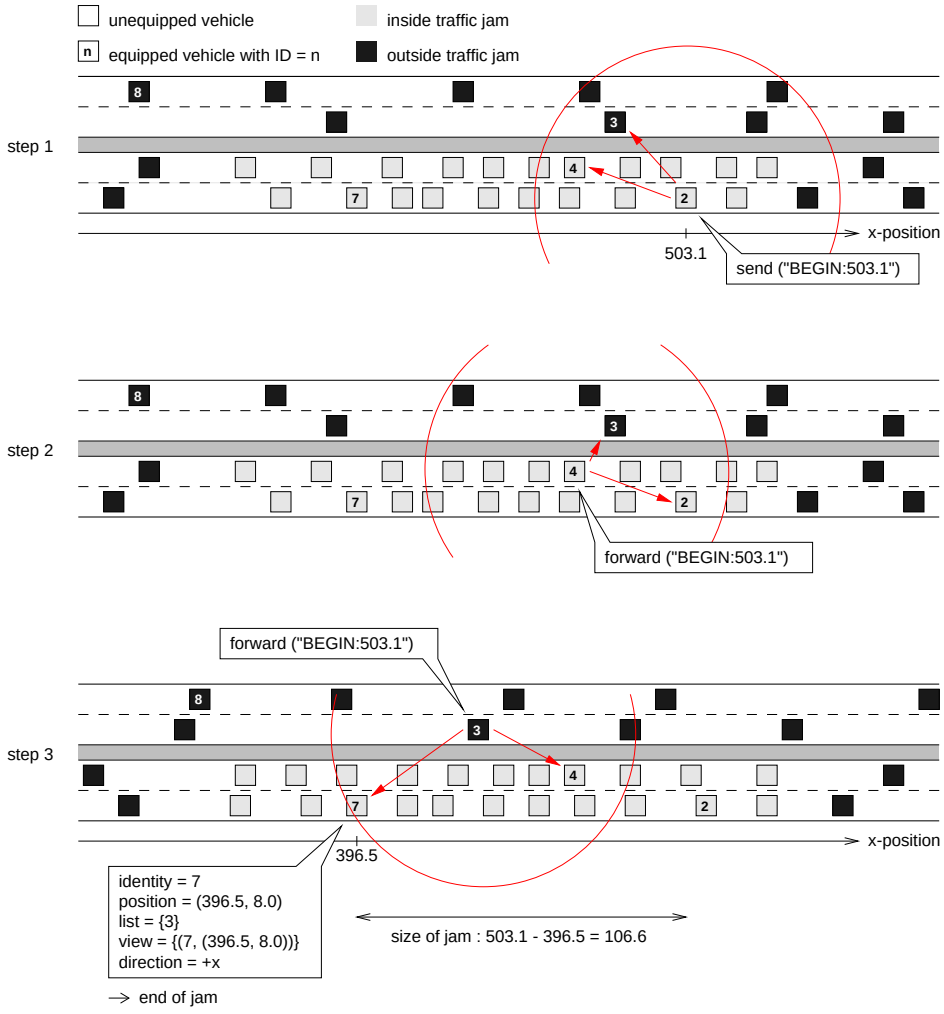


Figure 2.8: Example of routing in sparse network towards end of jam

vehicle # 4 may consider itself at the end of the jam. However, we limit the flooding process only by the number of hops a message can take, and by the lifetime of a message. Hence, vehicle # 4 still participates in spreading the message regardless of its current perspective on the situation. Assuming that the maximum number of hops is greater than one and that the lifetime has not expired yet, vehicle # 4 forwards in step two the message as seen before in Figure 2.6. This time, the message is already known by all its recipients. Vehicle # 3 continues driving in its direction while waiting for new neighbors. When vehicle # 7 comes into vicinity and announces its presence, vehicle # 3 forwards the pending message and thus closes the gap between the front and the end of the jam.

As prerequisites for our routing algorithm, an equipped vehicle is aware of is geographic position, the road and direction it is driving on, and its velocity. In this chapter, we sketched a routing algorithm that unicasts a message efficiently from one end of a traffic jam to the other. The algorithm requires the equipped vehicles that drive slowly on the highway to constitute a group. Other applications could benefit from forming a group as well e.g. police cars or fire engines that travel on streets. Therefore, we investigate the problem of group membership in the context of mobile ad hoc networks in the following chapter.

Chapter 3

Group Membership Service

Group membership has its roots in the research area of distributed systems. Therefore, we map the concept of mobile ad hoc networks to an asynchronous distributed system model.

> A group membership service in a distributed system maintains a list of the currently active and connected processes in the group. When this list changes (with new members joining and old ones departing or failing), the group membership service reports the change to the group members.

In this chapter, we describe the properties of the distributed system model by means of linear temporal logic. Two prominent characteristics distinguish our model from common distributed systems. First, we allow an unbounded number of processes to exist concurrently. None of the hosts are aware of an upper threshold on population size. Second, distributed applications—in the context of wide-area network like the Internet, or due to mobile hosts—are prone to temporary disconnections. Thus, recent work in group membership specification, e.g. [BDM98], relaxes the demand for agreement on a single view and allows multiple disjoint views to exist concurrently in different network components. We extend this idea and propose reducing the membership problem to the local environment of a host to cope with the severe conditions inherent to mobile ad hoc networks. A localized group membership service (LGMS) tracks the membership only of the adjacent neighbors. Changes in the localized group membership—existent neighbors join or leave the group voluntarily or crash, new members move into vicinity—

are installed as local views at each host. These views differ according to the neighborhood relation among vehicles and due to transmission failures.

3.1 Related Work

The group communication paradigm [Bir93] embodies a prominent technique in fault-tolerant and reliable distributed computing. Groups of member processes therein interact and communicate in order to achieve a common goal. A group communication system usually integrates a group membership service with a reliable multicast service.

The task of the group membership service is to keep members consistently informed about the current membership of a group by installing views. Processes can join and leave the group or even crash—all resulting in dynamic changes of the membership. Installed views consist of a set of members and reflect the perception of the group's membership. This requires the members to agree on the composition of a view.

In recent years, several approaches to group communication and to building fault-tolerant toolkits have been reported, including Transis [Tra], Ensemble [Ens], Newtop [EMS95], Jgroup [Jgr], and Spread [Spr]. However, no final agreement yet exists about a general specification of a partitionable group membership service. Anceaume et al. [ACBMT95] showed that trivial solutions are possible in proposed specifications for partitionable group membership services: (1) Each member process p installs views containing only p itself (*capricious view splitting*) or (2) members are allowed to install a priori determined views independent of the actual execution (*capricious view changes*). According to Anceaume et al., the Newtop [EMS95] group membership service does not prevent capricious view splitting and the group membership service specified in [DMS95] furthermore allows useless protocols that capriciously install views. We will later discuss our proposed specification in the light of useless protocols even though it is beyond the scope of this thesis to prove usefulness in general.

3.1.1 Mobile Systems

Prakash and Baldoni [PB98] present an architecture for group communication in the context of mobility. Three different types of mobile networks are considered: (1) A cellular and (2) a virtual cellular model in which base

stations are mobile, too, and (3) a fully mobile ad hoc network without base stations. All models apply only to connected networks.

The location of nodes plays an important role. Hence, the authors propose a "proximity layer" that links the group membership layer with the underlying medium access control (MAC) layer. The MAC layer of the mobile network provides point-to-point communication and beacons 'I am alive'-messages every t time units within transmission distance d. Then, a D-proximity test tries to find all nodes within distance D from a given node p. If $D \leq d$ then a node p determines its D-proximity set of nodes by just listening to the location stamped beacon messages. For $D = d$, this corresponds to our neighborhood service which uses heartbeat messages without the location.

The group construction protocol works on top of the proximity layer. There, a process p identifies all group members in D-proximity. Again, for $D = d$ this is analogous to our localized group membership service. The authors propose a three round protocol to solve the group construction task. First, p sends REQUEST messages to an a priori known superset S of the group members. Second, nodes receiving the request answer with acknowledge (ACK) or negative acknowledge (NACK) messages depending on their distance to p. In the third round, upon receiving all ACKs and NACKs from the set S, p sends JOIN messages to all nodes from which it has gotten positive acknowledgments.

Although the ideas of D-proximity and group construction are similar to our approach with a neighborhood and a localized group membership service, this work rather focuses on the solution of an informally stated group membership problem. Moreover, the limiting assumption of full connectivity does not hold in our model. Finally, the proposed architecture requires routing capabilities of the underlying network, which in turn is an active field of research in ad hoc networks. Routed messages in wireless ad hoc networks can be arbitrarily delayed or even lost which possibly prevents the proposed protocol to terminate.

3.1.2 View-Synchronous Group Communication

Fekete et al. [FLS97, FLS] propose a partitionable group communication service *VS* and an application using *VS*. The specification is split into safety requirements and performance and fault-tolerance requirements. The safety requirements are expressed by an abstract state machine that associates preconditions with effects. The performance and fault-tolerance requirements

are a set of properties that apply to executions of the state machine. The *VS* specification is tailored to the application of a totally ordered broadcast. However, the authors claim that other applications have used the proposed service as well.

The specification as a state machine is very dense and different from other approaches. Thus, it is difficult to compare the properties of the service with other work. Also, the authors integrate the membership service and the group communication service into one specification. As far as we can judge, one difference to our specification lies in the finite set of processes and in the initial view P_0 that every process has on the membership. We cannot assume such an initial set because the vehicles do not know the constellation in the future. Moreover, in our model the processes may start at an arbitrary time making an initial value senseless. Finally, the function to actively join or leave a group is omitted in *VS*.

3.1.3 Group Membership in Wide Area Networks

Keidar et al. [KSMD99] study the task of group membership in the context of wide area networks. Here, the membership service does not evolve from existing services in local area networks—in contrast, it resides on dedicated servers which are not involved in the communication among the group. This approach makes the service scalable both in terms of the number of groups and in the number of members in each group.

The membership service on dedicated servers uses an underlying network event notification service that handles failure detection of neighboring servers and local clients. Also, the communication between servers is reliable in the sense that a message either eventually arrives at its destination or else the notification service reports the link to be faulty. Upon receiving an event from the notification service, a membership server multicasts a "proposal" message to all other servers. This indicates that the system requires an efficient, scalable multicast service.

An interesting core idea of the wide area membership service is to avoid delivering obsolete views. The membership service waits for agreement among all the view members about what the view should be. It neither delivers a view without such agreement nor does it deliver an obsolete view when it has new information that the membership has changed. This implies that the algorithm may not terminate if the network cannot stabilize fast enough.

On the other hand, this policy avoids network congestion caused by control messages dealing with an outdated view.

The idea of waiting at the expense of not deciding in an unstable environment is appealing in the context of ad hoc networks where message overhead in an inherent unreliable communication scenario needs to be carefully observed. We use this idea when allowing the membership service to converge before installing a new view. However, due to constant changes in highly mobile ad hoc networks, the stabilization period is short and a time limit must prevent the membership service from waiting forever. Also, the inherent hierarchy in client/server approaches and predefined, dedicated servers do not exist in ad hoc networks. Therefore, we cannot apply this membership service in our environment.

3.1.4 Partitionable Light-Weight Groups

Rodrigues and Guo [RG00] describe a light-weight group service that is able to operate in partitionable networks. A light-weight group service maps multiple user groups onto a smaller number of instances. Then, a virtual synchronous implementation of a group membership service works on these fewer instances. Virtual synchrony ensures that all processes in the group receive consistent information about the group membership. Such algorithms do not scale well for a large number of groups. Therefore, the light-weight group service manages a pool of groups that are able to share common resources in order to enhance the performance of the virtual synchronous membership service. In the case of partitions, it is impossible to ensure the consistency of mapping decisions made in distinct parts of the network. Hence, partitionable operation requires reconciliation mechanisms when the network fragmentation is healed.

Such an approach suits system models that already employ a virtual synchronous group membership service. This assumption cannot be made in ad hoc networks. Furthermore, the key idea for operating in partitionable networks focuses on reconciliation procedures once the network becomes connected again. This implies that network partitioning occurs only temporarily. In contrast, we need a service that continues to operate correctly with respect to a possibly weaker specification, even during the time of partition.

3.1.5 Partitionable Systems

At the University of Bologna, the group communication paradigm has been studied and implemented for example in the Jgroup [Jgr] project. Advances in "partition-aware" group communication systems are reported in [BDMS98, BDM98, Mon00]. "Partition-aware" applications continue operating without blocking when the network fragments and reconfigure themselves when partitions merge. Babaoğlu et al. [BDM98] specify a partitionable group membership service that guarantees liveness and excludes trivial solutions. They give an implementation that satisfies the specification in distributed systems with a certain stability.

The asynchronous system model consists of a finite set of processes which communicate by passing messages between each pair of processes. Processes may crash and communication links can transiently fail. A discrete global clock is not accessible by the processes but helps to formulate the properties. The system model behaves benignly because it ensures eventual symmetry of reachable (unreachable) processes and fair channels.

Then, the partitionable group membership service (PGMS) comprises the properties of View Accuracy, View Completeness, View Coherency, View Order, and View Integrity. The authors discuss PGMS and show that for every implementation of PGMS a run exists that violates a property. Thus, it is impossible to solve PGMS. In the next step, failure detectors are employed to detect crashed processes. The failure detector exhibits the two properties of Strong Completeness and Eventual Strong Accuracy. Theoretically, the implementation of the required failure detectors is impossible, too, in asynchronous systems. However, they reflect the stability condition of the distributed system if they are satisfied. Together with the failure detector, the group membership becomes solvable and an implementation is presented. Finally, a reliable multicast service complementing PGMS is specified. Montresor [Mon00] uses the specification and algorithms in the context of object-oriented programming.

3.1.6 Summary on Related Work

We use the work on PGMS [BDM98] as a starting point for our membership service definition. Our system model differs from those presented so far because it allows a potentially infinite number of processes and these processes may start at arbitrary times. We cannot assume knowledge of the network

in terms of the size of population, the topology or the existence of dedicated servers.

In contrast to other work, we use certain time limits rather than stating properties to hold eventually. We assume that all processes finally crash (e.g. the driver parks the car) and the operation should be guaranteed to have finished before. Refer to Table 3.2 on page 75 that compares [BDM98] with our approach to a group membership service.

3.2 Notation

In the remaining part of this chapter, we specify formally the asynchronous system model and our proposed group membership service using temporal logic operators. This section briefly recalls the notation of Manna and Pnueli's temporal logic [MP92]. For the formal definition of the temporal logic operators refer to Appendix A. The time is linear and discrete, starting from an initial point. Thus, the past time operators can at most reach back until this starting point. State formulae are always evaluated from the current time. Besides the common boolean connectives $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), we use the quantifiers $\exists$ (existential) and $\forall$ (universal).

We make use of four basic future and four basic past operators. The $\bigcirc$ ("next") operator evaluates a formula in the following time step. Additionally, the abbreviation $\bigcirc^k$ denotes the k-times concatenation of $\bigcirc$. The $\Box$ ("henceforth," "always") operator describes all future time points. A bounded version $\Box_{\leq k}$ limits the validity of the $\Box$ operator to the next k time steps. If a formula holds at some future point, we use the $\Diamond$ ("eventually," "sometimes") operator. The bounded version $\Diamond_{\leq k}$ again limits the validity of the $\Diamond$ operator to the next k time steps. The last future operator is $\boldsymbol{\mathcal{U}}$; the binary operator expresses the validity of one formula until another formula holds.

The past operators are equivalent to the above mentioned future operators. The time step before the current time can be evaluated with the $\ominus$ ("previous") operator. Again, the k-times concatenation is $\ominus^k$. If a formula has always held or at least for the last k time steps, we write $\boxminus$ ("has-always-been") and $\boxminus_{\leq k}$ respectively. If we know that the formula has held once or once within the last k time steps, we use $\diamondminus$ ("once") and $\diamondminus_{\leq k}$ respectively. Finally, the $\boldsymbol{\mathcal{S}}$ ("since") operator applies to the case that a formula holds

Future	$\bigcirc$	"next"
	$\bigcirc^k$	k-times "next"
	$\Box$	"henceforth," "always"
	$\Box_{\leq k}$	bounded "henceforth," "always" (within next k time steps)
	$\Diamond$	"eventually," "sometimes"
	$\Diamond_{\leq k}$	bounded "eventually," "sometimes" (within next k time steps)
	$P \mathcal{U} Q$	"until" (Q happens eventually and until then P holds)
Past	$\ominus$	"previous"
	$\ominus^k$	k-times "previous"
	$\boxminus$	"has-always-been"
	$\boxminus_{\leq k}$	bounded "has-always-been" (within last k time steps)
	$\diamondminus$	"once"
	$\diamondminus_{\leq k}$	bounded "once" (within last k time steps)
	$P \mathcal{S} Q$	"since" (Q happened before and since then P holds)

Table 3.1: Overview on temporal logic operators

since another one has held at some past time. We derive the bounded operators from Henzinger et al. [HMP91]. Table 3.1 gives an overview of the temporal operators.

3.3 System Model

In this section, we define a model of the asynchronous distributed system matching our application in inter-vehicle communication. Henceforth, the system model covers all interacting entities during time. We use the temporal logic operators from [MP92, HMP91] introduced above to express logic formulas with respect to time. The model explains the behavior of single entities as well as the characteristics of the communication among them.

3.3.1 Time

The global time is modeled by a discrete global clock whose ticks are referenced by an infinite set isomorphic to the natural numbers $T \cong \mathbb{N}$. While

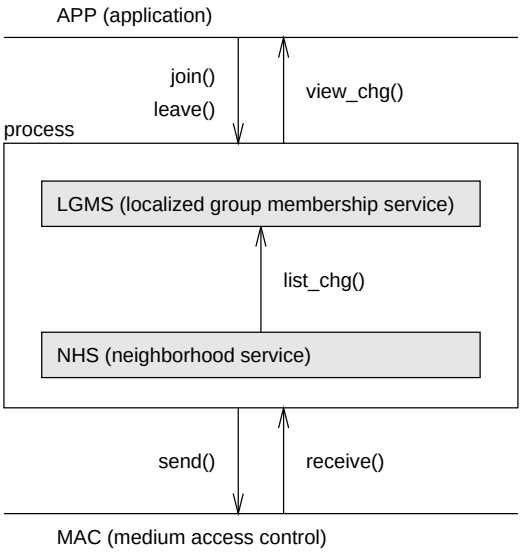


Figure 3.1: Architecture of a process

there is no correspondence between the ticks in the model and the real time measured in seconds, we can achieve an arbitrary fine granularity by inserting null events ϵ in the execution sequence of a process. However, processes have no access to the global clock because they run in an asynchronous manner.

Definition 3.1 (Global Time). *The global time T ticks through an infinite set isomorphic to the natural numbers $\mathbb{N}$.*

In the following, all definitions and properties correspond to a point in time. To enhance readability, we omit the explicit notation of time. However, as a reminder we mention in the informal description that a formula is to be evaluated with respect to time.

3.3.2 Processes

The distributed system consists of processes p which communicate solely via messages sent through a channel. Processes have a unique identifier p, ($p \in \mathbb{N}$) of which they are aware. Exactly one process exists on every equipped vehicle. Therefore, we will use the terms vehicle, node, host and process interchangeably. There is no common clock or common memory accessible from the processes, and the relative speed of processes is undetermined.

In contrast to most models found in the literature, the number of processes is unbounded but finite at every point in time. New processes can be created and existing processes can be destroyed. The latter corresponds to crash failures. Each process executes by performing events from a finite set S of valid events sequentially. The communication between processes is modeled by send() and receive() events. A process interacts with its application by join(), leave() and view_chg() events. The application signals with a join() event that it enters the group membership, and with a leave() event that it terminated its membership. The process performs a view_chg() event if a new view of the current group membership is installed at the process. Inside the process, two services communicate via a list_chg() event that establishes a new list of current neighbors. Note that the view of the membership is a subset of the list of current neighbors. While a process is idle it performs the null event ϵ. Figure 3.1 provides an overview of the architecture of a process.

The function σ captures the sequence of events performed by every process. In addition to the above mentioned events, the dead event δ models process failure that is described in detail in Section 3.3.5. As mentioned above, we omit the explicit notation of time in our definitions. For example, the global event history is a function of time and processes $\sigma(t, p)$. Using the temporal logic operators, we can reduce the parameter of time by writing $\sigma(t+1, p) \equiv \bigcirc\sigma(t, p)$ short as $\bigcirc\sigma(p)$.

Definition 3.2 (Global Event History). *The global event history σ describes the sequences of events that every possible process executes over time. Formally,*

$$\sigma : \mathbb{N} \rightarrow S \cup \{\delta, \epsilon\}$$

$$p \mapsto \sigma(p) := \begin{cases} \delta & \text{if } p \text{ is inactive} \\ \epsilon & \text{if } p \text{ is idle} \\ e \in S & \text{if } p \text{ performs event } e \end{cases}$$

There is no assumption on the underlying network topology. It is very unlikely that all vehicles are in transmission range of each other, hence the network structure is not fully connected (i.e. the corresponding graph is not a clique). Furthermore, we cannot even assume that the imposed network graph is connected all the time i.e. a path between every pair of nodes exists. In case part of the network becomes disconnected from another part, the network is called fragmented or partitioned.

Processes are neither aware of the other processes nor of the momentary network topology. If any service within a host presumes knowledge on these issues, it has to build it through the messages it has received lately.

3.3.3 Location and Mobility

Each process is associated with a location in time and space. Even though the location changes continuously while a vehicle is driving, we model location updates in discrete steps of the global clock.

Definition 3.3 (Global Location History). *The global location history λ describes the location of processes during time. Formally,*

$$\begin{aligned}\lambda &: \mathbb{N} \to \mathbb{R} \times \mathbb{R} \\ &p \mapsto \lambda(p) := (x, y) \text{ if } p \text{ is at location } (x, y)\end{aligned}$$

Depending on the location of vehicles in space, wireless communication is only possible when vehicles are in transmission range $r \in \mathbb{R}$ of each other. This imposes a neighborhood relation among processes at each global clock tick.

Definition 3.4 (Neighborhood Relation). *Two distinct processes p and q are neighbors at time t iff* [1] *their Euclidean distance is not greater than a given range $r \in \mathbb{R}$. Formally,*

$$\begin{aligned}&\leftrightarrow \subseteq \mathbb{N} \times \mathbb{N} \\ &\forall p, q \in \mathbb{N} : p \leftrightarrow q :\Leftrightarrow p \neq q \wedge \sqrt{(\lambda(p).x - \lambda(q).x)^2 + (\lambda(p).y - \lambda(q).y)^2} \leq r\end{aligned}$$

The set N_p denotes all neighbors of process p at a global clock tick. Formally,

$$\forall p \in \mathbb{N} : N_p := \{q \in \mathbb{N} | p \leftrightarrow q\}$$

It is easy to see that the neighborhood relation is commutative for a fixed point in time.

3.3.4 Communication

The messages are taken from a set $\mathcal{M}$ of valid messages.

[1] "iff" is short for "if and only if."

The radio channel is unreliable and thus messages can be lost; however, if the message is received then we assume it to be correct. Due to the broadcasting nature of radio waves, one send event causes receive events potentially at all processes in transmission range simultaneously. We assume that the transmission delay while broadcasting a radio message is negligible, i.e. the message will be received immediately. In ad hoc radio networks, the transmission power of antennas typically covers a range of a few hundred meters up to about one kilometer. Taking a transmission range of $r = 1$ km into account, the transmission delay a of radio waves traveling at the speed of light $c = 3 \cdot 10^8 \ \frac{\mathrm{m}}{\mathrm{s}}$ equals $a = \frac{r}{c} \approx 3.3$ microseconds. This provides an upper bound of time difference for receivers scattered 1 km apart in an actual radio network. Still, the unreliability of wireless communication can prevent some receivers in range from actually getting the message.

An important property of communication is causality: Every reception of messages must be caused by someone sending the message. The sender must be also in vicinity of the receiver.

Property 3.1 (Causal Communication). *A pair $\langle \sigma, \lambda \rangle$ of global event and location histories exhibits the causal communication property iff receiving a message m at a process p requires another process q in vicinity to send it. Formally,*

$$\forall p \in \mathbb{N} : \sigma(p) = \mathit{receive(m)} \Rightarrow \exists q \in N_p : \sigma(q) = \mathit{send(m)}$$

3.3.5 Failures

Processes can fail by crashing permanently at any time of execution. After a process has crashed, it performs the dead event δ. Also, processes may start later than the beginning of the global time. From $t = 0$ until a process is started for the first time ever, it also performs dead events δ. We call these processes and the crashed processes inactive, whereas processes are active if they are idle or if they perform events from the set of valid events S. Processes are not aware of the point in time that they fail. This means that an active process cannot determine when it will crash.

Property 3.2 (Permanent Crashes). *A global event history σ exhibits the property of permanent crashes iff processes that have performed events other than δ and then perform δ since then only perform δ. Formally,*

$$\forall p \in \mathbb{N} : (\sigma(p) \neq \delta \wedge \bigcirc \sigma(p) = \delta) \Rightarrow \Box \sigma(p) = \delta$$

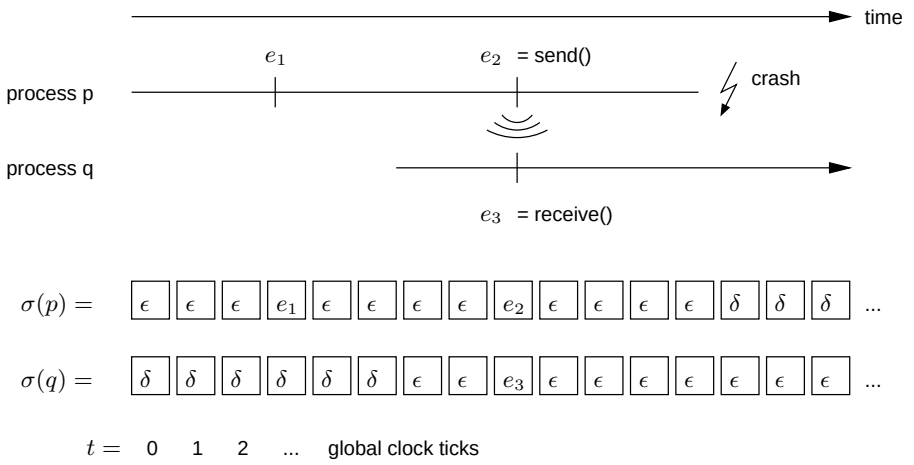


Figure 3.2: Modeling scheme of global event history

Property 3.3 (Finite Active Processes). *A global event history σ exhibits the finite active processes property iff the set of all active processes for a given time is always finite. Formally,*

$$\Box(|\{p \in \mathbb{N} | \sigma(p) \neq \delta\}| < \infty)$$

Other than crash failures, the nodes in the distributed system behave benignly. In particular, we do not consider arbitrary, malicious, or Byzantine faults like sending spurious messages or exhibiting any unpredictable behavior. Authorization mechanisms could prevent the system from intruders. However, applying such security features contradicts the desired openness of a network which aims to maximize its potential through many interacting entities. The compromise between easy access and restricted usage must be carefully crafted to let many people benefit from the application of ad hoc networks.

We illustrate the abstract model of the global event history with the schematic drawing in Figure 3.2.

A variety of reasons can inhibit communication over wireless links. In our model, communication failures are grouped into send and receive omission. We define processes which suffer from link failures to be elements of a set F_σ.

Definition 3.5 (Link Failures). *The set F associated with a global event history σ denotes processes that are subject to link failures (send or receive*

omission) at a global time tick. Formally,

$$F_\sigma \subseteq \{p \in \mathbb{N} | \sigma(p) \in \{\textsf{send()}, \textsf{receive()}\}\}$$

A send omission happens if the underlying medium access control (MAC) fails in claiming the channel for transmission before a time-out occurs. We propose a simple MAC that drops send requests while being busy with sending or receiving another packet. Then, a process experiences a send omission failure.

Definition 3.6 (Sending). *A process p successfully sends a message m iff p performs a* send(m) *event and does not suffer from a send omission fault. Formally,*

$$\text{snd} : \mathbb{N} \times \mathcal{M} \to \textsc{Boolean}$$
$$(p, m) \mapsto \text{snd}(p, m) := \begin{cases} \textsc{True} & \textit{if } \sigma(p) = \textsf{send(m)} \wedge p \notin F_\sigma \\ \textsc{False} & \textit{otherwise} \end{cases}$$

Packet loss in wireless radio communication can occur in the presence of strong multipath fading or because of shadowing effects if the chosen frequency demands line of sight. Additionally, atmospheric dilution may cause packet errors. We say that a receiving host suffers from a receive omission fault, if a packet is lost.

Definition 3.7 (Receiving). *A process p successfully receives a message m from a set M iff p performs a* receive(m) *event and sender and receiver do not suffer from an omission fault. Formally*[2]*,*

$$\text{rcv} : \mathbb{N} \times 2^{\mathcal{M}} \to \textsc{Boolean}$$
$$(p, M) \mapsto \text{rcv}(p, M) := \begin{cases} \textsc{True} & \textit{if } \exists m \in M : \\ & \qquad (\sigma(p) = \textsf{receive(m)} \wedge p \notin F_\sigma \wedge \\ & \qquad \exists q \in N_p : \text{snd}(q, m)) \\ \textsc{False} & \textit{otherwise} \end{cases}$$

For $|M| = 1$ we write $\text{rcv}(p, m)$ *short for* $\text{rcv}(p, \{m\})$.

Another well known problem in radio networks that lack full connectivity is packet loss due to hidden stations. Consider two senders q and s being out of range of each other but a receiver p sits in the middle of q and s hearing them

[2]For any set S given, 2^S denotes the power set of subsets of S.

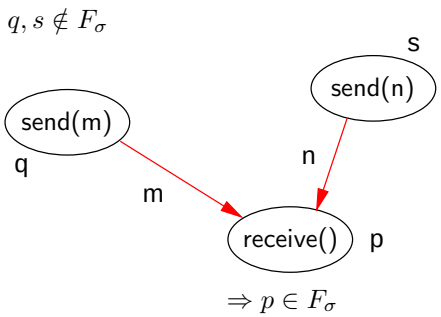


Figure 3.3: Example on hidden station problem

both. Now, q and s may start transmitting a packet simultaneously. In this case, the packets from q and s collide at the receiver C. We incorporate this effect in the hidden station property 3.4 defined below. There, we prohibit a process successfully performing the receive() event when two different senders are in vicinity at the same time. An example on a packet collision is depicted in Figure 3.3.

Property 3.4 (Hidden Station). *A triple $\langle \sigma, \lambda, F_\sigma \rangle$ of global event and location histories and link failures exhibits the hidden station property iff a receiving process p suffers from an omission fault when at least two different processes q, s in vicinity send messages. Formally,*

$$\forall p \in \mathbb{N} : (\sigma(p) = \mathit{receive}(\cdot) \wedge \exists q, s \in N_p,\ m, n \in \mathcal{M} : \mathrm{snd}(q, m) \wedge \mathrm{snd}(s, n)) \\ \Rightarrow p \in F_\sigma$$

3.3.6 Summary on System Model

The above introduced concepts comprise a system model which is called a run. The following specification of desired behavior applies to such runs. By observing the set of all possible runs, we can reason about whether a protocol satisfies certain properties within this system model.

Definition 3.8 (Run). *A run R is a triple $\langle \sigma, \lambda, F_\sigma \rangle$ of global event and location histories and link failures, if it satisfies the properties of causal communication 3.1, permanent crashes 3.2, finite active processes 3.3, and hidden stations 3.4.*

3.4 Neighborhood Service

Failure detectors are an effective means for reaching agreement in distributed systems that are prone to failures. At each process, a local module of the failure detector is installed and provides to its owner a negative list of processes that are suspected to be crashed or unreachable. However, in our model, we assume that the number of active processes over time is unbound in any run. Thus, a list of suspected processes could be infinitely long. We propose instead to employ a similar service that yields a positive list of processes which are expected to be alive at the moment. The service is based on a simple heartbeat mechanism that repeatedly beacons the own process identity to its neighbors with a fixed rate τ_{hb}. Such a beacon message is very short and should not be forwarded. The local neighborhood service of a process collects the heartbeats from other adjacent processes and maintains a list of current neighbors. It also sets a timer upon receiving a heartbeat from another process: If this timer expires without having received an update on the heartbeat, the process identity is then discarded from the list.

Property 3.5 (Heartbeats). *A run $R = \langle \sigma, \lambda, F_\sigma \rangle$ exhibits the heartbeat property, if every active process p sends out heartbeat messages* send(hb:p) *with a rate τ_{hb}. Formally,*

$$\forall p \in \mathbb{N} : \sigma(p) \neq \delta \Rightarrow \Diamond_{\leq \tau_{hb}} \sigma(p) = \textit{send(hb:p)}$$
$$\forall p \in \mathbb{N} : \sigma(p) = \textit{send(hb:p)} \Rightarrow \bigcirc^{\tau_{hb}} \sigma(p) = \textit{send(hb:p)}$$

With the help of the above defined heartbeats, we formalize the concept of a process being connected to another process. Note that this concept is inherently not symmetric; a process that receives consecutive heartbeats suspects another process to be its neighbor—no assumption is made on how the other process perceives that process. The beginning of a connection is the first reception of a heartbeat after the duration of at least the heartbeat rate in which the process has not received a heartbeat. The end of a connection equals the time-out of waiting for the next heartbeat. This interval denotes a transient connection of one process to another. Figure 3.4 shows an example of a connection to explain the concepts introduced here.

Definition 3.9 (Newly Connected). *A process q is newly connected to a process p at time t iff p receives a heartbeat from q at time t and has not*

received a heartbeat at $t - \tau_{hb}$. Formally,

$$\triangleright \subseteq \mathbb{N} \times \mathbb{N}$$
$$\forall p, q \in \mathbb{N} : \; q \triangleright p :\Leftrightarrow \mathrm{rcv}(p, hb{:}q) \wedge \neg \ominus^{\tau_{hb}} \mathrm{rcv}(p, hb{:}q)$$

Definition 3.10 (Disconnected). *A process q is disconnected from a process p at time t iff p received a heartbeat from q at time $t - \tau_{hb}$ but did not receive a heartbeat from q at time t. Formally,*

$$\triangleleft \subseteq \mathbb{N} \times \mathbb{N}$$
$$\forall p, q \in \mathbb{N} : \; q \triangleleft p :\Leftrightarrow \neg\, \mathrm{rcv}(p, hb{:}q) \wedge \ominus^{\tau_{hb}} \mathrm{rcv}(p, hb{:}q)$$

Definition 3.11 (Transiently Connected). *A process q is transiently connected to a process p at time t iff q has not been disconnected from p since q has been newly connected to p. Formally,*

$$\bowtie \subseteq \mathbb{N} \times \mathbb{N}$$
$$\forall p, q \in \mathbb{N} : \; q \bowtie p :\Leftrightarrow (\neg q \triangleleft p) \, \boldsymbol{\mathcal{S}} (q \triangleright p)$$

The relation "disconnected" is not complementary to the relation "transiently connected." Two processes can be neither transiently connected nor disconnected. Also, if a process gets disconnected from another process then they can still be neighbors. For example, one of the processes can suffer from a send or receive omission failure during the heartbeat.

Before we summarize the desired behavior of the membership service, we specify the effect of a process executing the list_chg() event.

Definition 3.12 (List of Neighbors). *A process p maintains a list L of current neighbors that is installed with the last list_chg(L) event at p. If p has not performed a list_chg() event yet or is crashed then the list is empty. Formally,*

$$list : \mathbb{N} \to 2^{\mathbb{N}}$$
$$p \mapsto list(p) := \begin{cases} L & \text{if } \sigma(p) \neq \delta \wedge \\ & \quad (\sigma(p) \neq \textit{list\_chg()}) \, \boldsymbol{\mathcal{S}} (\sigma(p) = \textit{list\_chg(L)}) \\ \emptyset & \text{otherwise} \end{cases}$$

Now, we use the notation introduced in this chapter to specify the neighborhood service (NHS). The service should react with list_chg() events if the

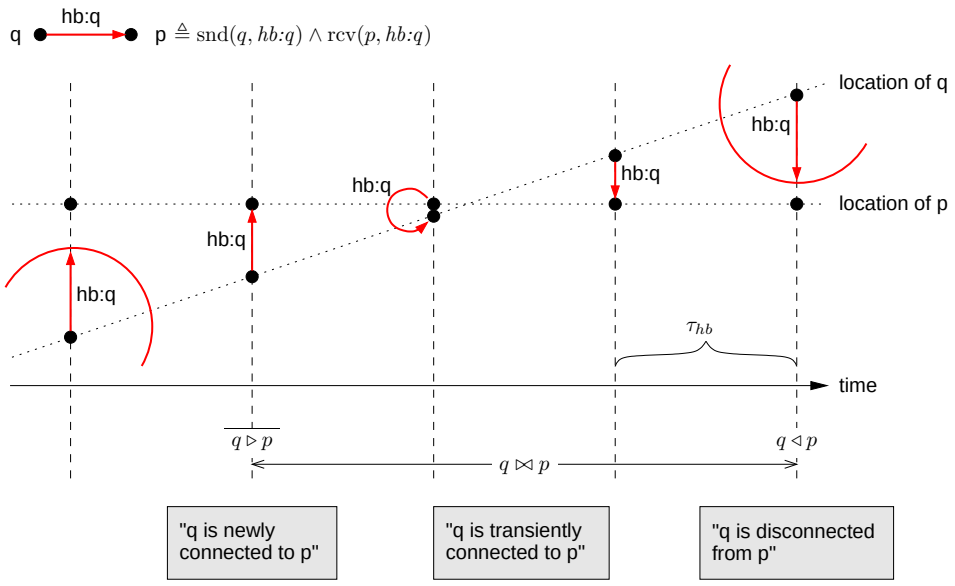


Figure 3.4: Example on heartbeats from process p's perspective

process gets newly connected or disconnected from another process. Also, the list reported to the upper layer must be accurate and complete such that it only includes those processes assumed to be neighbors from which it has recently received heartbeats.

NHS 1 (New Neighbors). *If a process q is newly connected to a process p at time t then p performs at least once a* list_chg() *event within the next $\tau_{hb} - 1$ time steps. Formally,*

$$\forall p, q \in \mathbb{N} : \; q \triangleright p \Rightarrow \Diamond_{\leq \tau_{hb}-1} \sigma(p) = \textit{list\_chg()}$$

NHS 2 (Leaving Neighbors). *If a process q is disconnected from a process p at time t then p performs at least once a* list_chg() *event within the next $\tau_{hb} - 1$ time steps. Formally,*

$$\forall p, q \in \mathbb{N} : \; q \triangleleft p \Rightarrow \Diamond_{\leq \tau_{hb}-1} \sigma(p) = \textit{list\_chg()}$$

NHS 3 (Accuracy). *If a process p installs a list by performing a* list_chg() *event at time t and q is transiently connected to a process p then the installed list includes q. Formally,*

$$\forall p, q \in \mathbb{N} : \; \sigma(p) = \textit{list\_chg()} \wedge q \bowtie p \Rightarrow q \in \mathit{list}(p)$$

NHS 4 (Completeness). *If a process p installs a list by performing a* list_chg() *event at time t and q is not transiently connected to a process p then the installed list excludes q. Formally,*

$$\forall p, q \in \mathbb{N} : \sigma(p) = \textit{list\_chg()} \wedge \neg q \bowtie p \Rightarrow q \notin list(p)$$

3.5 Localized Group Membership Service

For the sake of brevity, we assume that only one group exists in each run to omit group identifiers. This implies that in the case of multiple groups the characteristics of a group can be communicated as a small description of parameters such that processes can distinguish them. Processes decide upon local parameters for their own membership. We model this by the application issuing join() and leave() events to its own group membership layer. At each member process, a localized group membership service (LGMS) tracks the group membership of the adjacent neighbors. Changes in the localized group membership—existing neighbors join or leave the group voluntarily or crash, new member processes move into vicinity—are installed as views through the view_chg() event.

Definition 3.13 (Membership). *A process p is a member of the group at a global clock tick iff p has performed the* join() *event before and since then, p neither performed the* leave() *event nor crashed. Formally,*

$$member : \mathbb{N} \rightarrow \textsc{Boolean}$$

$$p \mapsto member(p) := \begin{cases} \textsc{True} & \textit{if } \sigma(p) \neq \delta \wedge \\ & \quad (\sigma(p) \neq \textit{leave()}) \, \boldsymbol{\mathcal{S}} (\sigma(p) = \textit{join()}) \\ \textsc{False} & \textit{otherwise} \end{cases}$$

For a meaningful group membership service, we assume that the application layer of an active process always alternates the join() and leave() events starting with the join() event. The property of correct applications formalizes this.

Property 3.6 (Correct Application). *A run $R = \langle \sigma, \lambda, F_\sigma \rangle$ exhibits the correct application property, if for every process p performing a* join() *event, a* leave() *event must be performed before p performs the next* join() *event. For every process p performing a* leave() *event, a* join() *event must be performed before p performs the next* leave() *event. Finally, a process p starting*

to operate, performs a join() *event before it may perform a* leave() *event. Formally,*

$$\forall p \in \mathbb{N} : \sigma(p) = \textit{join()} \Rightarrow \sigma(p) \neq \textit{join()} \mathcal{U}\, \sigma(p) = \textit{leave()}$$
$$\forall p \in \mathbb{N} : \sigma(p) = \textit{leave()} \Rightarrow \sigma(p) \neq \textit{leave()} \mathcal{U}\, \sigma(p) = \textit{join()}$$
$$\forall p \in \mathbb{N} : (\sigma(p) = \delta \wedge \bigcirc \sigma(p) \neq \delta) \Rightarrow \sigma(p) \neq \textit{leave()} \mathcal{U}\, \sigma(p) = \textit{join()}$$

The view of a member process is a set of process identifiers that are neighbors and members as well. A view reflects the current situation of the membership from the perspective of a certain process.

Definition 3.14 (View). *The view of a member process p at time t is a set V that has been installed with the last* view_chg(V) *event at p since p is a member. In particular, if p has not installed a view since it is a member, the list is empty. The view of a non-member is always empty. Formally,*

$$\textit{view} : \mathbb{N} \rightarrow 2^{\mathbb{N}}$$
$$p \mapsto \textit{view}(p) := \begin{cases} V & \textit{if } \textit{member}(p) \wedge \\ & \quad (\sigma(p) \neq \textit{view\_chg()} \wedge \sigma(p) \neq \textit{join()}) \\ & \quad\quad \mathcal{S}(\sigma(p) = \textit{view\_chg(V)}) \\ \emptyset & \textit{otherwise} \end{cases}$$

We introduce a timing value τ_{vc} for the view to change. After a process performs a join() and leave() event in the LGMS layer, the process must react within the next τ_{vc} time steps by installing a new view through the view_chg() event. Other situations in which a member process has to install a new view are drawn in Figure 3.5. There, three constellations require a member process p to adjust its view: If a becomes a member, p must include a into its view. If b leaves the group, p must remove b from its view. Finally, c must be excluded from p's view because $c \notin \textit{list}(p)$ and thus c is not a neighbor of p.

Now, we define the properties LGMS 1–5 of the sketched localized group membership service. We make use of the concepts introduced above and the neighborhood service of the underlying layer that reports changes in the neighborhood through the list_chg() event.

LGMS 1 (View Integrity). *(i) Every view installed at a member process p includes the process itself. Formally,*

$$\forall p \in \mathbb{N} : \textit{member}(p) \wedge \sigma(p) = \textit{view\_chg()} \Rightarrow p \in \textit{view}(p)$$

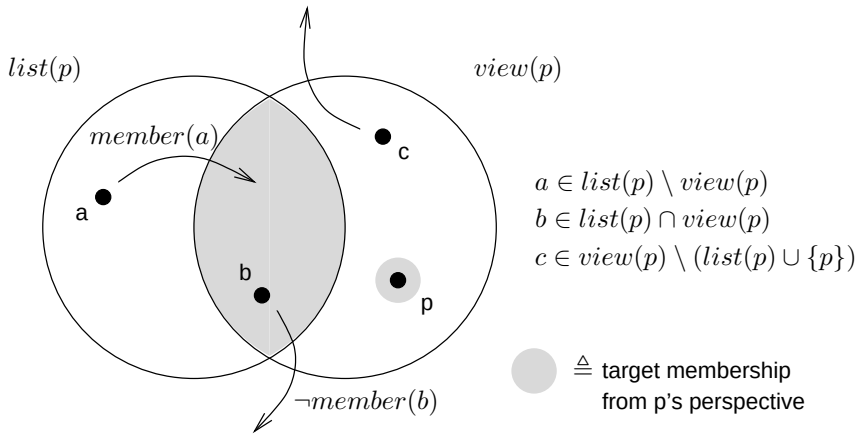


Figure 3.5: Three situations that require view changes at member p

(ii) Every view installed at a non-member process p is empty. Formally,

$$\forall p \in \mathbb{N} : \neg member(p) \wedge \sigma(p) = \textit{view\_chg()} \Rightarrow view(p) = \emptyset$$

LGMS 2 (Limit on Neighborhood). *Only neighbors are part of a view installed at a member process p. Formally,*

$$\forall p \in \mathbb{N} : member(p) \wedge \sigma(p) = \textit{view\_chg()} \Rightarrow view(p) \subseteq list(p) \cup \{p\}$$

LGMS 3 (View Accuracy). *If a member process p has a neighbor process q which is a member in its view, then q remains in p's view until q is not a neighbor anymore or p or q leaves the group. Formally,*

$$\begin{aligned}\forall p \in \mathbb{N} : & member(p) \wedge \exists q \in (view(p) \cap list(p)) \setminus \{p\} : member(q) \\ & \Rightarrow q \in view(p) \boldsymbol{\mathcal{U}} (q \notin list(p) \vee \neg member(q) \vee \neg member(p))\end{aligned}$$

LGMS 4 (View Completeness). *If a member process p has a neighbor process q which is not in p's view nor a member, then q is excluded form p's view until q is not a neighbor anymore or q becomes a member or p leaves the group. Formally,*

$$\begin{aligned}\forall p \in \mathbb{N} : & member(p) \wedge \exists q \in list(p) \setminus view(p) : \neg member(q) \\ & \Rightarrow q \notin view(p) \boldsymbol{\mathcal{U}} (q \notin list(p) \vee member(q) \vee \neg member(p))\end{aligned}$$

LGMS 5 (View Installation). *(i) If a process p joins or leaves the group, it installs a new view within the next τ_{vc} time steps. Formally,*

$$\forall p \in \mathbb{N} : \sigma(p) \in \{\textit{join()}, \textit{leave()}\} \Rightarrow \Diamond_{\leq \tau_{vc}} \sigma(p) = \textit{view\_chg()}$$

(ii) If a neighbor process q of a member process p is a member but not included in p's view, then p includes q in its view within the next τ_{vc} time steps or q is not a neighbor anymore or p or q leaves the group. Formally,

$$\begin{aligned}\forall p \in \mathbb{N} : & \, member(p) \wedge \exists q \in list(p) \setminus view(p) : member(q) \\ & \Rightarrow \Diamond_{\leq \tau_{vc}} (q \in view(p) \vee q \notin list(p) \vee \neg member(q) \vee \neg member(p))\end{aligned}$$

(iii) If a neighbor process q included in a member process p's view is not a member, then q is excluded from p's view within the next τ_{vc} time steps or q becomes a member or p leaves the group. Formally,

$$\begin{aligned}\forall p \in \mathbb{N} : & \, member(p) \wedge \exists q \in list(p) \cap view(p) : \neg member(q) \\ & \Rightarrow \Diamond_{\leq \tau_{vc}} (q \notin view(p) \vee (q \in list(p) \wedge member(q)) \vee \neg member(p))\end{aligned}$$

(iv) If a process q included in a member process p's view is not a neighbor, then q is excluded from p's view within the next τ_{vc} time steps or q becomes a neighbor and a member or p leaves the group. Formally,

$$\begin{aligned}\forall p \in \mathbb{N} : & \, member(p) \wedge \exists q \in view(p) \setminus (list(p) \cup \{p\}) \\ & \Rightarrow \Diamond_{\leq \tau_{vc}} (q \notin view(p) \vee (q \in list(p) \wedge member(q)) \vee \neg member(p))\end{aligned}$$

3.6 Some Thoughts on Impossibility

Before we reason about the service specifications in our system model, we extend the execution model. The two services NHS and LGMS of a process are mainly independent. Therefore, we see them as two distinct threads running concurrently within a process. To fit this into our model, we split each time step into two half-steps—the first corresponds to the NHS layer and the second equals the action in the LGMS layer. An exception from independence occurs when so called shared events happen for inactive processes and for both threads communicating with each other through the list_chg() event. Figure 3.6 illustrates this construction.

Property 3.7 (Shared Events). *A run $R = \langle \sigma, \lambda, F_\sigma \rangle$ with σ split into σ_{nhs} and σ_{lgm} exhibits the property of shared events, if both σ_{nhs} and σ_{lgm}*

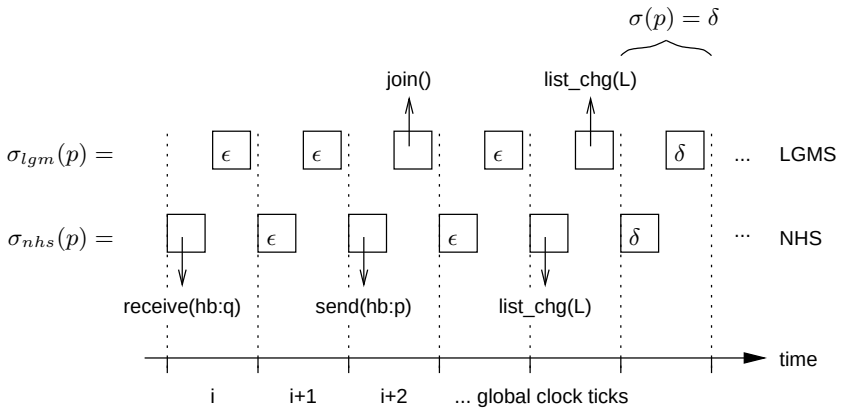


Figure 3.6: Modeling two concurrent threads

are inactive at the same clock tick and perform the ***list_chg()*** *event at the same clock tick. Formally,*

$$\forall p \in \mathbb{N} : \sigma_{nhs}(p) = \delta \Leftrightarrow \sigma_{lgm}(p) = \delta \; [\triangleq \sigma(p) = \delta]$$
$$\forall p \in \mathbb{N} : \sigma_{nhs}(p) = \textit{list\_chg()} \Leftrightarrow \sigma_{lgm}(p) = \textit{list\_chg()} \; [\triangleq \sigma(p) = \textit{list\_chg()}]$$

Ever since formal models of fault-tolerant systems have been used to specify properties of these systems, people have reasoned about impossibility, i.e. no implementation exists that meets a given specification. For group membership, Chandra et al. [CHTCB96] have proven the impossibility of providing a group membership service in asynchronous systems with crash failures. However, this result strictly applies to primary-partition membership services, which allow only one network component—called the primary component or partition—to continue running the service, whereas processes in other networks are considered faulty. In contrast, a partitionable membership service relaxes the rigorous demand of delivering the same sequence of views to all members and allows multiple disjoint views to exist concurrently in different network components. In the context of mobile ad hoc networks, temporary disconnections occur frequently. Hence, the service should be partitionable and therefore escapes the impossibility proof.

Escaping this impossibility result and creating a new membership service results in two problematic situations: The definition of the service can be too weak and is satisfied by useless protocols. Second, the specification can be too strong such that no protocol satisfies it in every possible run. The latter would yield another impossibility result. In this section, we show that the proposed LGMS is not solvable in our system model and discuss

the reasons. From there, we propose performance properties to mend these limitations. Finally, we point out in which sense the specification allows useless protocols to achieve LGMS and provide more detailed properties to overcome trivial solutions.

3.6.1 Impossibility in Our Model

Crashes

The NHS as well as the LGMS require processes to react upon certain events in a timely manner. It is easy to see that if a process crashes immediately after performing such a triggering event, it cannot carry out the mandatory reaction because of the permanent crashes property 3.2. To circumvent this problem, we explicitly add the possibility of crashes in property 3.5, NHS 1 and NHS 2, and LGMS 5. Abstractly, a formula $A \Rightarrow \Diamond_{\leq k} B$ is rewritten as $A \Rightarrow \Diamond_{\leq k}(B \vee \sigma(p) = \delta)$ for every p that is a scoped variable of a process by predicate B. Read the formula as "If A then within the next k time steps B happens or the process crashes."

Also, we incorporate in property 3.6, LGMS 3 and LGMS 4 that processes may crash. Manna and Pnueli [MP92] have defined the "until" operator $B \,\mathcal{U}\, C$ such that C must happen eventually. Thus, we replace formulas like $A \Rightarrow B \,\mathcal{U}\, C$ with $A \Rightarrow B \,\mathcal{U}(C \vee \sigma(p) = \delta)$ where p represents again every variable of a process scoped by predicate C. In these cases, read the formula as "If A then B holds until C happens or the process crashes."

Timing and Performance

Another problem in our membership service specification is the explicit notation of time limits like τ_{hb} and τ_{vc}. We need time limits because our system model allows every process to eventually crash and we aim to specify a meaningful service while processes are active. Therefore, we craft properties with timing constraints rather than allowing them to hold eventually. When these time limits are too short in an actual scenario, the service becomes impossible to be implemented.

Heartbeat. Consider a scenario with $\hat{N} > 1$ hosts; all in vicinity of each other. The heartbeat rate is $\tau_{hb} = \hat{N}$. Assume no communication failures i.e. $\Box F_\sigma = \emptyset$. All processes start to operate at time $t = 1$. We observe the

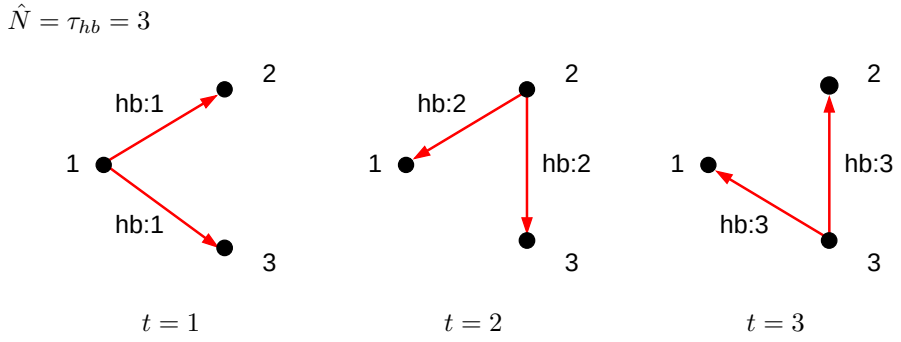


Figure 3.7: Example on impossibility due to small τ_{hb}

time interval $t \in \{1, ..., \hat{N}\}$ in which no process crashes. All processes are perfectly synchronized and send out their heartbeat messages one after each other:

$$\forall t \in \{1, ..., \hat{N}\} : \sigma_{nhs}(p) = \mathsf{send(hb:t)} \wedge$$
$$\forall p \in \{1, ..., \hat{N}\} \setminus \{t\} : \sigma_{nhs}(p) = \mathsf{receive(hb:t)}$$

An example for $\hat{N} = 3$ is depicted in Figure 3.7.

At time $t = 1$, the process 1 is newly connected to the other processes $2, ..., \hat{N}$. Then, NHS 1 requires all processes $2, ..., \hat{N}$ to perform a list_chg() event within the next $\tau_{hb} - 1 = \hat{N} - 1$ time steps. This contradicts that all processes already perform either send(hb) or receive(hb) events during the time steps $2, ..., \hat{N}$. As a consequence, τ_{hb} must be greater than the maximum of new neighbors that move into vicinity of a process and consecutively issue their heartbeats. Formally,

$$\tau_{hb} > \max\{k | \bigwedge_{i=0}^{k} \bigcirc^i (q_i \triangleright p \vee q_i \triangleleft p \vee \sigma_{nhs}(p) = \mathsf{send(hb:p)}), q_i \in \mathbb{N}, p \in \mathbb{N}\}$$

View Change. Naturally, we wish to set τ_{vc} as short as possible to provide an efficient membership service. However, processes that join or leave the group must inform the processes in their vicinity by sending a message regarding their membership status change. Also, in case the view becomes inaccurate when the neighborhood changes due to topology changes or lost heartbeats, the view must consolidate by exchanging messages among neighbors. If these messages get lost, the view installation cannot be guaranteed to be finished after τ_{vc} time steps as required by LGMS 5. Additionally, the number of messages regarding the view change cannot exceed τ_{vc} because

the process requires one time step for each message it receives. Therefore, a number of members among the neighbors greater than τ_{vc} that change their membership status contradicts our specification. Thus, we propose to use the time limit τ_{vc} to install the view regardless of whether the view is consolidated. This mechanism includes the possibility of an inaccurate view on the membership if messages during the consolidation get lost or too many changes are reported. We set a trust value $trust(p)$ upon the installation of a view at process p which is the ratio of determined neighbors plus one divided by the actual number of neighbors plus one. If the trust value is 1, the membership of all current neighbors is decided. We observe the value of $trust(p)$ in our simulation.

3.6.2 Trivial Solutions

On the other hand, when specifying a membership service, the question has to be posed whether the definition is too weak and and is satisfied by useless protocols. Two flavors of trivial solutions are prominent [ACBMT95]: capricious view splitting that allows singleton sets with only the own identity to be installed as views, and capricious view installation that allows members to install a priori determined views independent from the actual execution.

Capricious View Splitting

Consider a trivial protocol that immediately installs a view comprised of only the process itself upon joining the group.

$$\sigma_{lgm}(p) = \mathsf{join}() \Rightarrow \bigcirc \sigma_{lgm}(p) = \mathsf{view\_chg}(\{\mathsf{p}\})$$

Then, if a neighbor q of process p exists in $list(p)$ that is a member, we require in LGMS 5, (ii) p to install a view including q within the next τ_{vc} time steps. After this happened, LGMS 3 prevents p from switching back to the view of $\{p\}$ while q is a member. Thus, our specification prohibits trivial solutions that permanently install singleton sets as views. Still, in combination with the above mentioned strategy to accept inaccuracies in the membership, we do not guarantee LGMS 3 anymore. A careful consideration of the $trust()$ value must then preclude capricious view splitting. We propose to maximize the $trust()$ value. If process p has decided on q's membership because it received a positive message regarding q's membership, then p must include q in its view to increase the $trust()$ value.

Capricious View Installation

A group membership service usually requires that if certain events occur, then a new view of the group must be eventually installed to reflect the changes. In our model, we capture this in LGMS 5 where a view change is required after a certain time limit τ_{vc}. As pointed out in [ACBMT95], another requirement should then prevent capricious view changes, namely that a new view is installed *only* if certain events previously occurred. In our specification, we add LGMS 6 to overcome this problem.

LGMS 6 (View Justification). *If a process p installs a new view, one of the triggering events from the view installation property LGMS 5, (i)–(iv) happened before and since then no new view has been installed at p. Formally,*

$$\forall p \in \mathbb{N} : \sigma_{lgm}(p) = \mathit{view\_chg()} \Rightarrow (\sigma_{lgm}(p) \neq \mathit{view\_chg()} \, \mathcal{S} \, P)$$

where P is replaced by

$$\begin{aligned} P \leftarrow & \sigma_{lgm}(p) \in \{\mathit{join()}, \mathit{leave()}\} \vee && \textit{LGMS 5, (i)} \\ & (member(p) \wedge \exists q \in list(p) \setminus view(p) : member(q)) \vee && \textit{LGMS 5, (ii)} \\ & (member(p) \wedge \exists q \in list(p) \cap view(p) : \neg member(q)) \vee && \textit{LGMS 5, (iii)} \\ & (member(p) \wedge \exists q \in view(p) \setminus (list(p) \cup \{p\})) && \textit{LGMS 5, (iv)} \end{aligned}$$

3.7 Summary on Group Membership Service

In this chapter, we specified a localized group membership service that suits our application in highly mobile ad hoc networks. The discussion on related work has shown that defining a partitionable group membership service in asynchronous distributed systems is still an open question and an area of active research. Our specification was mainly inspired by [BDM98] although the application to highly mobile ad hoc networks made changes inevitable. We extended the idea of allowing different views to exists concurrently in distinct network partitions and reduced the membership problem to the local environment of each host. Thus, views at hosts typically differ according to the neighborhood relation. Table 3.2 compares [BDM98] with the specification of our localized group membership service (LGMS). Please note that the actual formulas of the membership service GM and LGMS are different

in spite of the same names. We use the name of a property to summarize the intention of the formula.

With the abstract system model and the membership service specification at hand, we discussed the problems of impossibility and trivial solutions that are reported in the literature [CHTCB96, ACBMT95]. From there, we provided additional characteristics of the system model and properties for the membership service. However, it is still impossible to implement the proposed service such that all properties are satisfied in every possible run. This is due to the generous specification of the distributed system in which we allow link failures to happen and hence messages to be lost arbitrarily. In order to prove the possibility of a solution in every execution, the system model must exhibit a certain benign behavior. Imposing a good nature of the system in turn limits the applicability of the theoretic results in reality. It is not the aim of this thesis to search for a benign system model which might not meet the requirements of applications in ad hoc networking. Rather, we focus on experiments with a localized group membership service by embedding the problem into a routing task in mobile ad hoc networks.

In the next chapter, we present an implementation of the membership service definition. We build an implementation of the routing problem on top of the membership algorithm. Then, we apply the protocols to highly mobile ad hoc networks in inter-vehicle communication and use the routing mechanism to determine the current size of a traffic jam.

Partitionable group membership service [BDM98]	Localized group membership service
Global clock with discrete time	
Unbounded set of processes with unique identifiers	
Asynchronous, processes have no access to global clock	
Channels connecting processes pairwise	Wireless channels connecting process star wise with processes
Causal communication	
Global history σ with event set $S \cup \{\epsilon\}$	Global history σ with event set $S \cup \{\delta, \epsilon\}$
Processes may crash prematurely and permanently	
Correct processes never crash	All processes eventually crash
	Processes may start at arbitrary times
Temporary link failures: processes pairwise (un-)reachable	Temporary link failures: send and receive omission
Eventual Symmetry of reachability (unreachability) Fair Channels	
	Neighborhood service: NHS 1 (New Neighbors), NHS 2 (Leaving Neighbors), NHS 3 (Accuracy), NHS 4 (Completeness)
Single process group	
Membership due to failures and crashes	Membership due to join(), leave() events and crashes
GM 1 (View Accuracy)	LGMS 3 (View Accuracy)
GM 2 (View Completeness)	LGMS 4 (View Completeness)
GM 3 (View Coherency), GM 4 (View Order)	LGMS 5 (View Installation), LGMS 6 (View Justification)
GM 5 (View Integrity)	LGMS 1 (View Integrity)
	LGMS 2 (Limit on Neighborhood)
Failure detectors: FD 1 (Strong Completeness), FD 2 (Eventual Strong Accuracy)	

Table 3.2: Comparison of [BDM98] with LGMS

Chapter 4

Implementing the Traffic Jam Detection

In this chapter, we integrate the concepts of anonymous routing, neighborhood and group membership service introduced in the previous chapters into an inter-vehicle communication system. As an example application, we use the system to detect a traffic jam on a highway. The application uses a new type of anonymous routing in highly mobile ad hoc networks that determines the recipient of a message not by an individual address but rather by geographic constraints and the momentary driving situation of equipped vehicles. The routing layer resides on top of a group membership service which maintains dynamically changing groups of equipped vehicles.

We use the Specification and Description Language (SDL) for developing and implementing our algorithms. SDL is an abstract specification language that provides enough details for an implementation. Professional tools generate executable code from an SDL model. The International Telecommunication Union (ITU) standardized SDL in their recommendation Z.100 [ITU99]. The key features of the language are (1) suitability for real-time and stimulus-response systems, (2) representation in a graphical form, (3) a model based on communicating processes which are extended finite state machines, and (4) object oriented description of SDL components. Although SDL is widely used in the telecommunications field, it is also now being applied to a diverse number of other areas ranging over aircraft, train control, medical and packaging systems.

SDL is a general purpose description language for communicating systems. The description of behavior is based on extended state machines that are

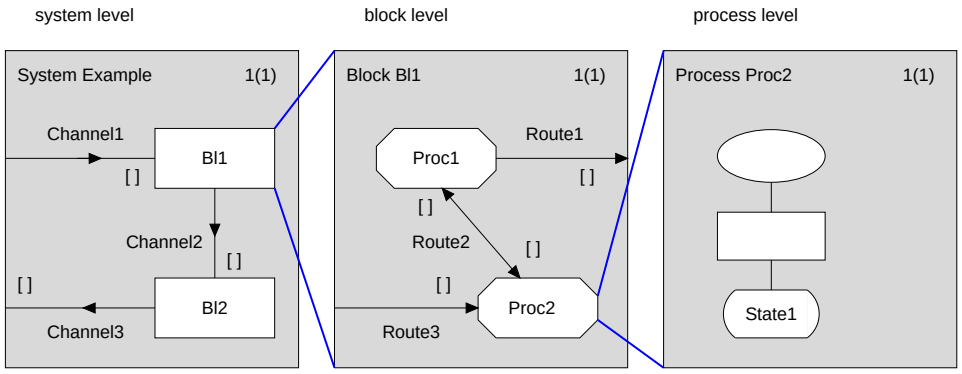


Figure 4.1: Hierarchical modeling with SDL

represented by SDL processes. An extended state machine consists of a number of states and of transitions connecting the states. One of the states is designated the initial state. Signals between SDL processes or between SDL processes and the environment of the system represent the communication abilities. Abstractions from the process level are denoted at block and system level. Thus, the modeled system consists of at least three hierarchical models (system, blocks, and processes) but can have additional levels at all stages. Figure 4.1 shows the hierarchies of SDL components. For a comprehensive introduction to SDL refer to [EHS97] or visit the web site of the SDL Forum Society [SDL].

4.1 Overview on System Architecture

The inter-vehicle communication system consists of four blocks. Figure 4.2 depicts the system architecture. The top block (*bTJam*) is responsible for detecting a traffic jam and uses two processes *pTJam* and *pMsgManagement* and a generic process type *ptResend*. The *pTJam* process issues the join and leave requests when the velocity of the vehicle exceeds or falls below certain values. Using the view on the local group membership, the *pTJam* process also classifies the vehicle to be at the beginning or at the end of the congestion. If the group has at least two members and the vehicle is at the border of the traffic jam, the *pTJam* process initiates a message to be sent to the other end of the congestion. Also, if the vehicle is at the border of the traffic jam and receives a message from the other end, the *pTJam* process calculates the distance between the originator of the message and itself to detect the size and position of the traffic jam.

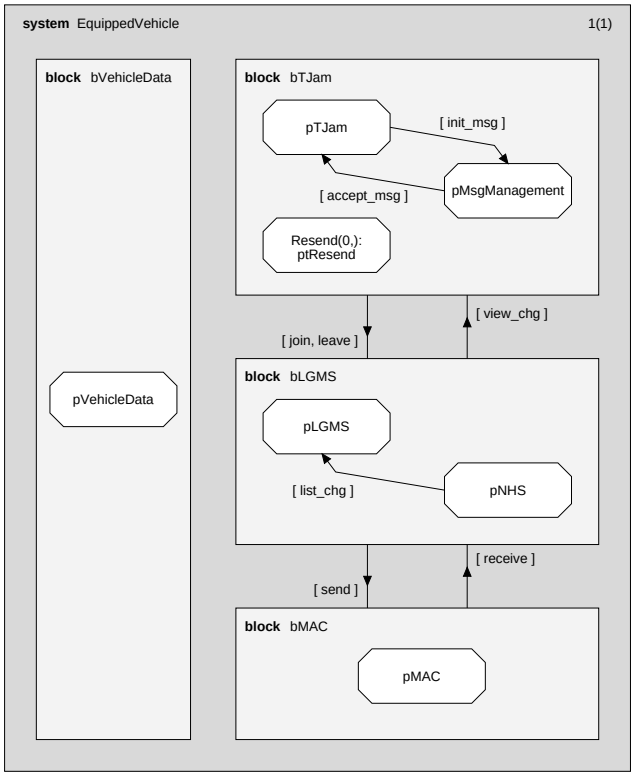


Figure 4.2: Overview on system architecture

The process *pMsgManagement* realizes the scoped flooding to send messages from one end to the other in a congestion. Therefore, the *pMsgManagement* process offers a service to initiate such a message. When receiving a message, the *pMsgManagement* process forwards it and delivers it to the upper layer, if the message is unknown. Thus, the *pMsgManagement* process must keep track of previously seen messages. We limit the propagation of a message to a certain number of hops that a message can take and to a given life time. If the message has been already received earlier by the system and is not new or if the life time has been expired, we discard it.

Every time it forwards a message, the block *bTJam* creates a new instance of the process type *ptResend* which handles the resending after a waiting time. When the block has forwarded a message, the corresponding process instance of *ptResend* terminates.

The *bTJam* block relies on the current membership of other vehicles nearby that is maintained by the block *bLGMS*. The *bLGMS* block contains the two processes *pLGMS* and *pNHS*. The localized group membership service (*pLGMS*) process installs views on the membership of neighbors when the view changes. Non-members have always an empty view. Members are always included in their local view. When the *bTJam* block joins the group, the *pLGMS* process announces this to its current neighbors. These respond with positive or negative messages regarding their membership depending on their actual speed. After the local view has consolidated, the *pLGMS* process notifies the upper layers about the new view.

In order to decide on whether all neighbors have responded yet to a join request, the system maintains a list of current neighbors. Hence, the *bLGMS* block provides a neighborhood service (*pNHS*) process. The *pNHS* process sends the own identity of the system periodically like a heartbeat. It also collects the heartbeat messages from other vehicles. When the *pNHS* process does not receive the heartbeat of existing neighbors anymore, it removes them from its list. If the NHS detects a heartbeat from unknown neighbors, it adds them to its list. The *pNHS* process reports all changes in the neighborhood to the *pLGMS* process.

As an interface to the wireless channel, we implement a simple medium access control in block *bMAC* which contains one process *pMAC*. If a send request reaches the process *pMAC* while being busy with either sending or receiving another data packet, the packet requested to be sent is dropped. If multiple receptions of packets overlap in time at the *pMAC* process, all

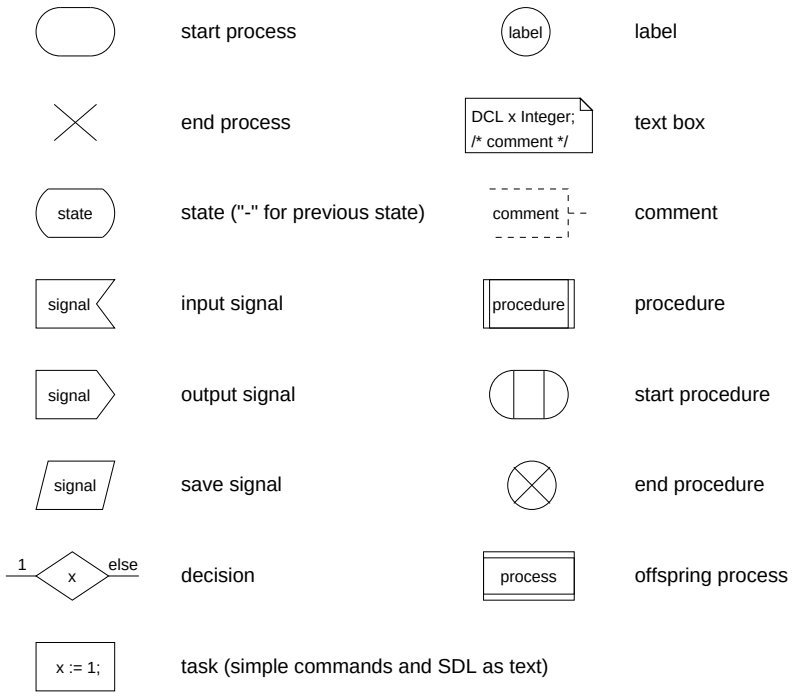


Figure 4.3: SDL symbols on process level

of them are discarded as collided packets. Otherwise, *pMAC* notifies the upper layers about a successfully received packet.

The block *bVehicleData* consists also of a single process, called *pVehicleData*. It maintains the current sensor data of the vehicle which consists of the geographical position and the velocity. The functionality of *pVehicleData* is simple and thus omitted in the following presentation of the system implementation. In the next sections, we describe the algorithms of each process in detail starting with the lowest layer. Refer to Figure 4.3 for an overview of SDL symbols on process level. Appendix C contains the complete SDL model.

4.2 Medium Access Control

pMAC The medium access control interfaces the communication system with the wireless medium. It offers two primitives to the upper layer—sending and receiving a packet. We model the wireless channel as another state machine that issues two consecutive events "PacketStart.ind" and "PacketEnd.ind" to each *pMAC* in vicinity of a transmitting process.

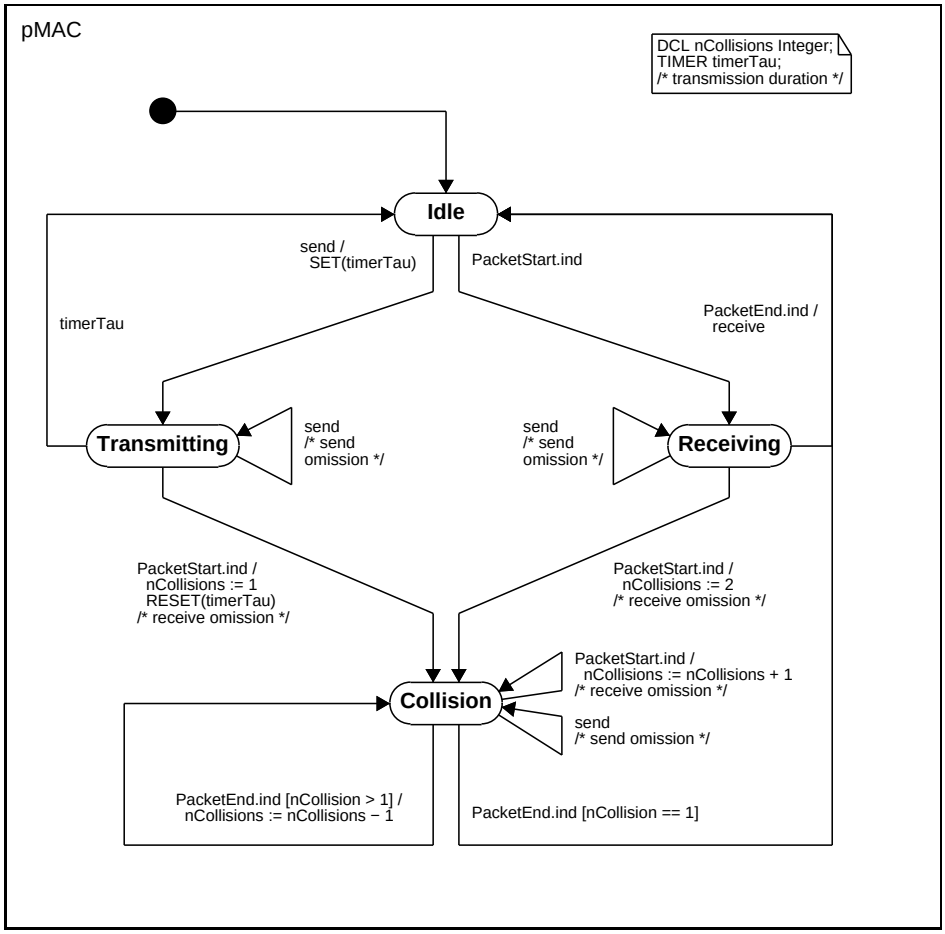


Figure 4.4: State chart of *pMAC* process

Thus, our implementation of the medium access control detects collisions and is responsible for not delivering collided packets to the upper layer. Refer to Figure 4.4 for the state chart of the process *pMAC*.

The process *pMAC* consists of four states "Idle," "Transmitting," "Receiving," and "Collision." The process starts in the "Idle" state. When the upper layer requests a packet to be sent, the process sets a timer to the transmission duration and switches into the "Transmitting" state. While transmitting, further send requests are omitted. When the previously set timer expires, the process switches back to the "Idle" state. If the wireless channel notifies the process of an incoming packet during the transmission, the process switches into the "Collision" state because radio communication prohibits receiving and sending at the same time.

If the wireless medium notifies an idle process *pMAC* about an incoming packet, the process enters the "Receiving" state. During the reception of the packet, the process drops send requests from the upper layer, again because simultaneously sending and receiving is impossible. When the medium finishes the transmission of the packet, the process *pMAC* delivers the packet to the upper layer and switches back into the "Idle" state. However, if the medium issues multiple notifications of incoming packets during the reception, the process changes to the "Collision" state and keeps track of the current number of overlapping packets. Every time the medium has finished sending, the process tests the counter of collisions. If the last collided packet is over, the process turns back to the "Idle" state.

4.3 Localized Group Membership Service

pNHS The neighborhood service employs a simple heartbeat mechanism to keep track of current neighbors. The local variables and the state machine for the *pNHS* process are depicted in Figure 4.5. The process *pNHS* contains only one state "Idle." Before this state is initially entered, the process sets the own timer with the heartbeat value. When its own timer expires, the process sends a heartbeat message. If the process receives a heartbeat from another equipped vehicle, it sets or resets the timer from an array of timers indexed by the sender's identity. The timer value is the heartbeat rate τ_{hb} multiplied with a factor n_{hb}. However, the larger n_{hb} is, the longer is the list of neighbors inaccurate when neighbors move away. In literature, the value of n_{hb} typically varies between one and three.

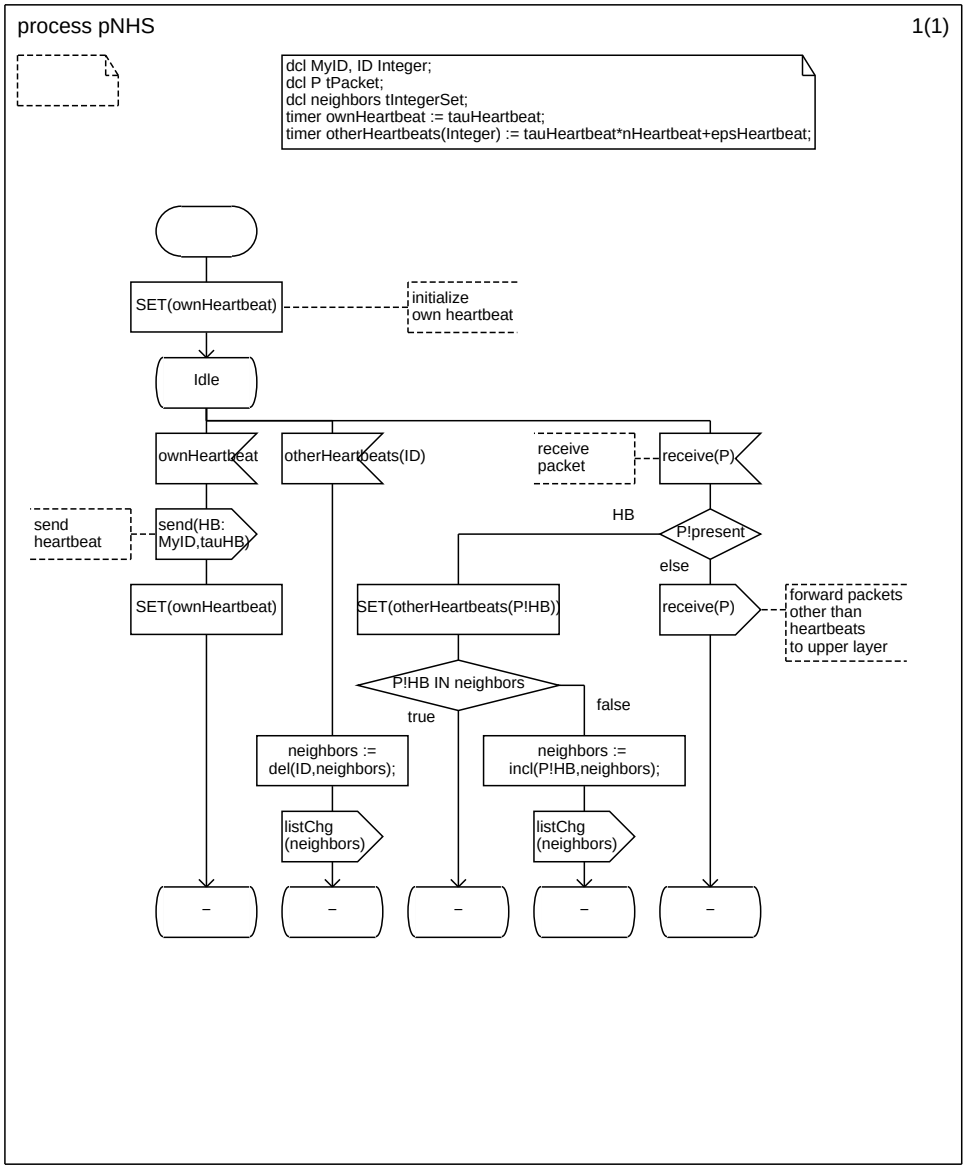


Figure 4.5: State machine in SDL of *pNHS* process

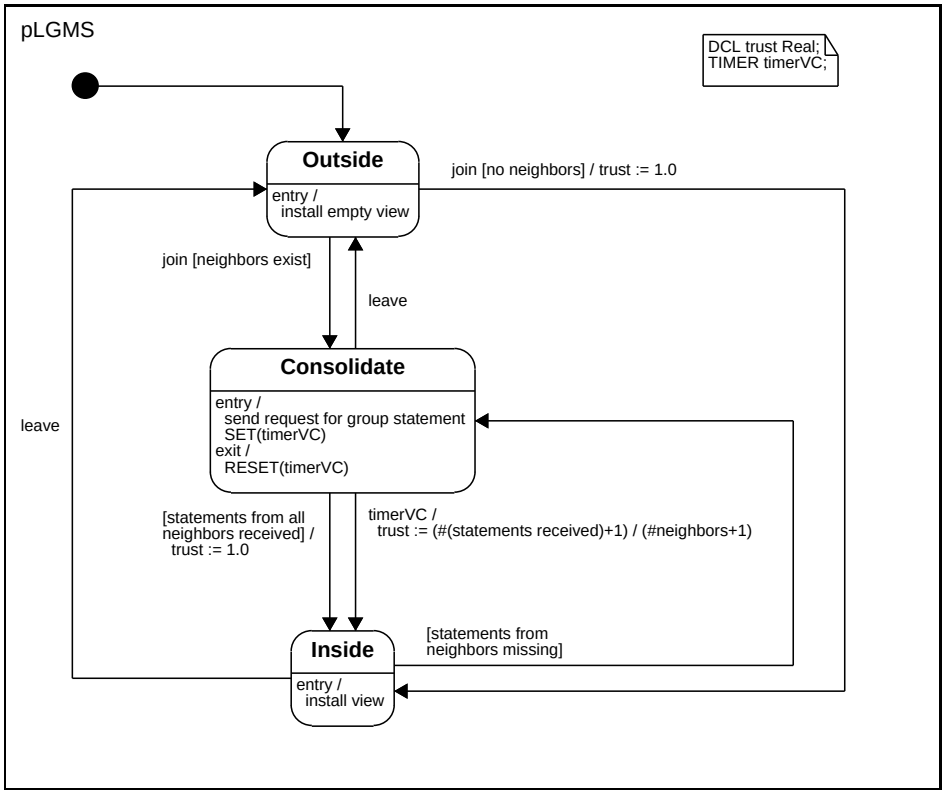


Figure 4.6: State chart of *pLGMS* process

Then, the process looks up the sender's identity in its list of neighbors. If the identity is already included the process returns to the "Idle" state. Otherwise, the process adds the identity of the sender to the list of current neighbors and signals to the upper layer that the list has changed. Once a timer from the timer array expires, the corresponding identity of the neighbor is removed from the list and the upper layer is notified about the changed list, before the process returns to the "Idle" state.

pLGMS The state machine for the localized group membership process contains three states "Outside," "Consolidate," and "Inside" as seen in Figure 4.6. Initially, the process is not a member and starts from the "Outside" state. When the upper block issues a signal to join the group, the process first switches to the "Consolidate" state. It also sends a message requesting all neighbors to respond with their membership status. While the process waits for the answers, it resides in the "Consolidate" state. As mentioned in the discussion on the specification of the group membership service in the

previous chapter, we limit the time to establish a view to τ_{vc}. Thus, the process sets a timer to τ_{vc} when it enters the "Consolidate" state. When either this timer expires or all current neighbors have responded, the LGMS installs a new view and switches into the "Inside" state. In the case that not all neighbors reacted on the join request within τ_{vc} time steps, the view is installed with a trust value < 1. The trust value is defined as the number of neighbors with a determined membership plus one divided by the size of the neighborhood plus one. We need to add one to both parts of the fraction to take the process itself into account. Especially with an empty neighborhood, the process immediately installs a view that consists only of the process itself with a trust value $= 1$. If the upper block wishes to leave the group, the process switches back to the "Outside" state and installs an empty view.

During the operation of *pLGMS*, the system reacts upon receiving a message from other vehicles requesting their current membership status. Depending on being outside the group or in either one of the "Inside" or "Consolidate" states, the process issues a negative or positive answer. We obey the possibility of multiple simultaneous receptions of the request message at other vehicles nearby. If all of those would respond immediately with their membership status, the answer messages would very likely collide at the originator of the request message. Thus, *pLGMS* delays the actual answer for a short time. This waiting time is randomly chosen according to a uniform distribution from the interval $t_{GWT} \in [0; t_{maxGWT}]$. To allow a view to consolidate after a request message has been sent, the maximum delay t_{maxGWT} should be shorter than the time limit to install a view τ_{vc}.

4.4 Detecting the Traffic Jam

pMsgManagement The state machine for the routing process consists of only one state "Idle" as seen in Figure 4.7. The process keeps track of the number of initiated messages which is initialized to the value zero. Also, the process maintains an array of known messages it has already seen. This array is indexed by the identity of the message's originator and the message's sequence number. The process preserves a copy of the current neighborhood to implement our approach towards routing messages in sparsely connected networks.

If the process receives a signal to originate a message, it increments the

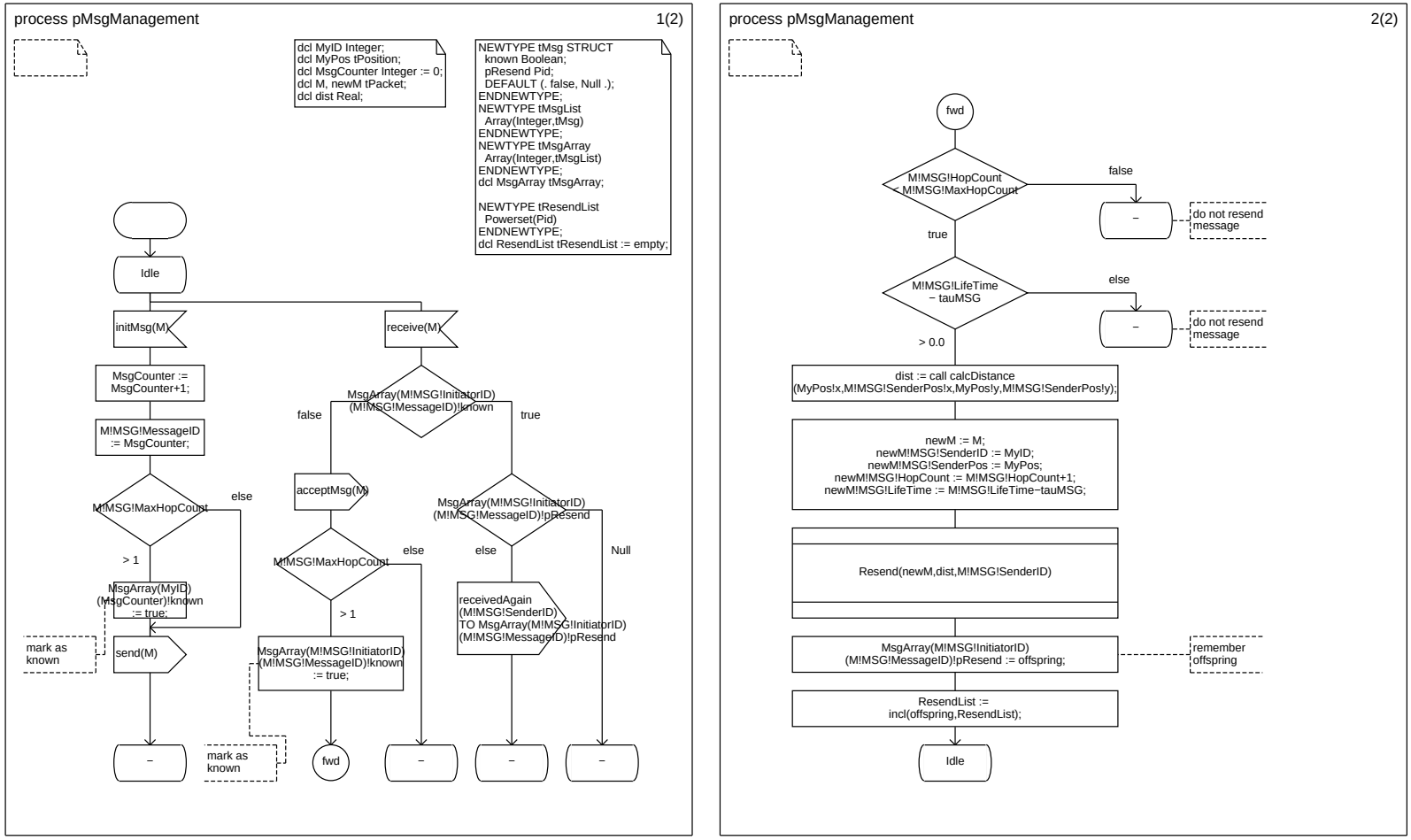


Figure 4.7: State machine of *pMsgManagement* process

counter for the sequence number by one and composes the message identity of this sequence number and the identity of the equipped vehicle. The process also marks the array of known messages at the tuple of own identity and the current message's sequence number if the maximum hop count is greater than one. Otherwise, the message will not be duplicated in the network and never return to the originator. Then, the process sends the message and returns to the "Idle" state.

In case it receives a message, the process first probes if the message is already known. If the array of known messages is marked at the tuple of originator and sequence number, we test whether an offspring process of type *ptResend* for this message is still pending. In this case, we update the resending process with the sender identity of the latest replica received. Then, we discard the message and return immediately to the "Idle" state. If the process received the message for the first time, we deliver the message to the traffic jam detection. Then, the tuple of originator and sequence number is used to mark the array of known messages. Now, the process compares the hop count of the message with the maximum hop count. If the actual hop count is smaller than the maximum, a process of the type *ptResend* is spawned that is responsible for forwarding the message. Otherwise, the process directly returns to the "Idle" state.

ptResend The process of type *ptResend* is started whenever a message must be forwarded. Multiple instances of *ptResend* exist concurrently if more than one message waits to be propagated. The state machine for *ptResend* is depicted in Figure 4.8. We distinguish between two states "WaitToResend" and "WaitForNeighbor." The parameters of the process include the message to be forwarded, the current distance from the sender of the message to determine the waiting time, and the sender's identity.

The process initializes a list of known senders with the identity of the message's sender that was passed as a parameter within the message. Then, we set a timer to the remaining life time of the message. The process computes the set difference of the current neighbors without the known senders to decide on the next step. If this set difference is not empty, the vehicle has neighbors other than the sender of the previously received message, and the process enters the "WaitToResend" state. A timer is set to a waiting time WT determined by the distance d of the vehicle to the sender of the message. The function for the time to wait WT yields a shorter value for

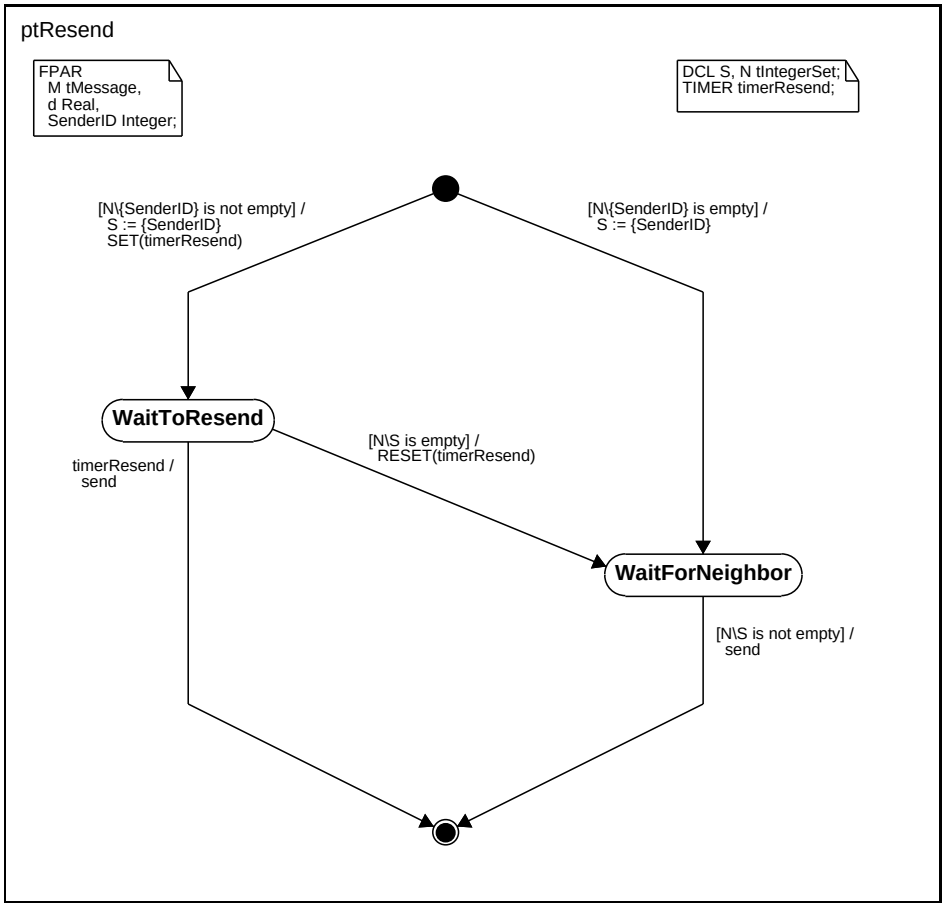


Figure 4.8: State machine of *ptResend* process type

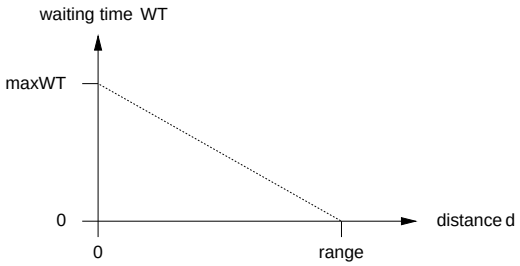


Figure 4.9: Function to determine waiting time depending on distance

more distant senders, such as given in equation 4.1 and plotted in Figure 4.9:

$$WT(d) = -\frac{t_{maxWT}}{r} \cdot \tilde{d} + t_{maxWT} \tag{4.1}$$

$$\tilde{d} = \min\{d, r\}$$

where d : distance to sender

t_{maxWT} : maximum waiting time

r : transmission range.

To understand our motivation to wait rather than to resend the message immediately, consider the broadcasting nature of radio waves. Multiple hosts can receive the same packet simultaneously. Then, an immediate resending would cause burst-like traffic on the channel. We try to avoid peak load by forcing the receivers to wait. Using the function WT, mainly hosts at the border of the reception area take part in forwarding the message quickly.

While the process awaits the moment to resend, it still updates the sets of neighbors and known senders. Then, it subtracts the set of known senders from the set of neighbors. If on any of these updates this set difference becomes empty, the process switches into the "WaitForNeighbor" state. Otherwise, it forwards the message after the timer expires and the calculated waiting time is over.

If at the creation time of the process the set difference of neighbors without known senders is already empty, then there are no new receivers nearby and the process enters the "WaitForNeighbor" state. In this mode, the process waits until an update of the set of neighbors occurs such that the set difference of neighbors without known senders is not empty anymore. Then, the system forwards the message.

During the operation of the process instance of *ptResend*, the timer can expire which was set to the remaining life time of the message. In this case, the process stops without sending the message. After forwarding the message, the process stops executing.

pTJam The state machine for the *pTJam* process is shown in Figures 4.11 and 4.12. The process contains two states "FreeFlow" and "InJam." Additionally, two boolean variables keep track of the vehicle's classification to be at the border of the jam—one for the beginning and one for the end of the traffic jam. The process starts executing in the "FreeFlow" state. We assume that the process gets constantly updates on the velocity of the vehicle. If the current velocity falls below a given threshold v_{jam}, the process issues a join request and switches into the "InJam" state. In the "InJam" state, the process continues monitoring the velocity. Once the vehicle becomes faster than a given threshold v_{free} again, the process leaves the group and switches back to the "FreeFlow" state.

The process *pTJam* continuously installs new views composed by the underlying localized group membership service. Outside the traffic jam, i.e. in the state "FreeFlow," a new view has no further effect and should be empty. In the "InJam" state, the vehicle installing a new view sets a timer to send out its own position if the timer is not already active. We determine the waiting time depending on the current speed of the vehicle. For higher velocity, the time until the position update will be sent is shorter. We use a simple linear descending function which has its maximum for stopped vehicles and its minimum of zero seconds waiting time, if the vehicle drives at least with velocity v_{free}, as given in equation 4.2 and plotted in Figure 4.13:

$$
\begin{aligned}
calcWT(v, t_{maxWT}) &= -\frac{t_{maxWT}}{v_{free}} \cdot \tilde{v} + t_{maxWT} \qquad (4.2)\\
\tilde{v} &= \min\{v, v_{free}\}\\
\text{where} \quad v &: \text{velocity}\\
t_{maxWT} &: \text{maximum waiting time}\\
v_{free} &: \text{upper threshold for velocity inside jam.}
\end{aligned}
$$

If a jammed vehicle installs a new view, it tests the size of the view. If the view contains only one member, the jammed vehicle classifies itself as being alone inside the jam and sets a corresponding timer to originate later

```
NEWTYPE tPosition STRUCT
  x,y Real;
  DEFAULT (. -1.0, 0.0 .);
ENDNEWTYPE;
SYNONYM invalidPos tPosition = (. -1.0, 0.0 .);
SYNONYM invalidSpeed Real = -1.0;

NEWTYPE tData CHOICE
  J`POS tPosition;
  J`ATBEGIN tPosition;
  J`ATEND tPosition;
  J`ALONE tPosition;
ENDNEWTYPE;
NEWTYPE tMessage STRUCT
  InitiatorID, MessageID, SenderID Integer;
  SenderPos tPosition;
  HopCount, MaxHopCount Integer;
  LifeTime Duration;
  data tData;
ENDNEWTYPE;

NEWTYPE tGroupState
  LITERALS inG, outG, reqG
ENDNEWTYPE;
SYNTYPE tGroupID = Real ENDSYNTYPE;
NEWTYPE tGroupMsg STRUCT
  ID Integer;
  gID tGroupID;
  gstate tGroupState;
ENDNEWTYPE;

NEWTYPE tPacket CHOICE
  MSG tMessage;
  GM tGroupMsg;
  HB Integer;
ENDNEWTYPE;

NEWTYPE tIntegerSet
  Powerset(Integer)
ENDNEWTYPE;

NEWTYPE tPosArray
  Array(Integer,tPosition)
ENDNEWTYPE;

NEWTYPE tJam STRUCT
  v tIntegerSet;
  vPos tPosArray;
ENDNEWTYPE;
```

Figure 4.10: SDL types for communication system

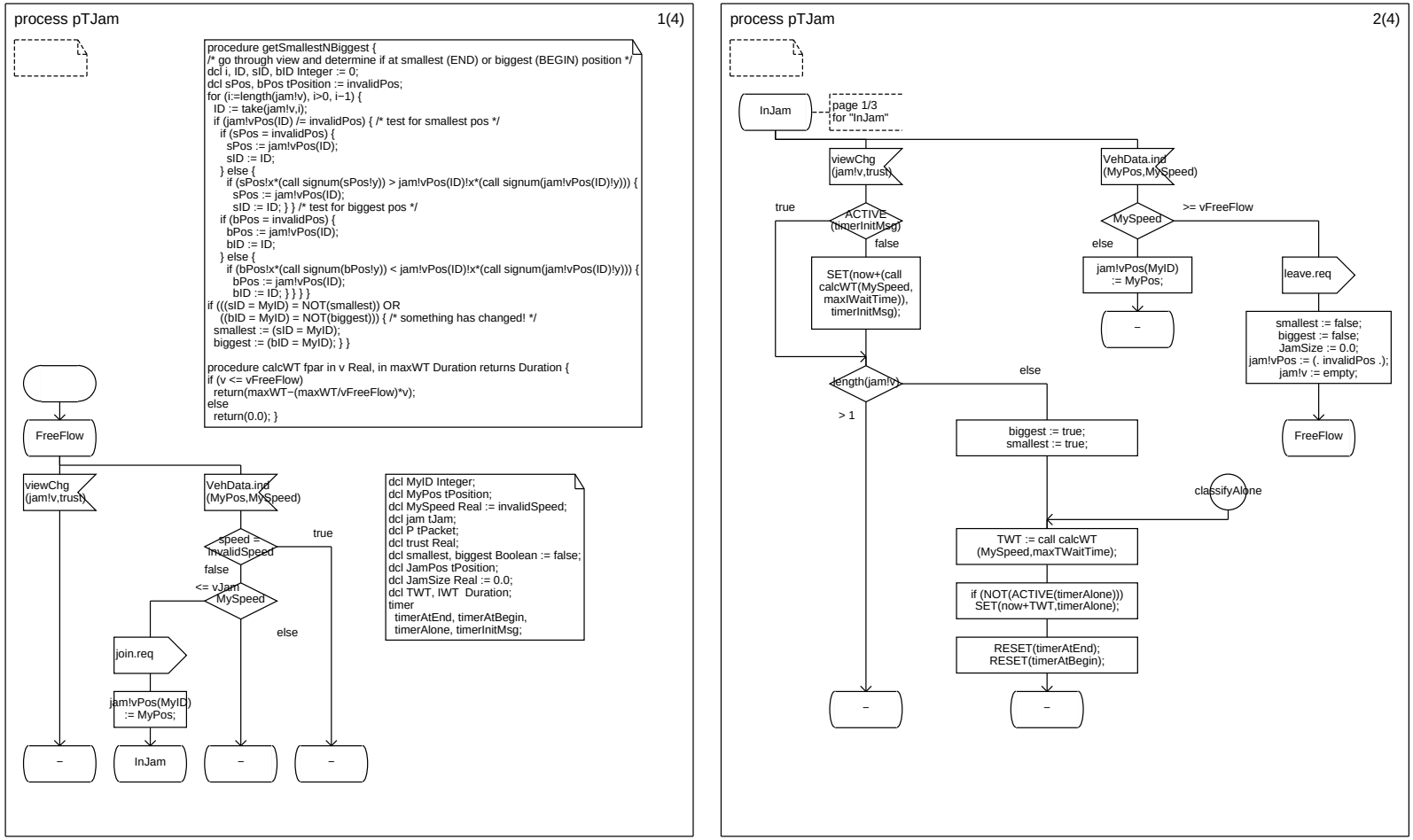


Figure 4.11: State machine of *pTJam* process (1 of 2)

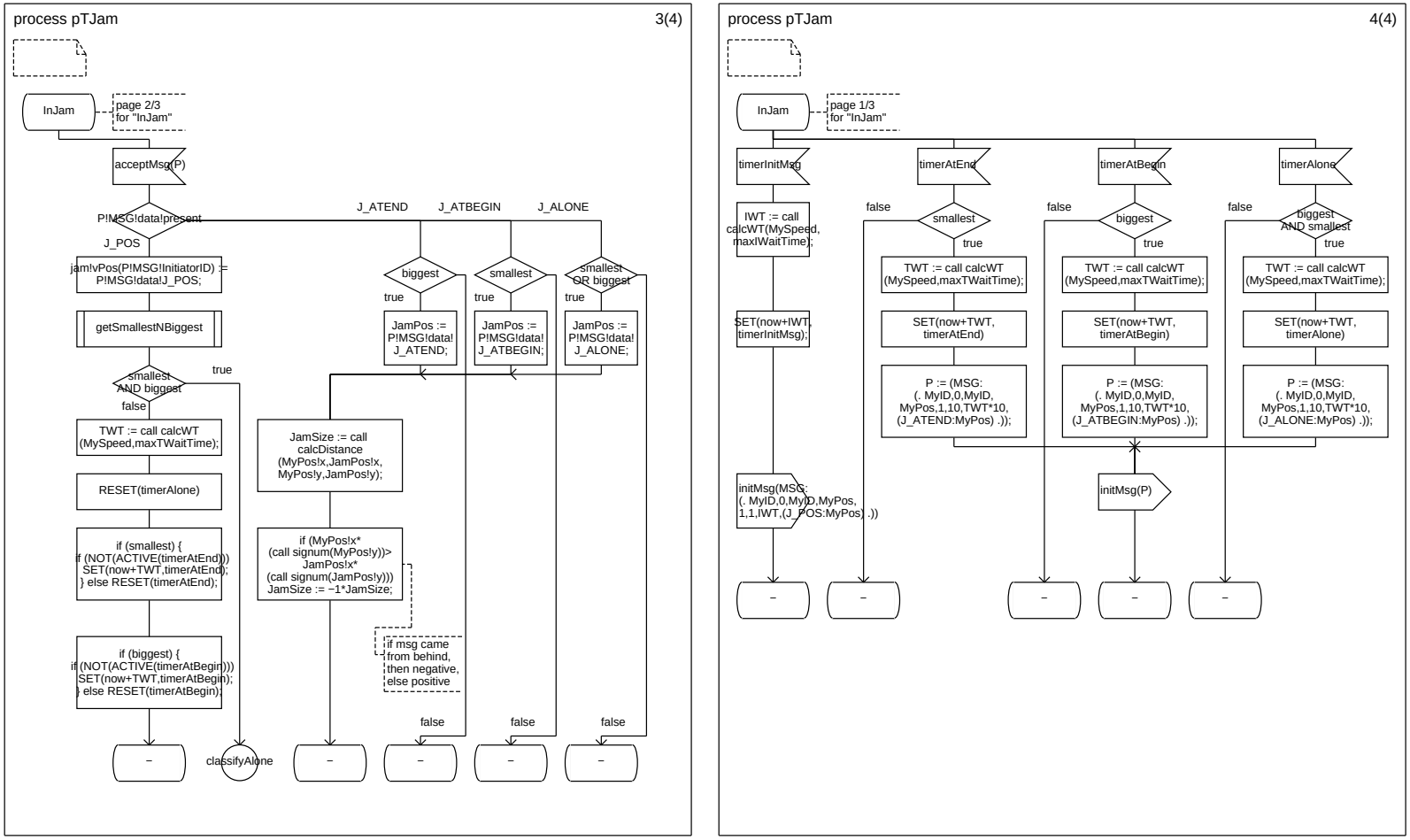


Figure 4.12: State machine of $pTJam$ process (2 of 2)

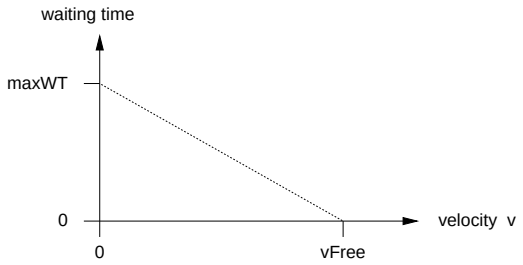


Figure 4.13: Function to determine waiting time depending on velocity

a message regarding the border of the jam. Again, if the timer is already active, the process skips this step.

Upon receiving a message from the lower layers, the process *pTJam* reacts according to the type of the message. For an overview on message types used in *pTJam* refer to Table 4.1. If the message contains a position update from another member (J_POS), the process saves the position and computes the current beginning and end of the jam based on the previously seen positions. If the vehicle itself is at the border of the traffic jam according to the collected positions, the process sets a timer to originate a message regarding the border of the traffic jam. The waiting time again is a function of the current velocity according to equation 4.2 and Figure 4.13.

In case the process *pTJam* receives such a message regarding the border of the traffic jam (J_ATEND, J_ATBEGIN, J_ALONE), the process determines from the settings of its variables whether it is a valid destination of the message—i.e. the vehicle is at the opposite end of the jam. Then, it accepts the message and calculates the current traffic jam size and position. We call this detecting the traffic jam.

Finally, if any of the timers expires while the vehicle is inside the "InJam" state, the process originates a message according to the settings of its variables and the current position. The corresponding timer is then set while the vehicle remains inside the traffic jam.

Type	Description	Max. hops	Rate	Max. time
J_POS	Position update inside jam	1	Periodically with waiting time depending on current velocity	10 s
J_ALONE	Border message if no jammed neighbors	10	Periodically with waiting time depending on current velocity	20 s
J_ATBEGIN	Border message if no jammed neighbors behind and at least one jammed neighbor in front	10	Periodically with waiting time depending on current velocity	20 s
J_ATEND	Border message if no jammed neighbors in front and at least one jammed neighbor behind	10	Periodically with waiting time depending on current velocity	20 s

Table 4.1: Messages initiated inside traffic jam

Chapter 5

Simulation of Proposed Protocols

In this chapter, we present the testbed for simulating the proposed protocols to detect traffic jams on highways. We apply the implemented inter-vehicle communication system in a realistic traffic scenario on a highway. Hence, the simulator consists of two parts. A microscopic traffic simulator generates a highway scenario in which a congestion in one driving direction occurs. Second, a simulation of the ad hoc network takes place on those vehicles that are equipped with the inter-vehicle communication system. In the next two sections, we describe the traffic simulator and the network simulator.

5.1 Microscopic Traffic Simulation

To analyze the proposed protocols, we simulate realistic highway traffic that is prone to jam formation. Our aim is to find a simple model that exhibits enough features such that a congestion is visible. Still, we need to keep the complexity of the traffic simulation at a low level in order to be able to focus on the behavior of the proposed communication protocols.

These protocols run as programs on single vehicles. Thus, the traffic model must be microscopic meaning that the vehicles are simulated as single entities. Opposed to this, macroscopic models consider not the dynamics of individual vehicles but the dynamics of vehicle density and average velocity. These quantities refer to a region of the road of sufficiently large spatial extent. Such macroscopic models can have better computational performance and are also suited to study the macroscopic nature of traffic jams.

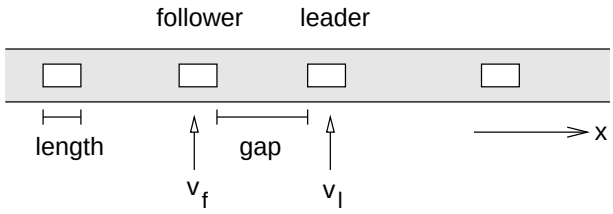


Figure 5.1: Krauß' model for update rules

However, our protocols react to certain values of vehicular data requiring a model of the individual vehicle. The challenge here lies in finding a simple description of a single vehicle's function that leads to a jam formation from a macroscopic perspective.

In this thesis, we utilize the microscopic traffic model described by Krauß [Kra97, Kra98, Jan98]. The model of a vehicle consists of four parameters and four rules for each time step. These parameters are the maximum velocity v_{max}, the maximum acceleration a, the maximum deceleration b and the amount of noise ε that introduces stochastic behavior to the model. The time is discrete and ticks with an interval $\Delta t = 1$ second. However, the spatial values of positions are continuous.

The rules of the model describe how the vehicle chooses a velocity and applies it to reach a new position in the next time step. The rules mirror three observations from human driving behavior.

(1) Drivers want to reach their goal as fast as possible.

(2) They do not want to collide with other vehicles.

(3) The human perception of speed and distance is inaccurate.

A noise term captures the latter and reduces the optimal velocity by a random value. The reaction time of the driver is set to $\tau = 1$ second. The update rules for the following vehicle are:

$$v_{safe} = v_l + \frac{gap - v_l \cdot \tau}{\tau_b + \tau} \quad \text{with } \tau_b = \frac{v_l + v_f}{2 \cdot b} \tag{5.1}$$

$$v_{desired} = \min[v_{max}, v_f + a \cdot \Delta t, v_{safe}] \tag{5.2}$$

$$v(t + \Delta t) = \max[0, v_{desired} - \varepsilon \cdot a \cdot \Delta t \cdot \text{random}()] \tag{5.3}$$

$$x(t + \Delta t) = x(t) + v(t + \Delta t) \cdot \Delta t, \tag{5.4}$$

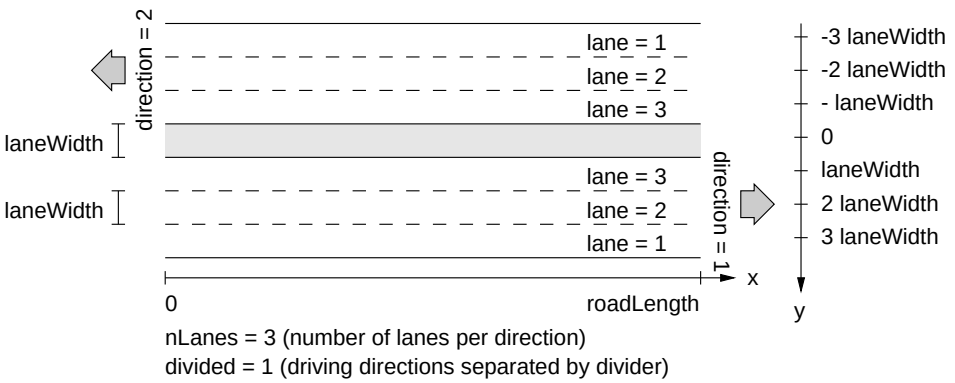


Figure 5.2: Road model for traffic simulation

where v_l is the velocity of the leader, v_f is the velocity of the follower and *gap* is the gap between the leader and the follower. Refer to Figure 5.1 for a schematic drawing.

The microscopic traffic model is applied to a highway scenario. Figure 5.2 gives an overview of the road model and the naming conventions that are used in the simulator program. The user can choose between an undivided and a divided highway model. In the latter case, extra space of the width of a lane is inserted between the driving directions. Other parameters of the road model are the total length, the width of a lane and the number of lanes per driving direction. We had to set the number of lanes per driving direction to one because the extension of the traffic model towards multilane traffic is not trivial. For more than one lane, the model requires lane changes to capture that traffic in a lane influences the traffic in a neighboring lane. We aim to keep the traffic model at a simple level because the main scope of this thesis focuses on ad hoc networks rather than traffic models.

For traffic simulations, the traffic density ρ describes the number of vehicles per kilometer and lane. The user can determine a density for each driving direction. We want to cause the traffic jam in direction one (that is where the x-positions of the vehicles increase over time.) Therefore, we set ρ to a high value in this direction and to a medium value for the oncoming driving direction.

The traffic scenario is initialized by spreading the vehicles on the lane according to ρ. With $\lambda = 1000 \cdot \frac{1}{\rho}$, we obtain the average distance between vehicles in meters. We start for each lane from the sink of the road. Then, we place vehicles with an offset of the average distance λ from the previous

vehicle on the lane until the source of the lane is reached. Starting the traffic simulation from this initialization, the chance that a traffic jam evolves is higher for higher density ρ. Still, we cannot control if, when and where the jam grows. Therefore, we disturb the initial setting by stopping the first $k = 5$ vehicles at the right hand side of the lane in direction one. For smaller $k < 5$, the traffic simulation did not cause a traffic jam in every run. The vehicles will not stop for long because the foremost vehicle quickly accelerates again having no predecessor in its lane. Nevertheless, the intrusion is enough to reliably cause a traffic jam for $k = 5$.

For evaluating our application in traffic jam detection, we need to define a traffic jam. In [Kra98, p. 38], we find the following definition:

> "A connected structure of vehicles, traveling at a velocity below a given threshold v_{thresh} will be called a jam, if this structure contains at least one stopped vehicle."

We adapt this idea and extend it towards a more realistic view. An important observation is that the own perception of drivers about being inside a traffic jam follows a hysteresis. A driver sees himself or herself being jammed if the speed falls below a certain threshold. From then on, the velocity must be significantly higher before the drivers feels that he or she is driving freely again. Another advantage of using a hysteresis is that the jammed state of a vehicle is not likely to switch back and forth if the velocity is varying around the threshold. We therefore use the following inductive definition of a traffic jam:

Definition 5.1 (Jammed Vehicles). *A stopped vehicle is jammed. A vehicle connected to a jammed vehicle and traveling at a velocity below a lower threshold v_{JIn}, is also jammed. A previously jammed vehicle only escapes the jam, if it is not connected to the preceding vehicle and if it travels at a velocity above an upper threshold v_{JOut}.*

A vehicle is called connected to its leader if it has to adjust its velocity because of the small gap between them. In terms of the model, the vehicle is connected if the minimum in rule 5.2 is v_{safe}. In compliance with [Kra98], we set the lower threshold to 50% of the maximum speed. The upper threshold for escaping the jam is set to 70% of the maximum speed.

Table 5.1 summarizes the parameters of the road and the traffic model. The values for the microscopic traffic model are derived from [Kra98]. We provide a conversion to the units of km/h and mph for velocity and acceleration.

Name	Description	Value
`roadLength`	Length of simulated road	10 km
`divided`	Type of highway	1 (“yes”)
`laneWidth`	Width of lanes	4 m
`nLanes`	Number of lanes per direction	1
`rho`	Traffic density per lane	15 vehs/km (direction one) 5 vehs/km (direction two)
`length`	Space occupied by vehicle	7.5 m
`vMax`	Maximum velocity	36 m/s ≈ 130 km/h ≈ 80 mph
`a`	Maximum acceleration	1.5 m/s^2 ≈ 6.1 km/h per s ≈ 3.8 mph per s
`b`	Maximum deceleration	4.5 m/s^2 ≈ 16 km/h per s ≈ 10 mph per s
`eps`	Influence of noise	1.5
`kStopped`	Number of stopped vehicles per lane to cause jam	5
`vJIn`	Lower threshold of velocity to become jammed	0.5 · `vMax`
`vJOut`	Upper threshold of velocity to escape jam	0.7 · `vMax`

Table 5.1: Parameters of traffic model

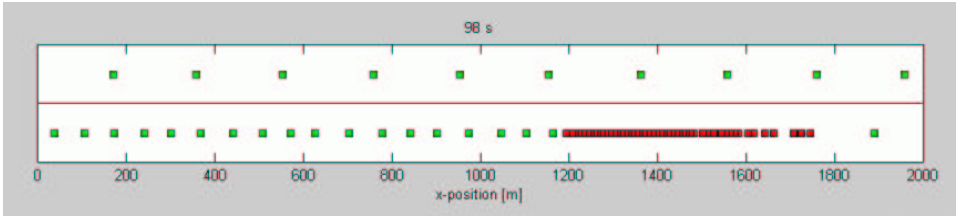


Figure 5.3: Screen shot of animated traffic simulation

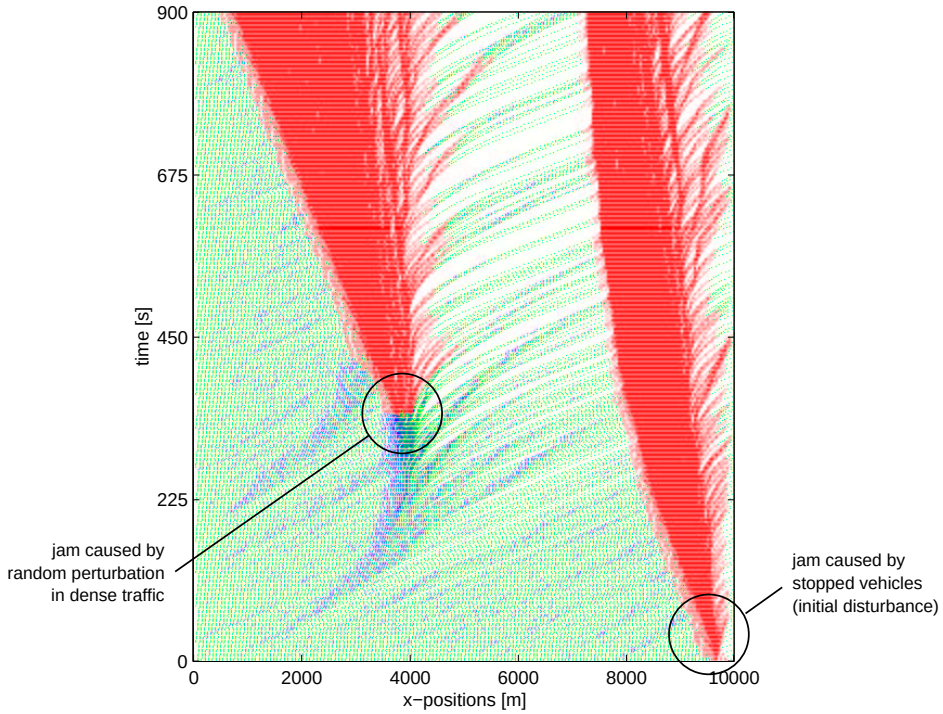


Figure 5.4: Example of space-time plot of traffic jam

Refer to Appendix B for our implementation of the microscopic traffic jam simulator. Figure 5.3 displays a screen shot of the traffic simulator running in animated mode. An animated sequence in AVI-format is included in the electronic version of this thesis. Please double-click on the small paper clip symbol next to Figure 5.3 to open the file. The traffic jam simulator outputs trace files with the position of vehicles during time. Each line of the trace file contains the time stamp, the vehicle identity and the position in Cartesian coordinates in the road model.

Figure 5.4 shows a space-time plot for a traffic jam simulation. Every line in the space-time plot corresponds to the position of vehicles at a time step. The colored dots in a line denote the longitudinal position of individual vehicles on the road. Green dots are vehicles in the state of free flow meaning that they need not adjust their speed because of the leading vehicle. Vehicles connected to the preceding vehicle that are not jammed, are colored blue. Red dots symbolize those vehicles that are jammed according to our definition. The simulation captures a duration of fifteen minutes.

5.2 Ad Hoc Network Simulation

The ad hoc network simulation reproduces the behavior and the interaction of multiple communication devices. On each equipped vehicle, exactly one communication system resides. The traffic simulation described in the previous Section 5.1 determines the location of vehicles and therefore of the communication units. The communication devices react upon stimuli form the outside world (change of velocity, reception of messages) and produce stimuli for other devices (send messages, alert driver). Thus, we use an event-driven simulation with discrete time.

In an event-driven simulation, the events that occur during the simulation are marked with a time stamp of the moment they happen. Then, all events are sorted in an event queue. The simulator processes the foremost events in the queue with the same, current time stamp. If the queue reaches an event with a greater time stamp, the simulator jumps with the simulated time to this next event and continues then to process events from the queue. Using this technique, the simulator skips the time in between events when nothing relevant happens.

To integrate the microscopic traffic jam simulation with the event-driven ad hoc network simulation, we generate a periodic event to update the positions of the vehicles. The traffic model computes new positions every second. Thus, the traffic simulation event happens once a second and the network simulator reads then the new positions from the trace file.

The simulator consists of the multiple modules for the equipped vehicles, a module controlling the traffic scenario, and a channel module that organizes the exchange of messages. Figure 5.5 provides an overview of the integrated simulator.

The scenario module controls the simulation run. It processes the trace file generated by the traffic simulator and updates the positions of the vehicles periodically. When the end of the trace file is reached, the scenario notifies all remaining vehicles and causes the simulator to shutdown properly.

The channel model assumes perfect reception of messages in transmission range and always drops packets outside the circular transmission area. Also, we model reception failures at stations that either send themselves or receive a message from another sender simultaneously. In the latter case when two or more transmissions overlap in time at the receiver, all of the messages are lost.

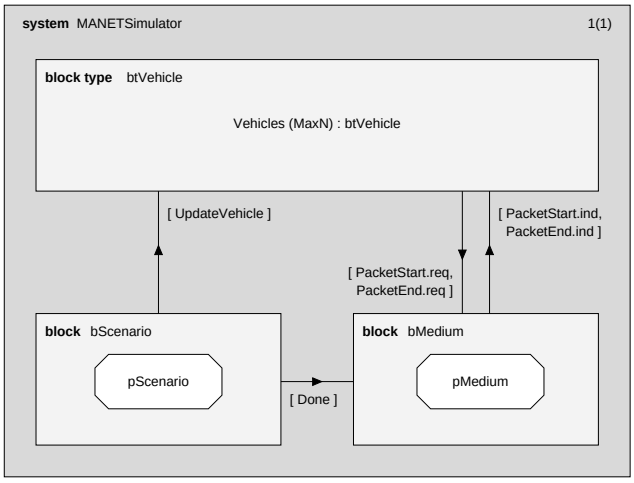


Figure 5.5: Overview of integrated simulator in SDL

Name	Description	Value
`transmissionRange`	r	600 m
`tauMSG`	Transmission duration of traffic jam message	5 ms
`tauGM`	Transmission duration of group message	1 ms
`tauMSG`	Transmission duration of heartbeat	0.5 ms
`maxHops`		20
`maxWT`	t_{maxWT}	40 ms
`tauHeartbeat`	τ_{hb}	1 s
`nHeartbeat`	n_{hb}	1
`epsHeartbeat`	Added to heartbeat timer to allow event of refreshing heartbeat happening before timer expires	0.0001 s
`maxGWT`	Maximum waiting time for answering group request	0.5 s
`tauViewChange`	τ_{vc}	3 s
`maxIWT`	Maximum time for position updates inside traffic jam	10 s
`maxTWT`	Maximum time for messages regarding border of traffic jam	20 s
`vFreeFlow`	v_{free}	70 km/h
`vJam`	v_{jam}	40 km/h

Table 5.2: Parameters of network simulation

Chapter 6

Evaluation of Proposed Protocols

In this chapter, we evaluate the proposed communication protocols for highly mobile ad hoc networks to detect traffic jams on highways. In the first section, we describe and define the metrics used to measure the performance of the protocols. In the second section, we discuss the results for the simulations runs that we carried out.

6.1 Metrics

We use different metrics for the different functions and layers of the implemented communication system. Before we explain the metrics for each layer, we introduce some concepts that apply to the entire remaining part.

For each simulation run, the set E contains all identities of equipped vehicles throughout the simulation run.

$$E \subset \mathbb{N} := \{\text{identities of all equipped vehicles in simulation run}\}$$

A simulation run is limited in time. Thus, we denote the time of a simulation run with a closed interval $T = [1; \max(T)]$ of values called clock ticks.

An active vehicle is a driving, equipped vehicle on the simulated road. The function A yields the set of currently active vehicles per clock tick.

$$\begin{aligned} A &: T \to 2^{\mathbb{N}} \\ &t \mapsto A(t) := \{n | n \in E \wedge n \text{ active at time } t\} \end{aligned}$$

We calculate mean values of the observed measures throughout a simulation run in order to characterize the whole simulation run and to compare this run with other runs. In these cases, we apply the method of independent replications analysis [Gol92, Cha93] to determine the statistical quality of the data we obtained from simulation. We stopped every simulation run when the time reached a certain limit. Thus, these runs fall into the category of terminating simulations. For each run, the pseudo random number generator starts with new initial seeds and hence produces a replication independent from other runs. For if the number of independent replications is large enough, a central limit theorem allows us to assume that the replicate outputs are approximately i.i.d.[1] normal. We then compute the half-width of the $100(1-\alpha)\%$ confidence interval as stated in (6.1) where $t(d,p)$ represents the p quantile of Student's t distribution with d degrees of freedom. We set the confidence level to 95% corresponding to $\alpha = 0.05$.

$$M_p \pm t(n-1, 1-\frac{\alpha}{2}) \cdot \sqrt{\frac{V_p}{n}} \tag{6.1}$$

where M_p : mean of data with parameter p

V_p : standard deviation of data with parameter p

n : number of independent replications

6.1.1 Packet Radio Communication

We implemented a simply strategy to organize the distributed access to the wireless medium. We aim to keep the number of parameters at a minimum level in order to focus on the higher level layers. Two types of malfunctions happen in our medium access control (MAC) scheme: (a) The request to send a packet from an upper layer arrives at the MAC layer during a transmission period, i.e. the MAC layer is busy with either sending or receiving. Then, the packet is dropped and a send omission occurs. (b) Multiple receptions overlap in time at a receiver. Then, all the packets are dropped at the receiver and are called collided.

We capture the above mentioned malfunctions of the MAC layer as the packet collision expressed by the function $\mathrm{pc}(p)$ per packet p requested to

[1] independently identically distributed

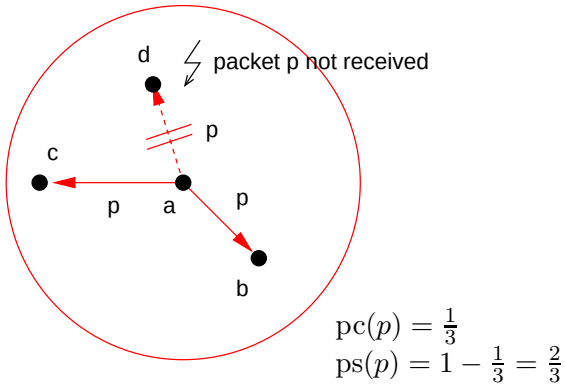


Figure 6.1: Example of packet collision and success

be sent during a simulation run.

$$\mathrm{pc}(p) := \begin{cases} 1 & \text{if send omission of } p \\ 0 & \text{if no send omission of } p \wedge \text{no receivers in range} \\ \frac{\#\text{ packets collided}}{\#\text{ receivers in range}} & \text{otherwise} \end{cases}$$

In contrast to the metrics used in literature, we allow other values than one and zero to characterize the sending of a packet. Due to the broadcasting nature of radio waves, a sender potentially reaches multiple destinations in a single sending procedure if more than one receiver exist in transmission range. Thus, we divide the number of receivers at whom the packet p collided by the number of potential receivers if such receivers exists and no send omission at the sender occurs. If a send omission happens, the packet already collides at the sender because the send request conflicts with an ongoing transmission. Hence, we set the packet collision to the maximum value of one. In case that no receivers are in range at the time of a sending procedure, no collisions take place neither at the sender nor at the (nonexistent) receivers corresponding to a packet collision value of zero.

We translate the values of the packet collision function ranging between zero and one into a metric that yields higher values if the quality of the measured MAC protocol is greater. Therefore, we define the metric packet success expressed by the function $\mathrm{ps}(p) := 1 - \mathrm{pc}(p)$ per packet p requested to be sent. Figure 6.1 depicts an example of packet collision and success. There, d is in range of the sending node a but d suffers from a receive omission. Hence, the packet collision ratio in this situation is $\frac{1}{3}$ and the packet success equals $1 - \frac{1}{3} = \frac{2}{3}$.

A simulation run is then characterized by the mean value of packet success which is the sum of all success values per packet divided by the total number of packets requested to be sent during the simulation run.

$$\text{ps} := \frac{\sum_p \text{ps}(p)}{\#\text{ packets requested to be sent}}$$

To further investigate our medium access scheme, we count the number of send requests that result in a collision at least at one receiver and the number of send omissions in a simulation run. We normalize both numbers by dividing each of them by the total of send requests that occur during the simulation run. We call these metrics collisions and send omissions per packets.

$$\text{collisions} := \frac{\#\text{ packets with collision at one or more receivers}}{\#\text{ send requests}}$$
$$\text{sendOmissions} := \frac{\#\text{ send omissions}}{\#\text{ send requests}}$$

6.1.2 Neighborhood Service

To evaluate the performance of the neighborhood service, we introduce the metric "accuracy of neighborhood." The accuracy of neighborhood is first defined per equipped vehicle. For each active vehicle, the two functions N and L denote the set of actual neighbors and the list of neighbors currently installed at this vehicle.

$$N : E \times T \to 2^{\mathbb{N}}$$
$$(e,t) \mapsto N(e,t) := \begin{cases} \emptyset & \text{if } e \notin A(t) \\ \{n | n \in A(t) \wedge \mathit{distance}(n,e) \leq r\} & \text{otherwise} \end{cases}$$
$$L : E \times T \to 2^{\mathbb{N}}$$
$$(e,t) \mapsto L(e,t) := \mathit{list}(e) \qquad \text{(as defined in 3.12 on page 63)}$$

Then, the accuracy of neighborhood per equipped vehicle is a function that maps every equipped vehicle and time to the fraction of the number of correctly classified neighbors versus the number of actual neighbors combined

with listed neighbors.

$$\text{aNHS} : E \times T \to \mathbb{R}$$
$$(e,t) \mapsto \text{aNHS}(e,t) := \begin{cases} 1 & \text{if } N(e,t) \cup L(e,t) = \emptyset \\ \frac{|N(e,t) \cap L(e,t)|}{|N(e,t) \cup L(e,t)|} & \text{otherwise} \end{cases}$$

The function $\text{aNHS}(e,t)$ is discrete with potential changes in values at position updates—this affects the set N—and at new installations of the list of neighbors—this affects the set L. For all other times, the value remains constant.

The accuracy of neighborhood for all equipped vehicles is defined as a function of time in the simulation run.

$$\text{aNHS} : T \to \mathbb{R}$$
$$t \mapsto \text{aNHS}(t) := \begin{cases} 1 & \text{if } A(t) = \emptyset \\ \frac{1}{|A(t)|} \cdot \sum_{i \in A(t)} \text{aNHS}(i,t) & \text{otherwise} \end{cases}$$

For each simulation run, we calculate the value aNHS as the integral over time of aNHS normalized by the length of simulation time.

$$\text{aNHS} := \frac{1}{\max(T)} \cdot \int_1^{\max(T)} \text{aNHS}(t)\, dt$$

Figure 6.2 shows an example of the accuracy of neighborhood in a simulation run that lasts 900 s with 5% equipped vehicles. The black curve is the function $\text{aNHS}(t)$. The normalized integral over the whole simulation time aNHS is plotted as a horizontal blue line and the value is plotted in blue, too.

6.1.3 Localized Group Membership Service

As discussed in Section 3.6.1, we force a member that is undecided on its view of the group to install a new view after a certain time limit is reached. Each member maintains information about the membership status of its adjacent neighbors. If the consolidation of this information takes too long, the member installs a view with those neighbors on which the member has already knowledge about their membership status. Thus, we define the $trust(m)$ of an installed view at member m by the following expression.

$$trust(m) := \frac{\#\text{ determined neighbors of } m+1}{\#\text{ neighbors of } m+1}$$

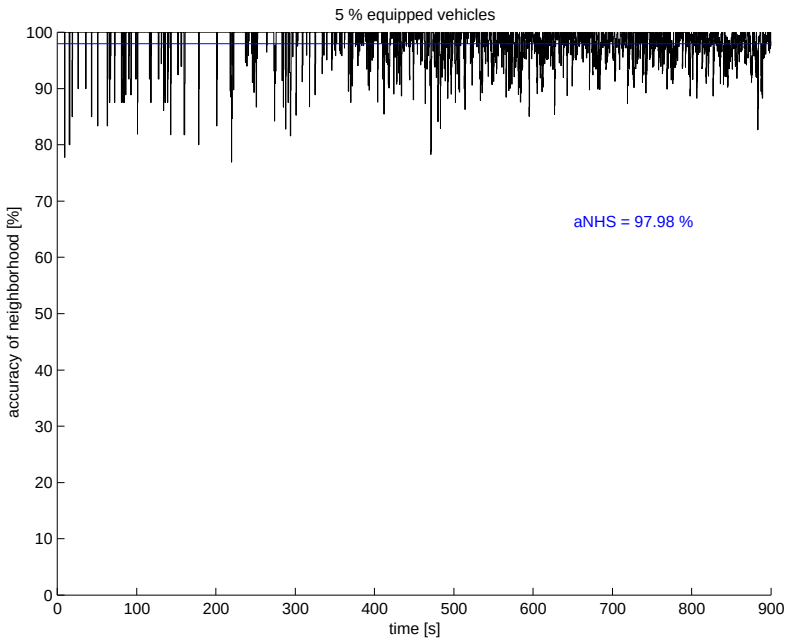


Figure 6.2: Example of accuracy of neighborhood

We add one to the divisor and the dividend because a view of a member always contains the member itself whereas the list of neighbors never contains the own identity. Hence, for an empty set of neighbors, a new member immediately installs a view that consists solely of the own identity with a trust value of 1.0.

For each simulation run, we measure the mean value of the trust values for all view installations during the run.

In addition to the trust value, we investigate the performance of the group membership service similarly to the neighborhood service by defining the metric "accuracy of views". The accuracy of views is first defined per equipped vehicle. For each active vehicle, the two functions M and V denote the set of actual members among the installed neighbors and the view on the local group membership currently installed at this vehicle.

$$M : E \times T \to 2^{\mathbb{N}}$$

$$(e,t) \mapsto M(e,t) := \begin{cases} \emptyset & \text{if } \neg member(e) \\ \{m | m \in L(e,t) \wedge member(m)\} & \text{otherwise} \end{cases}$$

$$V : E \times T \to 2^{\mathbb{N}}$$

$$(e,t) \mapsto V(e,t) := view(e) \qquad \text{(as defined in 3.14 on page 3.14)}$$

Then, the accuracy of views per equipped vehicle is a function that maps every equipped vehicle and time to the fraction of the number of correctly classified members versus the number of actual members among the neighbors combined with the members in the current view.

$$\text{aView} : E \times T \to \mathbb{R}$$
$$(e,t) \mapsto \text{aView}(e,t) := \begin{cases} 1 & \text{if } M(e,t) \cup V(e,t) = \emptyset \\ \frac{|M(e,t) \cap V(e,t)|}{|M(e,t) \cup V(e,t)|} & \text{otherwise} \end{cases}$$

The accuracy of views for all equipped vehicles is defined as a function of time in the simulation run.

$$\text{aView} : T \to \mathbb{R}$$
$$t \mapsto \text{aView}(t) := \begin{cases} 1 & \text{if } A(t) = \emptyset \\ \frac{1}{|A(t)|} \cdot \sum_{i \in A(t)} \text{aView}(i,t) & \text{otherwise} \end{cases}$$

For each simulation run, we calculate the value aView as the integral over time of aView normalized by the length of simulation time.

$$\text{aView} := \frac{1}{\max(T)} \cdot \int_1^{\max(T)} \text{aView}(t)\, dt$$

Figure 6.3 shows an example of the accuracy of neighborhood in a simulation run that lasts 900 s with 5% equipped vehicles. The black curve is the function aView(t). The normalized integral over the whole simulation time aView is plotted as a horizontal blue line and the value is plotted in blue, too.

6.1.4 Routing with SBR

We route a message regarding the border of the traffic jam to the opposite end using the proposed situation based routing (SBR) algorithm. In general, routing a message through a network can either reach the destination or fail to do so. However, in our context the destination of a message is a priori unknown and depends on the driving situation on the road. Moreover, the destination of a message is not fixed over time when the driving situation and the equipped vehicles at the border of the jam change. It is also possible that a message regarding the border of the jam is initiated at a moment when no valid destination exists. Especially for sparse deployment, an equipped

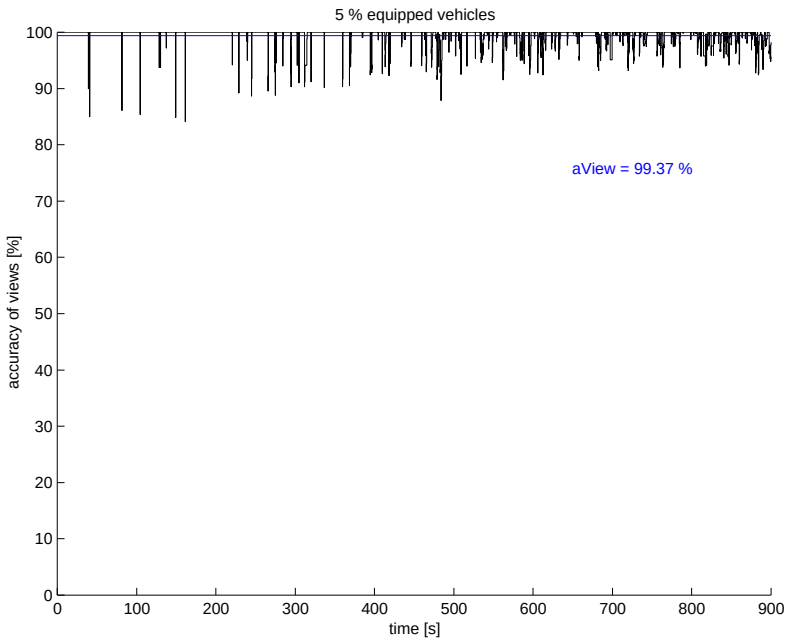


Figure 6.3: Example of accuracy of views

vehicle may be alone inside the jam but still sends out a message of type J_ALONE.

To evaluate this new type of routing task, we decide upon an equipped vehicle accepting the message whether the vehicle was indeed a destination of the message at that very moment from a global perspective. The message may later reach other equipped vehicles due to flooding, and may be still successfully accepted by them, if they are a valid destination—i.e. at the border of the traffic jam—then. We define the metric success of SBR as the following expression per message m originated.

$$\text{success}(m) := \frac{\#\text{ successfully accepted}}{\#\text{ accepted}} \qquad \text{, if message } m \text{ accepted}$$

For each simulation run, we compute the mean of the success values of all messages initiated.

$$M := \{m | m \text{ is initiated message}\}$$
$$\text{success} := \frac{\sum_{m \in M} \text{success}(m)}{|M|}$$

Next, we investigate the latency and the hop count of a successfully accepted SBR. The latency is the time difference of the time the message is successfully accepted subtracted by the time that the message was initiated. We also observe the number of hops that a successfully accepted message took since it was initiated. The message contains this number in one of its data fields. The hop count is not interchangeable with the latency of a successfully accepted message. Rather, the hop count denotes a minimum latency of the transmission duration multiplied with the number of hops. We expect the actual latency for hop counts > 1 to be longer than the minimum latency because intermediate hosts encounter different waiting times during the routing procedure.

Again, we use the mean values of all latencies and hop counts to characterize a simulation run.

$$\begin{aligned} M^* &:= \{m | m \text{ is successfully accepted message}\} \\ \text{latency} &:= \frac{\sum_{m \in M} \text{latency}(m)}{|M^*|} \\ \text{hopCount} &:= \frac{\sum_{m \in M} \text{hopCount}(m)}{|M^*|} \end{aligned}$$

Another important measure of routing strategies is the overhead of generated packets in the network. Here, we must distinguish packets sent and packets received because the numbers are different in case of point-to-multipoint communication. For this metric, we count the number of times the message was forwarded and we add one for the initial sending event. Then, we subtract the sum of all hop counts for every time the message was successfully accepted. The sum of hop counts denote the sending events which are necessary to reach the destination of the message. The metric overhead of an initiated message m is then

$$\text{overhead}(m) := 1 + \#\text{ forwardings} - \sum \text{hop count of successfully accepted.}$$

Note that in certain situations this metric can yield negative results. In the scenario depicted in Figure 6.4, the vehicle A in the middle of the traffic jam initiates a message because no neighbors of A are also jammed. Hence, A assumes to be at the border of the jam. The equipped vehicles B and C traveling in the opposite direction, receive the message from A and forward it

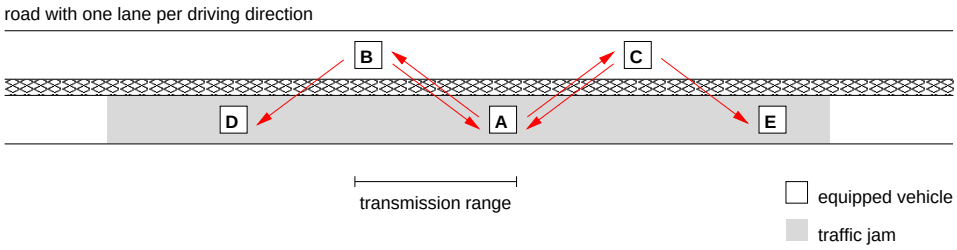


Figure 6.4: Example of negative overhead

to the vehicles D and E. These vehicles are also jammed and have no jammed neighbors in vicinity. Thus, D and E both accept the message successfully because they are indeed at the border of the traffic jam. Provided that D and E cause no more forwarding events of the message, the overhead equals

$$\text{overhead} = \underbrace{1+2}_{\substack{\text{sending of}\\ \text{A, B, and C}}} - \underbrace{(2+2)}_{\substack{\text{hop counts of}\\ \text{D and E}}} = -1.$$

Such extreme situations are rare because the flooding process tends to forward more messages than actually needed. Also, our definition of overhead does not contradict the motivation to measure the efficiency of a routing mechanism. A negative overhead symbolizes an efficient use of the broadcast capabilities of the wireless medium. To let the sum of hop counts exceed the number of sending events requires at least one sending procedure to reach more than one receiver on a valid path to a destination (as did A in our example.)

We calculate the mean of overhead values for all messages initiated during a simulation run.

$$\text{overhead} := \frac{\sum_{m \in M} \text{overhead}(m)}{|M|}$$

6.1.5 Traffic Jam Detection

The traffic jam at a jammed vehicle is detected when a message regarding the border of the traffic jam is received and the vehicle classifies itself as being at the opposite end of the jam. Then, the vehicle determines the jam size and position as the pair of distance to the originator of the message and its own position. We compare the detected traffic jam with the actual

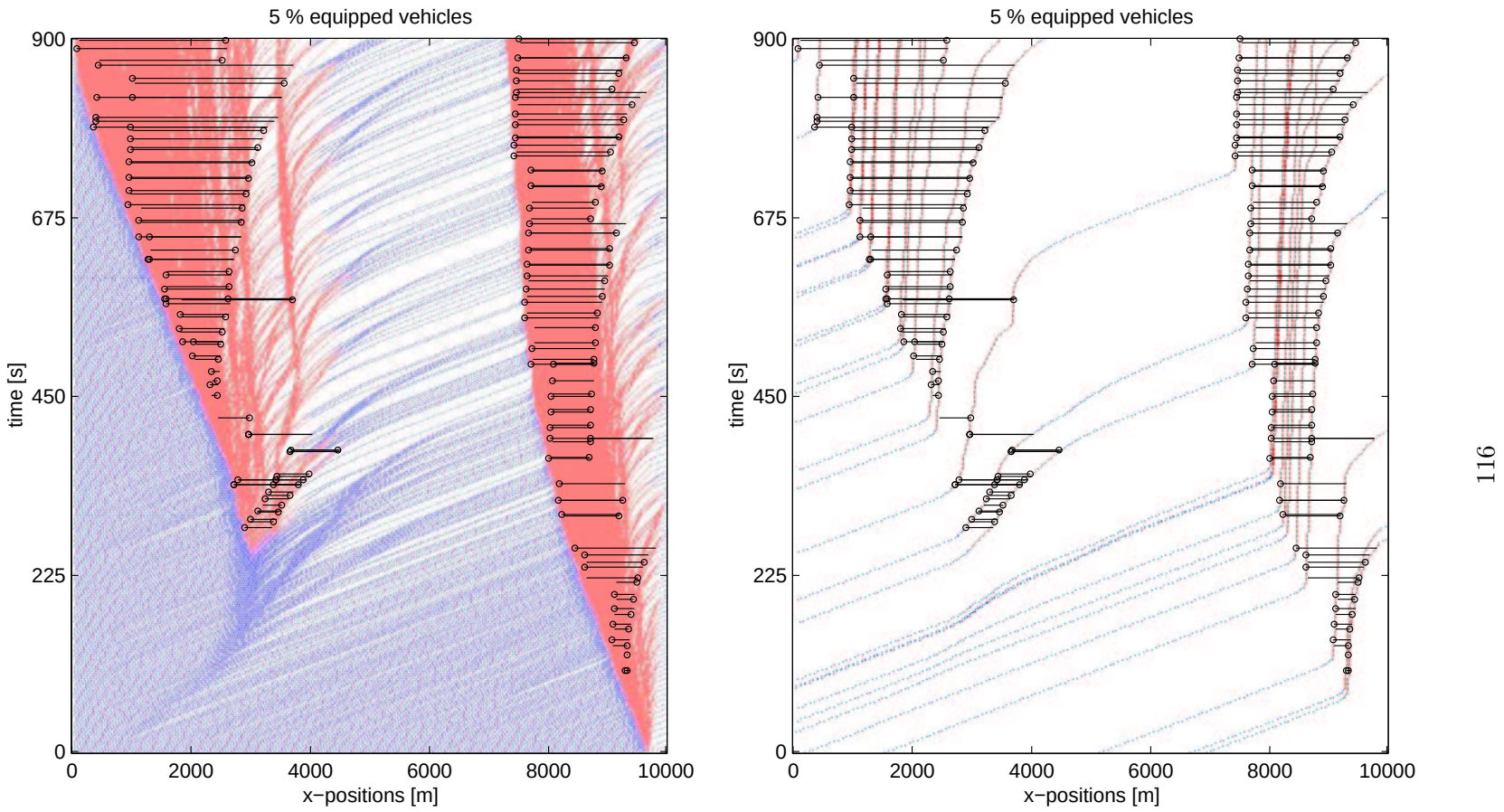
traffic jam on the road. Here, we distinguish between the situation with all vehicles on the road and the situation reduced to only the equipped vehicles. Refer to Figure 6.5 for an example of these two situations.

Each time an equipped vehicle detects a traffic jam, we plot a circle at its current position and a vertical line of the length corresponding to the detected size of the jam. Note that the vertical line points into the direction from where the jammed vehicle accepted the message. As a background, we used the space-time plot with all vehicles in Figure 6.5(a) and the space-time plot reduced to only the equipped vehicles in Figure 6.5(b). Depending on their state, the vehicles are colored blue while freely flowing, and red if they are caught in a traffic jam.

The example in Figure 6.5 shows that with only 5% equipped vehicles on the road, the traffic jam detections for an evolved jam works fairly well. Reduced to the situation with only equipped vehicles left, the two jams in this simulation run are detected almost perfectly by the outermost vehicles after 10 minutes, i.e. 600 seconds. However, we need to quantify and summarize the situation plotted in Figure 6.5 to compare it with other runs. Thus, we develop the metrics explained in the following.

The detected size and position of traffic jams is compared to the actual size and position of the jam. Therefore, we need to define a traffic jam first. We classify vehicles as jammed or freely flowing according to their state and velocity. If the vehicle is freely flowing and the velocity drops below a lower threshold, then the vehicle switches into the jammed state. If a jammed vehicle travels at a speed above an upper threshold, then the vehicle is called freely flowing again. Having all of them classified per time step, we arrange the vehicles driving into one direction on a scalar axis. Going from the sink to the source of driving vehicles, we group vehicles into one jam that are consecutively jammed. Some non-jammed vehicles are allowed to be inside a single jam structure, if the gap between them and the last jammed vehicle is smaller than a given threshold `macroView` of 500 meter. See the example in Figure 6.6 for an explanation of our traffic jam definition.

The situation at the time step printed in Figure 6.6 consists of two jams A and B although jam B has non-jammed vehicles in its structure. From a microscopic perspective, these non-jammed vehicles accelerate to escape the previous jam and are thus freely flowing. Their behavior is short-sighted due to the simple rules of the traffic model that take only the velocity of the predecessor into account. From a macroscopic view, the two vehicles in the



(a) all vehicles

(b) equipped vehicles

Figure 6.5: Example of traffic jam detection

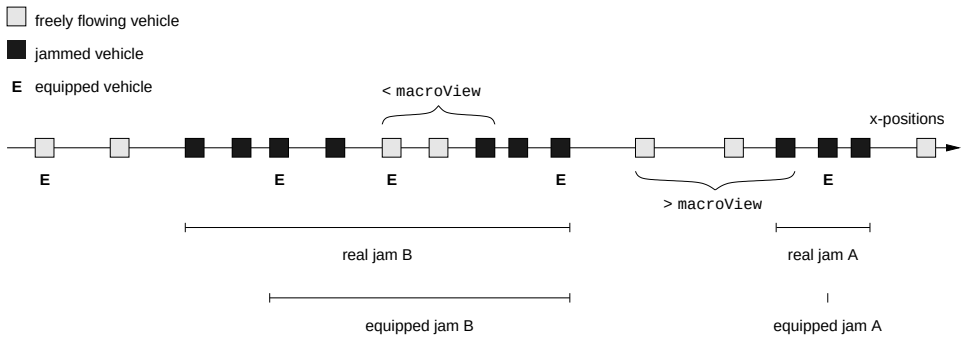


Figure 6.6: Example of traffic jam definition with `macroView`

middle of jam B will be caught in the next congestion after only a few time steps. Thus, we merge the last two chains of jammed vehicles into jam B.

We use the jam classification as described above of all vehicles in one time step to compute the jams reduced to equipped vehicles. Every jammed, equipped vehicle is mapped to a real jam. Then, the size and position of the equipped jam is determined by the outermost equipped vehicles in the real jams. Refer again to the example in Figure 6.6. Albeit the chain of jammed, equipped vehicles for jam B is interrupted by a non-jammed, equipped vehicle in the middle, all these vehicles belong to the real jam B and thus the outermost equipped vehicles define the equipped jam B. Inside the real jam A, there is only one equipped vehicle. It is therefore classified as the equipped jam A with a size of zero.

For every detection of a traffic jam, we match the receiving vehicle on one of the current real jams and equipped jams. Then, we compare the detected size and position with both types of jams—with the real jam that contains all vehicles, and with the equipped jam, which is reduced to the equipped vehicles. The absolute error eSizeAbs of the traffic jam size is defined by the following expression.

$$\text{eSizeAbs}_r := \begin{cases} \text{detected size} & \text{if not matched on jam} \\ \text{size of real jam} - \text{detected size} & \text{otherwise} \end{cases}$$

$$\text{eSizeAbs}_e := \begin{cases} \text{detected size} & \text{if not matched on jam} \\ \text{size of equipped jam} - \text{detected size} & \text{otherwise} \end{cases}$$

Note that the metric eSizeAbs can also yield negative values if the detected size is greater than the actual jam size.

The absolute error ePosAbs of the traffic jam position depends on the direction from where the jammed vehicle received the message. If the message came from behind, the jammed vehicle assumes to be at the beginning of the jam. Then, we calculate the difference of the current position from the beginning position of the traffic jam. In case that the vehicle detecting the traffic jam received the message from ahead, we compute the distance of the actual position and the end of the traffic jam.

$$\text{ePosAbs}_r := \begin{cases} 0 & \text{if vehicle not matched on real jam} \\ |x_{jamBegin} - x_{veh}| & \text{if message from behind} \\ |x_{jamEnd} - x_{veh}| & \text{if message from ahead} \end{cases}$$

$$\text{ePosAbs}_e := \begin{cases} 0 & \text{if vehicle not matched on equipped jam} \\ |x_{jamBegin} - x_{veh}| & \text{if message from behind} \\ |x_{jamEnd} - x_{veh}| & \text{if message from ahead} \end{cases}$$

The relative error eSizeRel of the traffic jam size is an additional metric to incorporate the current length of traffic jams into our observations. We divide the absolute error of the traffic jam size by the size of the actual jam if it is greater than zero. Otherwise, the result is undefined and dropped.

$$\text{eSizeRel}_r := \begin{cases} \perp & \text{if size of real jam} = 0 \\ \frac{\text{detected size}}{\text{size of real jam}} & \text{otherwise} \end{cases}$$

$$\text{eSizeRel}_e := \begin{cases} \perp & \text{if size of equipped jam} = 0 \\ \frac{\text{detected size}}{\text{size of equipped jam}} & \text{otherwise} \end{cases}$$

Finally, we count the number of times that an equipped vehicle detects a traffic jam during a simulation run. Then, the frequency of traffic jam detection is defined as the number of detections divided by the simulated time.

$$\text{frequency} := \frac{\text{\# traffic jam detections}}{\max(T)}$$

6.2 Results

The remaining part of this chapter presents the results for the metrics introduced above. As a free parameter, we varied the percentage of equipped

Equipped vehicles [%]	1	2	5	10	15	25	50	75	100
Simulated time [s]	900	900	900	300	300	120	100	50	25
Simulation runs	100	100	100	60	60	100	57	55	10

Table 6.1: Simulated time and number of runs per parameter

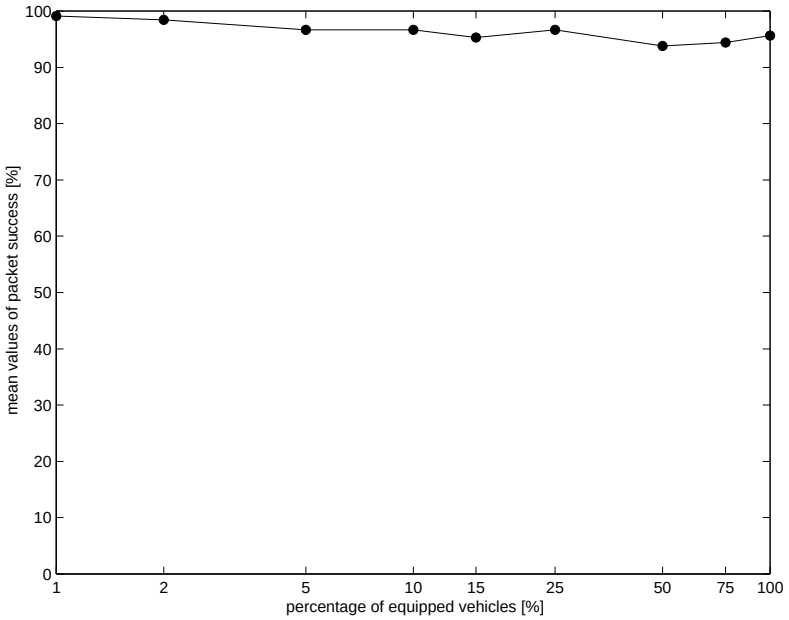
vehicles on the road—nine different settings were tested for 1, 2, 5, 10, 15, 25, 50, 75, and 100% of equipped vehicles on the road. For each parameter, we carried out up to 100 simulation runs. However, for more equipped vehicles driving on the road and actively participating in the networking, the simulator needed longer to complete simulation runs. For example, a single simulation run with 15% equipped vehicles on the road took about thirteen hours to simulate 900 s (15 minutes) of time. Thus, we adjusted the time to be simulated for the different parameter settings to allow more runs to be completed for each parameter set. Table 6.1 shows the simulated time for each parameter and the number of simulation runs that we carried out.

6.2.1 Packet Success, Collisions, and Send Omissions

The results for the metric of packet success are given in Figure 6.7. The diagram shows the mean values of packet success versus the percentage of equipped vehicles on the road. Note that we scaled the x-axis logarithmically to enhance visibility of the plotted results. The table contains the results along with confidence intervals for a confidence level of $\alpha = 5\%$.

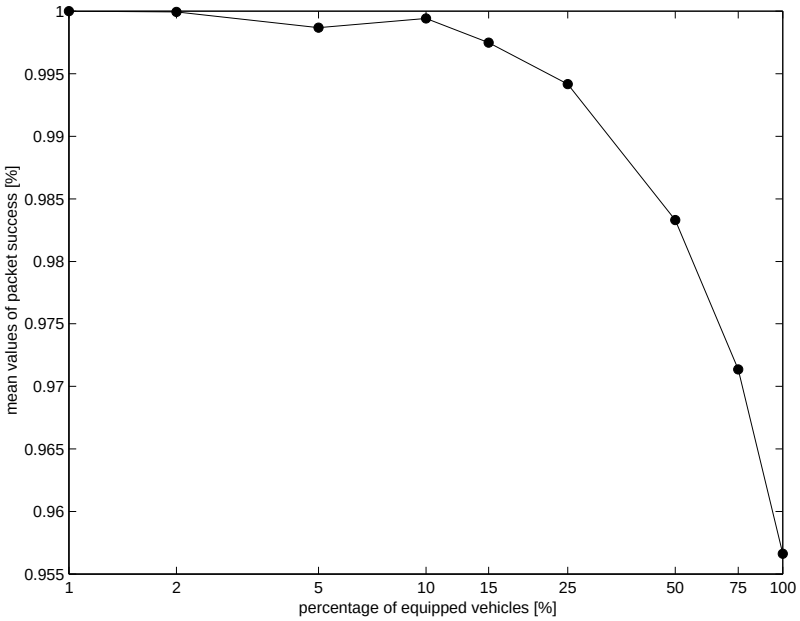
The curve is almost constant and slowly decays from a maximum of 99.1% for 1% of equipped vehicles on the road to 95.7% of packet success for all vehicles on the road being equipped. The global minimum of 93.8% is reached at 50% equipped vehicles. We obtained less data for higher percentages of equipped vehicles on the road due to shorter simulated time and due to a smaller number of simulation runs carried out. The size of the confidence intervals reflects this.

The mean packet success in a simulation run depends on the data packets requested to be sent and the density of the ad hoc network. Data packets are either heartbeats, group messages, or traffic jam messages. The first type of messages—heartbeats—are exchanged periodically and constantly, while other messages only occur in the presence of slow vehicles and slow neighbors. Hence, group messages and messages regarding the traffic jam are sporadic and occur more often if more vehicles are equipped on the road and more equipped vehicles drive inside a traffic jam. The simulated time



Percentage of equipped vehicles [%]	Packet success with confidence intervals for $\alpha = 5\%$ [%]
1	99.09 ± 0.13
2	98.40 ± 0.14
5	96.69 ± 0.45
10	96.69 ± 0.56
15	95.31 ± 0.69
25	96.65 ± 0.65
50	93.80 ± 0.34
75	94.40 ± 0.46
100	95.66 ± 0.54

Figure 6.7: Results for packet success



Percentage of equipped vehicles [%]	Packet success until $t = 25\ s$ with confidence intervals for $\alpha = 5\%$ [%]
1	100.00 ± 0.00
2	99.99 ± 0.01
5	99.87 ± 0.22
10	99.94 ± 0.04
15	99.75 ± 0.25
25	99.42 ± 0.38
50	98.33 ± 0.63
75	97.14 ± 0.67
100	95.66 ± 0.54

Figure 6.8: Results for packet success (normalized to 25 s)

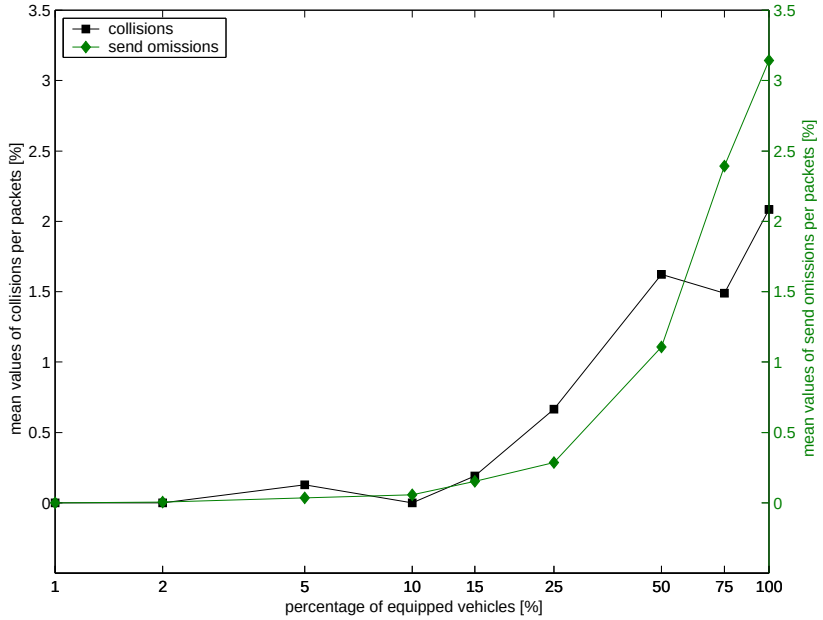
influences the size of the traffic jams and thus the metric of packet success. Therefore, we limit the simulated time to 25 s, which is the shortest time. We call this normalizing the metric to 25 s and provide the results in Figure 6.8. Note that we scaled the y-axis this time to the range of values.

For short simulation runs, the evolving jam is small and less vehicles perform group aggregation or send traffic jam messages. Hence, the remaining factor are the heartbeat messages and the density of the network. Both values increase for higher deployment of the system. The results show a linearly descending curve—note again that we scaled the x-axis logarithmically. The more vehicles are equipped on the road, the more receivers are in range but also the more heartbeats are issued which suffer from send omission or collisions. Hence, the packet success values degrade proportionally.

Figure 6.9 contains the results for collisions and send omissions in the medium access control. To compare the results with different parameters we again limited the simulation time for this investigation to 25 s which is the smallest simulation time carried out. The curves of collisions and send omissions start with near or equal zero values for small deployment of the system. The more vehicles are equipped, the more collisions and send omissions occur. While the collisions first supersede the send omissions, in the end, the send omissions grow stronger than the collisions for 75% and 100% equipped vehicles on the road. For all vehicles being equipped on the road, both curves reach their maximum of 2.1% collisions and 3.1% send omissions of all packets requested to be sent. However, the statistical quality shows that the size of confidence intervals is in the range of the mean values indicating that the data obtained from simulation runs is not sufficient.

To measure the packet success, the collisions and the send omissions gives us an idea about the reliability of the wireless medium which is accessed with our simplistic strategy. The results show that even in a densely populated scenario the packet loss is negligible considering the mean value of packet success being always well above 90 percent. Our metric punishes send omissions with a packet success of zero. However, send omissions pose a problem for high deployment of the system as the growing curve shows.

Extending the medium access control towards saving send requests that arrive during active transmissions or receptions would prevent send omissions and thus increase the performance of the system. As a drawback, the channel can suffer from additional traffic and hence congestion and packet collisions are more likely to occur. These collisions in turn can potentially cause packet



Percentage of equipped vehicles [%]	Collisions per packets with confidence intervals for $\alpha = 5\%$ [%]	send omissions per packets with confidence intervals for $\alpha = 5\%$ [%]
1	0.00 ± 0.00	0.00 ± 0.00
2	0.00 ± 0.00	0.01 ± 0.01
5	0.13 ± 0.26	0.04 ± 0.04
10	0.00 ± 0.00	0.06 ± 0.04
15	0.19 ± 0.28	0.15 ± 0.13
25	0.66 ± 0.61	0.29 ± 0.29
50	1.62 ± 0.76	1.11 ± 0.41
75	1.49 ± 0.66	2.39 ± 0.56
100	2.08 ± 0.38	3.14 ± 0.43

Figure 6.9: Results for collisions and send omissions (normalized to 25 s)

`nHeartbeat`	1	2	3
Accuracy of neighborhood	71.33 %	70.93 %	69.87 %

Table 6.2: Results of accuracy of neighborhood for different `nHeartbeat`

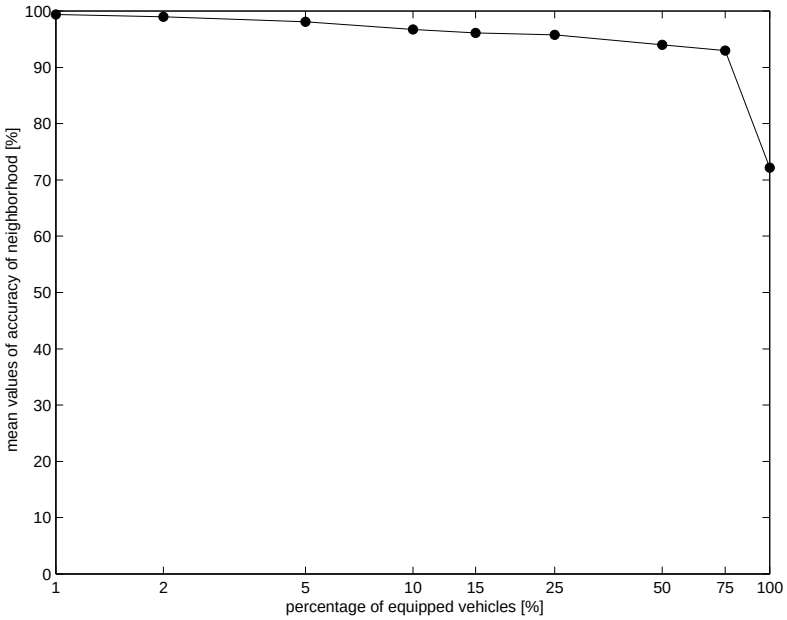
loss at receivers lowering the performance metrics. This buffering would also increase the complexity of the medium access control by introducing new parameters e.g. the buffer size.

6.2.2 Accuracy of Neighborhood

Figure 6.10 depicts the results of the accuracy of neighborhood. The mean values of the accuracy of neighborhood are always greater than 90% for 75% and less equipped vehicles on the road. A smaller percentage of equipped vehicles on the road yields better results for the neighborhood service. Having all vehicles being equipped on the road, the accuracy of neighborhood drops down significantly to only 72.2 percent.

This indicates clearly that heartbeats are lost either due to the system actively transmitting or receiving other messages or due to heartbeat messages colliding at receivers as a consequence of scarce bandwidth and the hidden station problem in multihop radio networks. The system we have implemented generates a heartbeat message exactly once every second. An equipped vehicle removes a neighbor from its list if it did not receive a new heartbeat after a period of exactly the heartbeat rate which is one second.

In [KK00], the authors propose to jitter a beacon transmission—i.e. a heartbeat in our terminology—around $[0.5B; 1.5B]$ where B is the beacon interval. There, a host would delete a neighbor from its list after not receiving a beacon within $4.5B$, three times the maximum beacon interval. We investigated one of the simulation runs with 100% deployment of the system and alternated the parameter `nHeartbeat` from one to three. This factor `nHeartbeat` multiplied with the heartbeat rate determines the time limit after which a neighbor is discarded if an equipped vehicle receives no further heartbeat. The other parameters and seeds for the random number generator for this simulation run remained fixed. Table 6.2 shows the results of accuracy of neighborhood as a function of the parameter `nHeartbeat`. Surprisingly, introducing such a laziness factor > 1 does not improve the accuracy of neighborhood. Moreover, the results slightly degrade from originally 71.3% to 69.9% for `nHeartbeat` $= 3$. We explain this with the additional inaccu-



Percentage of equipped vehicles [%]	aNHS with confidence intervals for $\alpha = 5\%$ [%]
1	99.42 ± 0.05
2	99.01 ± 0.10
5	98.06 ± 0.13
10	96.72 ± 0.33
15	96.09 ± 0.22
25	95.76 ± 0.17
50	93.99 ± 0.41
75	92.95 ± 0.25
100	72.16 ± 0.42

Figure 6.10: Results for accuracy of neighborhood

Percentage of equipped vehicles [%]	Trust values with confidence intervals for $\alpha = 5\%$ [%]
1	99.67 ± 0.23
2	99.69 ± 0.15
5	99.84 ± 0.03
10	99.90 ± 0.05
15	99.96 ± 0.01
25	99.93 ± 0.02
50	99.98 ± 0.01
75	99.97 ± 0.02
100	99.91 ± 0.08

Table 6.3: Results for trust of views

racy introduced when a neighbor moves out of sight and is not discarded until two or three times the heartbeat rate.

We attribute the poor results of accuracy of neighborhood for densely populated networks to the small bandwidth of the wireless radio channel. Already in 1970, Abramson [Abr70] (cited in [KL75]) showed that the capacity of ALOHA packet radio networks—i.e. with a random access scheme like ours—is as low as 18.4% for fixed packet size. The capacity of packet radio networks is the maximum throughput of packets as a function of offered traffic. Typically, the throughput in radio networks with random medium access schemes increases with higher load until the maximum is reached. Then, for greater network traffic offered, the throughput degrades again due to collisions. If the hosts in such a network state queue their generated packets to send them later—as done in carrier sense multiple access strategies, the load of the network constantly increases, more and more packets are lost and the network becomes unstable. Thus, a packet radio network with such a random medium access scheme should be operated at a small fraction of the theoretical capacity.

6.2.3 Trust and Accuracy of Views

We defined two metrics to evaluate the localized group membership service: the trust and the accuracy of installed views.

The results for the trust values of installed views are shown in Table 6.3. The mean values of the trust of installed views is always better than 99% with very small confidence intervals of 0.23% and less. This metric proves

that the time limit of 3 s we used to force a view installation, is almost never reached before the view at a member is consolidated.

Figure 6.11 contains the result for the accuracy of views. The accuracy of installed views at members is almost constant and always well above 90 percent. For all vehicles on the road being equipped, the accuracy of views has a global minimum of 94.6 percent.

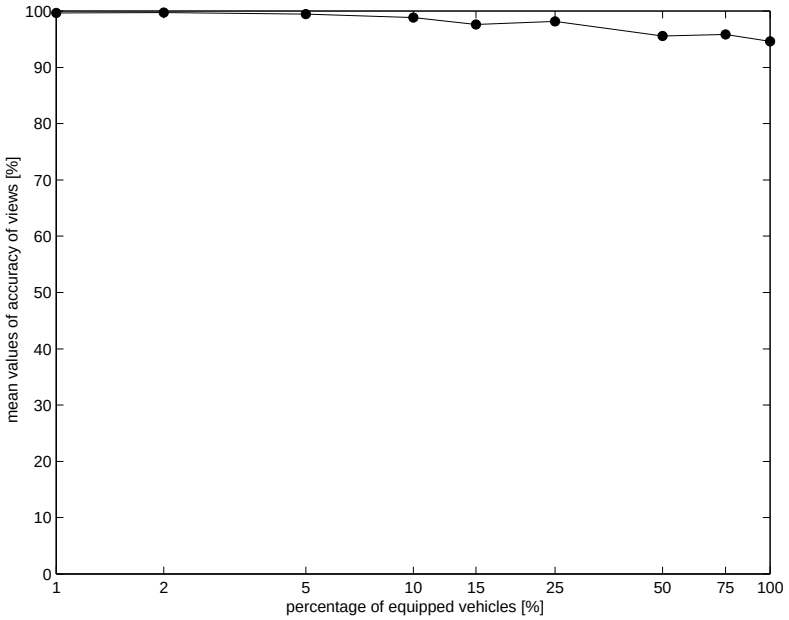
Again, we explain the lower values for more equipped vehicles on the road with more packets regarding the membership being lost. Note that in our metric we apply the accuracy of views to the list of neighbors kept at the vehicle and not to the actual neighborhood relation in the situation. Hence, the inaccurate neighborhood lists for 100% equipped vehicles on the road do not reduce the results for the accuracy of installed views.

6.2.4 Success and Overhead of SBR

Next, we evaluate the proposed routing algorithm SBR by applying the four metrics success, overhead, latency and hop count. The results of the success and the overhead of SBR are given in Figure 6.12. Figure 6.13 provides the results for the latency and the hop count of successfully accepted messages with SBR.

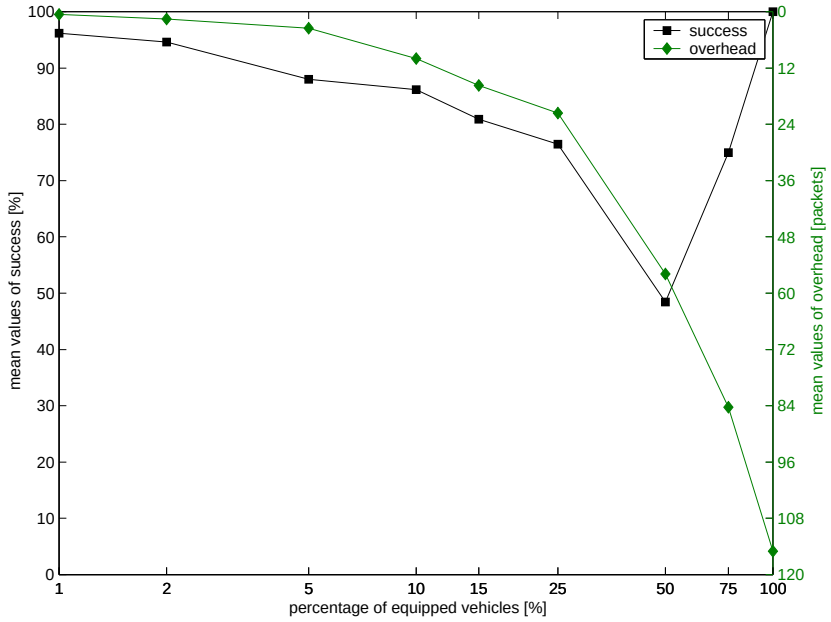
The diagram in Figure 6.12 shows the curves for the success of SBR (left y-axis) and the overhead in packets (right y-axis) as a function of the percentage of equipped vehicles on the road. Note that we turned the y-axis on the right hand side so that for both curves values at the top of the y-axis indicate better performance. We normalize the metric to a simulated time of 300 s or less. Thus, we are able to compare deployment rates of up to 15% equipped vehicles on the road. For higher deployment, we keep in mind that the simulated time is shorter.

The curve of success starts at a high value of 96.2% for 1% equipped vehicles on the road, and decreases slowly to 76.5% for 25% equipped vehicles on the road. For higher deployment, the success values suffer from the small amount of data we obtained from the shorter simulation runs expressed by large confidence intervals. We explain the decreasing success of SBR in the beginning of the curve with the dynamic of the traffic jam formation. The more vehicles are equipped on the road, the more often changes the vehicle at the border of the equipped traffic jam. Also, in spite of higher deployment, the outermost equipped vehicles move closer to the border of



Percentage of equipped vehicles [%]	aView with confidence intervals for $\alpha = 5\%$ [%]
1	99.65 ± 0.11
2	99.70 ± 0.04
5	99.47 ± 0.03
10	98.87 ± 0.22
15	97.64 ± 0.43
25	98.19 ± 0.13
50	95.60 ± 0.32
75	95.82 ± 0.23
100	94.64 ± 0.14

Figure 6.11: Results for accuracy of views



Percentage of equipped vehicles [%]	Success of SBR with confidence intervals for $\alpha = 5\%$ [%]	overhead of SBR with confidence intervals for $\alpha = 5\%$ [packets]
1	96.20 ± 3.86	0.60 ± 0.13
2	94.62 ± 2.43	1.51 ± 0.24
5	88.02 ± 5.51	3.48 ± 0.47
10	86.18 ± 3.14	9.95 ± 1.90
15	80.93 ± 5.48	15.75 ± 3.56
25	76.46 ± 4.96	21.64 ± 2.82
50	48.42 ± 13.51	55.89 ± 2.63
75	75.00 ± 50.78	84.31 ± 21.81
100	100.00 ± NaN[a]	115.00 ± NaN

[a] not a number

Figure 6.12: Results for success and overhead of SBR (normalized to 300 s)

the real traffic jam. There, we observe high fluctuation which causes the vehicles at the border to change faster than the equipped vehicles are able to exchange their positions and to decide on the classification of outermost vehicles.

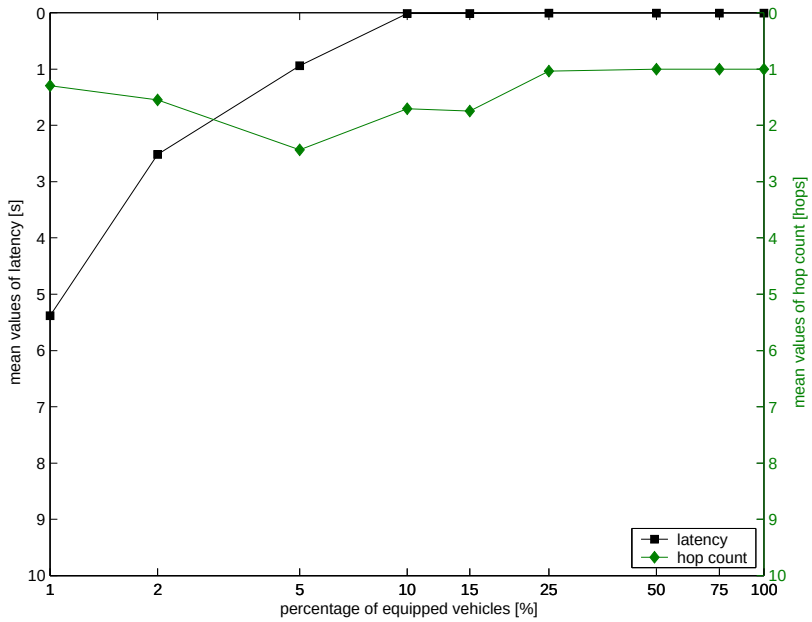
The overhead of SBR starts with small values like 0.6 packets for 1% equipped vehicles on the road. With more vehicles being equipped, the overhead grows linearly. Note that we scaled the x-axis logarithmically. This behavior is not surprising because the routing uses a flooding scheme. There, all receivers of a message forward the message until a maximum hop count or a maximum life time of the message is reached. We count the number of sending events which increase linearly with the number of hosts. For 50% and 100% equipped vehicles on the road, the amount of data is not sufficient to achieve good confidence of the results. Nevertheless, the maximum of 115 packets for all vehicles being equipped shows the trend of linear growth.

Compared to the almost constant success, the overhead indicates a problem for denser populated networks. Here, we recommend to drop packets that are scheduled to be forwarded, if the waiting host sees a certain number of replicas of this message. The recent advances in geographic routing like [KV99, KK00] propose to reduce flooding to certain areas and hosts that are closer to the destination than intermediate routers. A challenge then lies in the individual hosts estimating the current operational state of the network and when to switch to reduced flooding.

The diagram in Figure 6.13 displays the results of the latency and the hop counts of successfully accepted messages. Note that we turned both y-axes to let values at the top of the y-axis denoting better performance.

The latency of successfully routed packets is longer for sparsely populated scenarios and reaches the maximum of 5.4 s with a confidence interval of 2.8 s for 1% equipped vehicles on the road. For higher percentage of equipped vehicles on the road, the latency converges fast to 14 ms for 10% and 15% equipped vehicles, and to 5 ms for 25% and more equipped vehicles on the road. This equals the transmission duration of a message regarding the traffic jam which is 5 ms in our model. Therefore, these results correspond to a hop count of one. For less than 25% equipped vehicles on the road, the mean values of hop count vary between 1.29 and 2.43 hops.

The latency of successfully accepted messages for small deployment with 5% and less equipped vehicles on the road is greater than the hop count multiplied with the sum of transmission duration and maximum waiting



Percentage of equipped vehicles [%]	Latency of SBR with confidence intervals for $\alpha = 5\%$ [s]	hop count of SBR with confidence intervals for $\alpha = 5\%$ [hops]
1	5.384 ± 2.787	1.29 ± 0.08
2	2.517 ± 0.895	1.55 ± 0.07
5	0.941 ± 0.280	2.43 ± 0.15
10	0.014 ± 0.002	1.70 ± 0.12
15	0.014 ± 0.002	1.74 ± 0.13
25	0.005 ± 0.000	1.04 ± 0.04
50	0.005 ± 0.000	1.00 ± 0.00
75	0.005 ± 0.000	1.00 ± 0.00
100	0.005 ± NaN	1.00 ± NaN

Figure 6.13: Results for latency and hop count of SBR

Percentage of equipped vehicles [%]	Mean sizes of jams at max(T) with confidence intervals for $\alpha = 5\%$ [m] (all vehicles)	(equipped vehicles)
1	3962.0 ± 334.0	1308.3 ± 264.0
2	3985.2 ± 320.7	2367.4 ± 346.1
5	4015.4 ± 323.0	3026.3 ± 326.3
10	1337.8 ± 119.9	1091.9 ± 109.6
15	1301.9 ± 101.5	1176.6 ± 92.6
25	696.4 ± 20.5	551.1 ± 22.9
50	614.6 ± 24.0	542.4 ± 28.2
75	339.0 ± 16.7	312.1 ± 18.3
100	199.6 ± 0.0	199.6 ± 0.0

Table 6.4: Average jam sizes at end of simulation runs

time to forward the message. For 5% equipped vehicles on the road, this term equals to $2.43 \cdot (0.005 \text{ s} + 0.040 \text{ s}) = 0.11$ s. Instead of 110 ms, the latency is approximately nine times as high for 5% equipped vehicles on the road. This clearly shows the disconnected operation of the routing algorithm where the messages pending to be forwarded are kept until new receivers move into vicinity.

Table 6.4 summarizes the mean sizes of traffic jams at the end of the simulation runs in our scenarios. We compare the mean traffic sizes for jams reduced to equipped vehicles with the hop count for 1%, 2%, and 5% equipped vehicles on the road. If we divide the mean jam sizes by 2 to find an average jam size instead of the largest size at the end of simulation, and if we further divide the result by 600 m—the transmission range—we obtain the values 1.09, 1.97, and 2.52. These values are approximately the mean values of hop counts for 1%, 2%, and 5% equipped vehicles on the road.

For higher percentage of equipped vehicles on the road, we explain the very small hop counts and latency with the short simulated time. We simulated 120 s and less with 25% and more equipped vehicles on the road. During such a short time, the traffic jams evolving in our traffic model are almost not longer than our transmission range of 600 m or even shorter. Then, the hop count equals one and the latency is the transmission duration of 5 milliseconds. We expect higher hop counts and longer latencies if it was possible to obtain longer simulation runs.

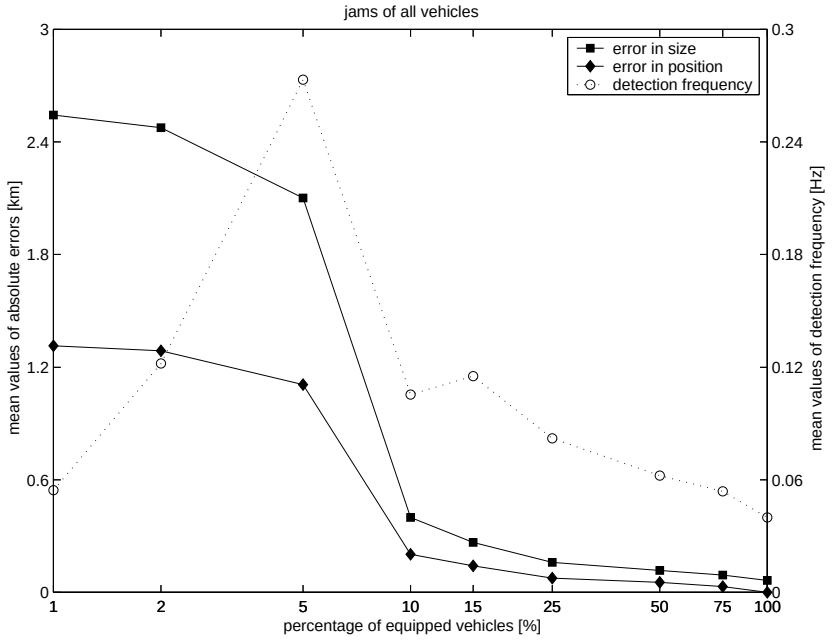
6.2.5 Error of Traffic Jam Detection

The last metric evaluates the quality of our application to detect traffic jams. Figures 6.14 and 6.15 display the results on the error of traffic jam detection for jams of all vehicles and for jams that contain only equipped vehicles, respectively. Table 6.6 summarizes the results along with the confidence intervals.

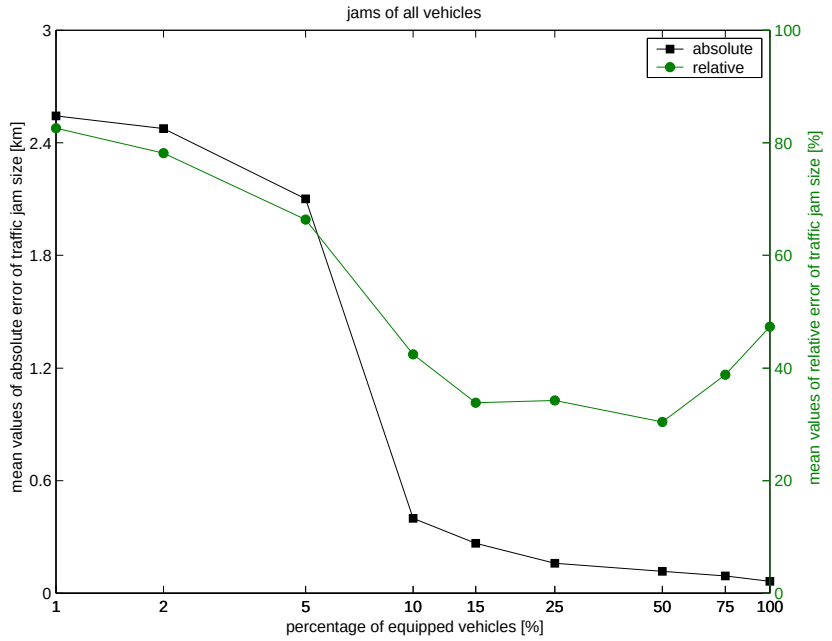
In Figure 6.14(a), we plot the absolute errors of traffic jam size and position as a function of the percentage of equipped vehicles on the road. Both curves show the same characteristics. The curves reach their maximum for the smallest deployment of 1% equipped vehicles on the road—an average error of 2543 m for the traffic jam size and an average error of 1314 m for the traffic jam position. For higher deployment of the system, the error shrinks fast and decreases to a minimum of 63 m for traffic jam size and 0 m for jam position in the scenario with all vehicles being equipped on the road. However, we obtained only one simulation run with results on the traffic jam detection for 100% deployment of the system. Therefore, the confidence of these results is not relevant.

Even though the absolute errors of traffic jam detection decrease for more vehicles being equipped on the road, we also investigate the relative error of the detected traffic jam size. The results along with the absolute error of traffic jam size are plotted in Figure 6.14(b). For relative errors, we divide every detected traffic jam size by the size of the actual jam if greater than zero. For higher deployment of the system, we reduced the simulated time and hence the resulting traffic jams are shorter. Therefore, the relative error of traffic jam size follows in the beginning the characteristics of the absolute error in position—starting from an error of 83% of the actual traffic jam size for 1% equipped vehicles on the road. Then, the relative error reaches its minimum of 30% misclassified jam size at a deployment of 50 percent. For higher percentages of equipped vehicles on the road, the relative error in traffic jam size detection increases slightly because the actual jam sizes decrease.

Figure 6.15 contains the results for traffic jam detection reduced to jams of equipped vehicles which show similar characteristics as the results for jams of all vehicles. In general, we achieve better results for jams that consist solely of equipped vehicles than for jams of all vehicles. For example, the maximum of the absolute errors in traffic jam detection is reached for 5% equipped vehicles on the road and averages to 1327 m error in size and 727 m

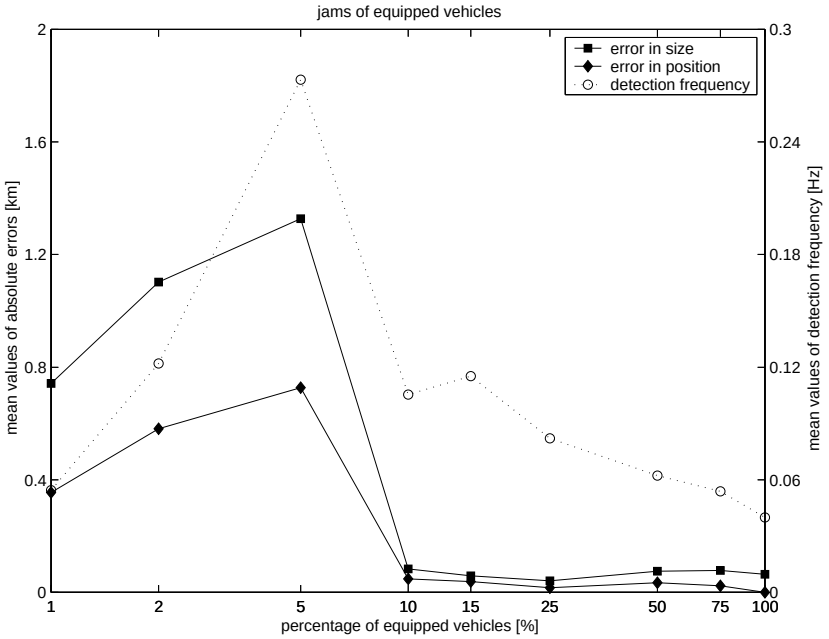


(a) absolute error of size and position

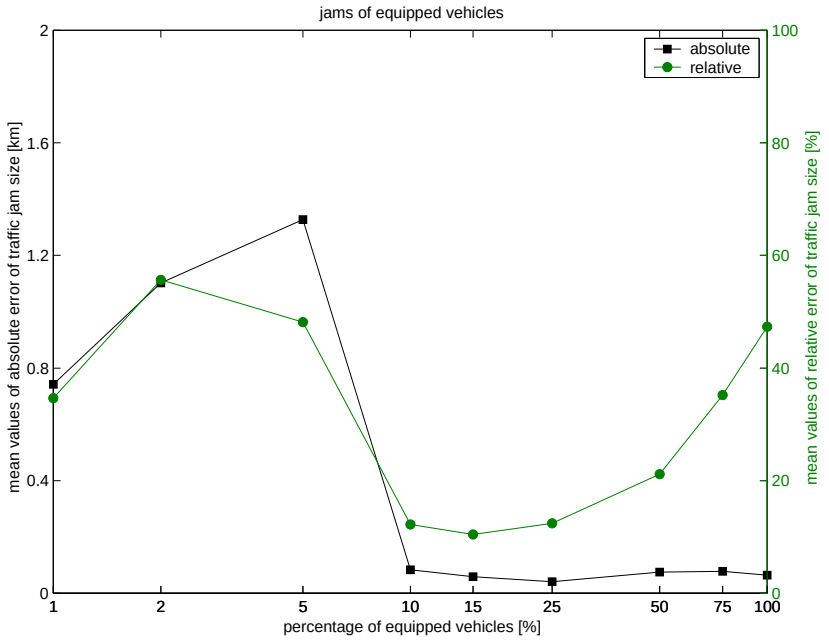


(b) absolute and relative error of size

Figure 6.14: Results for error of traffic jam detection (all vehicles)



(a) absolute error of size and position



(b) absolute and relative error of size

Figure 6.15: Results for error of traffic jam detection (equipped vehicles)

Percentage of equipped vehicles [%]	Detection frequency with confidence intervals for $\alpha = 5\%$ [Hz]
1	0.054 ± 0.009
2	0.122 ± 0.011
5	0.273 ± 0.012
10	0.105 ± 0.010
15	0.115 ± 0.007
25	0.082 ± 0.005
50	0.062 ± 0.009
75	0.054 ± 0.011
100	0.040 ± 0.000

Table 6.5: Results for detection frequency of traffic jams

error in position. The relative error of detected traffic jam size also increases slightly for higher deployment of the system because the actual traffic jam sizes are smaller due to shorter simulated time. The relative error varies between 10% and 56% of the actual size of the traffic jams.

We explain the fact that the average error of detected traffic jams increases slightly for very small deployment of 1% to 5% equipped vehicles on the road with the observed detection frequency. We added the mean values of detection frequency to figures 6.14(a) and 6.15(a). Table 6.5 summarizes the results of detection frequency along with confidence intervals. With 5% equipped vehicles on the road, we have more than five times as many detections as for 1% deployment of the system.

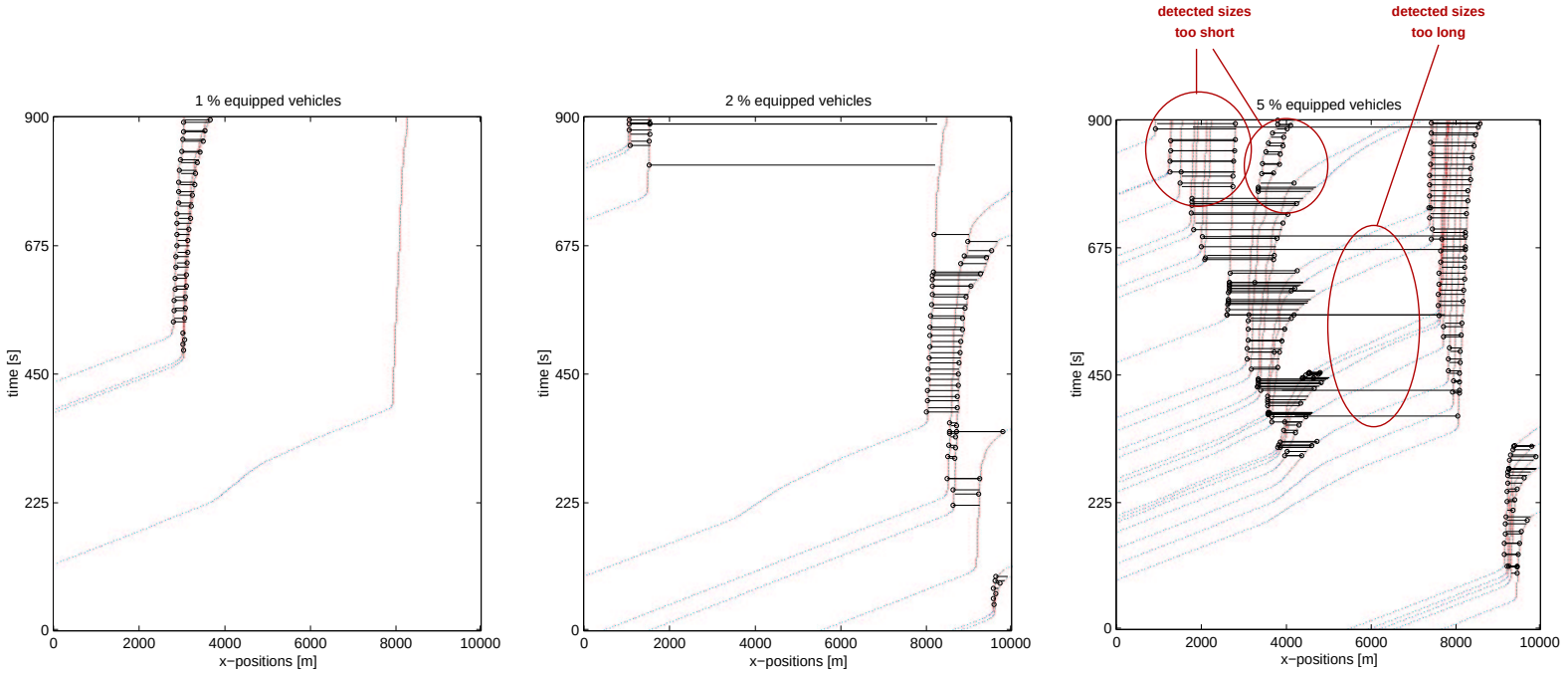
Figure 6.16 presents an example on the error of detected traffic jams increasing with higher deployment. Diagram 6.16(a) displays a simulation run with 1% equipped vehicles on the road. Although only the three vehicles on the left hand side detect the traffic jam, they happen to capture the size and the position perfectly. The single equipped vehicle on the right hand side lacks communication partners and hence never exchanges messages to detect the traffic jam.

With 2% equipped vehicles on the road—as seen in Figure 6.16(b), the equipped vehicles occasionally detect the traffic jam inaccurately. Especially equipped vehicles on the left hand side that receive a message from the jam on the right hand side via intermediate vehicles possibly traveling in the opposite direction cannot distinguish the two jams and extend the error of detection in the overall metric.

Finally, Figure 6.16(c) show frequent and most of the time accurate detection

	Percentage of equipped vehicles [%]	Absolute error of traffic jam size with confidence intervals for $\alpha = 5\%$ [m]	Absolute error of traffic jam position with confidence intervals for $\alpha = 5\%$ [m]	relative error of traffic jam size with confidence intervals for $\alpha = 5\%$
all vehicles	1	2543 ± 253	1314 ± 123	82.62 ± 3.19
	2	2475 ± 201	1287 ± 97	78.18 ± 1.79
	5	2101 ± 180	1107 ± 87	66.38 ± 1.91
	10	399 ± 54	202 ± 28	42.40 ± 3.71
	15	265 ± 25	140 ± 15	33.85 ± 2.43
	25	158 ± 11	75 ± 5	34.22 ± 1.96
	50	116 ± 10	54 ± 7	30.44 ± 3.00
	75	91 ± 14	30 ± 8	38.83 ± 5.43
	100	63 ± NaN	0 ± NaN	47.33 ± NaN
equipped vehicles	1	742 ± 189	354 ± 89	34.62 ± 7.10
	2	1102 ± 171	580 ± 80	55.63 ± 9.56
	5	1327 ± 174	727 ± 83	48.17 ± 3.25
	10	83 ± 35	48 ± 21	12.20 ± 3.82
	15	59 ± 13	38 ± 9	10.44 ± 2.17
	25	41 ± 3	16 ± 2	12.42 ± 1.00
	50	75 ± 9	35 ± 6	21.23 ± 2.75
	75	77 ± 12	23 ± 8	35.21 ± 5.77
	100	63 ± NaN	0 ± NaN	47.33 ± NaN

Table 6.6: Results for error of traffic jam detection (all and equipped vehicles)



(a) rare detection with accurate traffic jam sizes

(b) detection with occasionally misclassified traffic jam sizes

(c) frequent detection with some inaccurate traffic jam sizes

Figure 6.16: Example of detection of traffic jams degrading with increasing deployment

of the traffic jams. However, having a higher frequency of detection also increases the average error of the detected jam. We circled the regions of frequent misclassification where the detected jam is either too long or too short.

6.3 Summary on Evaluation

In this chapter, we have presented metrics to evaluate the performance of different layers in our proposed inter-vehicle communication system.

We measured the packet success to capture the amount of lost packets compared to the successfully received packets using point-to-multipoint communication links. The results show that for small deployment of the system, the loss of packets is negligible and the simple medium access control scheme is sufficient. For denser populated networks with 25% and more equipped vehicles on the road, the packet success gradually decreases and the amount of send omissions and collisions increases. This indicates the need for a more sophisticated MAC scheme once the system is more deployed.

For future implementations, we recommend a hybrid medium access strategy. There, the medium access control distinguishes two types of sending requests. With the first type, the packets are sent immediately or dropped (as our implementation currently does)—which is suitable for short messages like heartbeats or possibly the group membership status. The second type of packets are queued in a sending buffer in case the transceiver is busy when the request arrives. Longer messages regarding the traffic jam would fall into this category. If the medium access control applies such a strategy, the buffer length and crafting a scheme to schedule waiting packets for retry—all hosts that wait to send a packet, shall not start immediately after the current transmission is over—requires careful consideration.

The implementation of the neighborhood service is evaluated by the metric of accuracy of neighborhood. The results yield high accuracies for almost all deployment rates. However, from the data we have obtained we cannot fully explain, why the accuracy drops significantly for 100% deployment of the system.

In contrast to the neighborhood service, the accuracy of the installed views on the local membership reaches high values well above 90% for all deployment rates. The measured mean trust values of installed views are higher than 99% for all deployment rates. Both metrics show that the combination

of our proposed algorithm with the setting of parameters achieve the task of maintaining the local group membership even for small percentages of equipped vehicles on the road.

The results on our proposed routing scheme are evaluated with respect to accepted messages. Thus, we depend on the traffic jam formation causing enough messages to be initiated and accepted to study the routing mechanism. The simulation time for high deployment of the system has been reduced because the performance of the simulator did not allow longer runs. Therefore, the results for SBR apply mostly to small percentage of equipped vehicles on the road. Here, we clearly see the fragmentation of the ad hoc network occurring for 5% and less equipped vehicles on the road, because the latency of messages is much longer than expected for a connected topology. We achieve high success values above 85% for accepted messages even in the presence of network partitions. Our approach indeed overcomes the disconnection of the network.

Again, the reduced simulation time for higher deployment of the system makes it difficult to compare these results with those for small percentages of equipped vehicles on the road. The error of traffic jam detection shows that the task to accurately measure the traffic jam cannot be fulfilled with our algorithm. The dynamics of the application—new vehicles constantly enter and leave the jam—lead to an inherently inaccurate jam detection because the mechanism to let the vehicles classify who is at the border of the jam must converge first. If the algorithm broadcast its position more often, the agreement would be reached earlier but also the channel usage increases. This opposes the approach to keep the channel accessible most of the time.

We recommend to apply a strategy to average out the detected traffic jams over a period of time. As seen in the example simulation runs, the misclassified traffic jams are rare in comparison to the correct classification. Then, an equipped vehicle would observe the evolution of the jam for a few minutes until the knowledge is consolidated.

Chapter 7

Conclusion

Advances in the technology of wireless communication and small devices have drawn the attention of research to mobile ad hoc networks. Designing communication protocols and applications for ad hoc networks poses a difficult challenge due to mobility, constrained bandwidth, and the absence of fixed infrastructure or dedicated master stations. The network topology changes frequently and is unpredictable which causes problems in standard routing schemes that heavily employ routing tables to find a path to the destination.

Inspired by an application of ad hoc networking to inter-vehicle communication, we conclude that the routing task shifts from explicitly naming the destination of a message towards anonymously addressing messages. Abstractly, an anonymous message contains a description of its destination such that hosts receiving the message decide using their knowledge whether they belong to the intended recipients. We propose a situation based routing (SBR) mechanism which utilizes a flooding scheme to achieve the task of anonymous routing. The anonymous routing task is more general than geographic addressing which is recently reported in literature. Also, some approaches use geographic constraints to aid the routing procedure rather than describing the packet's destination solely by the position.

We advocate anonymous routing to solve the problem of detecting a traffic jam on highways. Vehicles at the border of the traffic jam send a message to the other end of the congestion without knowing the identity of the destination. In anonymous routing, the vehicles decide on their own if they are at the border of the jam. This requires an underlying mechanism that provides the positions of other jammed vehicles in the vicinity of the jammed

vehicles.

To achieve this, we map the concept of group formation known from distributed systems to the constitution of a group among the jammed vehicles on the road. We categorize the ad hoc network as an asynchronous distributed system with mobile processes, prone to network disconnection and process crashes. We extend the idea of partitionable group membership to cope with the system immanent problems of mobility and large scale deployment. We propose a localized group membership service (LGMS) that reduces the membership information to the adjacent neighbors of a host.

We develop and implement the LGMS together with the underlying neighborhood service (NHS). To simulate the behavior of it, we supplement the communication system with a higher application layer that detects traffic jams based on the local group membership. We embed the network simulation in a realistic traffic jam scenario on a highway.

The accuracy and the measured trust values of the installed views on the local membership reach high values well above 90% for all deployment rates. Both metrics show that the combination of our proposed algorithm with the chosen setting of parameters achieve the task of maintaining the local group membership.

We focused on sparse deployment of the system which is likely to occur soon after such a system is introduced to the market. In this state, the resulting ad hoc network tends to be disconnected. We tailor our SBR scheme to operate in spite of network disconnection. The simulation shows indeed that for 5% and less equipped vehicles on the road, SBR overcomes the network fragmentation. We reduced the simulated time of the traffic scenario for high deployment of the system because of the computing performance of the simulator. Hence, the results for higher percentage of equipped vehicles on the road bear less statistical quality but nevertheless show a general trend.

The results of the application to detect traffic jams reflect the difficulty of such a task. For small deployment of the system, the vehicles detect the vehicle at the other end of the traffic jam quite reliably, if the message overcomes possible gaps in the network topology. When the percentage of equipped vehicles on the road increases, the high fluctuation at the border makes it difficult to reach exactly the outermost vehicle inside the jam. Thus, the error of traffic jam detection does not reach zero for full deployment. To improve the detection, we need to enrich the algorithm with more intelligent

features.

We recommend a strategy that averages out the detected traffic jams over a period of time. As seen in the simulation runs, the misclassified traffic jams are rare in comparison to the correct classification. Then, an equipped vehicle would observe the evolution of the jam for a few minutes until the knowledge is classified as consolidated.

7.1 Future Directions

In order to compare the performance of different routing protocols, the routing task to be achieved must be coherent. We could not perform such comparisons because the paradigm of anonymous routing is relatively new and rarely addressed in literature. If other applications in mobile ad hoc networking benefit from the approach to anonymous routing, the metrics to evaluate routing protocols should be standardized.

Another open problem is to study the assumptions on a benign system in which we can prove that the proposed implementation of NHS and LGMS satisfies the specification of the services. From the discussion on group membership, we isolated core requirements in timing which can be used as a starting point. As a trade off, imposing a good nature of the system model limits the applicability of the theoretic results to reality where unsuspected failures can happen.

Finally, the rather simple protocols of the communication system's components suit the application in sparsely populated networks but should be optimized for heavier deployment of the system. As sketched above, more sophisticated approaches using the history and other sensor data of the vehicle can extend the accuracy of the traffic jam detection and prevent sporadic misclassifications. Another idea for future implementation includes adjusting the transmission range of radio communication such that an optimum number of neighbors is reached on average. On the other hand, increasing the complexity of the protocols and the population of the network in our simulation framework requires more computing power to achieve results within a reasonable amount of time.

We feel that transferring the concepts of distributed systems to the area of mobile ad hoc networks and vice versa provides valuable insight to understand both areas. The active discussions about a partitionable group

membership service definition are still fueling the field of distributed systems. We hope that an interdisciplinary exchange of ideas between these two ongoing research areas continues to grow.

Appendix A

Temporal Logic Operators

This chapter gives a formal definition of the temporal logic operators used in Chapter 3 as found in [MP92]. The time $T = 0, 1, 2, ...$ is isomorphic to the set of natural numbers $\mathbb{N}$. A temporal formula p consists of variables, functions, quantifiers, and boolean and temporal operators. A computation γ is a sequence of states $c_0, c_1, c_1, ...$ Each state c_i assigns values to the variables. Then, a temporal formula can be interpreted in a state by evaluating the formula with the variables replaced by their values in c_i. We write $(\gamma, i) \vDash p$ if and only if p interpreted at time i in computation γ is TRUE. Table A.1 gives an example for each operator.

For $k \geq 0, (k \in \mathbb{N})$:

The "next" Operator $\bigcirc$

$$\begin{aligned} (\gamma, i) &\vDash \bigcirc p \quad \text{iff} \quad (\gamma, i+1) \vDash p \\ (\gamma, i) &\vDash \bigcirc^k p \quad \text{iff} \quad (\gamma, i+k) \vDash p \end{aligned}$$

The "always" Operator $\Box$

$$\begin{aligned} (\gamma, i) &\vDash \Box p \quad \text{iff} \quad (\gamma, j) \vDash p \text{ for all } j \geq i \\ (\gamma, i) &\vDash \Box_{\leq k}\, p \quad \text{iff} \quad (\gamma, j) \vDash p \text{ for all } i \leq j \leq i+k \end{aligned}$$

The "eventually" Operator $\Diamond$

$$\begin{aligned} (\gamma, i) &\vDash \Diamond p \quad \text{iff} \quad (\gamma, j) \vDash p \text{ for some } j \geq i \\ (\gamma, i) &\vDash \Diamond_{\leq k}\, p \quad \text{iff} \quad (\gamma, j) \vDash p \text{ for some } i \leq j \leq i+k \end{aligned}$$

The "until" Operator $\mathcal{U}$

$$(\gamma, i) \models p \,\mathcal{U}\, q \quad \text{iff} \quad \text{there exists a } l \geq i \text{ such that } (\gamma, l) \models q \text{ and for every } j, i \leq j < l \text{ is } (\gamma, j) \models p$$

The "previous" Operator $\ominus$

$$(\gamma, i) \models \ominus p \quad \text{iff} \quad i > 0 \text{ and } (\gamma, i-1) \models p$$
$$(\gamma, i) \models \ominus^k p \quad \text{iff} \quad i \geq k \text{ and } (\gamma, i-k) \models p$$

The "has-always-been" Operator $\boxminus$

$$(\gamma, i) \models \boxminus p \quad \text{iff} \quad (\gamma, j) \models p \text{ for all } 0 \leq j \leq i$$
$$(\gamma, i) \models \boxminus_{\leq k}\, p \quad \text{iff} \quad i \geq k \text{ and } (\gamma, j) \models p \text{ for all } i-k \leq j \leq i$$

The "once" Operator $\diamondminus$

$$(\gamma, i) \models \diamondminus p \quad \text{iff} \quad (\gamma, j) \models p \text{ for some } 0 \leq j \leq i$$
$$(\gamma, i) \models \diamondminus_{\leq k}\, p \quad \text{iff} \quad i \geq k \text{ and } (\gamma, j) \models p \text{ for some } i-k \leq j \leq i$$

The "since" Operator $\mathcal{S}$

$$(\gamma, i) \models p \,\mathcal{S}\, q \quad \text{iff} \quad \text{there exists a } 0 \leq l \leq i \text{ such that } (\gamma, l) \models q \text{ and for every } j, l < j \leq i \text{ is } (\gamma, j) \models p$$

i	0	1	2	3	4	5	6	7	8	...
x	1	1	2	3	1	1	2	3	3	(3 cont'd)
$\bigcirc(x=3)$	F	F	T	F	F	F	T	T	T	...
$\bigcirc^2(x=3)$	F	T	F	F	F	T	T	T	T	...
$\Box(x=3)$	F	F	F	F	F	F	F	T	T	...
$\Box_{\leq 1}(x=1)$	T	F	F	F	T	F	F	F	F	...
$\Diamond(x=2)$	T	T	T	T	T	T	T	F	F	...
$\Diamond_{\leq 2}(x=2)$	T	T	T	F	T	T	T	F	F	...
$(x>1)\,\boldsymbol{\mathcal{U}}(x=3)$	F	F	T	T	F	F	T	T	T	...
$\ominus(x=1)$	F	T	T	F	F	T	T	F	F	...
$\ominus^2(x=1)$	F	F	T	T	F	F	T	T	F	...
$\boxminus(x=1)$	T	T	F	F	F	F	F	F	F	...
$\boxminus_{\leq 1}(x=1)$	F	T	F	F	F	F	F	F	F	...
$\diamondminus(x=2)$	F	F	T	T	T	T	T	T	T	...
$\diamondminus_{\leq 2}(x=2)$	F	F	T	T	T	F	T	T	T	(F cont'd)
$(x<3)\,\boldsymbol{\mathcal{S}}(x=1)$	T	T	T	F	T	T	T	F	F	...

Table A.1: Example on temporal logic operators

Appendix B

Traffic Jam Simulator

This chapter includes the M-files used to implement the microscopic traffic simulator. It corresponds to the model introduced in Section 5.1. The simulator runs under MATLAB®. The microscopic traffic jam simulator is implemented in the function "mtjs." It uses two functions "calcV" and "calcY."

To invoke the traffic simulation, the user gives five arguments to the function "mtjs." The first parameter is the number of seconds to be simulated. The second parameter determines a pause time in seconds after each simulated time step. For non-animated usage, this parameter should be set to zero. The third parameter is a switch between a silent and an animated mode. In animated mode, the layout of the road is printed into a figure and the vehicles are drawn as small squares. During the simulation, the vehicles are animated and redrawn at their new positions after each time step. Additionally, the squares are colored according to the state of the vehicles. Green stands for vehicles that are not connected to the leading vehicle. Blue vehicles are connected but not jammed. Jammed vehicles are colored red. The fourth parameter of the function "mtjs" is a prefix of filenames which is used for storing trace files of the simulation. Finally, the fifth parameter points to a file which contains the settings for the traffic simulation. For example, a five minute simulation without animation can be started with

```
>> mtjs(300,0,1,‘data/example‘,‘data/.parameter‘)
```

Then, the program reads the traffic simulation parameters from the file "data/.parameter," generates a log file "data/example_matlab.log," and saves

[0]MATLAB® is a registered trademark of The MathWorks, Inc.

the trace files under "data/example.position," "data/example.state," and "data/example.speed."

B.1 MATLAB M-files

mtjs.m

```
function mtjs(endTime,factor,silent,name,pname)
% function mtjs(endTime,factor,silent,name,pname)
% ---------------------------------------
% microscopic traffic jam simulator
% jam caused by stopped vehicle at the right hand side
% endTime: sumlation runs for "endTime" seconds
% factor: speed factor for time elapsing
% silent: determines output modus (0: animated figure, 1: no output)
% name: create trace files and save them to "<name>.position",
%       "<name>.state", and "<name>.speed"
%       log date and time to "<name>.log"
% pname: get parameters form file "pname"

% open log file for output
fid_log = fopen([name '_matlab.log'],'w');
if fid_log == -1
   fprintf(2,'ERROR: cannot open %s for writing!\n',[name '_matlab.log'])
   return
end
fprintf(fid_log,'Start simulation at %s.\n',datestr(now,0));
fprintf(fid_log,'Simulating %d s with factor %.1f.\n',endTime,factor);

% load parameters
parameters = load(pname);
roadLength = parameters(1,1);
divided = parameters(2,1);
laneWidth = parameters(3,1);
nLanes = parameters(4,1);
rho = [parameters(5,1) parameters(6,1)];
length = parameters(7,1);
vMax = parameters(8,1);
a = parameters(9,1);
b = parameters(10,1);
eps = parameters(11,1);
kStopped = parameters(12,1);
vStopped = parameters(13,1);

if ~silent
   % prepare figure for output
   newplot
   axis ij
   xtra = divided*0.5*laneWidth; % extra space for divider
   % shrink height of plot according to nLanes and divided
   set(gcf,'Units','points')
   fpos = get(gcf,'Position');
```

```
   set(gcf,'Position',[fpos(1,1) fpos(1,2) fpos(1,3) 50*2*nLanes+100+50*xtra])
   set(gcf,'Units','pixels')
   set(gca,'Units','points')
   apos = get(gca,'Position');
   h = 50*2*nLanes+50*xtra;
   set(gca,'Position',[apos(1,1) apos(1,2)+0.5*(apos(1,4)-h) 800 h])
   set(gcf,'Units','normalized')
   % set other properties
   set(gca,'YLim',[-nLanes*laneWidth-xtra nLanes*laneWidth+xtra])
   ticks = laneWidth+xtra:laneWidth:(nLanes-1)*laneWidth+xtra;
   set(gca,'YTick',sort([-ticks ticks]))
   set(gca,'YTickLabel',{},'YGrid','on','XLim',[0 roadLength],'Box','on')
   set(gcf,'RendererMode','manual','Renderer','painters','DoubleBuffer','on')
   xlabel('x-position [m]')
   % draw divider
   if xtra > 0
      rectangle('Position',[0,-xtra,roadLength,2*xtra],'FaceColor',[0.9,0.9,0.9])
   else
      hold on
      plot([0 roadLength],[0 0],'-k')
      hold off
   end
   % create label that displays simulated time
   htime = title('xxx s');
end

% open trace files for output
fid_pos = fopen([name '.position'],'w');
if fid_pos == -1
   fprintf(2,'ERROR: cannot open %s for writing!\n',[name '.position'])
   return
end
fid_sta = fopen([name '.state'],'w');
if fid_sta == -1
   fprintf(2,'ERROR: cannot open %s for writing!\n',[name '.state'])
   return
end
fid_spd = fopen([name '.speed'],'w');
if fid_spd == -1
   fprintf(2,'ERROR: cannot open %s for writing!\n',[name '.speed'])
   return
end
% write comments
fprintf(fid_pos,'%%TRACE\n');
fprintf(fid_pos,'%% time id x y z rx ry rz\n');
fprintf(fid_sta,'%%STATE\n');
fprintf(fid_sta,'%% time id state(1 = free flow, ');
fprintf(fid_sta,'2 = connected but not jammed, 3 = jammed)\n');
fprintf(fid_spd,'%%SPEED\n');
fprintf(fid_spd,'%% time id speed\n');

global vehicles;
vehicles = cell(2,nLanes);
% holds queues of vehicles for each direction in each lane
```

```
% vehicles are tuples (ID, x, y, v, handle, conn_state, jam_state)
% conn_state: 1 = free flow, 2 = connected
% jam_state: 0 = not jammed, 1 = jammed

% initialization
hd = 0;
mcolors = ['g' 'b' 'r']; % marker colors indicating jam state
lambda = 1000*(rho.^(-1)); % constant average distance between vehicles in m
nextID = 1; % ID of next vehicle created
for dir = 1:2
   for lane = 1:nLanes
      nextX = sign(2-dir)*roadLength-sign(1.5-dir)*lambda(dir);
      hold on
      while (nextX>=0) & (nextX<=roadLength)
         % add vehicles
         x = nextX;
         y = calcY(dir,lane,divided,laneWidth,nLanes);
         v = vMax-(eps*a)*rand; % randomly reduced velocity
         if ~silent
            cbstr = sprintf('callbackfun(%d,%d)',nextID,nLanes);
            hd = plot(x,y, ...
               's','MarkerEdgeColor','k','MarkerSize',5, ...
               'MarkerFaceColor','g','ButtonDownFcn',cbstr);
         end
         vehicles{dir,lane} = [vehicles{dir,lane}; nextID x y v hd 1 0];
         nextID = nextID+1;
         nextX = x-sign(1.5-dir)*lambda(dir);
      end
      hold off
   end
end
if ~silent
   set(htime,'String','1 s')
end

% stop first vehicles in direction == 1 to cause jam
for lane = 1:nLanes
   for k = 1:min([kStopped size(vehicles{1,lane},1)])
      vehicles{1,lane}(k,4) = 0;
      vehicles{1,lane}(k,7) = 1; % jam_state is jammed
      if ~silent
         set(vehicles{1,lane}(k,5),'MarkerFaceColor','r');
      end
   end
end

% start simulation
for time = 1:endTime
   hold on
   % add new vehicles if necessary
   for dir = 1:2
      for lane = 1:nLanes
         last = size(vehicles{dir,lane},1);
         if last == 0 % lane is empty
```

```
            x = sign(2-dir)*roadLength-sign(1.5-dir)*lambda(dir);
        else
            lastX = vehicles{dir,lane}(last,2);
            x = lastX-sign(1.5-dir)*lambda(dir);
        end
        if (x>=0) & (x<=roadLength)
            % add new vehicle
            y = calcY(dir,lane,divided,laneWidth,nLanes);
            v = vMax-(eps*a)*rand; % randomly reduced velocity
            if ~silent
                cbstr = sprintf('callbackfun(%d,%d)',nextID,nLanes);
                hd = plot(x,y, ...
                    's','MarkerEdgeColor','k','MarkerSize',5, ...
                    'MarkerFaceColor','g','ButtonDownFcn',cbstr);
            end
            vehicles{dir,lane} = [vehicles{dir,lane}; nextID x y v hd 1 0];
            nextID = nextID+1;
        end
    end
end
% update driving vehicles
for dir = 1:2
    for lane = 1:nLanes
        remove = []; % collects vehicles to be removed
        vLeader = 0;
        for n = size(vehicles{dir,lane},1):-1:1
            % go through queue of vehicles starting from the last one
            if n > 1
                gap = abs(vehicles{dir,lane}(n-1,2)-vehicles{dir,lane}(n,2))-length;
                vLeader = vehicles{dir,lane}(n-1,4);
            else % first vehicle in queue
                gap = Inf;
            end
            [vehicles{dir,lane}(n,4),vehicles{dir,lane}(n,6)] = ...
                calcV(gap,vehicles{dir,lane}(n,4),vLeader,vMax,a,b,eps);
            newX = vehicles{dir,lane}(n,2)+sign(1.5-dir)*vehicles{dir,lane}(n,4);
            if (newX>=0) & (newX<=roadLength)
                vehicles{dir,lane}(n,2) = newX; % update x-position
                if ~silent
                    set(vehicles{dir,lane}(n,5),'XData',newX); % animate plot
                end
                % set jammed state if vehicle stopped
                if vehicles{dir,lane}(n,4) == 0
                    vehicles{dir,lane}(n,7) = 1;
                end
            else % remove vehicle
                if ~silent
                    delete(vehicles{dir,lane}(n,5));
                end
                remove = [remove n];
            end
        end
        % delete vehicles from queue
        vehicles{dir,lane}(remove,:) = [];
```

```
   end
end
% classify traffic jams and color vehicles:
% green = free flow, blue = connected but not jammed, red = jammed
% save data to traces
for dir = 1:2
   for lane = 1:nLanes
      connToJam = 0;
      state = 0;
      for n = 1:size(vehicles{dir,lane},1)
         conn_state = vehicles{dir,lane}(n,6);
         jammed = vehicles{dir,lane}(n,7);
         v = vehicles{dir,lane}(n,4);
         % decide about switching jam state
         if ~jammed % not jammed yet
            if (v < vStopped) | (connToJam & (conn_state == 2)) % enter jam
               % literally stopped or connected to jammed vehicle
               jammed = 1;
            end
         else % already jammed
            if (conn_state == 1) & (v > 0.5*vMax) % escape jam
               jammed = 0;
            end
         end
         % set jam state
         if jammed
            connToJam = 1;
            vehicles{dir,lane}(n,7) = 1;
            state = 3;
            if ~silent
               set(vehicles{dir,lane}(n,5),'MarkerFaceColor','r');
            end
         else % not jammed
            connToJam = 0;
            vehicles{dir,lane}(n,7) = 0;
            state = conn_state;
            if ~silent
               set(vehicles{dir,lane}(n,5), ...
                  'MarkerFaceColor',mcolors(1,conn_state));
            end
         end
         % write traces
         ID = vehicles{dir,lane}(n,1);
         x = vehicles{dir,lane}(n,2);
         y = vehicles{dir,lane}(n,3);
         rz = (sign(y)-1)*(-90);
         v = vehicles{dir,lane}(n,4);
         fprintf(fid_pos,'%.0f %d %.2f %.2f 0 0 0 %.0f\n',time,ID,x,y,rz);
         fprintf(fid_sta,'%.0f %d %d\n',time,ID,state);
         fprintf(fid_spd,'%.0f %d %.2f\n',time,ID,v);
      end
   end
end
hold off
```

```
   pause(factor);
   % update time string
   timestr = sprintf('%d s',time);
   if ~silent
      set(htime,'String',timestr)
   end
end
fprintf(fid_log,'Simulation ended at %s.\n',datestr(now,0));

% close files
fclose(fid_pos);
fclose(fid_sta);
fclose(fid_spd);
fclose(fid_log);
```

calcV.m

```
function [v,c] = calcV(gap,vf,vl,vMax,a,b,eps)
% function [v,c] = calcV(gap,vf,vl,vMax,a,b,eps)
% ----------------------------------------------
% calculates new velocity of follower from "gap" between vehicles
% and from velocities of follower and leader "vf", "vl" according
% to Krauss-model
% returns connected state in "c" corresponding to the following:
%   1: - free flow
%   2: - connected

c = 1;
if gap == Inf
   % no predecessor
   v = vMax-(eps*a)*rand;
else
   vSafe = vl+(gap-vl)/((vl+vf)/(2*b)+1);
   [vDes,i] = min([vMax vf+a vSafe]);
   if i == 3
      c = 2; % choosing vSafe indicates being connected
   end
   v = max([0 vDes-(eps*a)*rand]);
end
```

calcY.m

```
function y = calcY(dir,lane,divided,laneWidth,nLanes)
% function y = calcY(dir,lane,divided,laneWidth,nLanes)
% -----------------------------------------------------
% calculates y-position from "lane" in "dir"
% the rightmost lane in each direction is labeled 1
% "dir" == 1 is up, "dir" == 2 is down
% "divided" == 0: driving directions not divided,
%              1: driving directions separated

y = sign(-dir+1.5);
factor = nLanes-lane+1;
if divided == 0
```

```
   factor = factor-0.5;
end
y = y*factor*laneWidth;
```

Appendix C

Communication Network Simulator

This chapter contains the SDL diagrams that implement the communication network simulator. From the SDL notation, we generated executable code that reads the traces of equipped vehicles as computed by the traffic simulator described in Appendix B. The simulator writes back the results to files using the same prefix of these trace names.

An external text file similar to Figure C.1 provides the individual parameters for a simulation run. The environment variable "SDTEXTSYNFILE" must point to this parameter file. Also, the directory with the executable program of the simulator contains a textfile "siminit.com" as seen in Figure C.2 that stores the commands executed by the simulator. Then, we are able to run the simulator stand-alone from the command line:

```
$ MANETSimulator_smc
```

The remaining part of this chapter consists of all diagrams that together form the simulator used for our experiments. For an overview on the SDL system structure refer to the tree in Figure C.3.

```
/* variable SDL parameters */
SeedHBOffset 26205
SeedTJOffset 10589
Seed4Seed 15389
nHeartbeat 1.0
TRACE_PATH '..\..\matlab\traffic\data\run4\free2\run5'
SimulationTime 900.0
```

Figure C.1: Example on external parameter file for SDL simulator

```
Set-Trace System MANETSimulator 0
Go
Exit
```

Figure C.2: Command file for SDL simulator

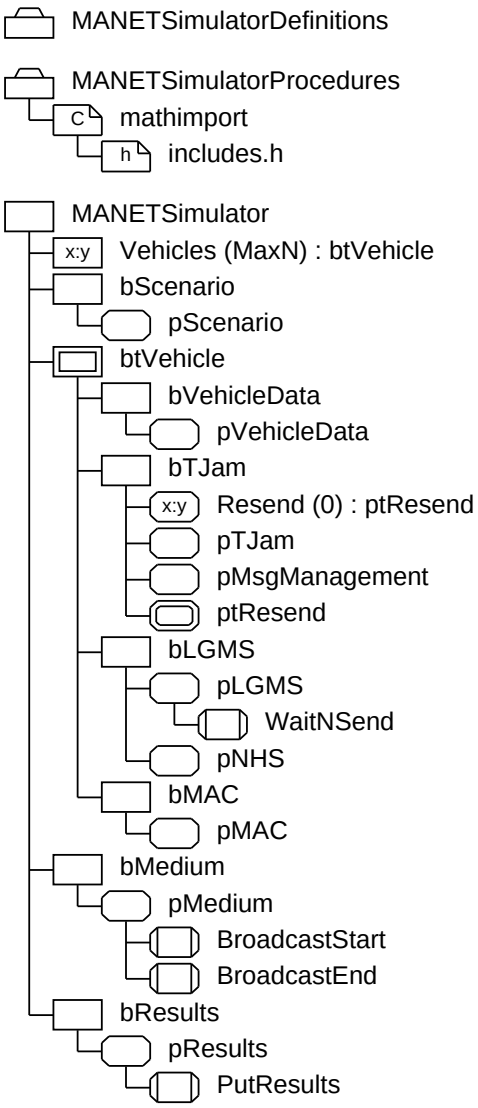
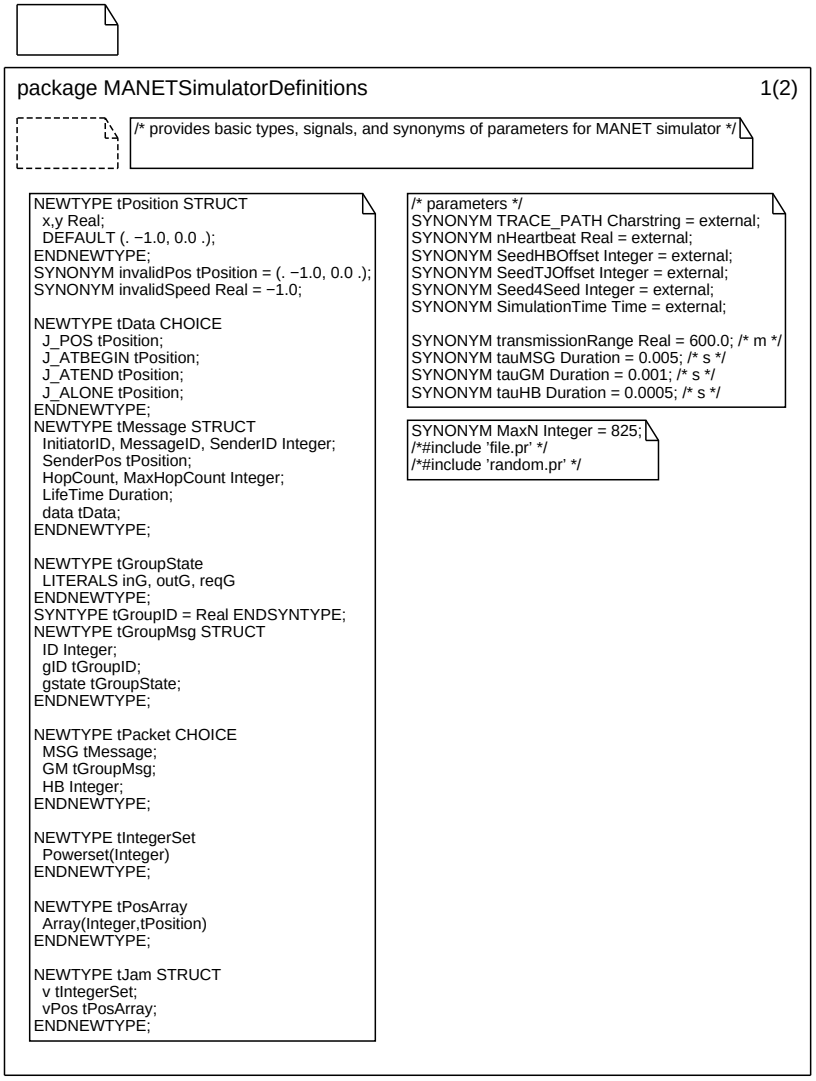
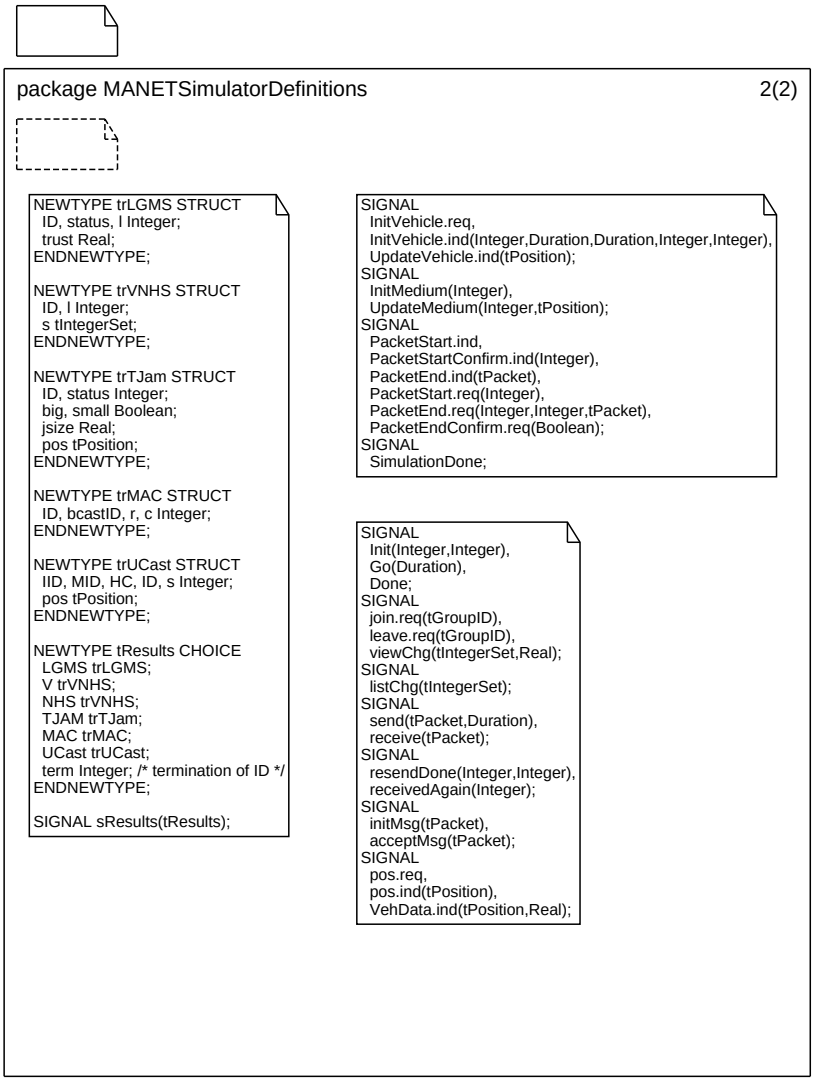


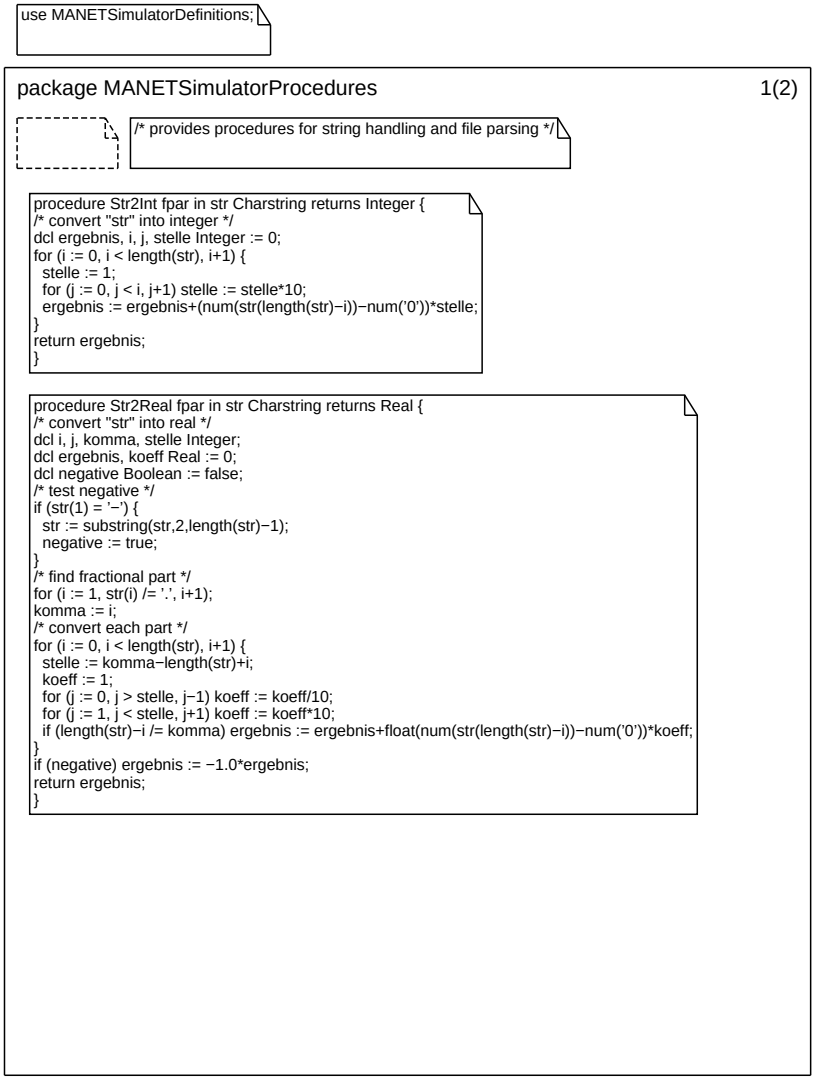
Figure C.3: Overview on SDL system structure

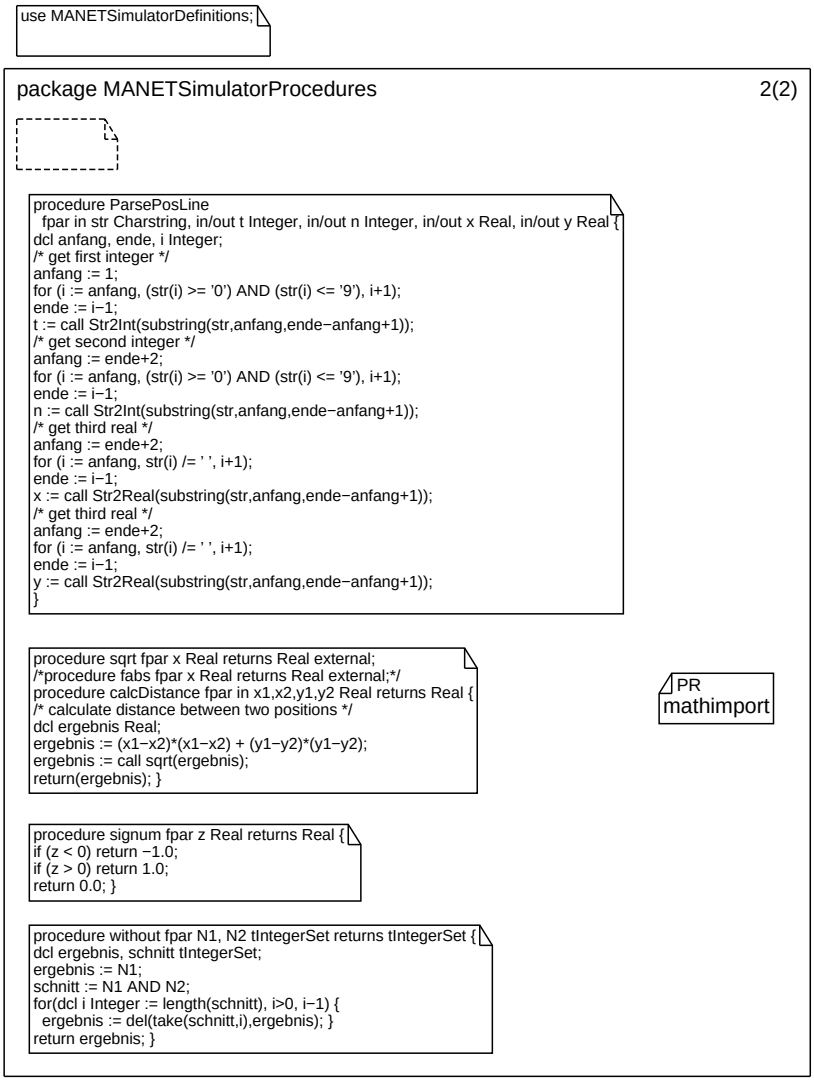
C.1 Definitions



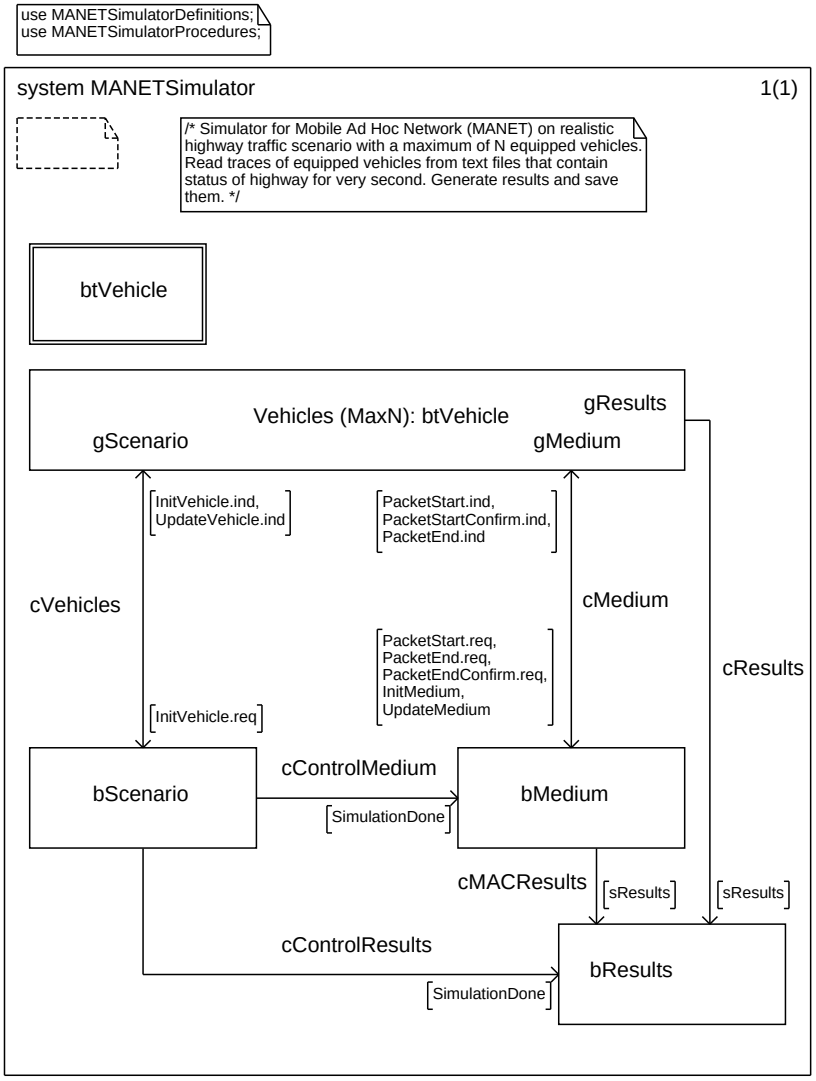

package MANETSimulatorDefinitions 2(2)
NEWTYPE trLGMS STRUCT
ID, status, l Integer;
trust Real;
ENDNEWTYPE;
NEWTYPE trVNHS STRUCT
ID, l Integer;
s tIntegerSet;
ENDNEWTYPE;
NEWTYPE trTJam STRUCT
ID, status Integer;
big, small Boolean;
jsize Real;
pos tPosition;
ENDNEWTYPE;
NEWTYPE trMAC STRUCT
ID, bcastID, r, c Integer;
ENDNEWTYPE;
NEWTYPE trUCast STRUCT
IID, MID, HC, ID, s Integer;
pos tPosition;
ENDNEWTYPE;
NEWTYPE tResults CHOICE
LGMS trLGMS;
V trVNHS;
NHS trVNHS;
TJAM trTJam;
MAC trMAC;
UCast trUCast;
term Integer; /* termination of ID */
ENDNEWTYPE;
SIGNAL sResults(tResults);
SIGNAL
InitVehicle.req,
InitVehicle.ind(Integer,Duration,Duration,Integer,Integer),
UpdateVehicle.ind(tPosition);
SIGNAL
InitMedium(Integer),
UpdateMedium(Integer,tPosition);
SIGNAL
PacketStart.ind,
PacketStartConfirm.ind(Integer),
PacketEnd.ind(tPacket),
PacketStart.req(Integer),
PacketEnd.req(Integer,Integer,tPacket),
PacketEndConfirm.req(Boolean);
SIGNAL
SimulationDone;
SIGNAL
Init(Integer,Integer),
Go(Duration),
Done;
SIGNAL
join.req(tGroupID),
leave.req(tGroupID),
viewChg(tIntegerSet,Real);
SIGNAL
listChg(tIntegerSet);
SIGNAL
send(tPacket,Duration),
receive(tPacket);
SIGNAL
resendDone(Integer,Integer),
receivedAgain(Integer);
SIGNAL
initMsg(tPacket),
acceptMsg(tPacket);
SIGNAL
pos.req,
pos.ind(tPosition),
VehData.ind(tPosition,Real);

C.2 Procedures

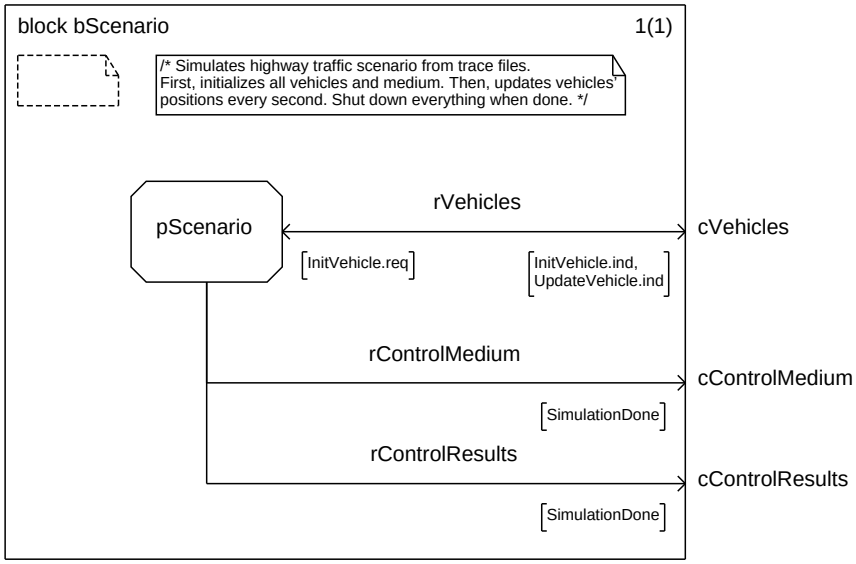


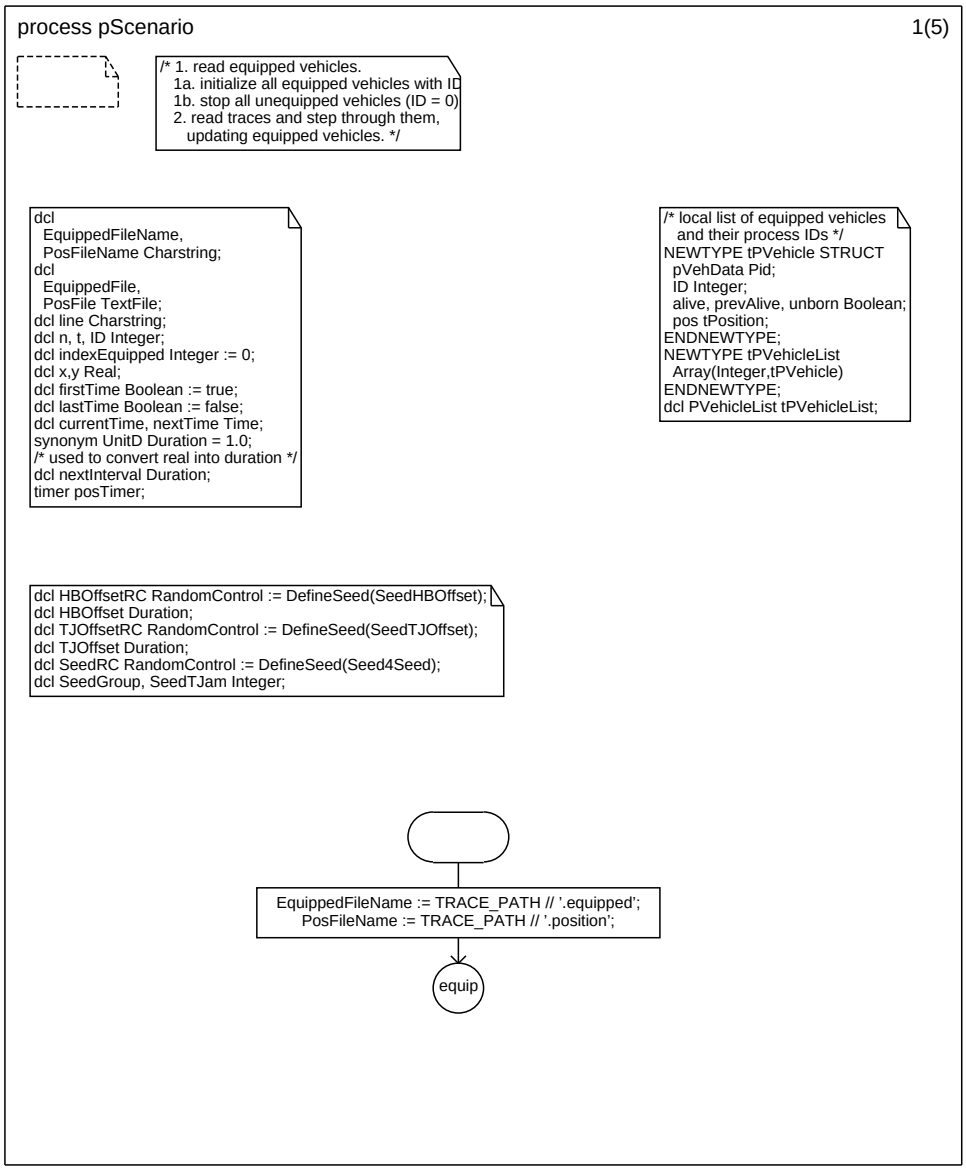

use MANETSimulatorDefinitions;
package MANETSimulatorProcedures
2(2)
procedure ParsePosLine
fpar in str Charstring, in/out t Integer, in/out n Integer, in/out x Real, in/out y Real {
dcl anfang, ende, i Integer;
/* get first integer */
anfang := 1;
for (i := anfang, (str(i) >= '0') AND (str(i) <= '9'), i+1);
ende := i−1;
t := call Str2Int(substring(str,anfang,ende−anfang+1));
/* get second integer */
anfang := ende+2;
for (i := anfang, (str(i) >= '0') AND (str(i) <= '9'), i+1);
ende := i−1;
n := call Str2Int(substring(str,anfang,ende−anfang+1));
/* get third real */
anfang := ende+2;
for (i := anfang, str(i) /= ' ', i+1);
ende := i−1;
x := call Str2Real(substring(str,anfang,ende−anfang+1));
/* get third real */
anfang := ende+2;
for (i := anfang, str(i) /= ' ', i+1);
ende := i−1;
y := call Str2Real(substring(str,anfang,ende−anfang+1));
}
procedure sqrt fpar x Real returns Real external;
/*procedure fabs fpar x Real returns Real external;*/
procedure calcDistance fpar in x1,x2,y1,y2 Real returns Real {
/* calculate distance between two positions */
dcl ergebnis Real;
ergebnis := (x1−x2)*(x1−x2) + (y1−y2)*(y1−y2);
ergebnis := call sqrt(ergebnis);
return(ergebnis); }
PR
mathimport
procedure signum fpar z Real returns Real {
if (z < 0) return −1.0;
if (z > 0) return 1.0;
return 0.0; }
procedure without fpar N1, N2 tIntegerSet returns tIntegerSet {
dcl ergebnis, schnitt tIntegerSet;
ergebnis := N1;
schnitt := N1 AND N2;
for(dcl i Integer := length(schnitt), i>0, i−1) {
ergebnis := del(take(schnitt,i),ergebnis); }
return ergebnis; }

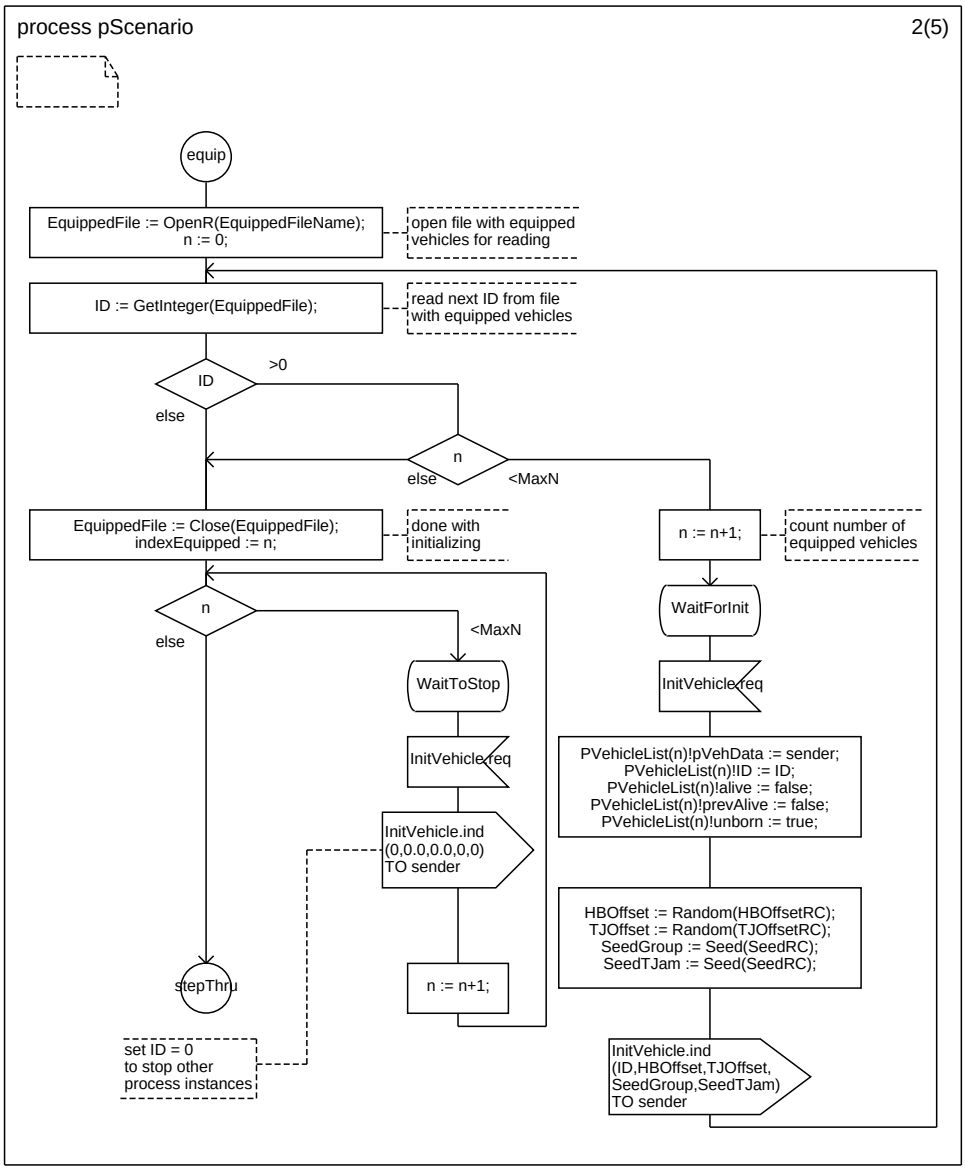
C.3 Simulator

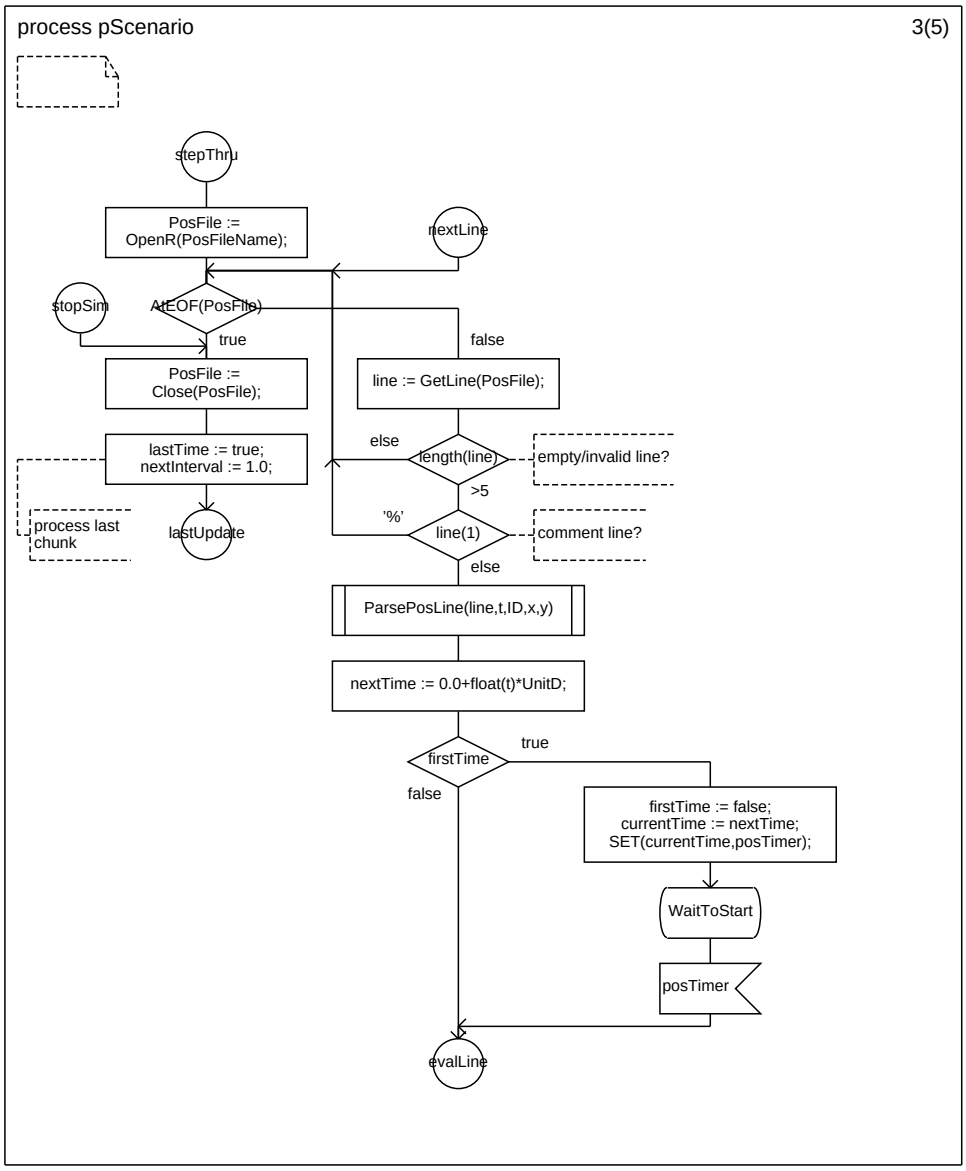


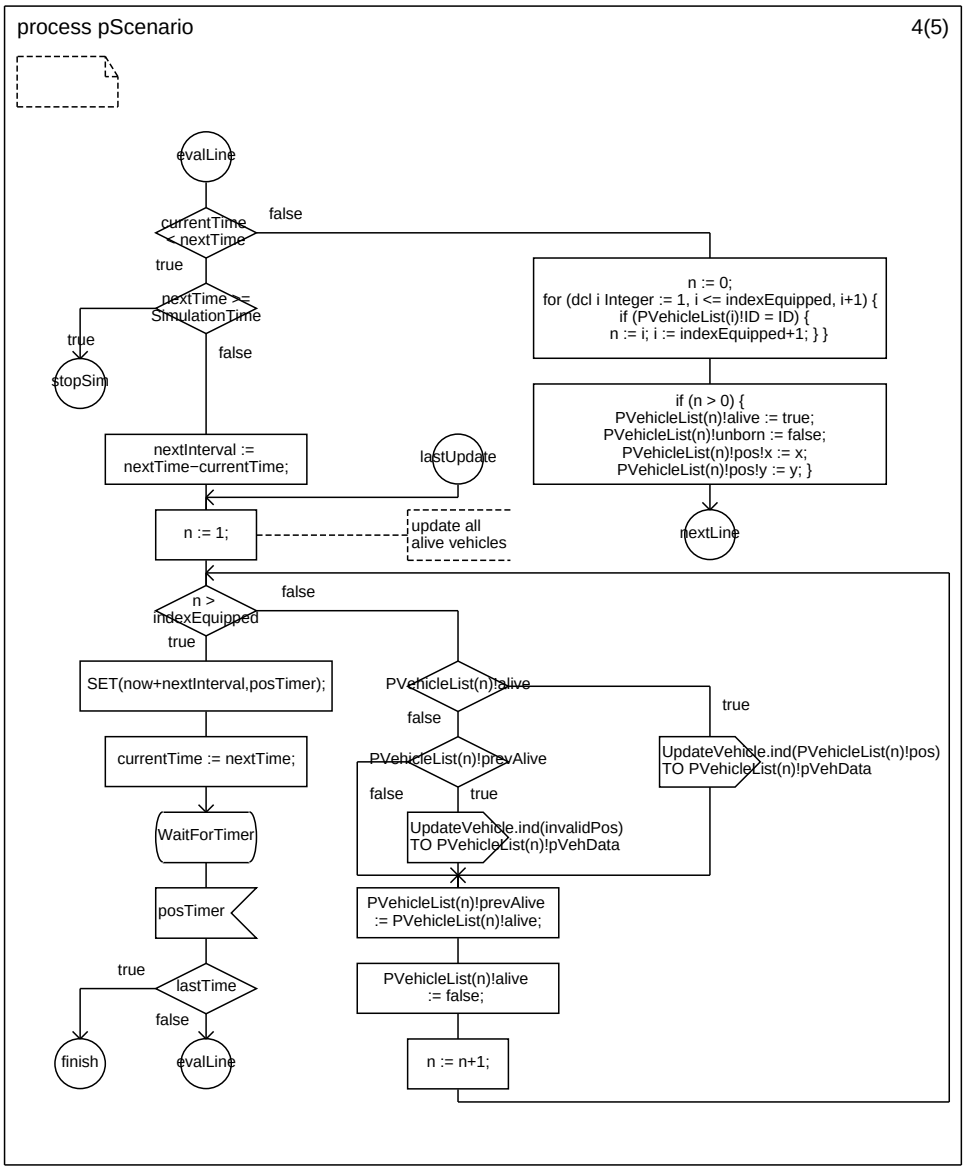
C.3.1 Scenario

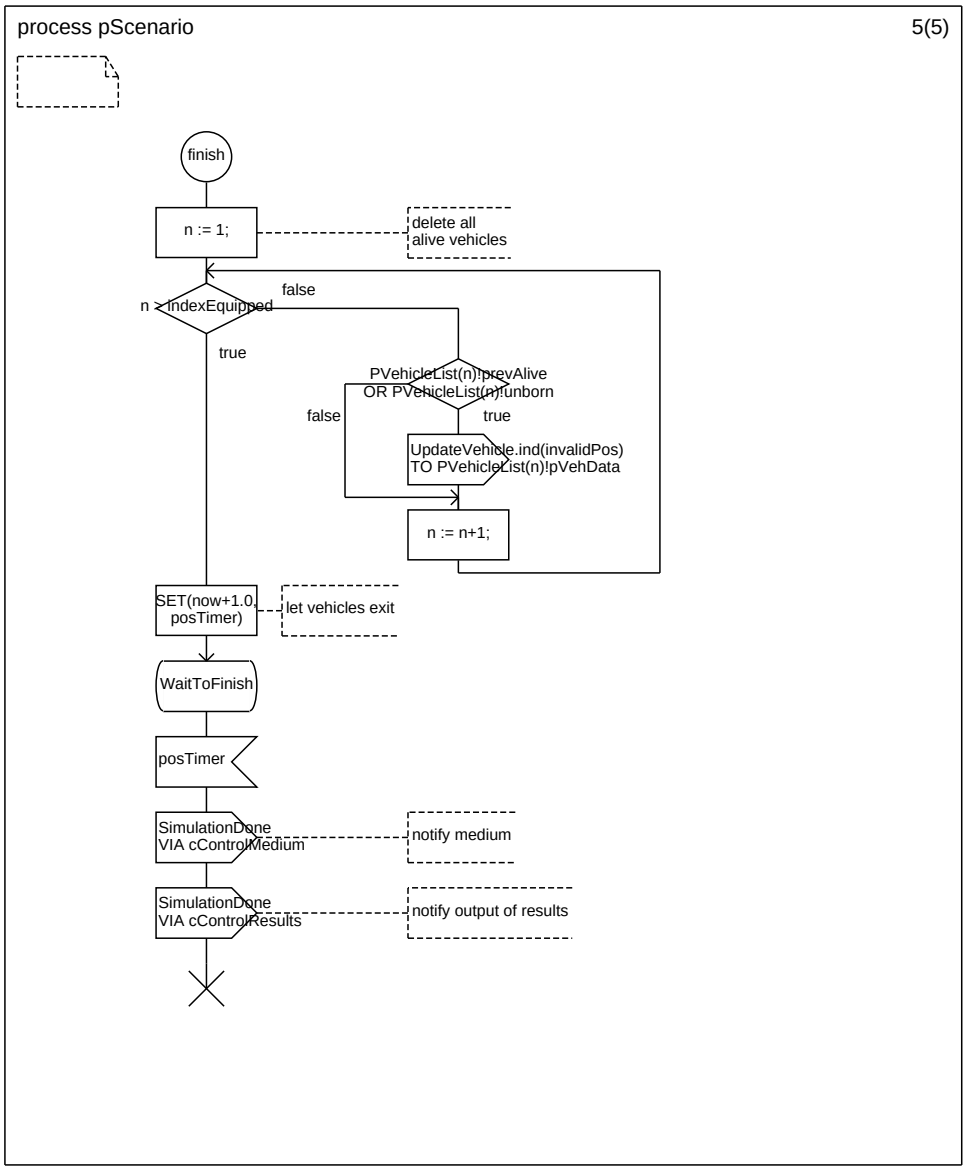



process pScenario
1(5)
/* 1. read equipped vehicles.
1a. initialize all equipped vehicles with ID
1b. stop all unequipped vehicles (ID = 0)
2. read traces and step through them,
updating equipped vehicles. */
dcl
EquippedFileName,
PosFileName Charstring;
dcl
EquippedFile,
PosFile TextFile;
dcl line Charstring;
dcl n, t, ID Integer;
dcl indexEquipped Integer := 0;
dcl x,y Real;
dcl firstTime Boolean := true;
dcl lastTime Boolean := false;
dcl currentTime, nextTime Time;
synonym UnitD Duration = 1.0;
/* used to convert real into duration */
dcl nextInterval Duration;
timer posTimer;
/* local list of equipped vehicles
and their process IDs */
NEWTYPE tPVehicle STRUCT
pVehData Pid;
ID Integer;
alive, prevAlive, unborn Boolean;
pos tPosition;
ENDNEWTYPE;
NEWTYPE tPVehicleList
Array(Integer,tPVehicle)
ENDNEWTYPE;
dcl PVehicleList tPVehicleList;
dcl HBOffsetRC RandomControl := DefineSeed(SeedHBOffset);
dcl HBOffset Duration;
dcl TJOffsetRC RandomControl := DefineSeed(SeedTJOffset);
dcl TJOffset Duration;
dcl SeedRC RandomControl := DefineSeed(Seed4Seed);
dcl SeedGroup, SeedTJam Integer;
EquippedFileName := TRACE_PATH // '.equipped';
PosFileName := TRACE_PATH // '.position';
equip

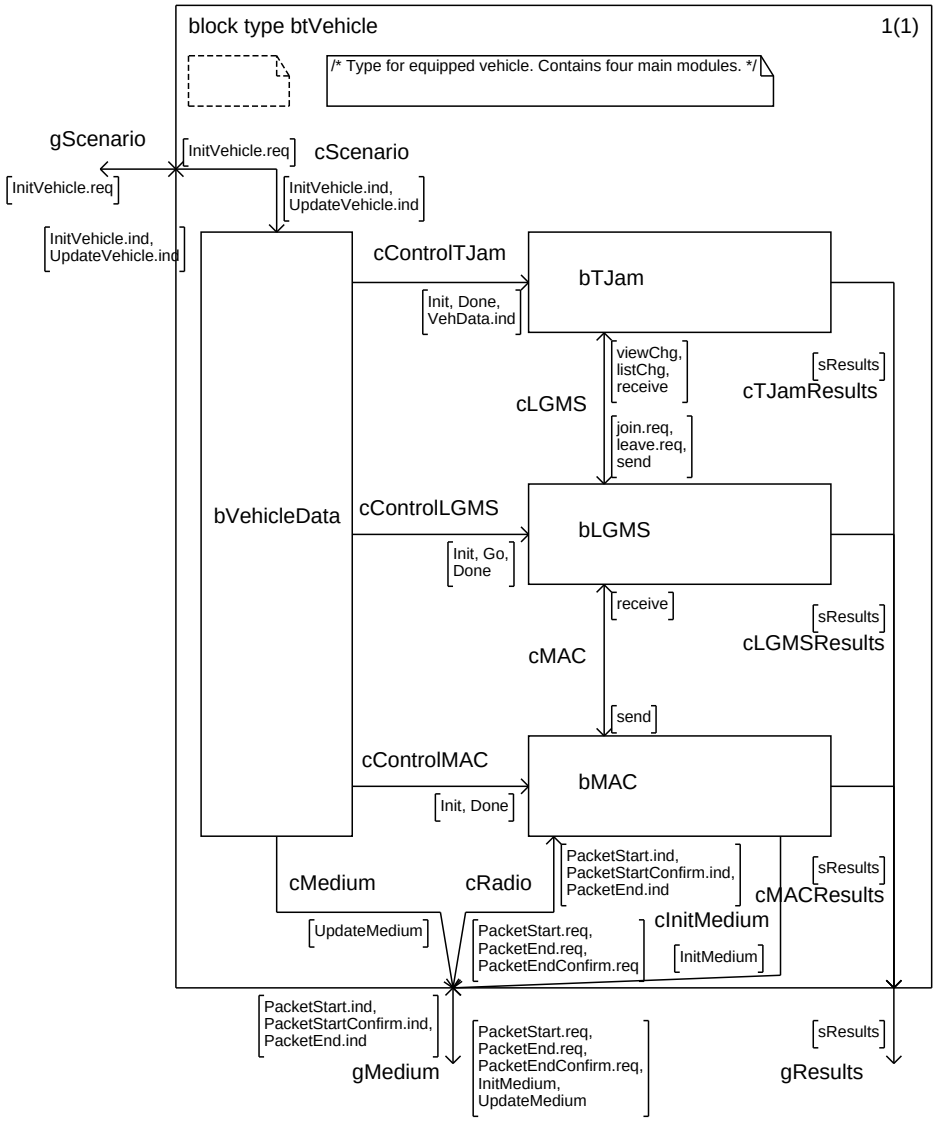

process pScenario
2(5)
equip
EquippedFile := OpenR(EquippedFileName);
n := 0;
open file with equipped vehicles for reading
ID := GetInteger(EquippedFile);
read next ID from file with equipped vehicles
ID
>0
else
n
else
<MaxN
EquippedFile := Close(EquippedFile);
indexEquipped := n;
done with initializing
n := n+1;
count number of equipped vehicles
n
else
<MaxN
WaitForInit
WaitToStop
InitVehicle.req
InitVehicle.req
PVehicleList(n)!pVehData := sender;
PVehicleList(n)!ID := ID;
PVehicleList(n)!alive := false;
PVehicleList(n)!prevAlive := false;
PVehicleList(n)!unborn := true;
InitVehicle.ind
(0,0.0,0.0,0,0)
TO sender
HBOffset := Random(HBOffsetRC);
TJOffset := Random(TJOffsetRC);
SeedGroup := Seed(SeedRC);
SeedTJam := Seed(SeedRC);
stepThru
n := n+1;
set ID = 0
to stop other
process instances
InitVehicle.ind
(ID,HBOffset,TJOffset,
SeedGroup,SeedTJam)
TO sender

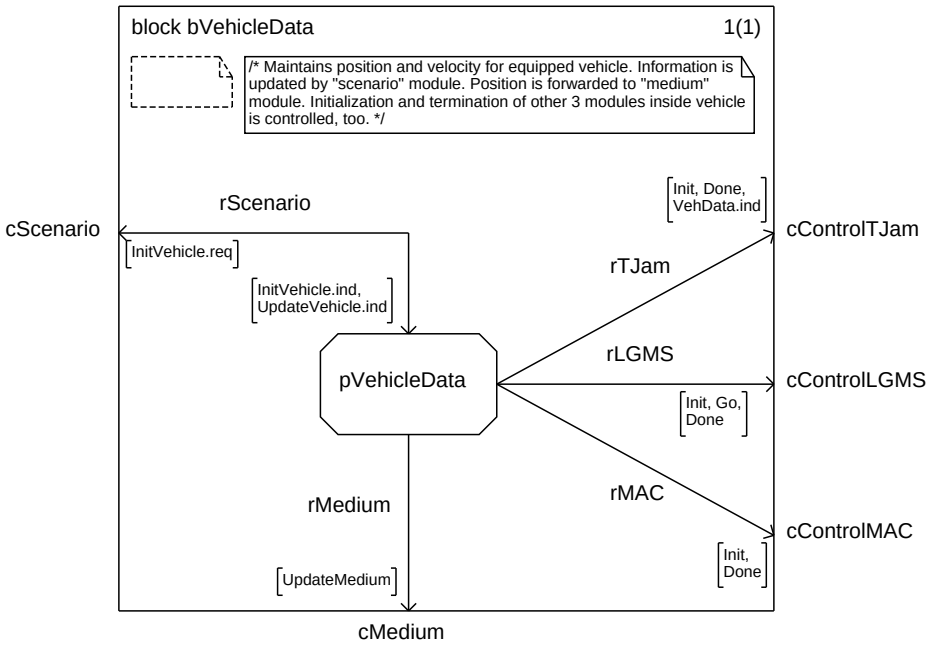

process pScenario
3(5)
stepThru
PosFile :=
OpenR(PosFileName);
nextLine
stopSim
AtEOF(PosFile)
true
false
PosFile :=
Close(PosFile);
line := GetLine(PosFile);
lastTime := true;
nextInterval := 1.0;
else
length(line)
empty/invalid line?
>5
process last
chunk
lastUpdate
'%'
line(1)
comment line?
else
ParsePosLine(line,t,ID,x,y)
nextTime := 0.0+float(t)*UnitD;
firstTime
true
false
firstTime := false;
currentTime := nextTime;
SET(currentTime,posTimer);
WaitToStart
posTimer
evalLine

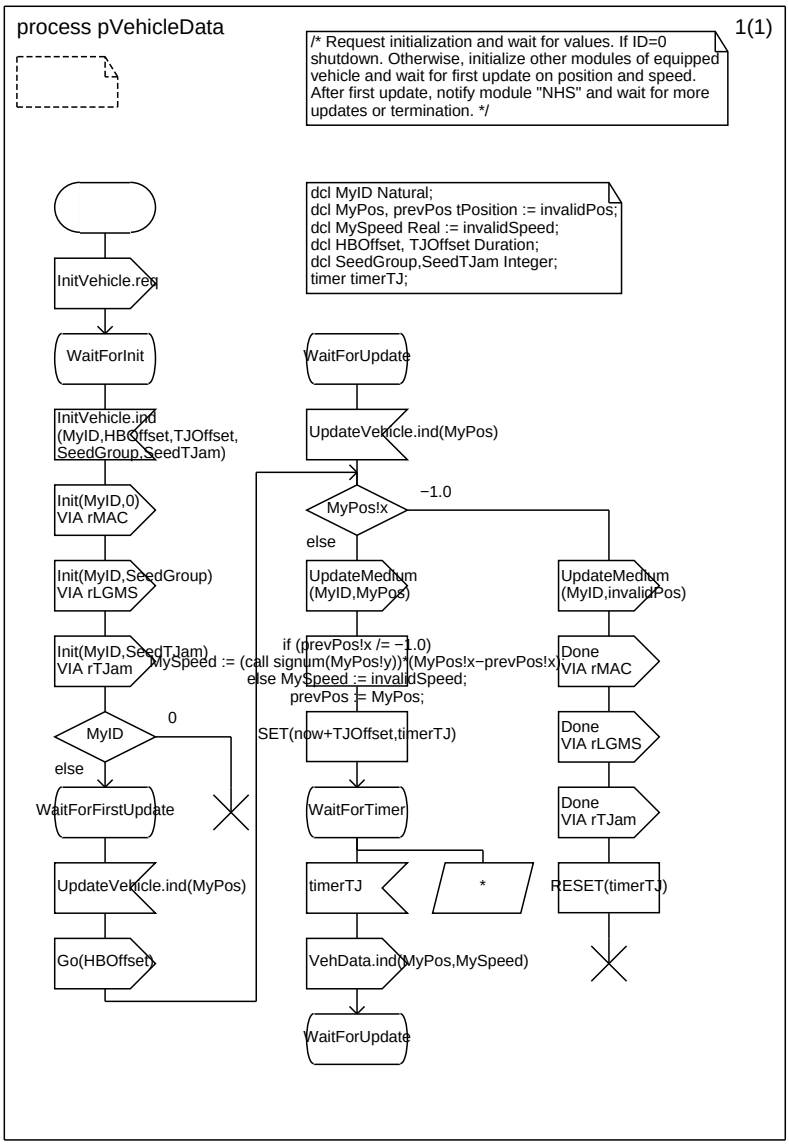

process pScenario
4(5)
evalLine
currentTime < nextTime
false
true
nextTime >= SimulationTime
true
stopSim
false
nextInterval := nextTime−currentTime;
lastUpdate
n := 1;
update all alive vehicles
n := 0;
for (dcl i Integer := 1, i <= indexEquipped, i+1) {
if (PVehicleList(i)!ID = ID) {
n := i; i := indexEquipped+1; } }
if (n > 0) {
PVehicleList(n)!alive := true;
PVehicleList(n)!unborn := false;
PVehicleList(n)!pos!x := x;
PVehicleList(n)!pos!y := y; }
nextLine
n > indexEquipped
false
true
SET(now+nextInterval,posTimer);
currentTime := nextTime;
WaitForTimer
posTimer
true
lastTime
false
finish
evalLine
PVehicleList(n)!alive
false
true
PVehicleList(n)!prevAlive
false
true
UpdateVehicle.ind(invalidPos) TO PVehicleList(n)!pVehData
UpdateVehicle.ind(PVehicleList(n)!pos) TO PVehicleList(n)!pVehData
PVehicleList(n)!prevAlive := PVehicleList(n)!alive;
PVehicleList(n)!alive := false;
n := n+1;

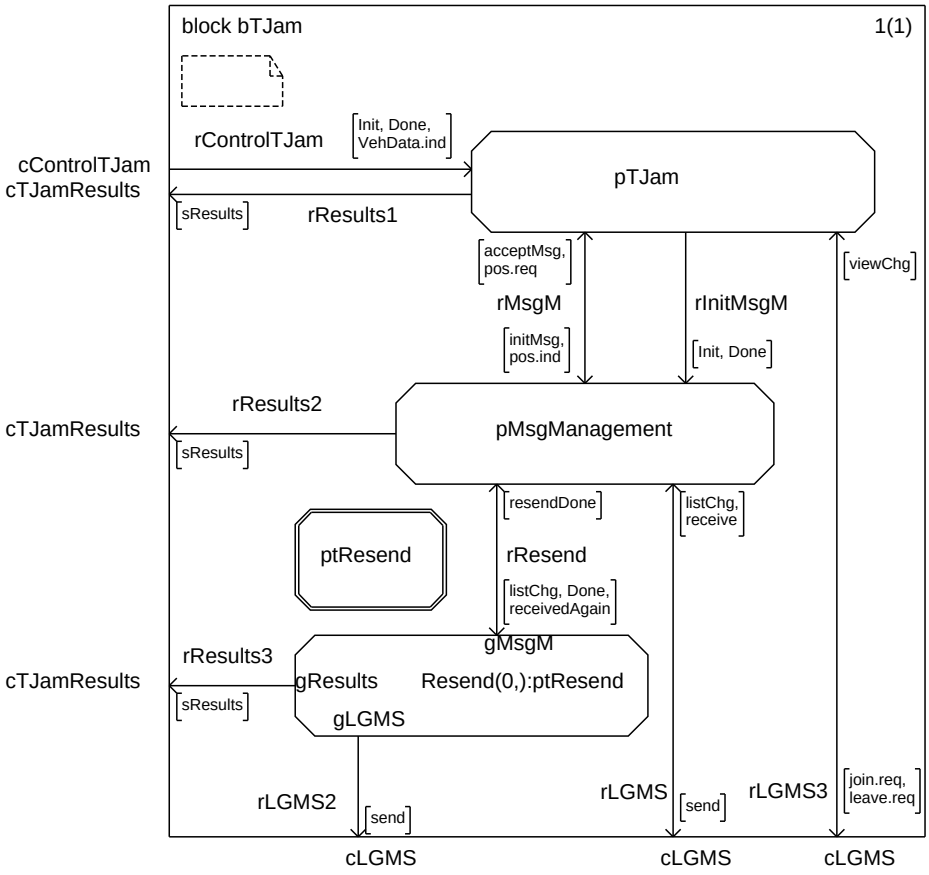

process pScenario
5(5)
finish
n := 1;
delete all
alive vehicles
n < IndexEquipped
false
true
PVehicleList(n)!prevAlive
OR PVehicleList(n)!unborn
false
true
UpdateVehicle.ind(invalidPos)
TO PVehicleList(n)!pVehData
n := n+1;
SET(now+1.0,
posTimer)
let vehicles exit
WaitToFinish
posTimer
SimulationDone
VIA cControlMedium
notify medium
SimulationDone
VIA cControlResults
notify output of results

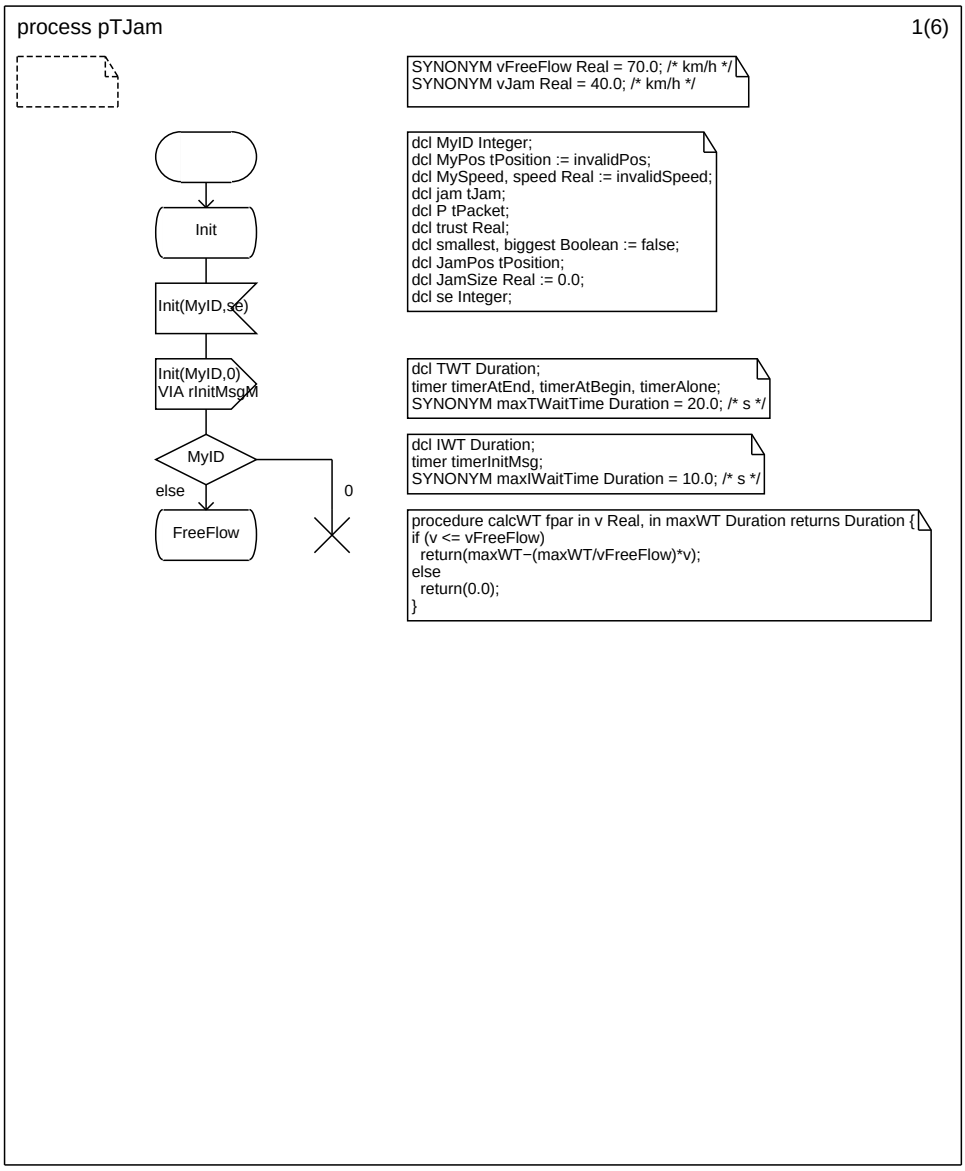
C.3.2 Equipped Vehicle

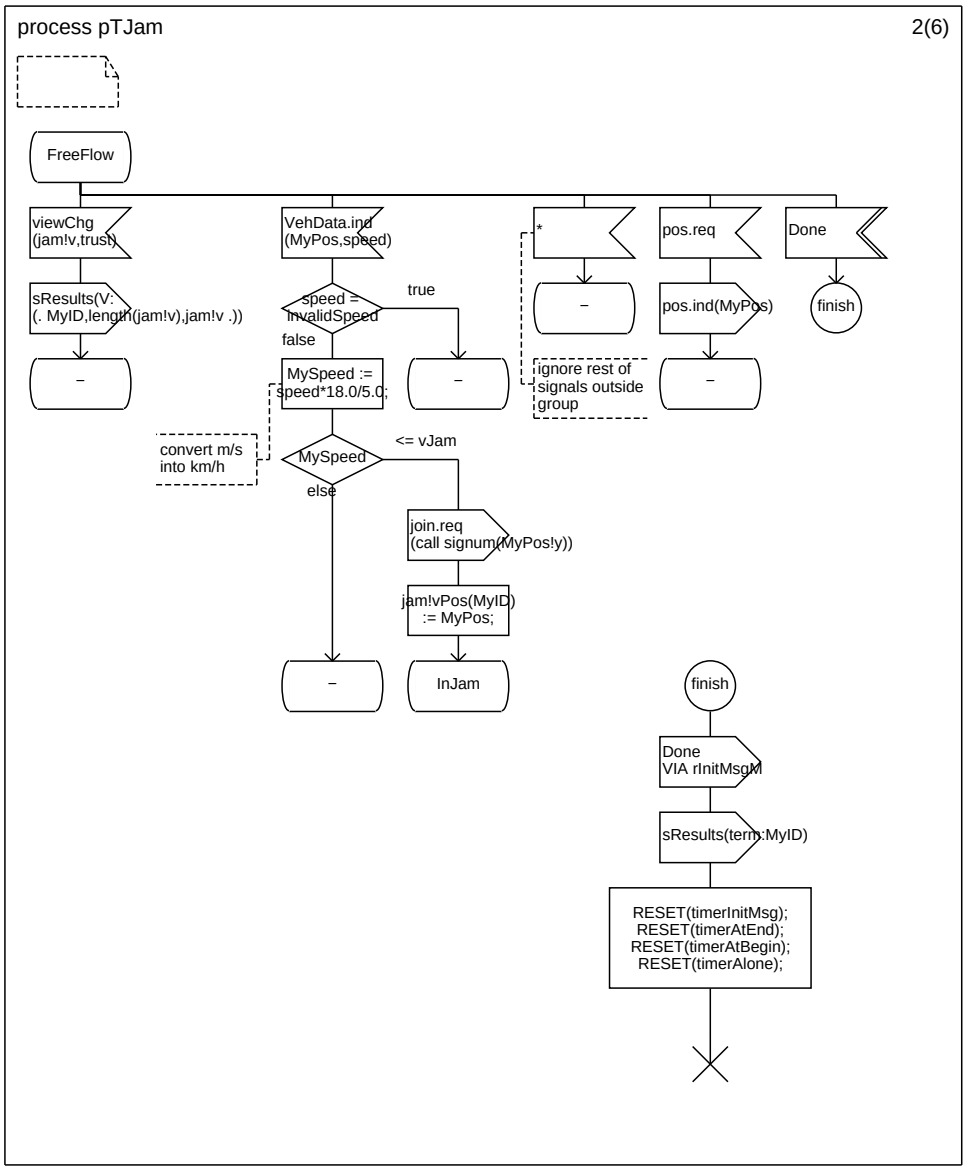


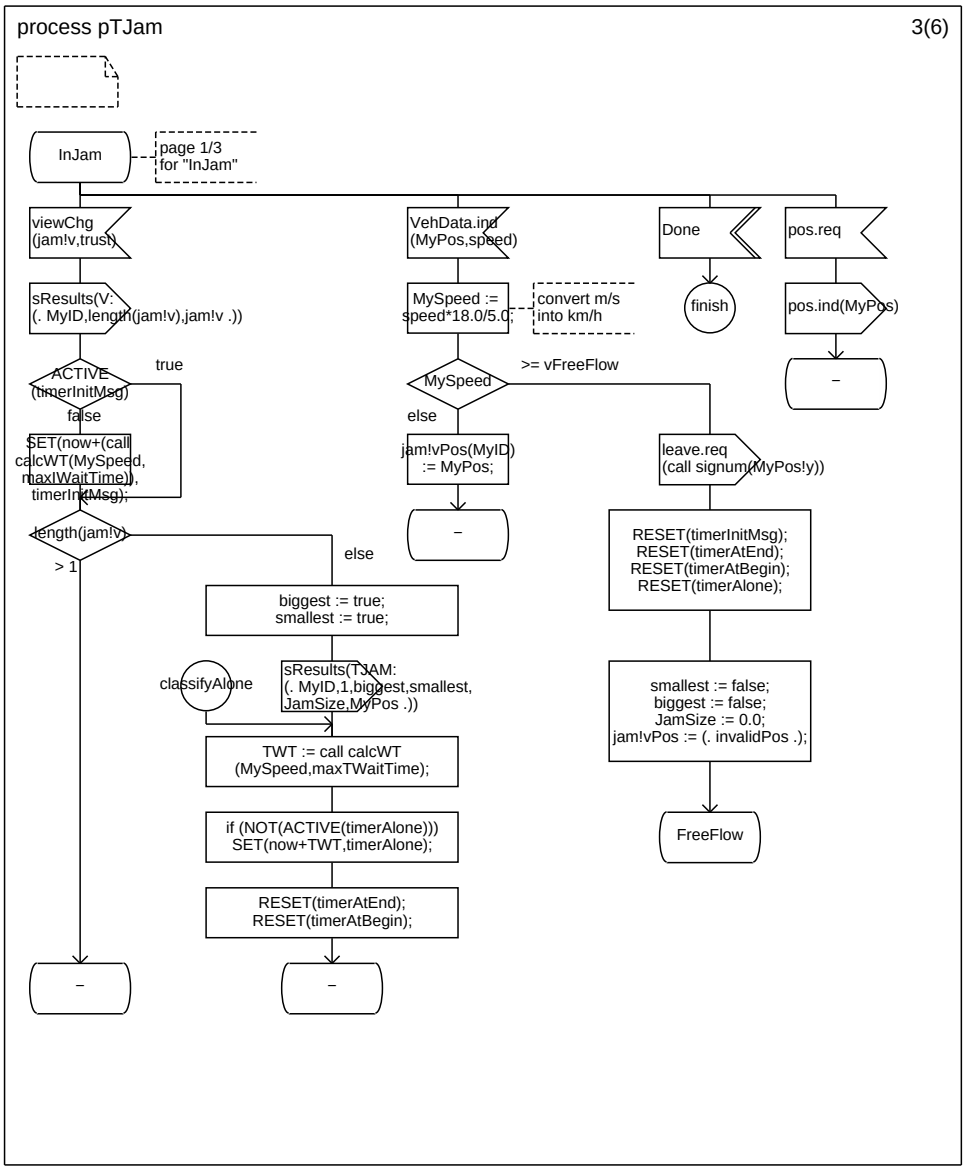

block bVehicleData
1(1)
/* Maintains position and velocity for equipped vehicle. Information is updated by "scenario" module. Position is forwarded to "medium" module. Initialization and termination of other 3 modules inside vehicle is controlled, too. */
cScenario
rScenario
InitVehicle.req
InitVehicle.ind,
UpdateVehicle.ind
pVehicleData
rTJam
Init, Done,
VehData.ind
cControlTJam
rLGMS
Init, Go,
Done
cControlLGMS
rMAC
Init,
Done
cControlMAC
rMedium
UpdateMedium
cMedium

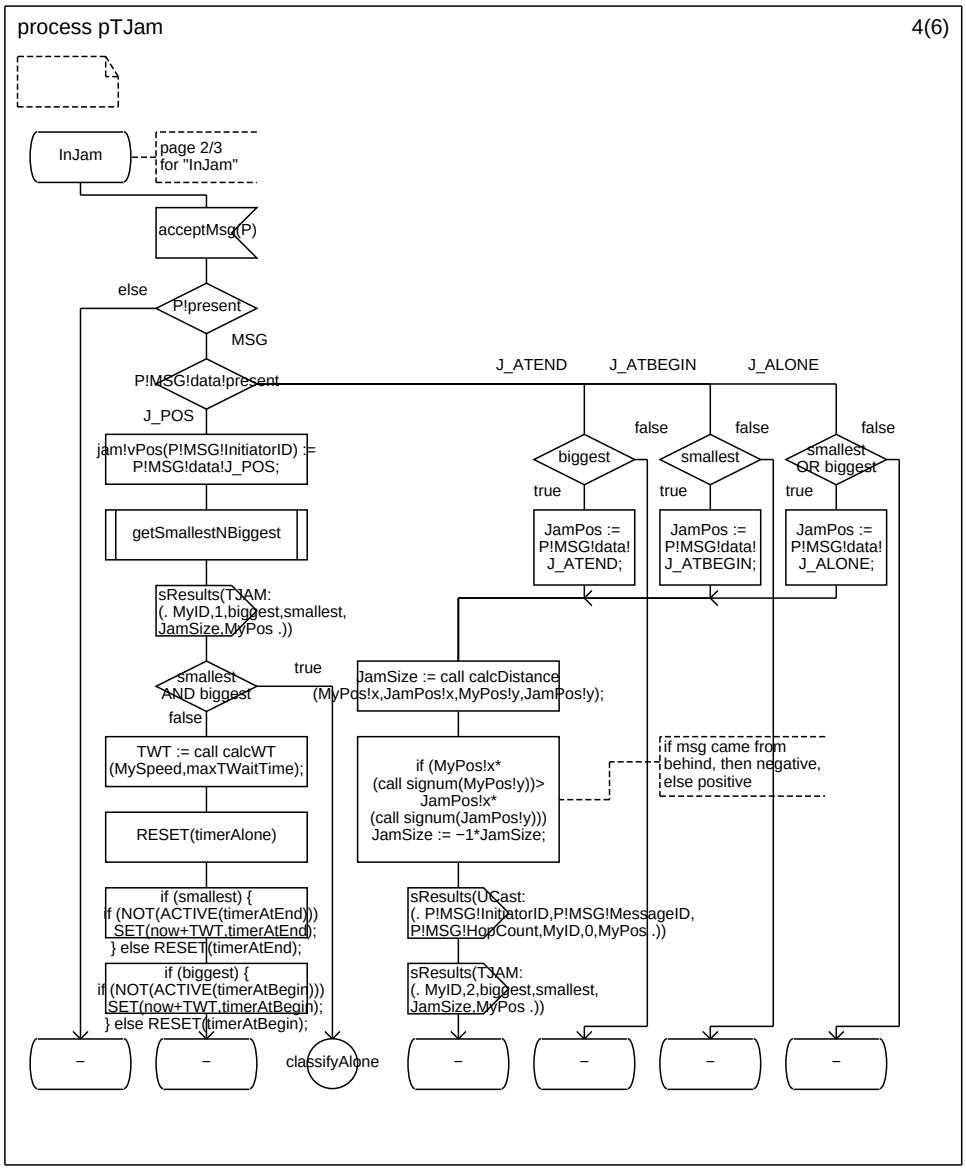

process pVehicleData
1(1)
/* Request initialization and wait for values. If ID=0 shutdown. Otherwise, initialize other modules of equipped vehicle and wait for first update on position and speed. After first update, notify module "NHS" and wait for more updates or termination. */
dcl MyID Natural;
dcl MyPos, prevPos tPosition := invalidPos;
dcl MySpeed Real := invalidSpeed;
dcl HBOffset, TJOffset Duration;
dcl SeedGroup,SeedTJam Integer;
timer timerTJ;
InitVehicle.req
WaitForInit
InitVehicle.ind (MyID,HBOffset,TJOffset, SeedGroup,SeedTJam)
Init(MyID,0) VIA rMAC
Init(MyID,SeedGroup) VIA rLGMS
Init(MyID,SeedTJam) VIA rTJam
MyID
0
else
WaitForFirstUpdate
UpdateVehicle.ind(MyPos)
Go(HBOffset)
WaitForUpdate
UpdateVehicle.ind(MyPos)
MyPos!x
−1.0
else
UpdateMedium (MyID,MyPos)
if (prevPos!x /= −1.0)
MySpeed := (call signum(MyPos!y))*(MyPos!x−prevPos!x)/
else MySpeed := invalidSpeed;
prevPos := MyPos;
SET(now+TJOffset,timerTJ)
WaitForTimer
timerTJ
*
VehData.ind(MyPos,MySpeed)
WaitForUpdate
UpdateMedium (MyID,invalidPos)
Done VIA rMAC
Done VIA rLGMS
Done VIA rTJam
RESET(timerTJ)

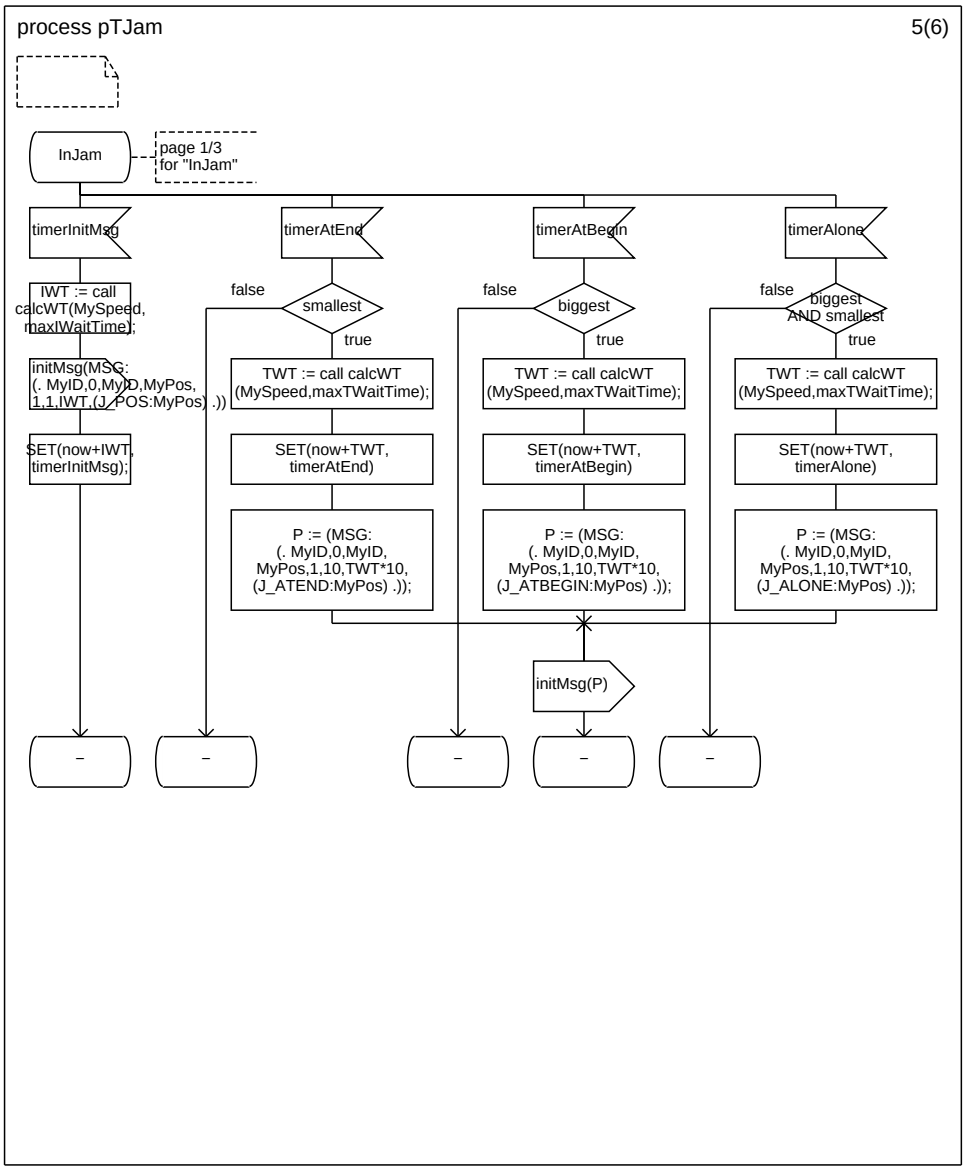

block bTJam
1(1)
rControlTJam
Init, Done,
VehData.ind
cControlTJam
cTJamResults
pTJam
sResults
rResults1
acceptMsg,
pos.req
viewChg
rMsgM
rInitMsgM
initMsg,
pos.ind
Init, Done
rResults2
cTJamResults
pMsgManagement
sResults
resendDone
listChg,
receive
ptResend
rResend
listChg, Done,
receivedAgain
gMsgM
rResults3
cTJamResults
gResults
Resend(0,):ptResend
sResults
gLGMS
rLGMS2
send
rLGMS
send
rLGMS3
join.req,
leave.req
cLGMS
cLGMS
cLGMS


process pTJam
1(6)
SYNONYM vFreeFlow Real = 70.0; /* km/h */
SYNONYM vJam Real = 40.0; /* km/h */
Init
dcl MyID Integer;
dcl MyPos tPosition := invalidPos;
dcl MySpeed, speed Real := invalidSpeed;
dcl jam tJam;
dcl P tPacket;
dcl trust Real;
dcl smallest, biggest Boolean := false;
dcl JamPos tPosition;
dcl JamSize Real := 0.0;
dcl se Integer;
Init(MyID,se)
Init(MyID,0)
VIA rInitMsgM
dcl TWT Duration;
timer timerAtEnd, timerAtBegin, timerAlone;
SYNONYM maxTWaitTime Duration = 20.0; /* s */
MyID
dcl IWT Duration;
timer timerInitMsg;
SYNONYM maxIWaitTime Duration = 10.0; /* s */
else
0
FreeFlow
procedure calcWT fpar in v Real, in maxWT Duration returns Duration {
if (v <= vFreeFlow)
return(maxWT−(maxWT/vFreeFlow)*v);
else
return(0.0);
}


process pTJam
2(6)
FreeFlow
viewChg
(jam!v,trust)
VehData.ind
(MyPos,speed)
*
pos.req
Done
sResults(V:
(. MyID,length(jam!v),jam!v .))
speed =
InvalidSpeed
true
false
pos.ind(MyPos)
finish
MySpeed :=
speed*18.0/5.0;
ignore rest of
signals outside
group
convert m/s
into km/h
MySpeed
<= vJam
else
join.req
(call signum(MyPos!y))
jam!vPos(MyID)
:= MyPos;
InJam
finish
Done
VIA rInitMsgM
sResults(term:MyID)
RESET(timerInitMsg);
RESET(timerAtEnd);
RESET(timerAtBegin);
RESET(timerAlone);

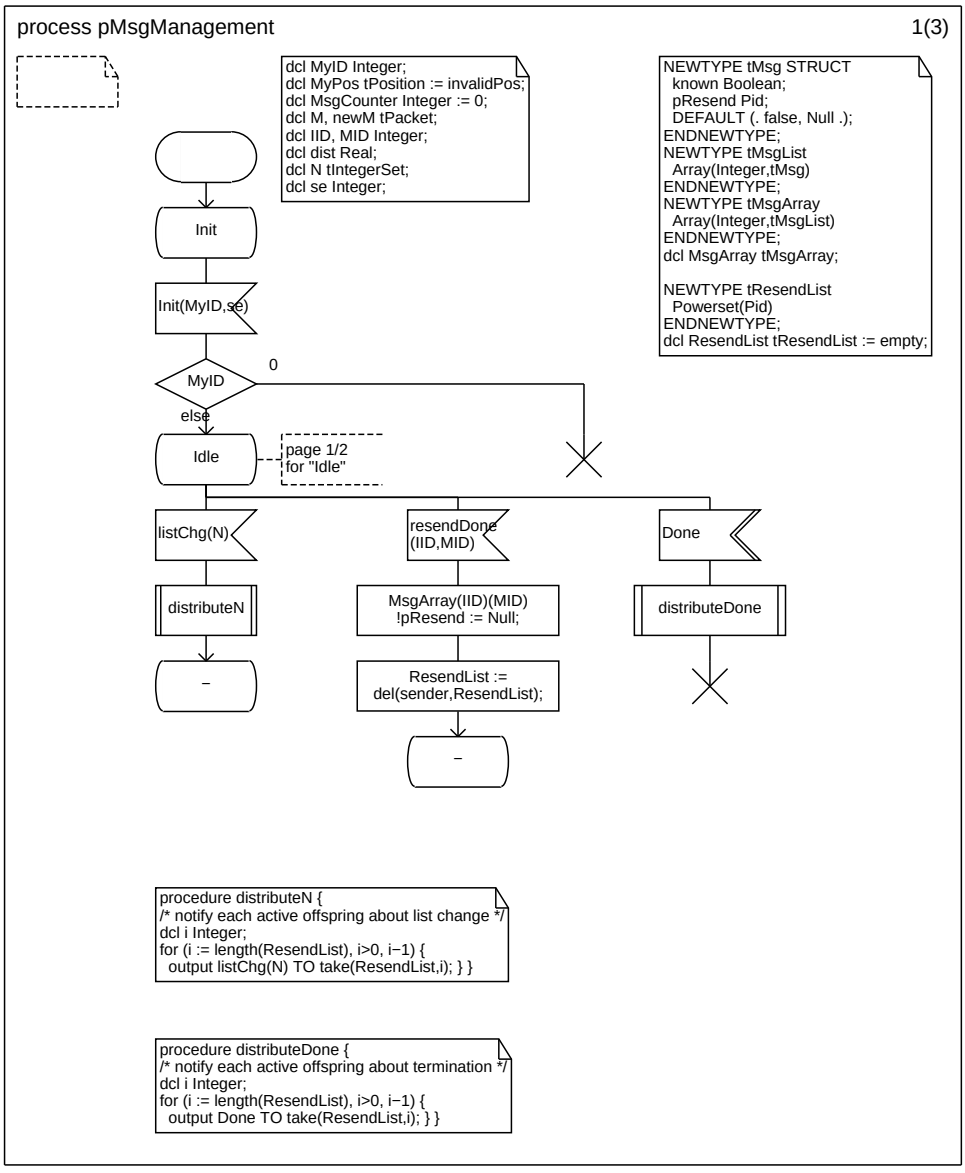

process pTJam
3(6)
InJam
page 1/3
for "InJam"
viewChg
(jam!v,trust)
VehData.ind
(MyPos,speed)
Done
pos.req
sResults(V:
(. MyID,length(jam!v),jam!v .))
MySpeed :=
speed*18.0/5.0;
convert m/s
into km/h
finish
pos.ind(MyPos)
ACTIVE
(timerInitMsg)
true
false
MySpeed
>= vFreeFlow
else
SET(now+(call
calcWT(MySpeed,
maxTWaitTime)),
timerInitMsg);
jam!vPos(MyID)
:= MyPos;
leave.req
(call signum(MyPos!y))
length(jam!v)
> 1
else
RESET(timerInitMsg);
RESET(timerAtEnd);
RESET(timerAtBegin);
RESET(timerAlone);
biggest := true;
smallest := true;
classifyAlone
sResults(TJAM:
(. MyID,1,biggest,smallest,
JamSize,MyPos .))
smallest := false;
biggest := false;
JamSize := 0.0;
jam!vPos := (. invalidPos .);
TWT := call calcWT
(MySpeed,maxTWaitTime);
FreeFlow
if (NOT(ACTIVE(timerAlone)))
SET(now+TWT,timerAlone);
RESET(timerAtEnd);
RESET(timerAtBegin);

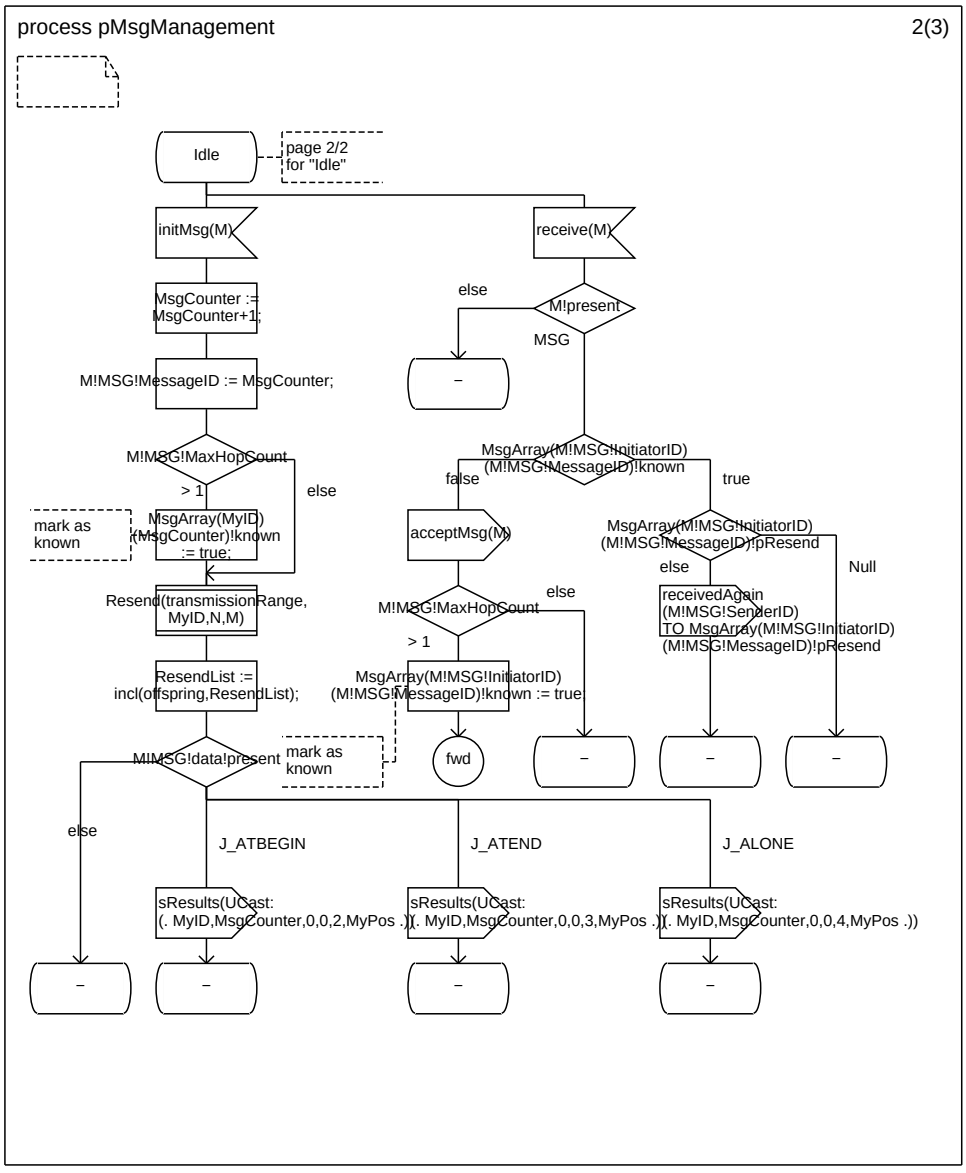

process pTJam
4(6)
InJam
page 2/3
for "InJam"
acceptMsg(P)
else
P!present
MSG
P!MSG!data!present
J_ATEND
J_ATBEGIN
J_ALONE
J_POS
jam!vPos(P!MSG!InitiatorID) :=
P!MSG!data!J_POS;
biggest
smallest
smallest
OR biggest
false
false
false
true
true
true
JamPos :=
P!MSG!data!
J_ATEND;
JamPos :=
P!MSG!data!
J_ATBEGIN;
JamPos :=
P!MSG!data!
J_ALONE;
getSmallestNBiggest
sResults(TJAM:
(. MyID,1,biggest,smallest,
JamSize,MyPos .))
smallest
AND biggest
true
JamSize := call calcDistance
(MyPos!x,JamPos!x,MyPos!y,JamPos!y);
false
TWT := call calcWT
(MySpeed,maxTWaitTime);
if (MyPos!x*
(call signum(MyPos!y))>
JamPos!x*
(call signum(JamPos!y)))
JamSize := −1*JamSize;
if msg came from
behind, then negative,
else positive
RESET(timerAlone)
if (smallest) {
if (NOT(ACTIVE(timerAtEnd)))
SET(now+TWT,timerAtEnd);
} else RESET(timerAtEnd);
sResults(UCast:
(. P!MSG!InitiatorID,P!MSG!MessageID,
P!MSG!HopCount,MyID,0,MyPos .))
if (biggest) {
if (NOT(ACTIVE(timerAtBegin)))
SET(now+TWT,timerAtBegin);
} else RESET(timerAtBegin);
sResults(TJAM:
(. MyID,2,biggest,smallest,
JamSize,MyPos .))
–
–
classifyAlone
–
–
–
–

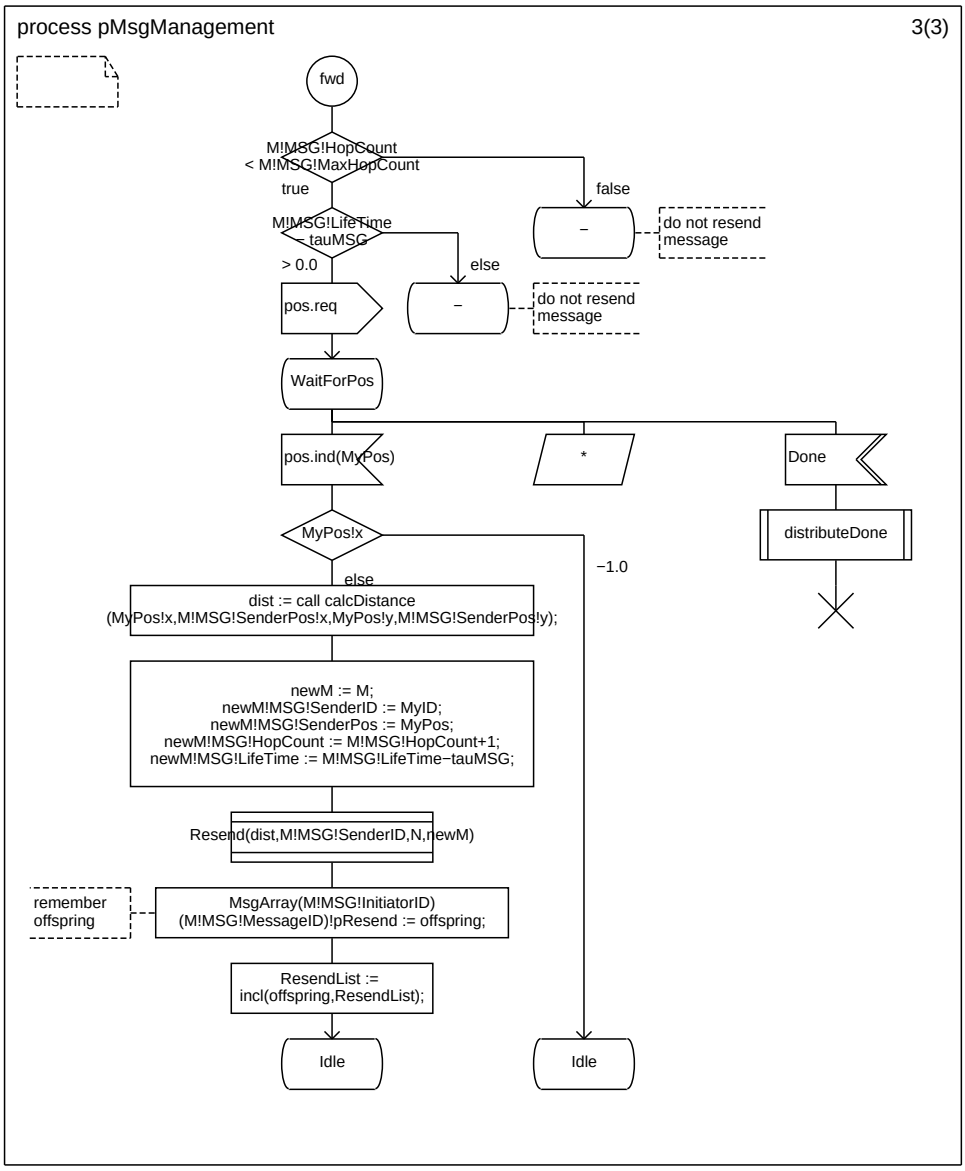

process pTJam
5(6)
InJam
page 1/3
for "InJam"
timerInitMsg
timerAtEnd
timerAtBegin
timerAlone
IWT := call calcWT(MySpeed, maxIWaitTime);
initMsg(MSG: (. MyID,0,MyID,MyPos, 1,1,IWT,(J_POS:MyPos) .))
SET(now+IWT, timerInitMsg);
false
smallest
true
TWT := call calcWT (MySpeed,maxTWaitTime);
SET(now+TWT, timerAtEnd)
P := (MSG: (. MyID,0,MyID, MyPos,1,10,TWT*10, (J_ATEND:MyPos) .));
false
biggest
true
TWT := call calcWT (MySpeed,maxTWaitTime);
SET(now+TWT, timerAtBegin)
P := (MSG: (. MyID,0,MyID, MyPos,1,10,TWT*10, (J_ATBEGIN:MyPos) .));
false
biggest AND smallest
true
TWT := call calcWT (MySpeed,maxTWaitTime);
SET(now+TWT, timerAlone)
P := (MSG: (. MyID,0,MyID, MyPos,1,10,TWT*10, (J_ALONE:MyPos) .));
initMsg(P)
–
–
–
–
–

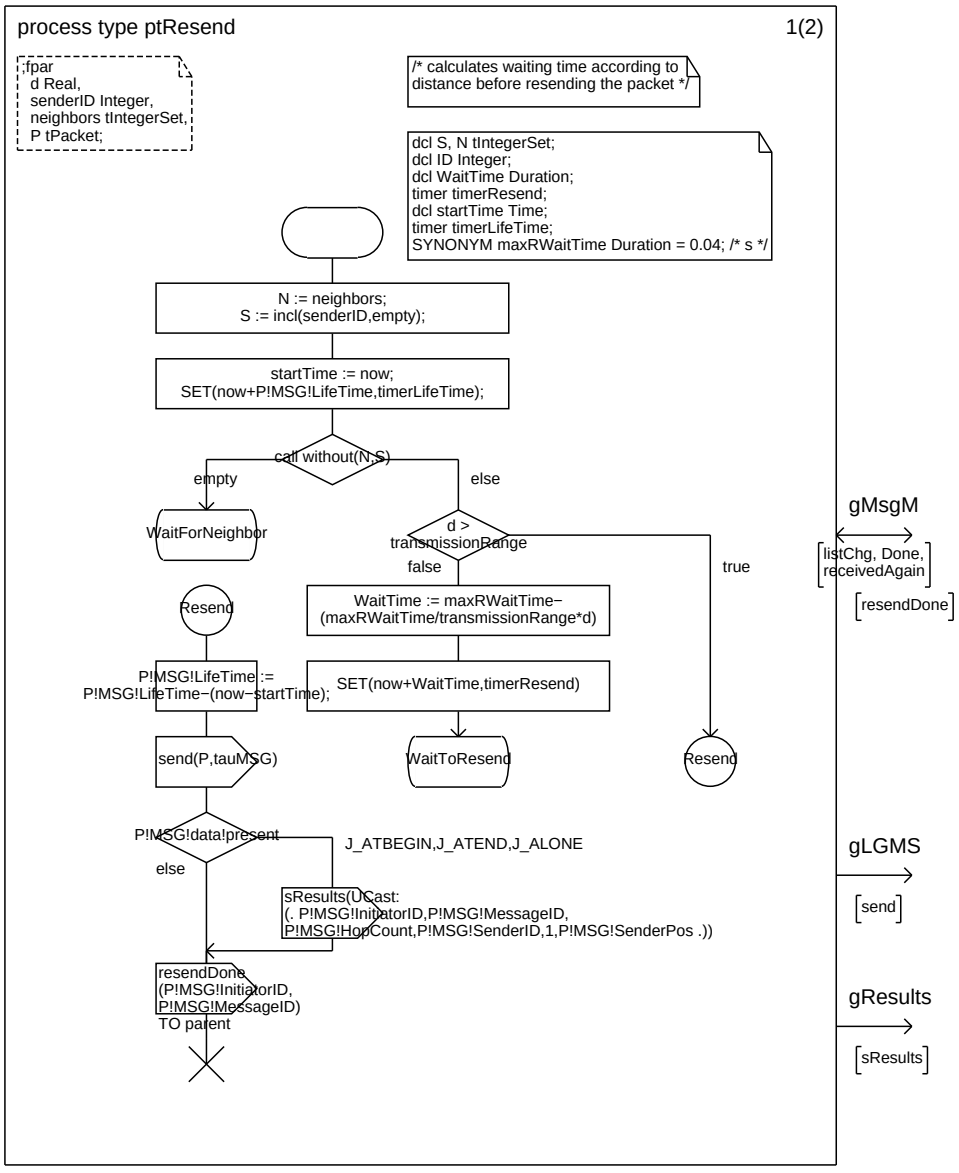
process pTJam 6(6)

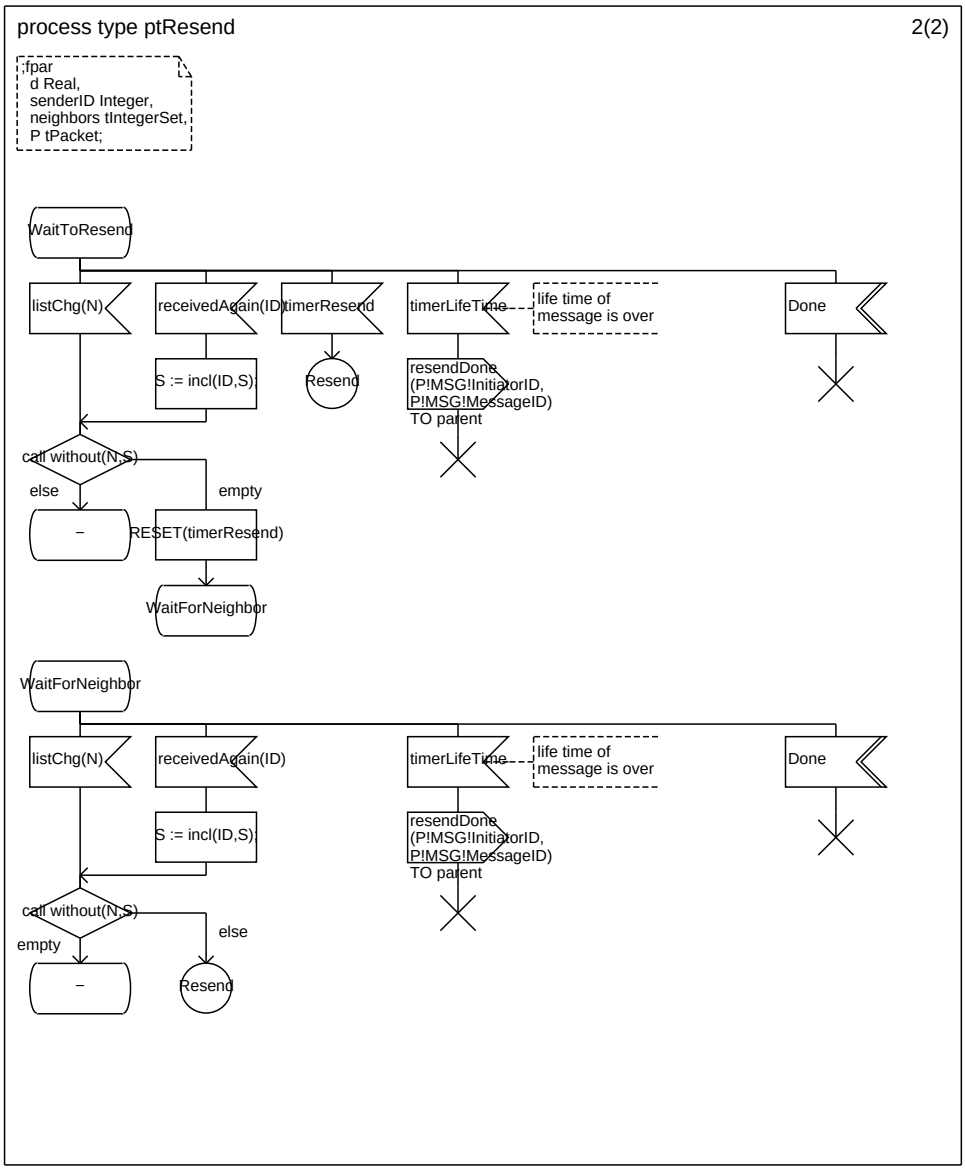
```
procedure getSmallestNBiggest {
/* go through view and determine if at smallest (END) or biggest (BEGIN) position */
dcl i, ID, sID, bID Integer := 0;
dcl sPos, bPos tPosition := invalidPos;
for (i:=length(jam!v), i>0, i−1) {
  ID := take(jam!v,i);
  if (jam!vPos(ID) /= invalidPos) {
    /* test for smallest pos */
    if (sPos = invalidPos) {
      sPos := jam!vPos(ID);
      sID := ID;
    } else {
      if (sPos!x*(call signum(sPos!y)) > jam!vPos(ID)!x*(call signum(jam!vPos(ID)!y))) {
        sPos := jam!vPos(ID);
        sID := ID; } }
    /* test for biggest pos */
    if (bPos = invalidPos) {
      bPos := jam!vPos(ID);
      bID := ID;
    } else {
      if (bPos!x*(call signum(bPos!y)) < jam!vPos(ID)!x*(call signum(jam!vPos(ID)!y))) {
        bPos := jam!vPos(ID);
        bID := ID; } } } }
if (((sID = MyID) = NOT(smallest)) OR
    ((bID = MyID) = NOT(biggest))) {
  /* something has changed! */
  smallest := (sID = MyID);
  biggest := (bID = MyID);
  output sResults(TJAM:(. MyID,1,biggest,smallest,JamSize,MyPos .)); } }
```

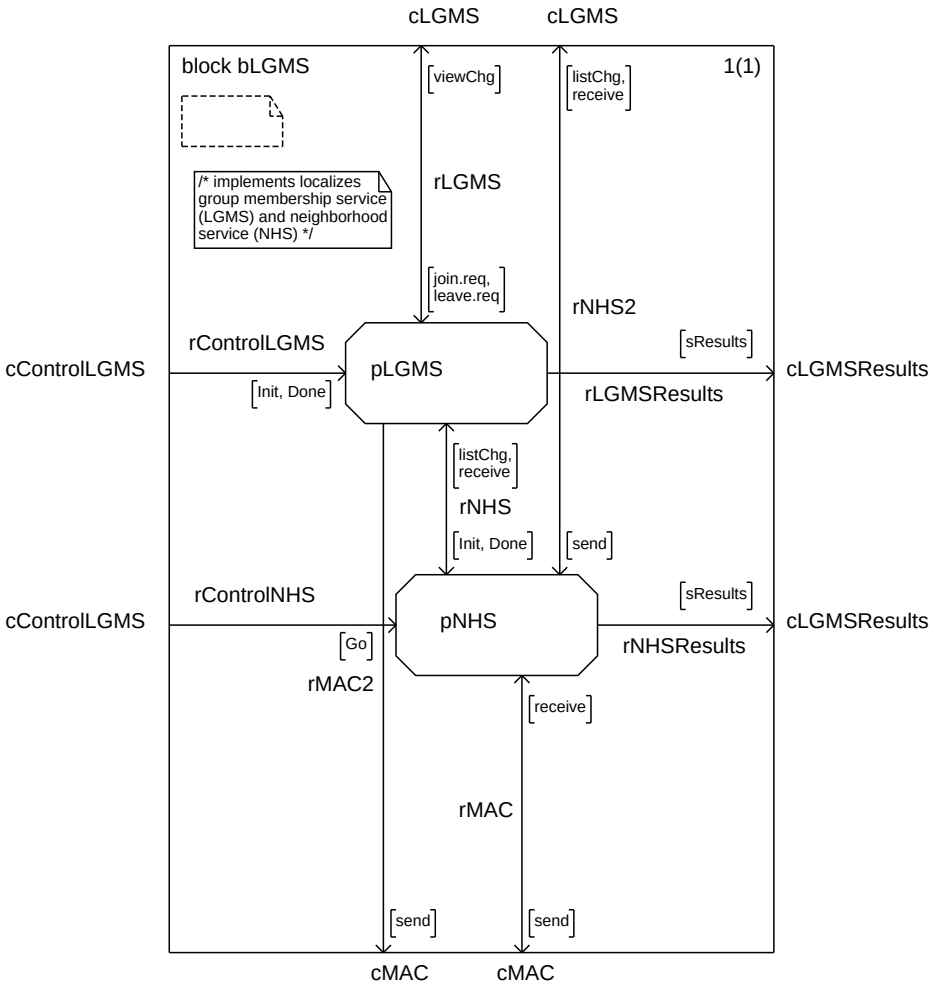

process pMsgManagement
1(3)
dcl MyID Integer;
dcl MyPos tPosition := invalidPos;
dcl MsgCounter Integer := 0;
dcl M, newM tPacket;
dcl IID, MID Integer;
dcl dist Real;
dcl N tIntegerSet;
dcl se Integer;
NEWTYPE tMsg STRUCT
known Boolean;
pResend Pid;
DEFAULT (. false, Null .);
ENDNEWTYPE;
NEWTYPE tMsgList
Array(Integer,tMsg)
ENDNEWTYPE;
NEWTYPE tMsgArray
Array(Integer,tMsgList)
ENDNEWTYPE;
dcl MsgArray tMsgArray;
NEWTYPE tResendList
Powerset(Pid)
ENDNEWTYPE;
dcl ResendList tResendList := empty;
Init
Init(MyID,se)
MyID
0
else
Idle
page 1/2
for "Idle"
listChg(N)
resendDone
(IID,MID)
Done
distributeN
MsgArray(IID)(MID)
!pResend := Null;
distributeDone
ResendList :=
del(sender,ResendList);
procedure distributeN {
/* notify each active offspring about list change */
dcl i Integer;
for (i := length(ResendList), i>0, i−1) {
output listChg(N) TO take(ResendList,i); } }
procedure distributeDone {
/* notify each active offspring about termination */
dcl i Integer;
for (i := length(ResendList), i>0, i−1) {
output Done TO take(ResendList,i); } }

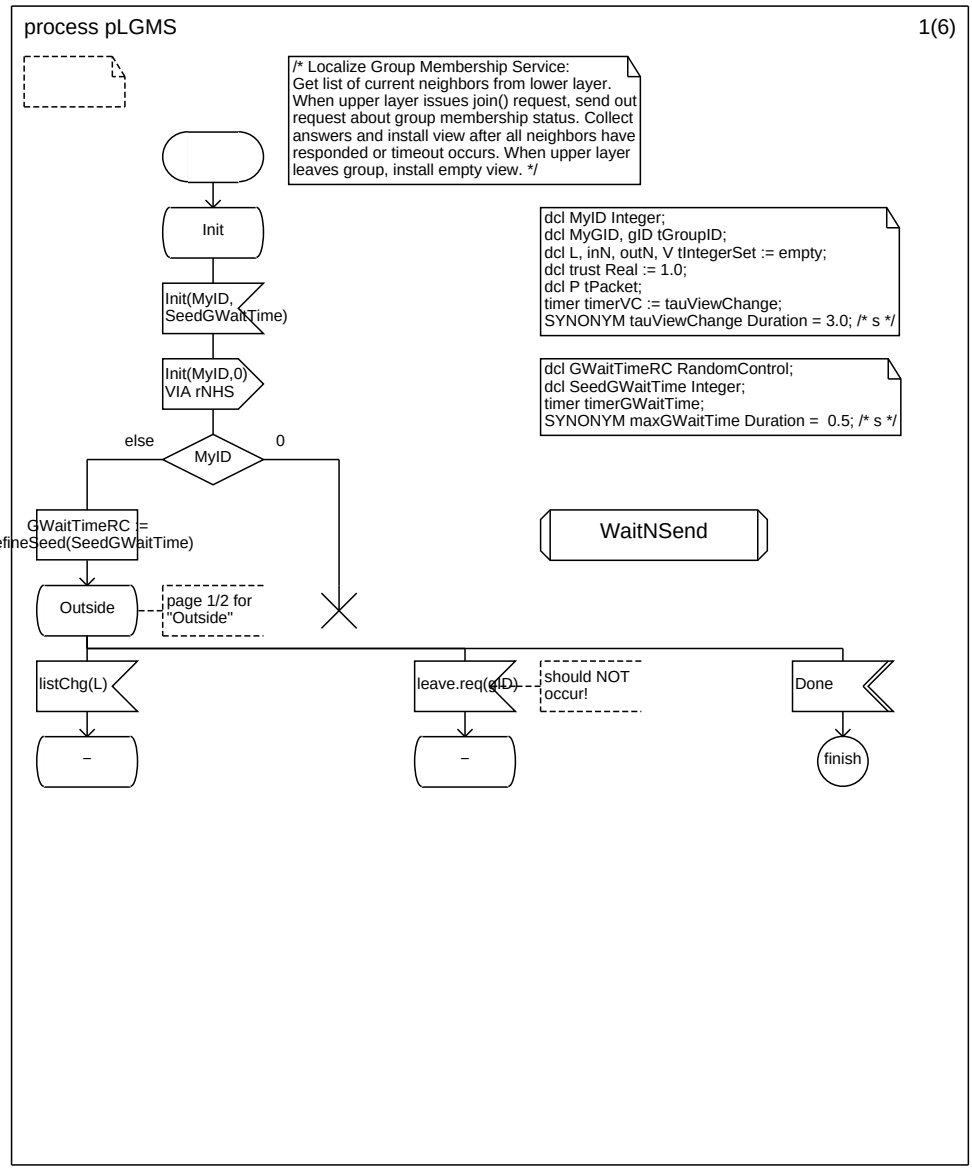

process pMsgManagement
2(3)
Idle
page 2/2
for "Idle"
initMsg(M)
receive(M)
MsgCounter :=
MsgCounter+1;
else
M!present
MSG
M!MSG!MessageID := MsgCounter;
–
M!MSG!MaxHopCount
> 1
else
MsgArray(M!MSG!InitiatorID)
(M!MSG!MessageID)!known
false
true
mark as
known
MsgArray(MyID)
(MsgCounter)!known
:= true;
acceptMsg(M)
MsgArray(M!MSG!InitiatorID)
(M!MSG!MessageID)!pResend
else
Null
Resend(transmissionRange,
MyID,N,M)
else
M!MSG!MaxHopCount
receivedAgain
(M!MSG!SenderID)
TO MsgArray(M!MSG!InitiatorID)
(M!MSG!MessageID)!pResend
> 1
ResendList :=
incl(offspring,ResendList);
MsgArray(M!MSG!InitiatorID)
(M!MSG!MessageID)!known := true
M!MSG!data!present
mark as
known
fwd
–
–
–
else
J_ATBEGIN
J_ATEND
J_ALONE
sResults(UCast:
(. MyID,MsgCounter,0,0,2,MyPos .))
sResults(UCast:
(. MyID,MsgCounter,0,0,3,MyPos .))
sResults(UCast:
(. MyID,MsgCounter,0,0,4,MyPos .))
–
–
–
–

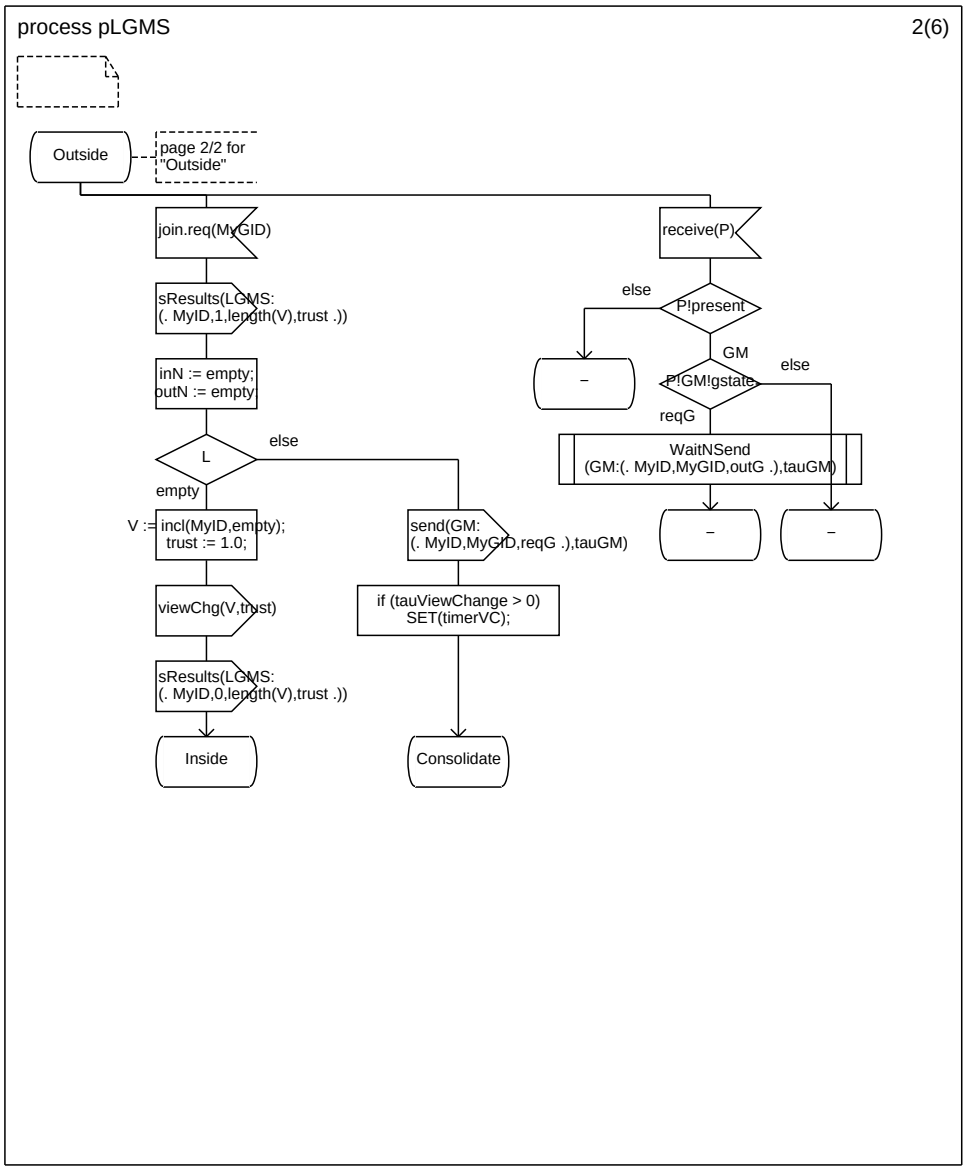

process pMsgManagement
3(3)
fwd
M!MSG!HopCount
< M!MSG!MaxHopCount
true
false
M!MSG!LifeTime
> tauMSG
do not resend
message
> 0.0
else
pos.req
do not resend
message
WaitForPos
pos.ind(MyPos)
*
Done
distributeDone
MyPos!x
−1.0
else
dist := call calcDistance
(MyPos!x,M!MSG!SenderPos!x,MyPos!y,M!MSG!SenderPos!y);
newM := M;
newM!MSG!SenderID := MyID;
newM!MSG!SenderPos := MyPos;
newM!MSG!HopCount := M!MSG!HopCount+1;
newM!MSG!LifeTime := M!MSG!LifeTime−tauMSG;
Resend(dist,M!MSG!SenderID,N,newM)
remember
offspring
MsgArray(M!MSG!InitiatorID)
(M!MSG!MessageID)!pResend := offspring;
ResendList :=
incl(offspring,ResendList);
Idle
Idle

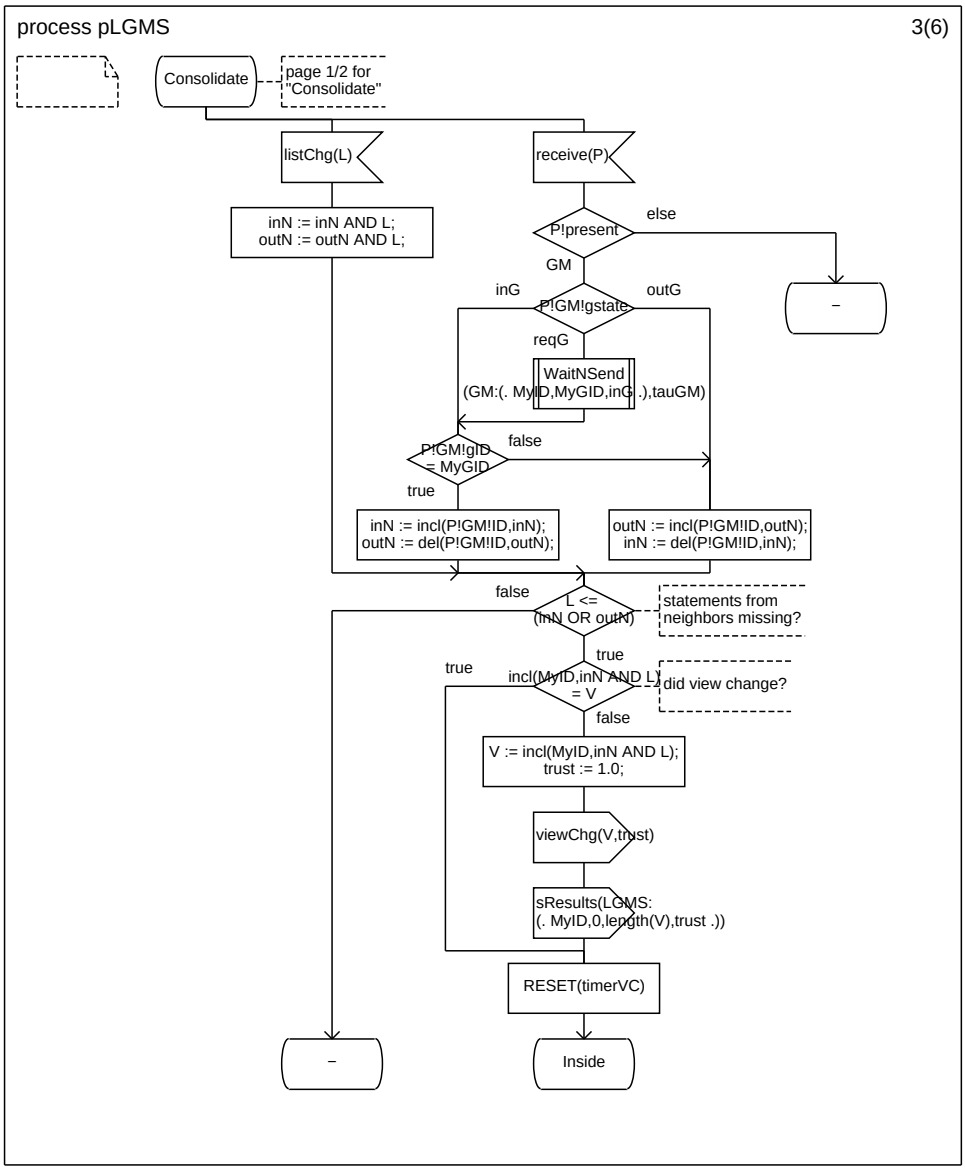

process type ptResend
1(2)
fpar
d Real,
senderID Integer,
neighbors tIntegerSet,
P tPacket;
/* calculates waiting time according to distance before resending the packet */
dcl S, N tIntegerSet;
dcl ID Integer;
dcl WaitTime Duration;
timer timerResend;
dcl startTime Time;
timer timerLifeTime;
SYNONYM maxRWaitTime Duration = 0.04; /* s */
N := neighbors;
S := incl(senderID,empty);
startTime := now;
SET(now+P!MSG!LifeTime,timerLifeTime);
call without(N,S)
empty
else
WaitForNeighbor
d > transmissionRange
false
true
Resend
WaitTime := maxRWaitTime− (maxRWaitTime/transmissionRange*d)
P!MSG!LifeTime := P!MSG!LifeTime−(now−startTime);
SET(now+WaitTime,timerResend)
send(P,tauMSG)
WaitToResend
Resend
P!MSG!data!present
J_ATBEGIN,J_ATEND,J_ALONE
else
sResults(UCast: (. P!MSG!InitiatorID,P!MSG!MessageID, P!MSG!HopCount,P!MSG!SenderID,1,P!MSG!SenderPos .))
resendDone (P!MSG!InitiatorID, P!MSG!MessageID) TO parent
gMsgM
listChg, Done, receivedAgain
resendDone
gLGMS
send
gResults
sResults

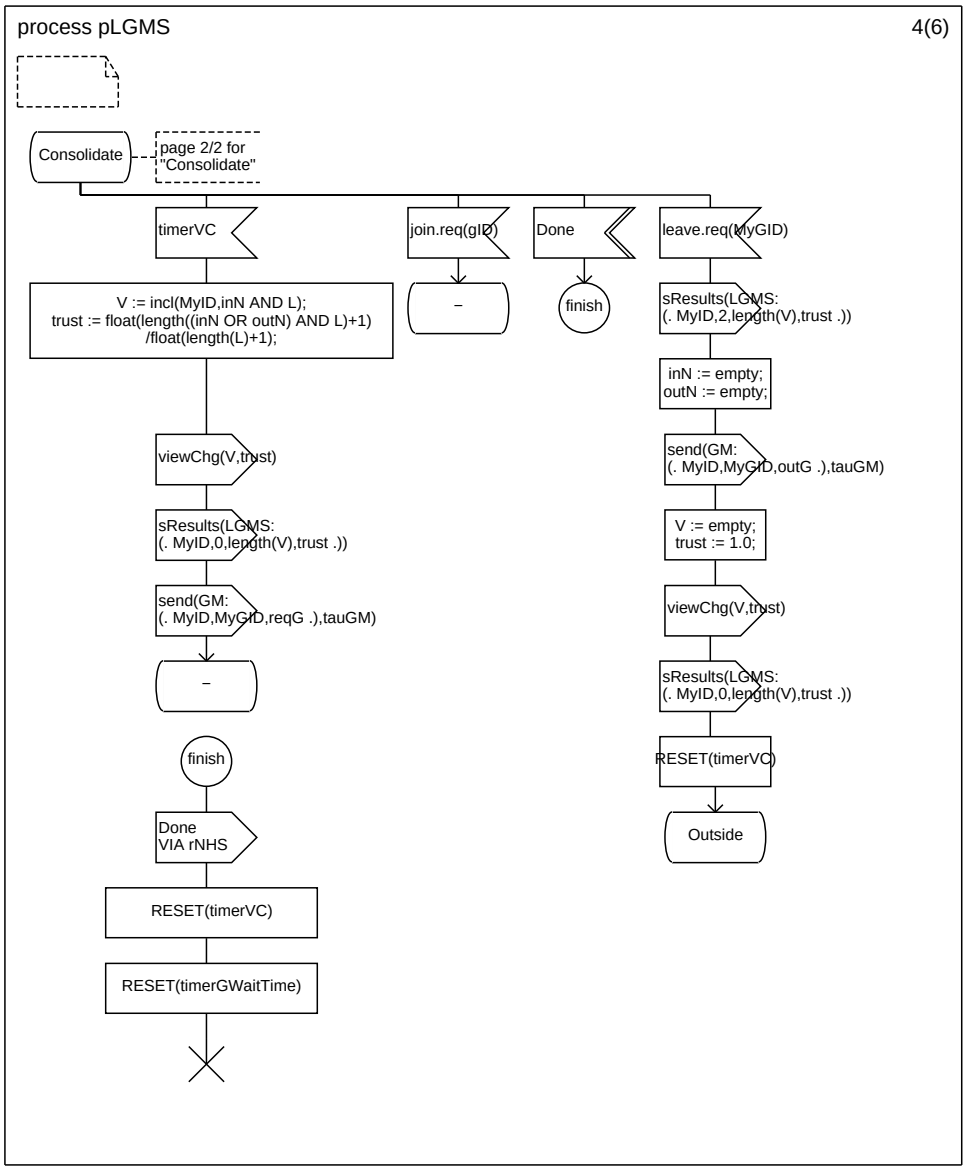

process type ptResend
2(2)
;fpar
d Real,
senderID Integer,
neighbors tIntegerSet,
P tPacket;
WaitToResend
listChg(N)
receivedAgain(ID)
timerResend
timerLifeTime
life time of message is over
Done
S := incl(ID,S)
Resend
resendDone (P!MSG!InitiatorID, P!MSG!MessageID) TO parent
call without(N,S)
else
empty
–
RESET(timerResend)
WaitForNeighbor
WaitForNeighbor
listChg(N)
receivedAgain(ID)
timerLifeTime
life time of message is over
Done
S := incl(ID,S)
resendDone (P!MSG!InitiatorID, P!MSG!MessageID) TO parent
call without(N,S)
empty
else
–
Resend

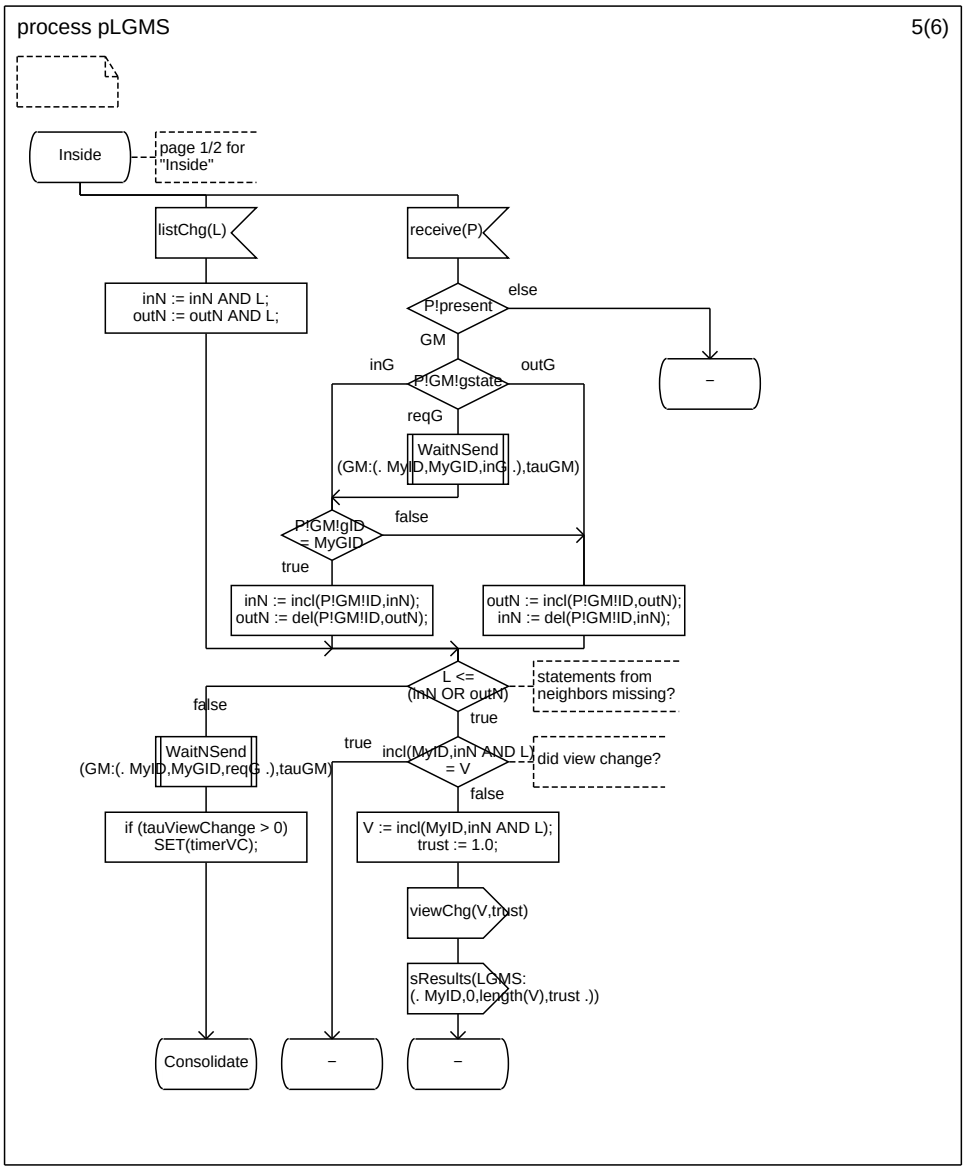

cLGMS
cLGMS
block bLGMS
1(1)
viewChg
listChg,
receive
/* implements localizes
group membership service
(LGMS) and neighborhood
service (NHS) */
rLGMS
join.req,
leave.req
rNHS2
rControlLGMS
sResults
cControlLGMS
pLGMS
cLGMSResults
Init, Done
rLGMSResults
listChg,
receive
rNHS
Init, Done
send
rControlNHS
sResults
cControlLGMS
pNHS
cLGMSResults
Go
rNHSResults
rMAC2
receive
rMAC
send
send
cMAC
cMAC

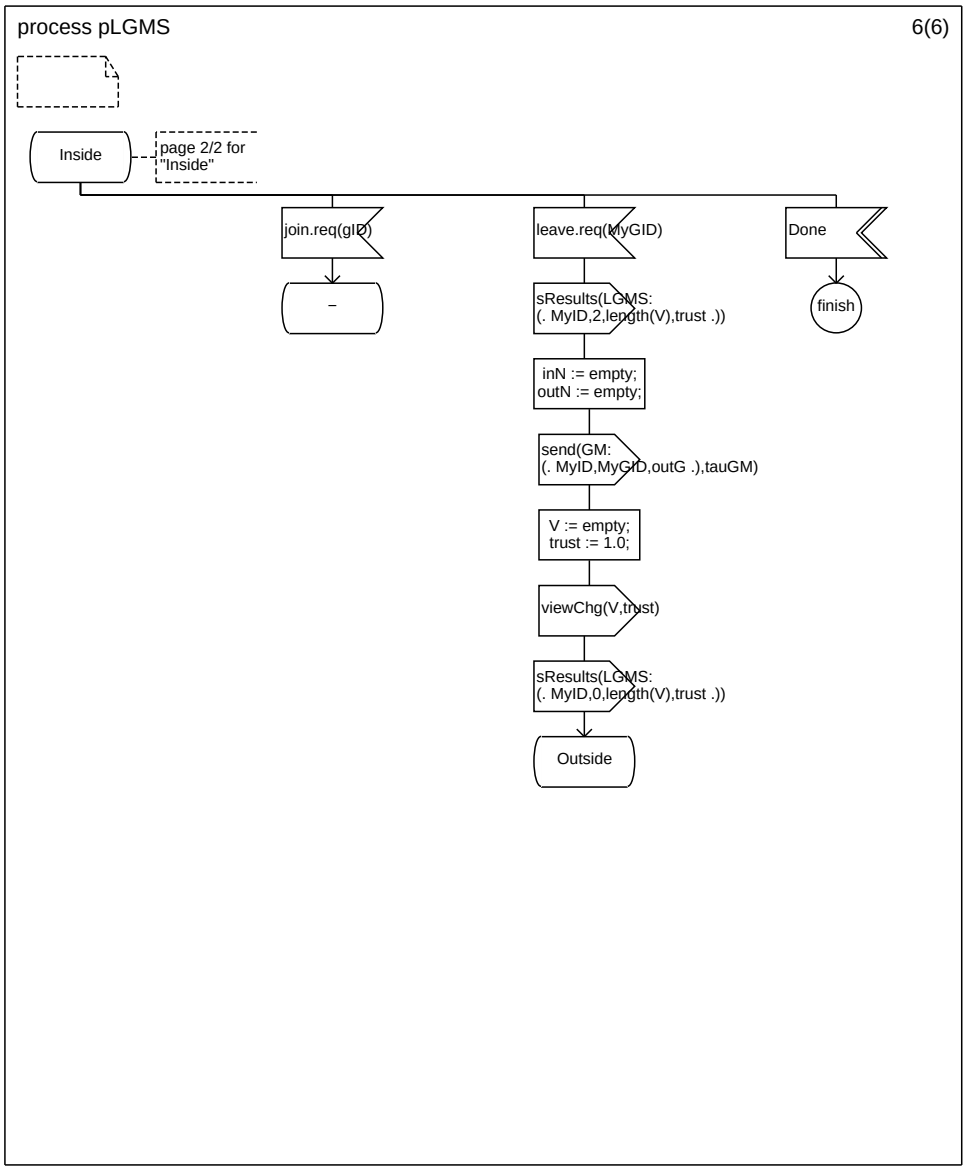

process pLGMS
1(6)
/* Localize Group Membership Service:
Get list of current neighbors from lower layer.
When upper layer issues join() request, send out request about group membership status. Collect answers and install view after all neighbors have responded or timeout occurs. When upper layer leaves group, install empty view. */
Init
dcl MyID Integer;
dcl MyGID, gID tGroupID;
dcl L, inN, outN, V tIntegerSet := empty;
dcl trust Real := 1.0;
dcl P tPacket;
timer timerVC := tauViewChange;
SYNONYM tauViewChange Duration = 3.0; /* s */
Init(MyID, SeedGWaitTime)
Init(MyID,0) VIA rNHS
dcl GWaitTimeRC RandomControl;
dcl SeedGWaitTime Integer;
timer timerGWaitTime;
SYNONYM maxGWaitTime Duration = 0.5; /* s */
else
MyID
0
GWaitTimeRC := DefineSeed(SeedGWaitTime)
WaitNSend
Outside
page 1/2 for "Outside"
listChg(L)
leave.req(gID)
should NOT occur!
Done
–
–
finish

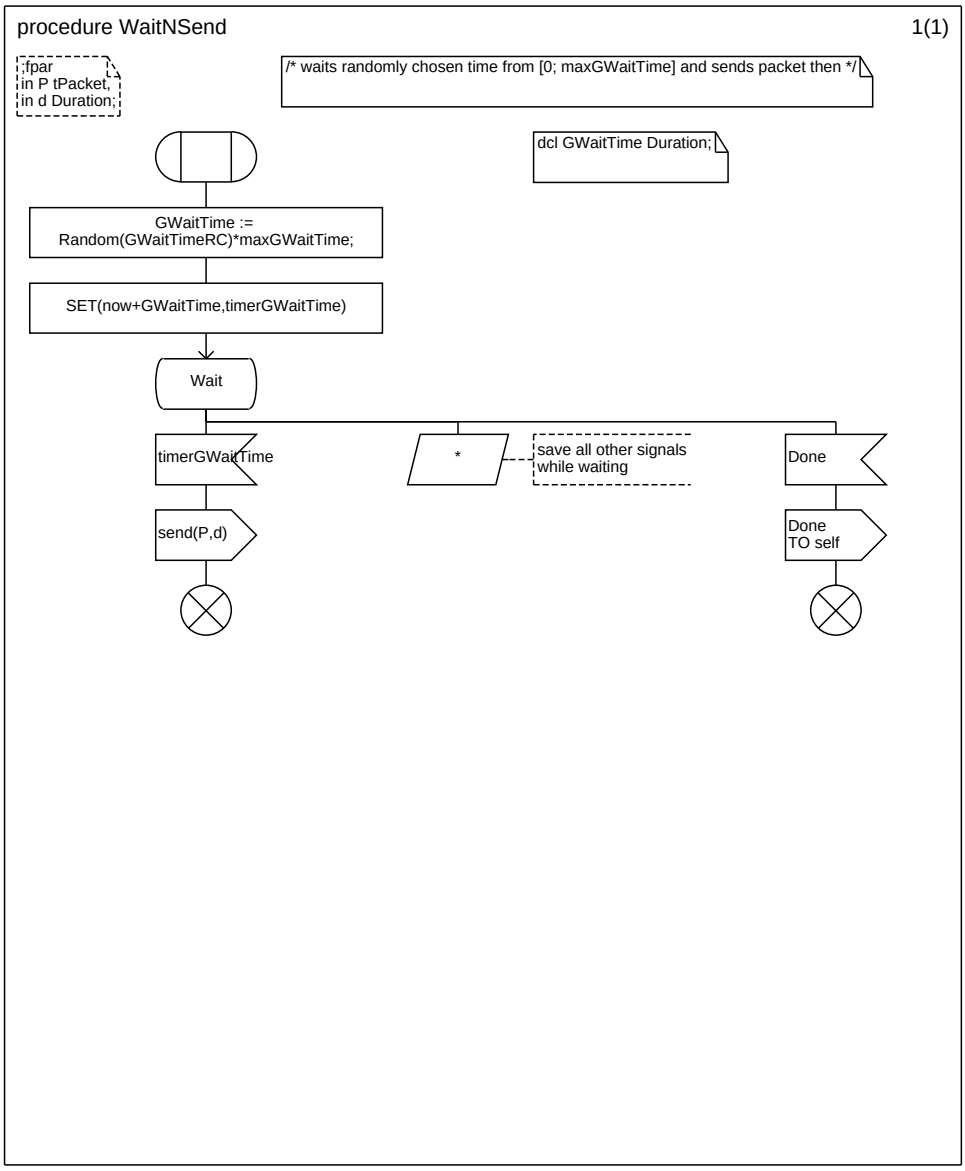

process pLGMS
2(6)
Outside
page 2/2 for "Outside"
join.req(MyGID)
receive(P)
sResults(LGMS: (. MyID,1,length(V),trust .))
else
P!present
GM
else
P!GM!gstate
reqG
inN := empty; outN := empty
–
WaitNSend (GM:(. MyID,MyGID,outG .),tauGM)
L
else
empty
V := incl(MyID,empty); trust := 1.0;
send(GM: (. MyID,MyGID,reqG .),tauGM)
–
–
viewChg(V,trust)
if (tauViewChange > 0) SET(timerVC);
sResults(LGMS: (. MyID,0,length(V),trust .))
Inside
Consolidate

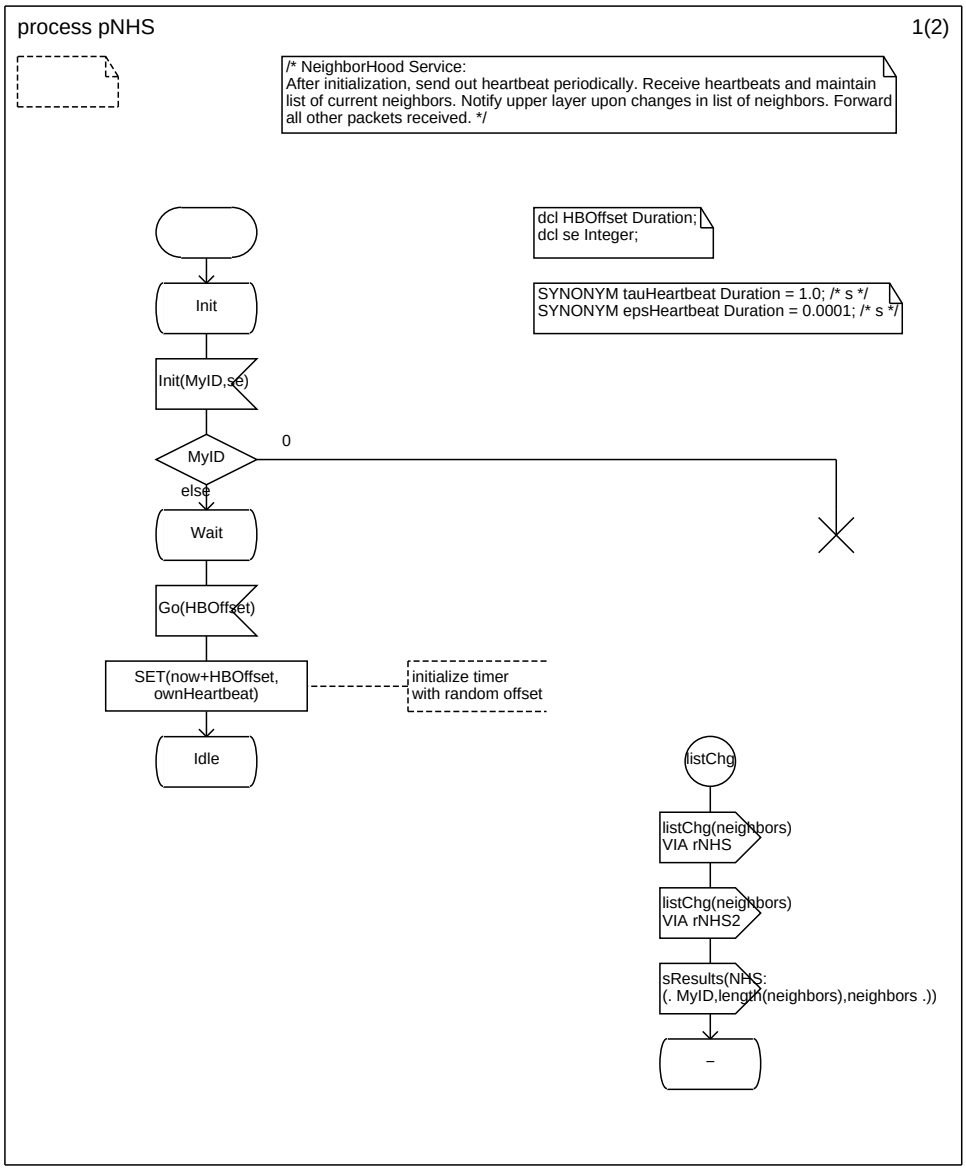

process pLGMS
3(6)
Consolidate
page 1/2 for "Consolidate"
listChg(L)
receive(P)
inN := inN AND L;
outN := outN AND L;
P!present
else
GM
inG
outG
P!GM!gstate
–
reqG
WaitNSend
(GM:(. MyID,MyGID,inG .),tauGM)
P!GM!gID = MyGID
false
true
inN := incl(P!GM!ID,inN);
outN := del(P!GM!ID,outN);
outN := incl(P!GM!ID,outN);
inN := del(P!GM!ID,inN);
false
L <= (inN OR outN)
statements from neighbors missing?
true
true
incl(MyID,inN AND L) = V
did view change?
false
V := incl(MyID,inN AND L);
trust := 1.0;
viewChg(V,trust)
sResults(LGMS: (. MyID,0,length(V),trust .))
RESET(timerVC)
–
Inside

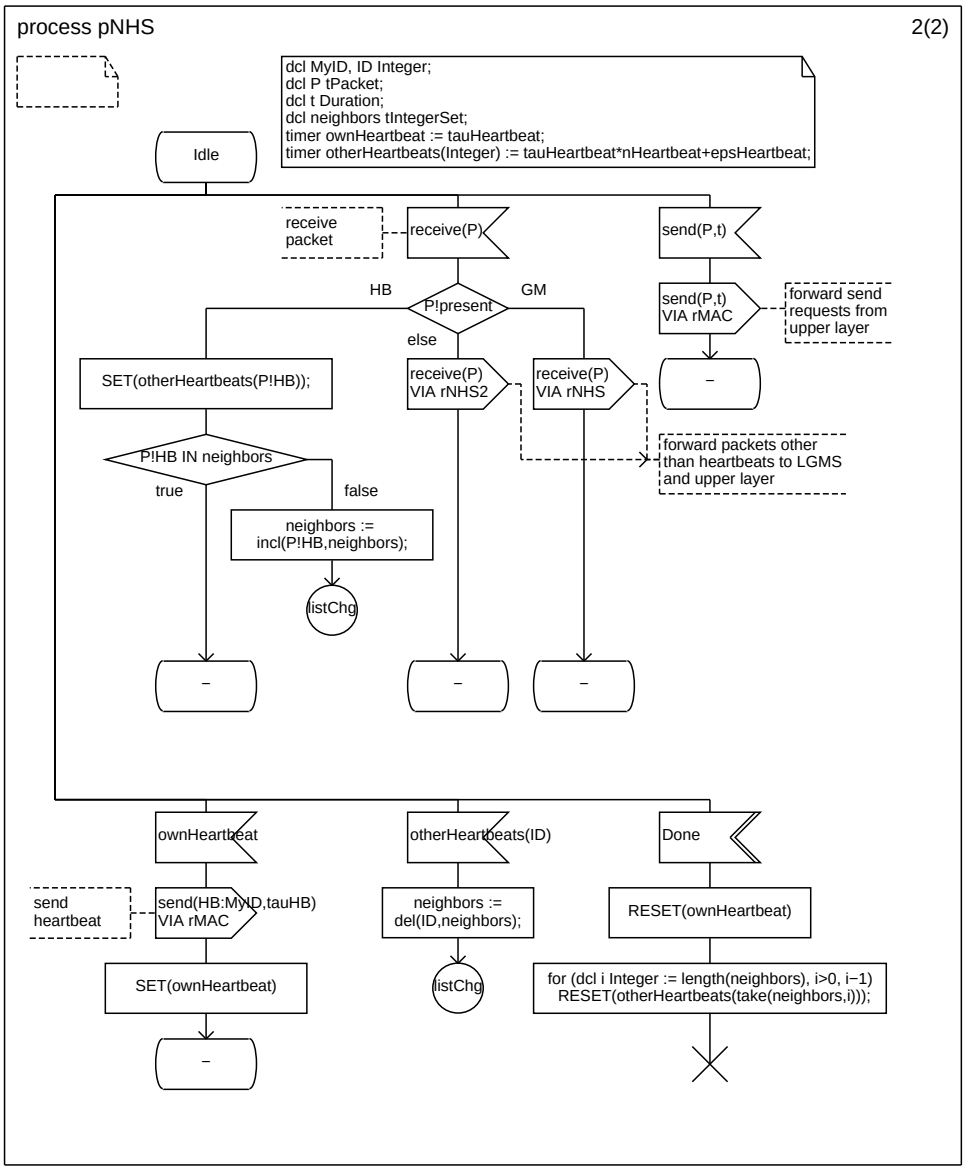

process pLGMS
4(6)
Consolidate
page 2/2 for "Consolidate"
timerVC
join.req(gID)
Done
leave.req(MyGID)
V := incl(MyID,inN AND L);
trust := float(length((inN OR outN) AND L)+1) /float(length(L)+1);
finish
sResults(LGMS: (. MyID,2,length(V),trust .))
inN := empty;
outN := empty;
viewChg(V,trust)
sResults(LGMS: (. MyID,0,length(V),trust .))
send(GM: (. MyID,MyGID,reqG .),tauGM)
send(GM: (. MyID,MyGID,outG .),tauGM)
V := empty;
trust := 1.0;
viewChg(V,trust)
sResults(LGMS: (. MyID,0,length(V),trust .))
RESET(timerVC)
Outside
finish
Done VIA rNHS
RESET(timerVC)
RESET(timerGWaitTime)

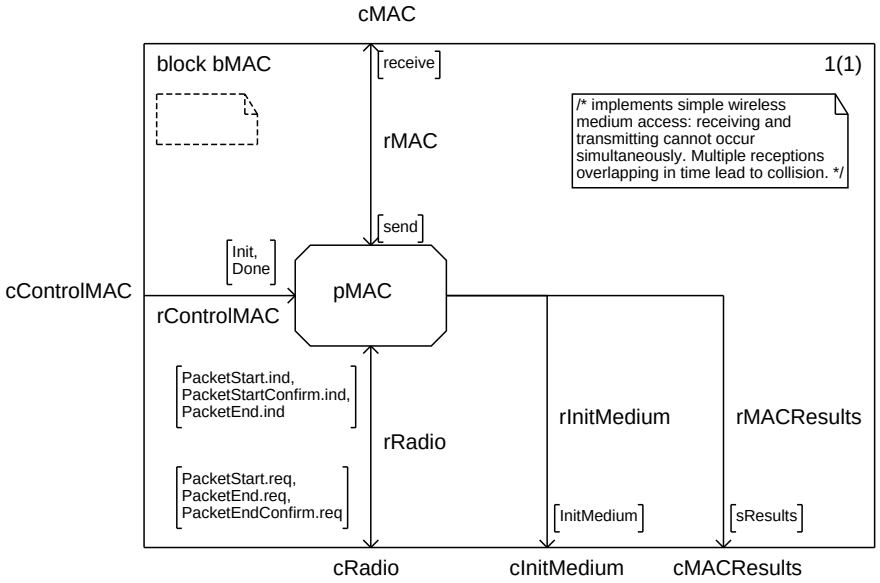

process pLGMS
5(6)
Inside
page 1/2 for "Inside"
listChg(L)
receive(P)
inN := inN AND L;
outN := outN AND L;
P!present
else
GM
inG
outG
P!GM!gstate
–
reqG
WaitNSend
(GM:(. MyID,MyGID,inG .),tauGM)
P!GM!gID = MyGID
false
true
inN := incl(P!GM!ID,inN);
outN := del(P!GM!ID,outN);
outN := incl(P!GM!ID,outN);
inN := del(P!GM!ID,inN);
L <= (inN OR outN)
statements from neighbors missing?
false
true
WaitNSend
(GM:(. MyID,MyGID,reqG .),tauGM)
true
incl(MyID,inN AND L) = V
did view change?
false
if (tauViewChange > 0) SET(timerVC);
V := incl(MyID,inN AND L);
trust := 1.0;
viewChg(V,trust)
sResults(LGMS: (. MyID,0,length(V),trust .))
Consolidate
–
–

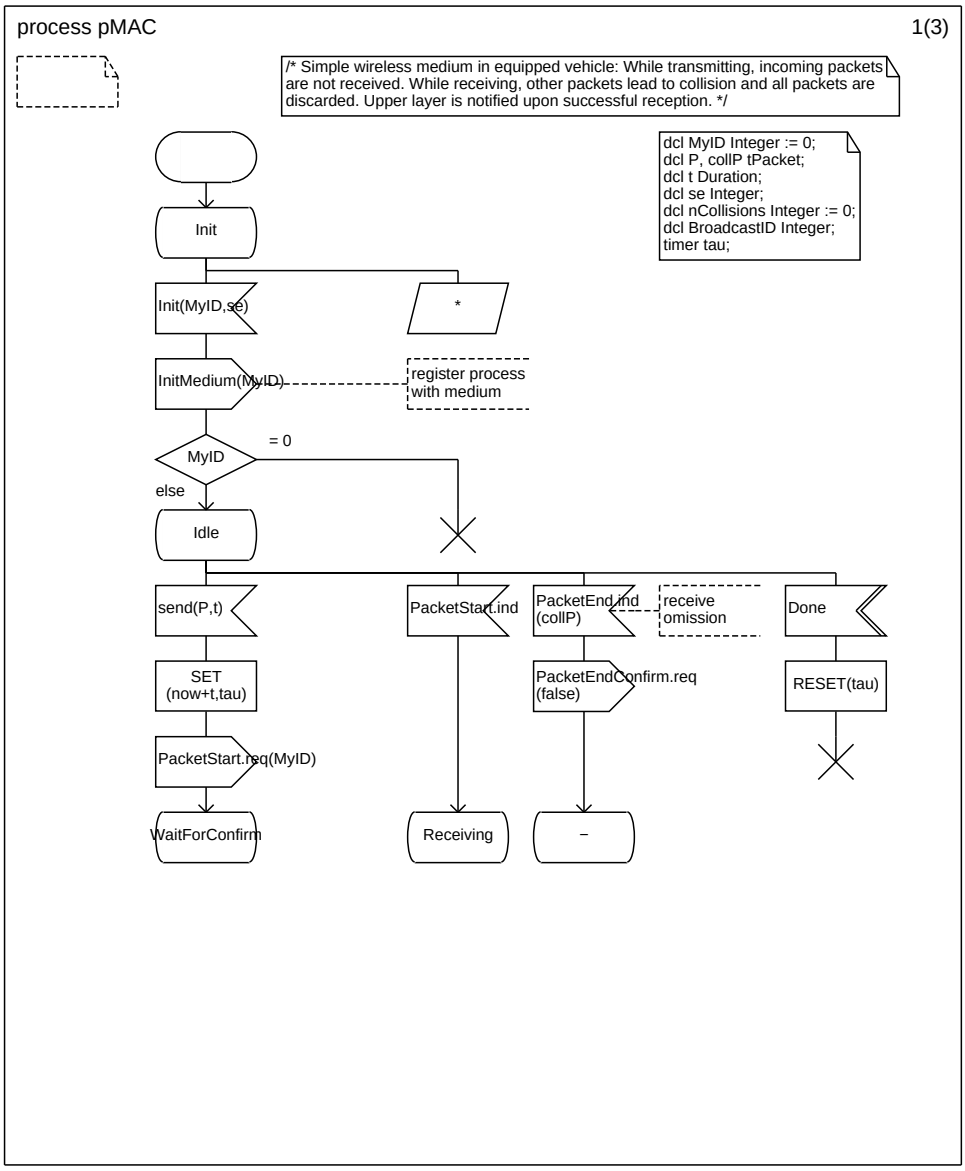

process pLGMS
6(6)
Inside
page 2/2 for "Inside"
join.req(gID)
–
leave.req(MyGID)
sResults(LGMS: (. MyID,2,length(V),trust .))
inN := empty; outN := empty;
send(GM: (. MyID,MyGID,outG .),tauGM)
V := empty; trust := 1.0;
viewChg(V,trust)
sResults(LGMS: (. MyID,0,length(V),trust .))
Outside
Done
finish

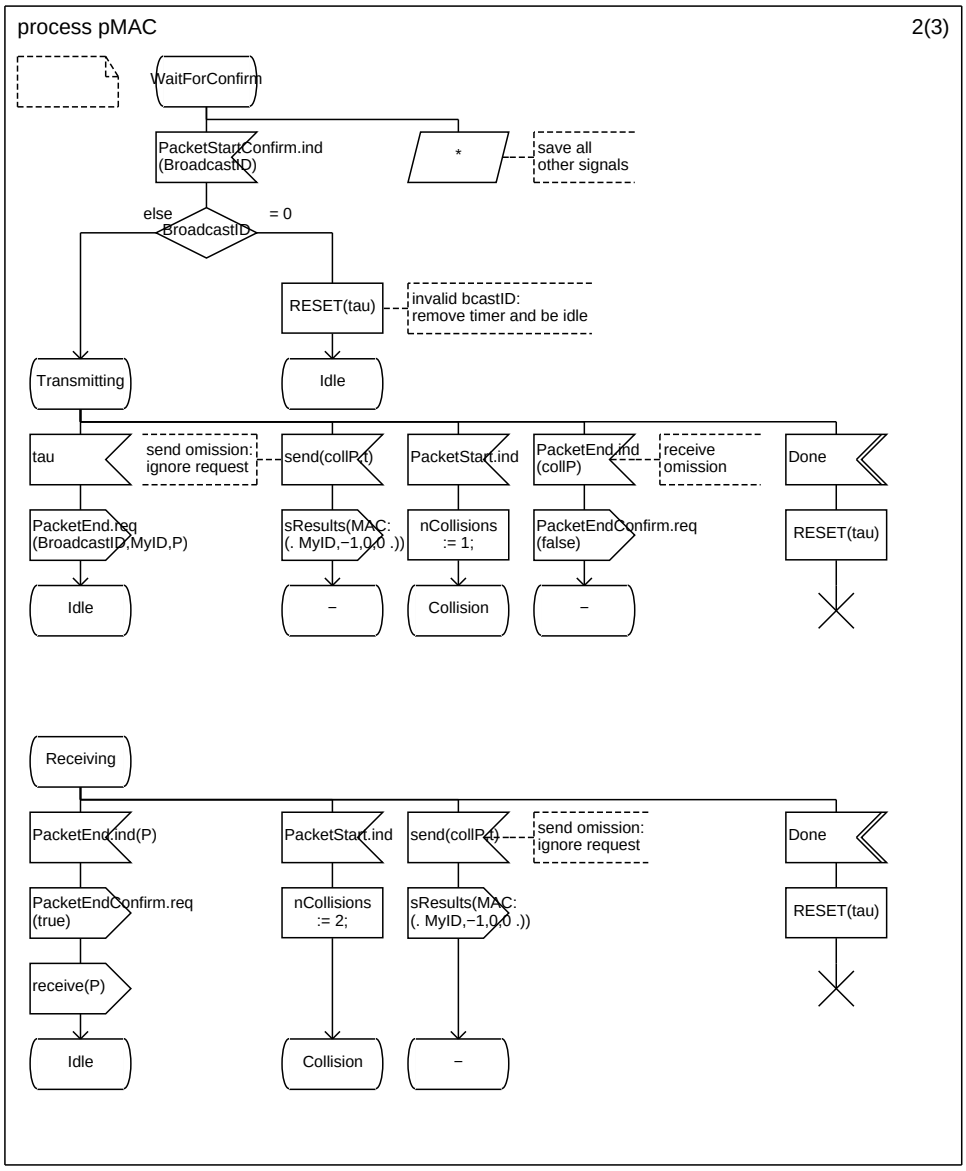

procedure WaitNSend
1(1)
;fpar
in P tPacket,
in d Duration;
/* waits randomly chosen time from [0; maxGWaitTime] and sends packet then */
dcl GWaitTime Duration;
GWaitTime :=
Random(GWaitTimeRC)*maxGWaitTime;
SET(now+GWaitTime,timerGWaitTime)
Wait
timerGWaitTime
*
save all other signals
while waiting
Done
send(P,d)
Done
TO self

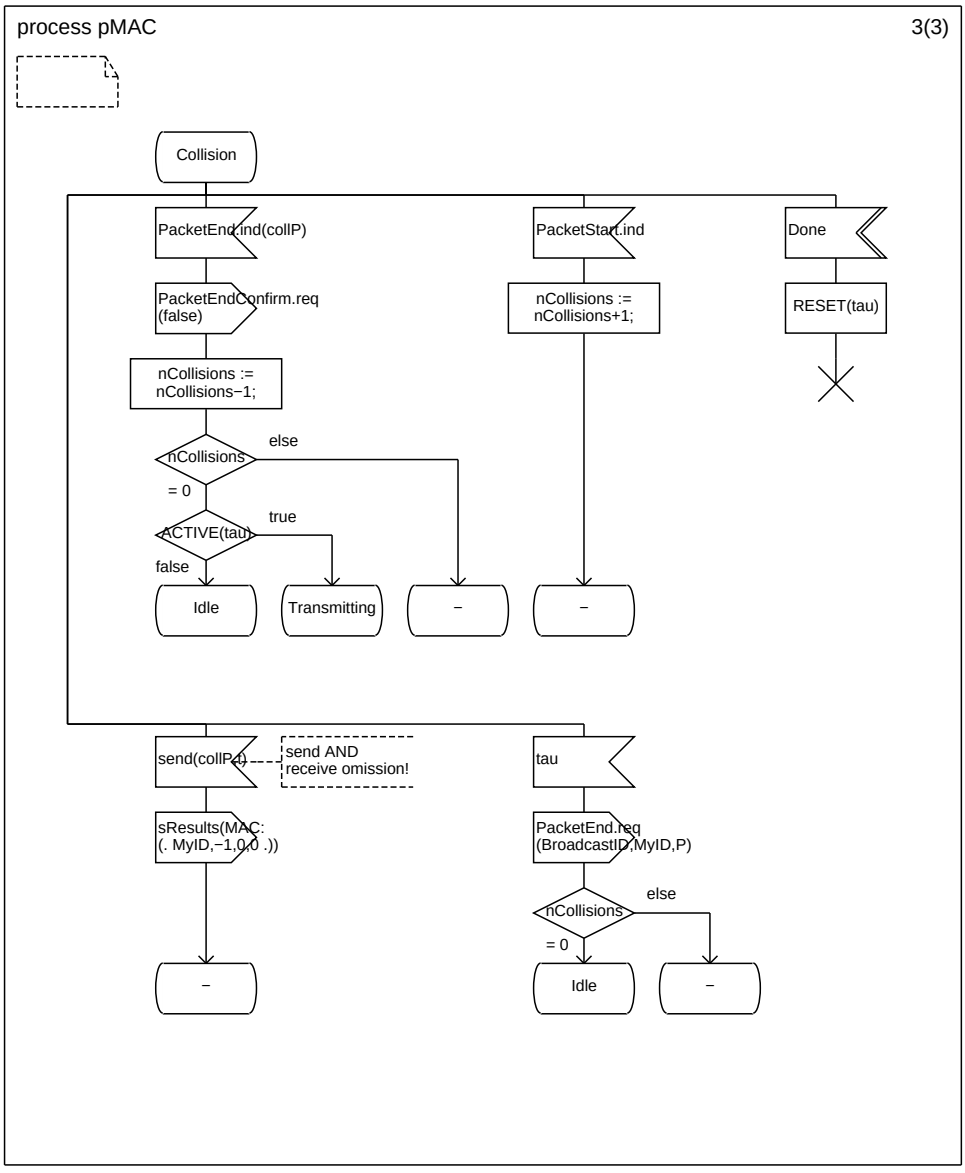

process pNHS
1(2)
/* NeighborHood Service:
After initialization, send out heartbeat periodically. Receive heartbeats and maintain list of current neighbors. Notify upper layer upon changes in list of neighbors. Forward all other packets received. */
dcl HBOffset Duration;
dcl se Integer;
SYNONYM tauHeartbeat Duration = 1.0; /* s */
SYNONYM epsHeartbeat Duration = 0.0001; /* s */
Init
Init(MyID,se)
MyID
0
else
Wait
Go(HBOffset)
SET(now+HBOffset, ownHeartbeat)
initialize timer with random offset
Idle
listChg
listChg(neighbors) VIA rNHS
listChg(neighbors) VIA rNHS2
sResults(NHS: (. MyID,length(neighbors),neighbors .))


process pNHS
2(2)
dcl MyID, ID Integer;
dcl P tPacket;
dcl t Duration;
dcl neighbors tIntegerSet;
timer ownHeartbeat := tauHeartbeat;
timer otherHeartbeats(Integer) := tauHeartbeat*nHeartbeat+epsHeartbeat;
Idle
receive packet
receive(P)
send(P,t)
HB
P!present
GM
else
send(P,t) VIA rMAC
forward send requests from upper layer
SET(otherHeartbeats(P!HB));
receive(P) VIA rNHS2
receive(P) VIA rNHS
–
forward packets other than heartbeats to LGMS and upper layer
P!HB IN neighbors
true
false
neighbors := incl(P!HB,neighbors);
listChg
–
–
–
ownHeartbeat
otherHeartbeats(ID)
Done
send heartbeat
send(HB:MyID,tauHB) VIA rMAC
neighbors := del(ID,neighbors);
RESET(ownHeartbeat)
SET(ownHeartbeat)
listChg
for (dcl i Integer := length(neighbors), i>0, i−1) RESET(otherHeartbeats(take(neighbors,i)));
–

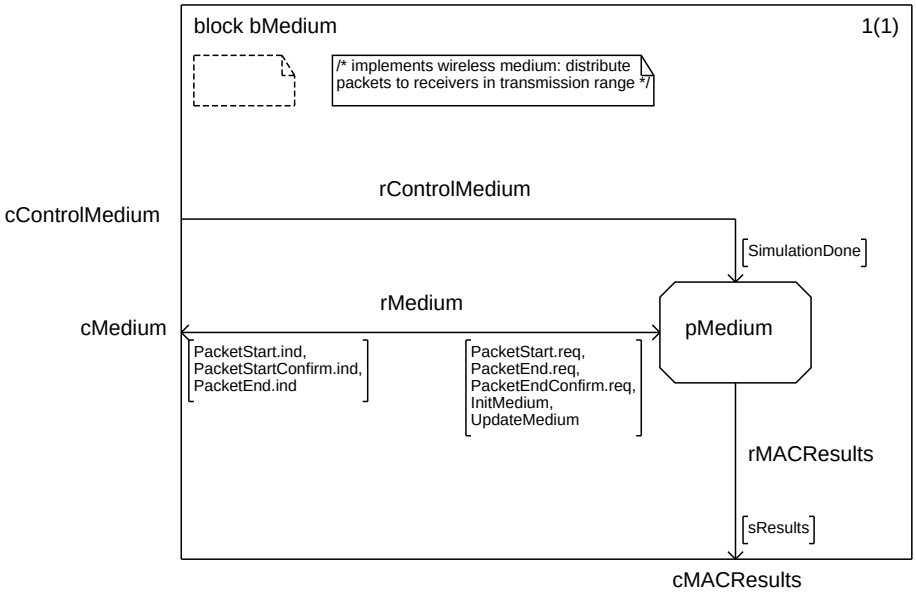

cMAC
block bMAC
1(1)
receive
rMAC
/* implements simple wireless medium access: receiving and transmitting cannot occur simultaneously. Multiple receptions overlapping in time lead to collision. */
send
Init,
Done
cControlMAC
pMAC
rControlMAC
PacketStart.ind,
PacketStartConfirm.ind,
PacketEnd.ind
rRadio
rInitMedium
rMACResults
PacketStart.req,
PacketEnd.req,
PacketEndConfirm.req
InitMedium
sResults
cRadio
cInitMedium
cMACResults

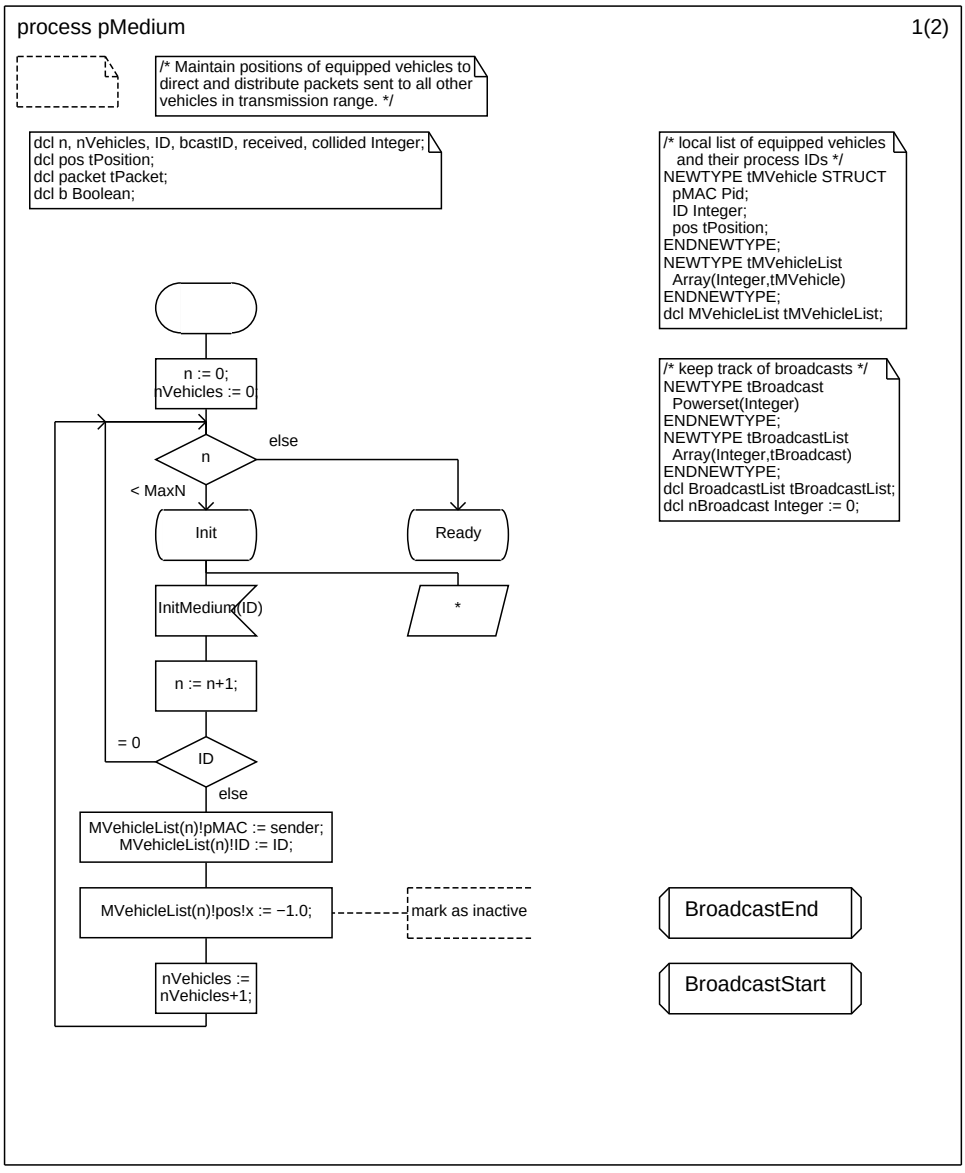

process pMAC
1(3)
/* Simple wireless medium in equipped vehicle: While transmitting, incoming packets are not received. While receiving, other packets lead to collision and all packets are discarded. Upper layer is notified upon successful reception. */
dcl MyID Integer := 0;
dcl P, collP tPacket;
dcl t Duration;
dcl se Integer;
dcl nCollisions Integer := 0;
dcl BroadcastID Integer;
timer tau;
Init
Init(MyID,se)
*
InitMedium(MyID)
register process with medium
MyID
= 0
else
Idle
send(P,t)
PacketStart.ind
PacketEnd.ind (collP)
receive omission
Done
SET (now+t,tau)
PacketEndConfirm.req (false)
RESET(tau)
PacketStart.req(MyID)
WaitForConfirm
Receiving
–

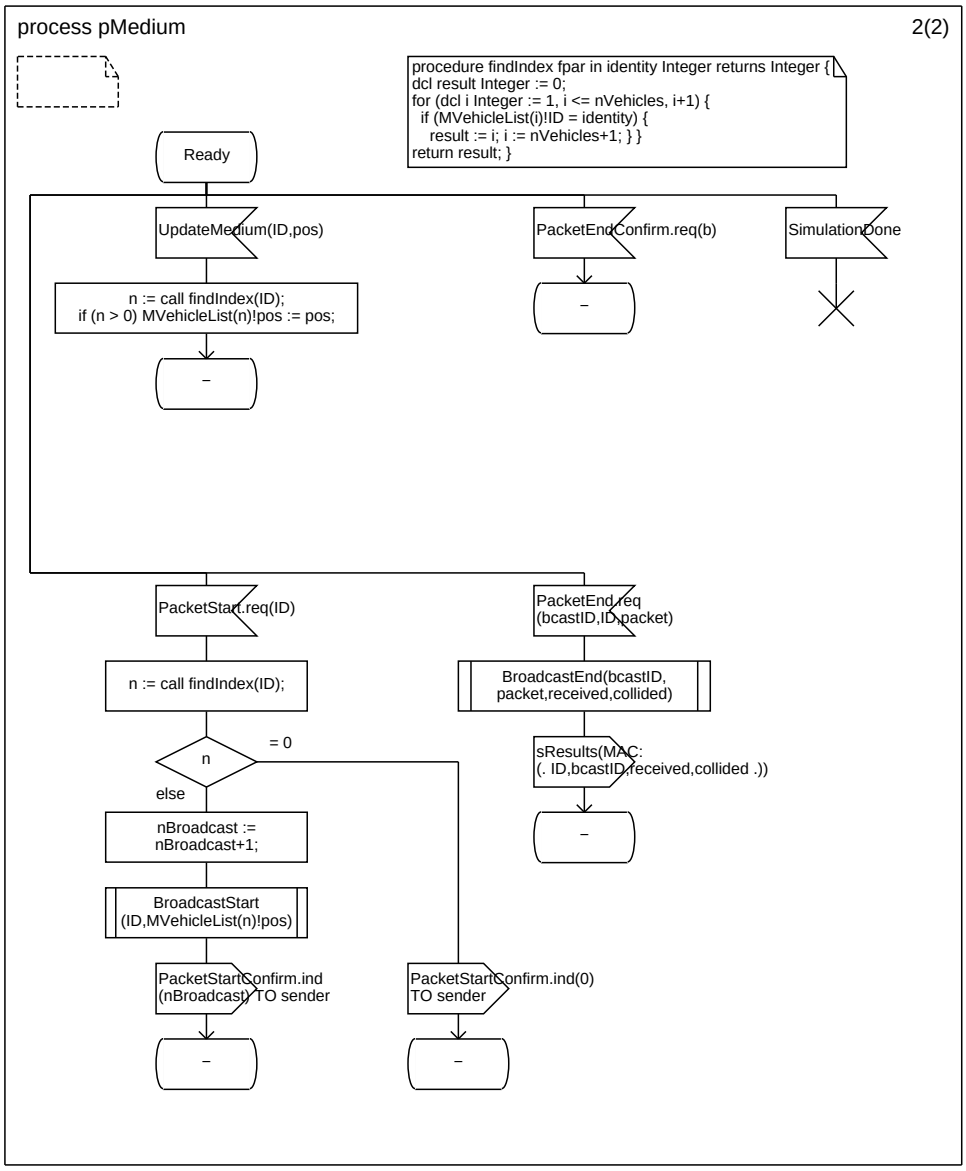

process pMAC
2(3)
WaitForConfirm
PacketStartConfirm.ind
(BroadcastID)
*
save all
other signals
else
BroadcastID
= 0
RESET(tau)
invalid bcastID:
remove timer and be idle
Transmitting
Idle
tau
send omission:
ignore request
send(collP,1)
PacketStart.ind
PacketEnd.ind
(collP)
receive
omission
Done
PacketEnd.req
(BroadcastID,MyID,P)
sResults(MAC:
(. MyID,−1,0,0 .))
nCollisions
:= 1;
PacketEndConfirm.req
(false)
RESET(tau)
Idle
−
Collision
−
Receiving
PacketEnd.ind(P)
PacketStart.ind
send(collP,1)
send omission:
ignore request
Done
PacketEndConfirm.req
(true)
nCollisions
:= 2;
sResults(MAC:
(. MyID,−1,0,0 .))
RESET(tau)
receive(P)
Idle
Collision
−

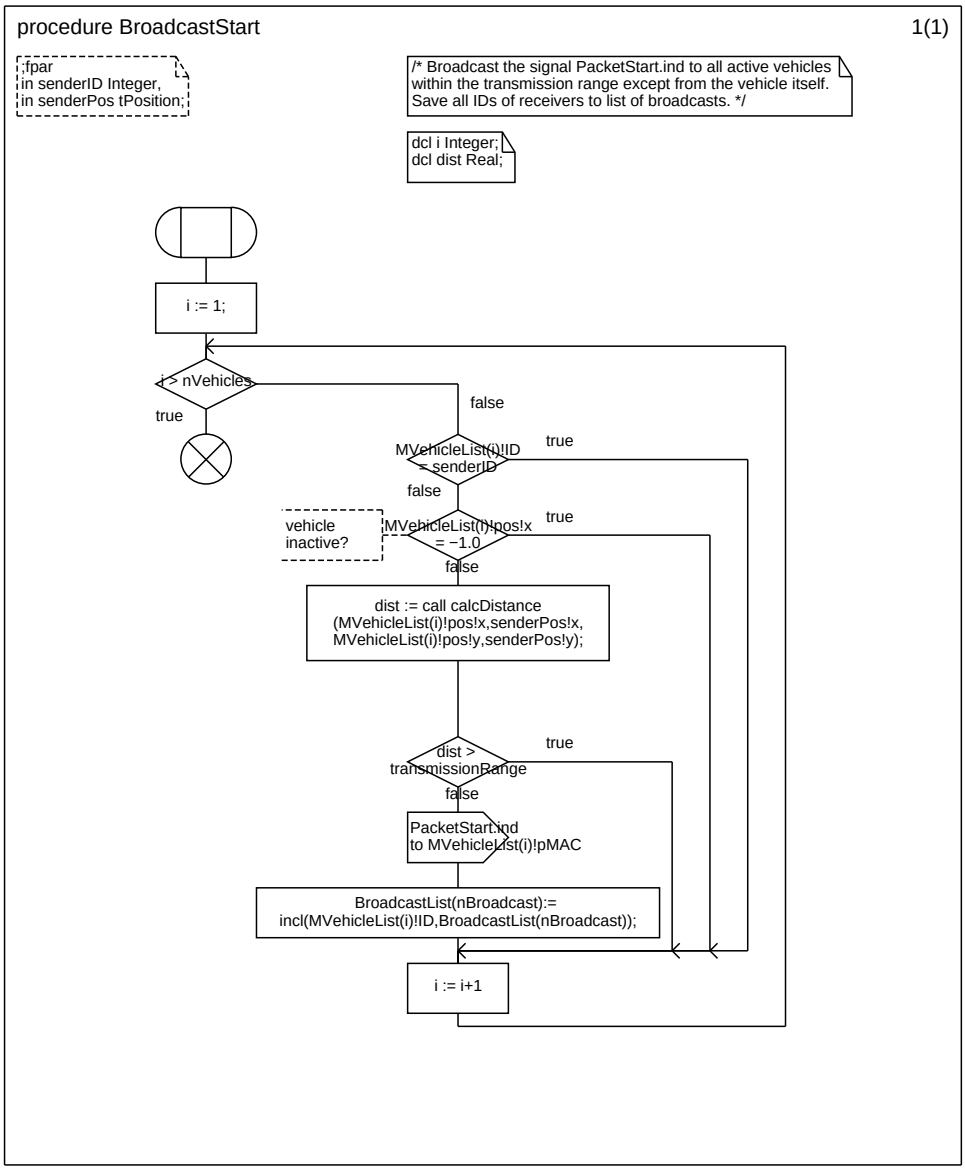

process pMAC
3(3)
Collision
PacketEnd.ind(collP)
PacketStart.ind
Done
PacketEndConfirm.req
(false)
nCollisions :=
nCollisions+1;
RESET(tau)
nCollisions :=
nCollisions−1;
else
nCollisions
= 0
true
ACTIVE(tau)
false
Idle
Transmitting
−
−
send(collP,1)
send AND
receive omission!
tau
sResults(MAC:
(. MyID,−1,0.0 .))
PacketEnd.req
(BroadcastID,MyID,P)
else
nCollisions
= 0
−
Idle
−

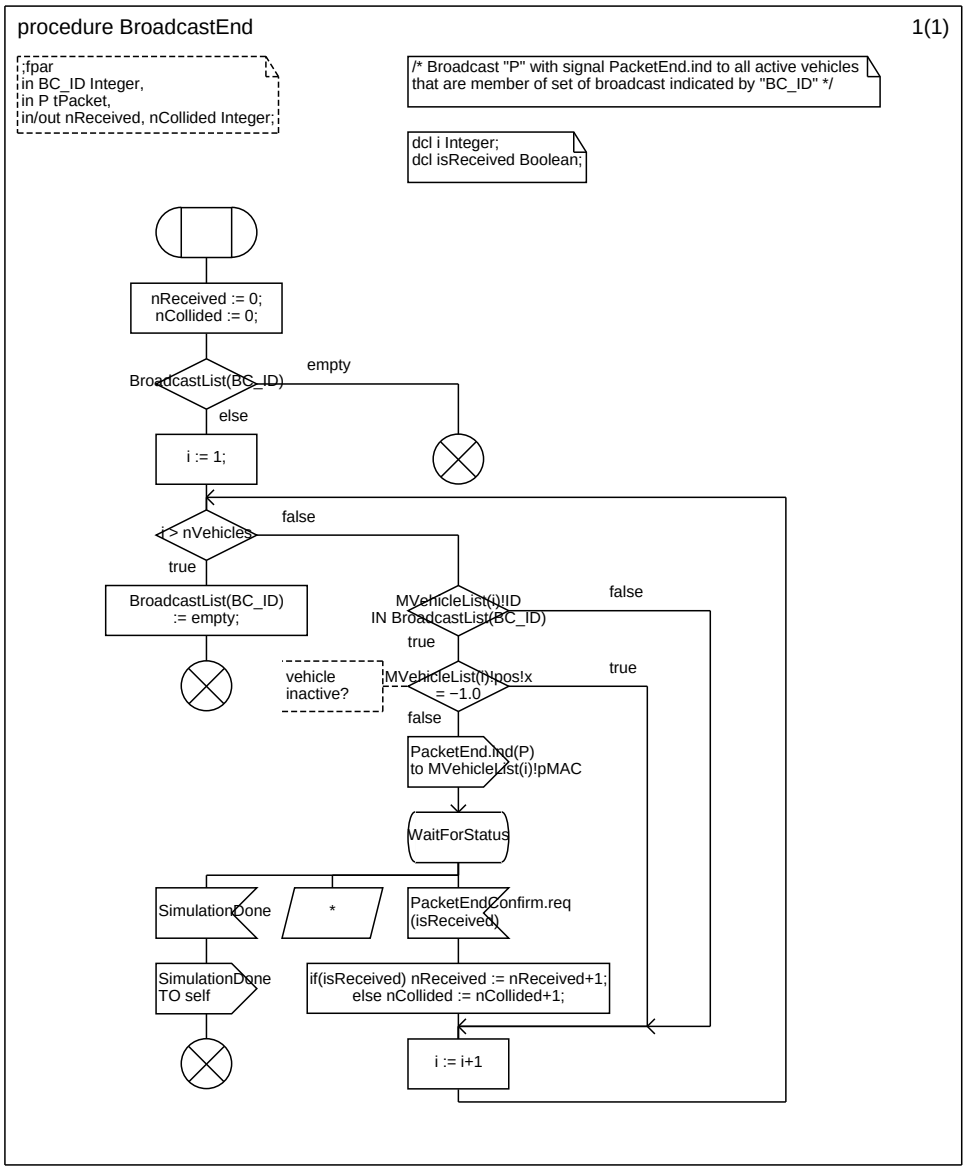
C.3.3 Medium


block bMedium
1(1)
/* implements wireless medium: distribute packets to receivers in transmission range */
rControlMedium
cControlMedium
SimulationDone
rMedium
cMedium
pMedium
PacketStart.ind,
PacketStartConfirm.ind,
PacketEnd.ind
PacketStart.req,
PacketEnd.req,
PacketEndConfirm.req,
InitMedium,
UpdateMedium
rMACResults
sResults
cMACResults

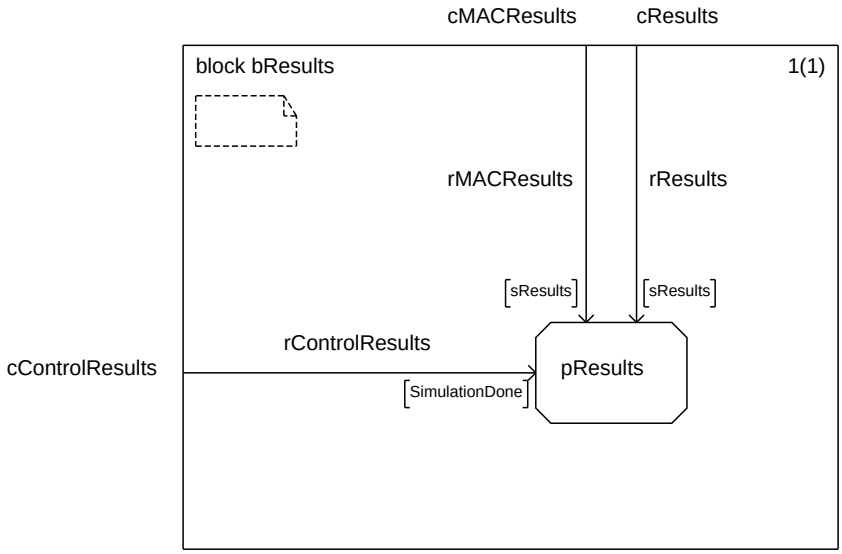

process pMedium
1(2)
/* Maintain positions of equipped vehicles to direct and distribute packets sent to all other vehicles in transmission range. */
dcl n, nVehicles, ID, bcastID, received, collided Integer;
dcl pos tPosition;
dcl packet tPacket;
dcl b Boolean;
/* local list of equipped vehicles and their process IDs */
NEWTYPE tMVehicle STRUCT
pMAC Pid;
ID Integer;
pos tPosition;
ENDNEWTYPE;
NEWTYPE tMVehicleList
Array(Integer,tMVehicle)
ENDNEWTYPE;
dcl MVehicleList tMVehicleList;
n := 0;
nVehicles := 0
n
else
< MaxN
Init
Ready
/* keep track of broadcasts */
NEWTYPE tBroadcast
Powerset(Integer)
ENDNEWTYPE;
NEWTYPE tBroadcastList
Array(Integer,tBroadcast)
ENDNEWTYPE;
dcl BroadcastList tBroadcastList;
dcl nBroadcast Integer := 0;
InitMedium(ID)
*
n := n+1;
= 0
ID
else
MVehicleList(n)!pMAC := sender;
MVehicleList(n)!ID := ID;
MVehicleList(n)!pos!x := −1.0;
mark as inactive
nVehicles :=
nVehicles+1;
BroadcastEnd
BroadcastStart

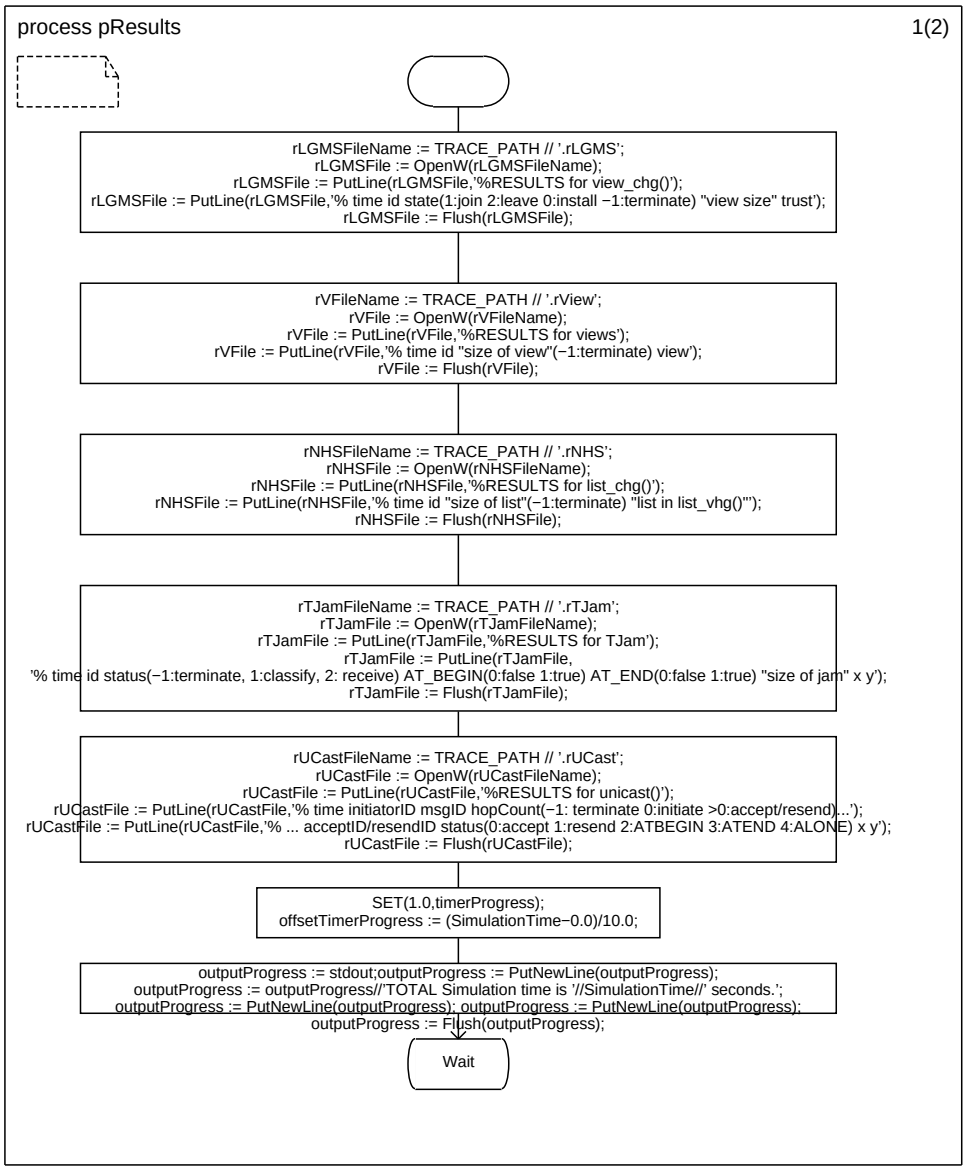

process pMedium
2(2)
procedure findIndex fpar in identity Integer returns Integer {
dcl result Integer := 0;
for (dcl i Integer := 1, i <= nVehicles, i+1) {
if (MVehicleList(i)!ID = identity) {
result := i; i := nVehicles+1; } }
return result; }
Ready
UpdateMedium(ID,pos)
PacketEndConfirm.req(b)
SimulationDone
n := call findIndex(ID);
if (n > 0) MVehicleList(n)!pos := pos;
–
–
PacketStart.req(ID)
PacketEnd.req
(bcastID,ID,packet)
n := call findIndex(ID);
BroadcastEnd(bcastID,
packet,received,collided)
n
= 0
else
sResults(MAC:
(. ID,bcastID,received,collided .))
nBroadcast :=
nBroadcast+1;
–
BroadcastStart
(ID,MVehicleList(n)!pos)
PacketStartConfirm.ind
(nBroadcast) TO sender
PacketStartConfirm.ind(0)
TO sender
–
–

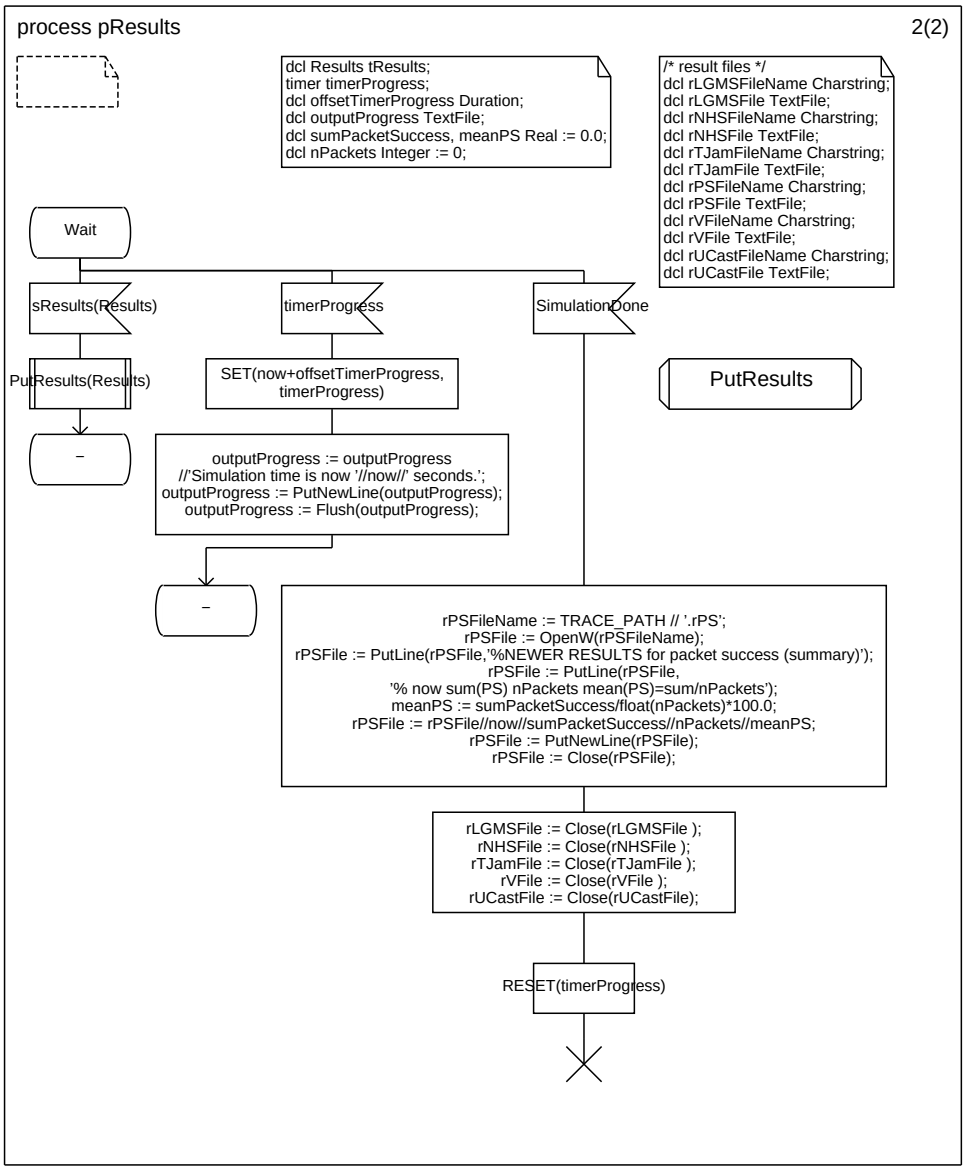

procedure BroadcastStart
1(1)
;fpar
in senderID Integer,
in senderPos tPosition;
/* Broadcast the signal PacketStart.ind to all active vehicles within the transmission range except from the vehicle itself. Save all IDs of receivers to list of broadcasts. */
dcl i Integer;
dcl dist Real;
i := 1;
i > nVehicles
true
false
MVehicleList(i)!ID = senderID
true
false
vehicle inactive?
MVehicleList(i)!pos!x = −1.0
true
false
dist := call calcDistance (MVehicleList(i)!pos!x,senderPos!x, MVehicleList(i)!pos!y,senderPos!y);
dist > transmissionRange
true
false
PacketStart.ind to MVehicleList(i)!pMAC
BroadcastList(nBroadcast):= incl(MVehicleList(i)!ID,BroadcastList(nBroadcast));
i := i+1

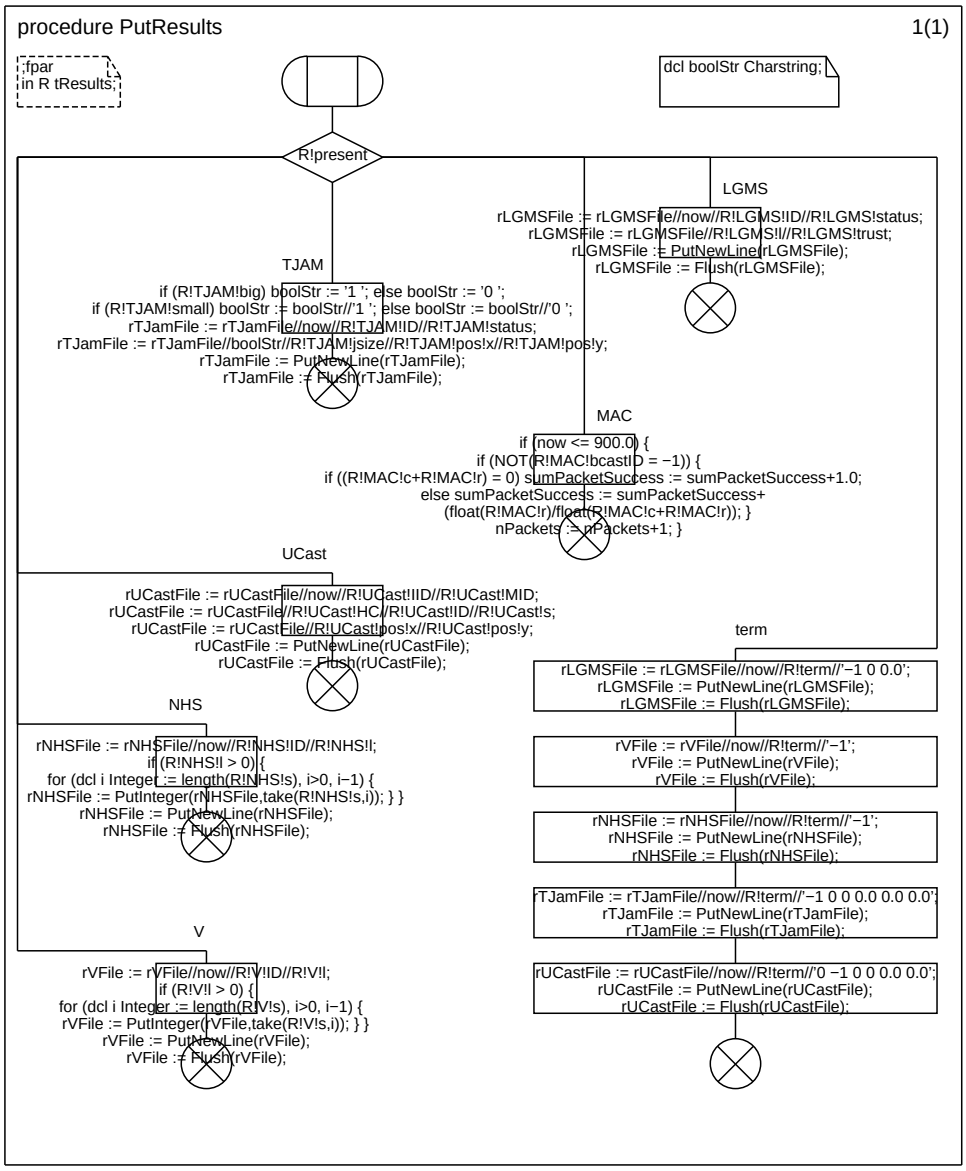

procedure BroadcastEnd
1(1)
/fpar
in BC_ID Integer,
in P tPacket,
in/out nReceived, nCollided Integer;
/* Broadcast "P" with signal PacketEnd.ind to all active vehicles that are member of set of broadcast indicated by "BC_ID" */
dcl i Integer;
dcl isReceived Boolean;
nReceived := 0;
nCollided := 0;
BroadcastList(BC_ID)
empty
else
i := 1;
i > nVehicles
false
true
BroadcastList(BC_ID)
:= empty;
MVehicleList(i)!ID
IN BroadcastList(BC_ID)
false
true
vehicle
inactive?
MVehicleList(i)!pos!x
= −1.0
true
false
PacketEnd.ind(P)
to MVehicleList(i)!pMAC
WaitForStatus
SimulationDone
*
PacketEndConfirm.req
(isReceived)
SimulationDone
TO self
if(isReceived) nReceived := nReceived+1;
else nCollided := nCollided+1;
i := i+1

C.3.4 Results


cMACResults
cResults
block bResults
1(1)
rMACResults
rResults
sResults
sResults
rControlResults
cControlResults
pResults
SimulationDone


process pResults
1(2)
rLGMSFileName := TRACE_PATH // '.rLGMS';
rLGMSFile := OpenW(rLGMSFileName);
rLGMSFile := PutLine(rLGMSFile,'%RESULTS for view_chg()');
rLGMSFile := PutLine(rLGMSFile,'% time id state(1:join 2:leave 0:install −1:terminate) "view size" trust');
rLGMSFile := Flush(rLGMSFile);
rVFileName := TRACE_PATH // '.rView';
rVFile := OpenW(rVFileName);
rVFile := PutLine(rVFile,'%RESULTS for views');
rVFile := PutLine(rVFile,'% time id "size of view"(−1:terminate) view');
rVFile := Flush(rVFile);
rNHSFileName := TRACE_PATH // '.rNHS';
rNHSFile := OpenW(rNHSFileName);
rNHSFile := PutLine(rNHSFile,'%RESULTS for list_chg()');
rNHSFile := PutLine(rNHSFile,'% time id "size of list"(−1:terminate) "list in list_vhg()"');
rNHSFile := Flush(rNHSFile);
rTJamFileName := TRACE_PATH // '.rTJam';
rTJamFile := OpenW(rTJamFileName);
rTJamFile := PutLine(rTJamFile,'%RESULTS for TJam');
rTJamFile := PutLine(rTJamFile,
'% time id status(−1:terminate, 1:classify, 2: receive) AT_BEGIN(0:false 1:true) AT_END(0:false 1:true) "size of jam" x y');
rTJamFile := Flush(rTJamFile);
rUCastFileName := TRACE_PATH // '.rUCast';
rUCastFile := OpenW(rUCastFileName);
rUCastFile := PutLine(rUCastFile,'%RESULTS for unicast()');
rUCastFile := PutLine(rUCastFile,'% time initiatorID msgID hopCount(−1: terminate 0:initiate >0:accept/resend)...');
rUCastFile := PutLine(rUCastFile,'% ... acceptID/resendID status(0:accept 1:resend 2:ATBEGIN 3:ATEND 4:ALONE) x y');
rUCastFile := Flush(rUCastFile);
SET(1.0,timerProgress);
offsetTimerProgress := (SimulationTime−0.0)/10.0;
outputProgress := stdout;outputProgress := PutNewLine(outputProgress);
outputProgress := outputProgress//'TOTAL Simulation time is '//SimulationTime//' seconds.';
outputProgress := PutNewLine(outputProgress); outputProgress := PutNewLine(outputProgress);
outputProgress := Flush(outputProgress);
Wait


process pResults
2(2)
dcl Results tResults;
timer timerProgress;
dcl offsetTimerProgress Duration;
dcl outputProgress TextFile;
dcl sumPacketSuccess, meanPS Real := 0.0;
dcl nPackets Integer := 0;
/* result files */
dcl rLGMSFileName Charstring;
dcl rLGMSFile TextFile;
dcl rNHSFileName Charstring;
dcl rNHSFile TextFile;
dcl rTJamFileName Charstring;
dcl rTJamFile TextFile;
dcl rPSFileName Charstring;
dcl rPSFile TextFile;
dcl rVFileName Charstring;
dcl rVFile TextFile;
dcl rUCastFileName Charstring;
dcl rUCastFile TextFile;
Wait
sResults(Results)
timerProgress
SimulationDone
PutResults(Results)
SET(now+offsetTimerProgress, timerProgress)
PutResults
–
outputProgress := outputProgress //'Simulation time is now '//now// seconds.';
outputProgress := PutNewLine(outputProgress);
outputProgress := Flush(outputProgress);
–
rPSFileName := TRACE_PATH // '.rPS';
rPSFile := OpenW(rPSFileName);
rPSFile := PutLine(rPSFile,'%NEWER RESULTS for packet success (summary)');
rPSFile := PutLine(rPSFile,
'% now sum(PS) nPackets mean(PS)=sum/nPackets');
meanPS := sumPacketSuccess/float(nPackets)*100.0;
rPSFile := rPSFile//now//sumPacketSuccess//nPackets//meanPS;
rPSFile := PutNewLine(rPSFile);
rPSFile := Close(rPSFile);
rLGMSFile := Close(rLGMSFile);
rNHSFile := Close(rNHSFile);
rTJamFile := Close(rTJamFile);
rVFile := Close(rVFile);
rUCastFile := Close(rUCastFile);
RESET(timerProgress)


procedure PutResults
1(1)
fpar
in R tResults;
dcl boolStr Charstring;
R!present
LGMS
rLGMSFile := rLGMSFile//now//R!LGMS!ID//R!LGMS!status;
rLGMSFile := rLGMSFile//R!LGMS!//R!LGMS!trust;
rLGMSFile := PutNewLine(rLGMSFile);
rLGMSFile := Flush(rLGMSFile);
TJAM
if (R!TJAM!big) boolStr := '1 '; else boolStr := '0 ';
if (R!TJAM!small) boolStr := boolStr//'1 '; else boolStr := boolStr//'0 ';
rTJamFile := rTJamFile//now//R!TJAM!ID//R!TJAM!status;
rTJamFile := rTJamFile//boolStr//R!TJAM!jsize//R!TJAM!pos!x//R!TJAM!pos!y;
rTJamFile := PutNewLine(rTJamFile);
rTJamFile := Flush(rTJamFile);
MAC
if (now <= 900.0) {
if (NOT(R!MAC!bcastID = −1)) {
if ((R!MAC!c+R!MAC!r) = 0) sumPacketSuccess := sumPacketSuccess+1.0;
else sumPacketSuccess := sumPacketSuccess+
(float(R!MAC!r)/float(R!MAC!c+R!MAC!r)); }
nPackets := nPackets+1; }
UCast
rUCastFile := rUCastFile//now//R!UCast!IID//R!UCast!MID;
rUCastFile := rUCastFile//R!UCast!HC//R!UCast!ID//R!UCast!s;
rUCastFile := rUCastFile//R!UCast!pos!x//R!UCast!pos!y;
rUCastFile := PutNewLine(rUCastFile);
rUCastFile := Flush(rUCastFile);
term
rLGMSFile := rLGMSFile//now//R!term//'−1 0 0.0';
rLGMSFile := PutNewLine(rLGMSFile);
rLGMSFile := Flush(rLGMSFile);
NHS
rNHSFile := rNHSFile//now//R!NHS!ID//R!NHS!l;
if (R!NHS!l > 0){
for (dcl i Integer := length(R!NHS!s), i>0, i−1) {
rNHSFile := PutInteger(rNHSFile,take(R!NHS!s,i)); } }
rNHSFile := PutNewLine(rNHSFile);
rNHSFile := Flush(rNHSFile);
rVFile := rVFile//now//R!term//'−1';
rVFile := PutNewLine(rVFile);
rVFile := Flush(rVFile);
rNHSFile := rNHSFile//now//R!term//'−1';
rNHSFile := PutNewLine(rNHSFile);
rNHSFile := Flush(rNHSFile);
rTJamFile := rTJamFile//now//R!term//'−1 0 0 0.0 0.0 0.0';
rTJamFile := PutNewLine(rTJamFile);
rTJamFile := Flush(rTJamFile);
V
rVFile := rVFile//now//R!V!ID//R!V!l;
if (R!V!l > 0) {
for (dcl i Integer := length(R!V!s), i>0, i−1) {
rVFile := PutInteger(rVFile,take(R!V!s,i)); } }
rVFile := PutNewLine(rVFile);
rVFile := Flush(rVFile);
rUCastFile := rUCastFile//now//R!term//'0 −1 0 0 0.0 0.0';
rUCastFile := PutNewLine(rUCastFile);
rUCastFile := Flush(rUCastFile);

Bibliography

[Abr70] N. Abramson. The ALOHA system – another alternative for computer communications. In *Proceedings of the Fall 1970 AFIPS Computer Conference*, volume 37, pages 281–285, 1970. 126

[ACBMT95] Emmanuelle Anceaume, Bernadette Charron-Bost, Pascale Minet, and Sam Toueg. On the formal specification of group membership services. Technical Report TR95-1534, Cornell University, Computer Science Department, August 25, 1995. 48, 72, 73, 74

[AF96] Masayoshi Aoki and Haruki Fujii. Inter-vehicle communication: Technical issues on vehicle control application. *IEEE Communications Magazine*, 34(10):90–93, October 1996. 18

[BDM98] Özalp Babaoğlu, Renzo Davoli, and Alberto Montresor. Group communication in partitionable systems: Specification and algorithms. Technical Report UBLCS-98-1, University of Bologna, Italy. Department of Computer Science., April 1998. 15, 47, 52, 52, 52, 53, 73, 73, 75, 75

[BDMS98] Özalp Babaoğlu, Renzo Davoli, Alberto Montresor, and Roberto Segala. System support for partition-aware network applications. In *18th International Conference on Distributed Computing Systems*, pages 184–191, May 1998. 52

[BH00] Linda Briesemeister and Günter Hommel. Overcoming fragmentation in mobile ad hoc networks. *Journal of Communications and Networks.*, 2(3):182–187, September 2000. ISSN 1229-2370. 24, 38

[Bir93] Kenneth P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, December 1993. 48

[BJM98] J. Broch, D. Johnson, and D. Maltz. The dynamic source routing protocol for mobile ad hoc networks, December 1998. IETF Internet Draft (work in progress). http://www.ietf.org/internet-drafts/draft-ietfmanet-dsr-01.txt. 34

[BMJ+98] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *4th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 85–97, October 1998. 23, 24

[BMSU99] Prosenjit Bose, Pat Morin, Ivan Stojmenovic, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Third International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 48–55, August 1999. 23, 30, 31

[BSH00] Linda Briesemeister, Lorenz Schäfers, and Günter Hommel. Disseminating messages among highly mobile hosts based on inter-vehicle communication. In *IEEE Intelligent Vehicles Symposium*, pages 522–527, October 2000. 18, 43, 43

[CGZ98] C.-C. Chiang, M. Gerla, and L. Zhang. Forwarding group multicast protocol (FGMP) for multihop, mobile wireless networks. *ACM-Baltzer Journal of Cluster Computing: Special Issue on Mobile Computing*, 1(2), 1998. 23

[Cha93] J. M. Charnes. Statistical analysis of output processes. In *Winter Simulation Conference*, pages 41–49, December 1993. 106

[CHTCB96] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernadette Charron-Bost. On the impossibility of group membership. In *15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 322–330, May 1996. 20, 69, 74

[CL00] Daniel Câmara and Antonio Alfredo F. Loureiro. A novel routing algorithm for ad hoc networks. In *33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, USA, January 2000. 23, 29

[DMS95] Danny Dolev, Dalia Malki, and Ray Strong. A framework for partitionable membership service. Technical Report 95-4, Institute of Computer Science, The Hebrew University of Jerusalem, March 1995. (16 pages). 48

[EGHK99] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *5th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 263–270, 1999. 30

[EHS97] Jan Ellsberger, Dieter Hogrefe, and Amardeo Sarma. *SDL – Formal Object-oriented Language for Communicating Systems.* Prentice Hall, 1997. 78

[EMS95] Paul D. Ezhilchelvan, Raimundo A. Macêdo, and Santosh K. Shrivastava. Newtop: A fault-tolerant group communication protocol. In *15th International Conference on Distributed Computing Systems (ICDCS'95)*, pages 296–306, May 30–June 2 1995. 48, 48

[Ens] The Ensemble Distributed Communication System. Department of Computer Science, Cornell University, http://www.cs.cornell.edu/Info/Projects/Ensemble/. 48

[EWB87] A. Ephremides, J. Wieselthier, and D. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE*, 75(1):56–73, January 1987. 25

[FLS] Alan Fekete, Nancy Lynch, and Alex Shvartsman. Specifying and using a partitionable group communication service. To appear in ACM Transactions on Computer Systems, http://www.toc.lcs.mit.edu/tds/. 49

[FLS97] Alan Fekete, Nancy Lynch, and Alex Shvartsman. Specifying and using a partionable group communication service. In *16th*

Annual ACM Symposium on Principles of Distributed Computing, pages 53–62, 21–24 August 1997. 49

[GF97] Ottmar Gehring and Hans Fritz. Lateral control concepts for truck platooning in the CHAUFFEUR project. In *4th World Congress on Intelligent Transport Systems*, October 1997. 18

[Gol92] D. Goldsman. Simulation output analysis. In *Winter Simulation Conference*, pages 97–103, December 1992. 106

[GT95] M. Gerla and J. Tzu-Chieh Tsai. Multicluster, mobile, multimedia radio network. *ACM/Baltzer Wireless Networks*, 1(3):255–265, August 1995. 25

[Har62] Frank Harary. The maximum connectivity of a graph. *Proc. of the National Academy of Sciences*, 48:1142–1146, 1962. 26

[HMP91] Tom Henzinger, Zohar Manna, and Amir Pnueli. Temporal proof methodologies for real-time systems. In *18th ACM Symposium on Principles of Programming Languages*, pages 353–366, January 1991. 54, 54

[HOTV99] Christopher Ho, Katia Obraczka, Gene Tsudik, and Kumar Viswanath. Flooding for reliable multicast in multi-hop ad hoc networks. In *3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DialM)*, pages 64–71, August 1999. 23

[IET] Internet Engineering Task Force (IETF). Mobile Ad Hoc Networks (MANET) Working Group Charter. http://www.ietf.org/html.charters/manet-charter.html. 17

[IGE00] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *6th Annual International Conference on Mobile Computing and Networking*, pages 56–67, August 2000. 26

[IN99] Tomasz Imielinski and Julio C. Navas. GPS-based geographic addressing, routing, and resource discovery. *Communications of the ACM*, 42(4):86–92, April 1999. 24, 27

[ITU99] International Telecommunication Union. *ITU-T Recommendation Z.100. Specification and description language (SDL)*, September 1999. 77

[Jan98] Stefan Janz. Mikroskopische Minimalmodelle des Straßenverkehrs. Master's thesis, Mathematisch Naturwissenschaftliche Fakultät, Universität zu Köln, May 1998. 98

[Jgr] The Jgroup Project. Department of Computer Science, University of Bologna, http://www.cs.unibo.it/projects/jgroup/. 48, 52

[JPS99] Rahul Jain, Anuj Puri, and Raja Sengupta. Geographical routing using partial information for wireless ad hoc networks. Technical Report M99/69, University of California, Berkeley, December 20, 1999. 34, 36, 42

[KK92] Werner Kremer and Wolfgang Kremer. Vehicle density and communication load estimation in mobile radio local area networks (MR-LANs). In *42nd Vehicular Technology Society Conference*, volume 2, pages 698–704, May 1992. 18

[KK00] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *6th Annual International Conference on Mobile Computing and Networking*, pages 243–254, August 2000. 24, 33, 34, 36, 42, 124, 130

[KL75] L. Kleinrock and S. S. Lam. Packet switching in a multiaccess broadcast channel: performance evaluation. *IEEE trans. on commun.*, 23(4):410–422, April 1975. 126

[Kra97] Stefan Krauß. Microscopic traffic simulation: Robustness of a simple approach. Technical report, Deutsches Zentrum für Luft- und Raumfahrt, 1997. 98

[Kra98] Stefan Krauß. *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics*. PhD thesis, Mathematisches Institut, Universität zu Köln, April 1998. 98, 100, 100, 100

[KSMD99] Idit Keidar, Jeremy Sussman, Keith Marzullo, and Danny Dolev. A client-server oriented algorithm for virtually synchronous group membership in WANs. Technical Report CS1999-0623, University of California, San Diego, Computer Science and Engineering, June 1999. 50

[KV98] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. Technical Report TR98-012, Texas A&M University, June 1, 1998. 27, 28, 29

[KV99] Young-Bae Ko and Nitin H. Vaidya. Geocasting in mobile ad hoc networks: Location based multicast algorithms. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, February 1999. 28, 38, 130

[KV00] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. *Wireless Networks*, 6(4):307–321, July 2000. 27

[Li95] Wei-Yi Li. Design and implementation of digital radio communications link for platoon control experiments. PATH Research Report UCB-ITS-PRR-95-2, University of California, Berkeley, January 1995. 18

[LJC+00] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *6th Annual International Conference on Mobile Computing and Networking*, pages 120–130, August 2000. 31, 33

[LMM99] Meng-Jang Lin, Keith Marzullo, and Stefano Masini. Gossip versus deterministic flooding: Low message overhead and high reliability for broadcasting on small networks. Technical Report CS1999-0637, University of California, San Diego, Computer Science and Engineering, November 18, 1999. 25

[LR00] Qun Li and Daniela Rus. Sending messages to mobile users in disconnected ad-hoc wireless networks. In *6th Annual International Conference on Mobile Computing and Networking*, pages 44–55, August 2000. 36

[MJK+00] Robert Morris, John Jannotti, Frans Kaashoek, Jinyang Li, and Douglas S. J. De Couto. CarNet: A scalable ad hoc wireless network system. In *Proceedings of the 9th ACM SIGOPS European workshop: Beyond the PC: New Challenges for the Operating System*, Kolding, Denmark, September 2000. 18, 32

[Mob00] The First Annual Workshop on Mobile Ad Hoc Networking & Computing, August 2000. http://www.cs.tamu.edu/faculty/vaidya/mobihoc/2000/workshop.html. 23

[Mob01] ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc), October 2001. http://www.cs.tamu.edu/faculty/vaidya/mobihoc/2001/main.html. 23

[Mon00] Alberto Montresor. *System Support for Programming Object-Oriented Dependable Applications in Partitionable Systems*. PhD thesis, University of Bologna, Italy, March 2000. Technical Report UBLCS-2000-10. 52, 52

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992. 21, 53, 54, 70, 145

[NI97] Julio C. Navas and Tomasz Imielinski. GeoCast – geographic addressing and routing. In *3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 66–76, September 1997. 27, 28

[ns2] The Network Simulator – ns-2 http://www.isi.edu/nsnam/ns/. 27

[OKFF98] Masaya Ohtomo, Ryouji Kimura, Shigeki Fukushima, and Noboru Fujii. Automatic following system utilizing vehicle-to-vehicle communication. In *IEEE International Conference on Intelligent Vehicles*, pages 381–384, October 1998. 18

[PB94] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Com-*

munications Architectures, Protocols and Applications, pages 234–244, August 1994. 35

[PB98] R. Prakash and R. Baldoni. Architecture for group communication in mobile systems. In *17th IEEE Symposium on Reliable Distributed Systems*, pages 235–244, October 1998. 48

[PR97] Elena Pagani and Gian Paolo Rossi. Reliable broadcast in mobile multihop packet networks. In *Third Annual ACM/IEEE International Conference on Mobile Computing and Networking*, September 1997. 24

[PR99] Elena Pagani and Gian Paolo Rossi. Providing reliable and fault tolerant broadcast delivery in mobile ad-hoc networks. *Mobile Networks and Applications*, 4(3):175–192, 1999. 24

[RG00] Luís Rodrigues and Katherine Guo. Partitionable light-weight groups. In *20th IEEE International Conference on Distributed Computing Systems (ICDCS'20)*, pages 38–45, April 2000. 51

[RT99] Elizabeth Royer and C-K Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications Magazine*, pages 46–55, April 1999. 23, 24

[SDL] The SDL Forum Society. http://www.sdl-forum.org. 78

[SL99] Ivan Stojmenovic and Xu Lin. GEDIR: Loop-free location based routing in wireless networks. In *IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 1025–1028, November 1999. 30

[SL00] Ivan Stojmenovic and Xu Lin. Power aware localized routing in wireless networks. In *IEEE International Parallel and Distributed Processing Symposium*, pages 371–376, May 2000. 30

[Spr] The Spread Wide Area Group Communication system. Center for Networking and Distributed Systems, Johns Hopkins University, http://www.spread.org/. 48

[SR00] Ivan Stojmenovic and Mark Russell. Depth first search and location based localized routing and QoS routing in wireless

networks. In *IEEE International Conference on Parallel Processing*, pages 173–180, August 2000. 24, 30, 31, 42

[SV93] Sonia R. Sachs and Pravin Varaiya. A communication system for the control of automated vehicles. PATH Technical Memorandum 93-5, University of California, Berkeley, September 1993. 18

[TK84] H. Takagi and L. Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. *IEEE Transactions on Communications*, 32(3):246–257, March 1984. 36

[Tra] The Transis Project Home Page. Computer Science Department, The Hebrew University of Jerusalem, http://www.cs.huji.ac.il/~transis/. 48

[VB00] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-200006, Duke University, April 2000. 37

[WRW98] Christopher K. H. Wilson, Seth Rogers, and Shawn Weisenburger. The potential of precision maps in intelligent vehicles. In *IEEE International Conference on Intelligent Vehicles*, pages 419–422, October 1998. 39

[ZS00] Hu Zhou and Suresh Singh. Content based multicast (CBM) in ad hoc networks. In *First Annual Workshop on Mobile Ad Hoc Networking and Computing*, pages 51–60, August 2000. 35, 42

The numbers at the end of each entry are back references to the pages on which the citation occurs.