# Formal Description of a Distributed Location Service for Mobile Ad Hoc Networks

András Benczúr[1], Uwe Glässer[2] and Tamás Lukovszki[3]

[1]Computer Science Research Institute of the Hungarian Academy of Sciences,
benczur@sztaki.hu
[2]School of Computing Science, Simon Fraser University, BC, Canada,
glaesser@cs.sfu.ca
[3]Heinz Nixdorf Institute, Paderborn University, Germany,
tamas@hni.upb.de

**Abstract.** We define here a *distributed abstract state machine* (DASM) [7] of the network or routing layer of mobile ad hoc networks [13]. Such networks require routing strategies substantially different from those used in static communication networks, since storing and updating large routing tables at mobile hosts would congest the network with administration packets very fast. In [1], the *hypercubic location service* is presented, which considers a very strong definition of fault-tolerance thereby improving state-of-the-art ad hoc routing protocols in several respects. Our goal in modeling the protocols for the distributed location service and the position based routing is twofold. First, we support the definition and validation of wireless communication protocols and implementations based thereon. Second, we feel that the abstract computation model naturally reflects the *layering principle* of communication architectures in combination with an uncompromisingly local view of the application domain. Thus we can identify fundamental semantic concepts, such as concurrency, reactivity and asynchronism, directly with the related concepts as imposed by the given application context.

## 1 Introduction

In this paper, we define a *distributed abstract state machine* (DASM) model [7] of the network or routing layer protocol for mobile ad hoc networks. Such networks are designed for wireless communication and require no fixed infrastructure as the mobile hosts also perform routing tasks [13]. Thus, they are particularly suitable, for instance, for rescue operations in regions affected by a natural disaster. The dynamics of self-organizing mobile networks require routing strategies substantially different from those used in static communication networks. Storing and updating large routing tables at mobile hosts would congest the network with administration packets very fast.

Ad hoc routing received a lot of attention in recent years [13]. Several routing protocols have been developed. The so-called *proactive* protocols, like DSDV [15], continuously maintain route information for all destinations, whereas the *reactive* ones construct the routes for the destinations as required, e.g. DSR [9] and TORA [14]. Neither type of those strategies is fully satisfying however: the proactive ones suffer

from a congestion of administrative messages when devices begin to move relative fast; reactive strategies in contrast have the drawback of long network search routes and thus high initiation costs.

Li et al. [11] present a routing strategy which combines the advantages of proactive and reactive routing. Their *geographic location service* (GLS), for each mobile node, consists of a few other randomly selected nodes that act as its location servers. Based on the assumption that every node can determine its current geographic position at any time—e.g., by using a global positioning system or GPS—, the nodes periodically resend position information to the location servers by the same routing mechanism used for normal data transmission, e.g., simple geographic forwarding. To initiate communication, we find a location server of the destination node and obtain its geographic position. A location server can be determined by visiting a well defined sequence of nodes such that each node acts as a location server of the next node in the sequence.

In [1], the *hypercubic location service* is presented, which considers a very strong definition of fault-tolerance, thereby improving state-of-the-art ad hoc routing protocols in several respects. Our goal in modeling the protocols for the distributed location service and the position based routing is twofold. First, we support the definition and validation of wireless communication protocols and implementations based thereon. Second, we feel that the abstract computation model naturally reflects the *layering principle* of communication architectures [2] in combination with an uncompromisingly local view of the application. Thus we can identify fundamental semantic concepts, such as concurrency, reactivity and asynchronism, directly with the related concepts as imposed by the given application context.

The paper is organized as follows. Section 2 briefly outlines the characteristic features of the applied protocols and the abstract representation of the network topology for geographic ad hoc routing. Section 3 introduces the abstract data structures and computation model underlying our mathematical definition of the network model. The protocols for position based routing and the distributed location service are formally defined in Section 4 and Section 5 respectively. Section 6 concludes the paper.

## 2 Efficient Ad Hoc Routing

The routing protocol in [1] is divided into two sublayers, one for the location service and one for position based routing between known locations, as illustrated in Fig. 1. These two sublayers cooperatively form the network layer of a mobile ad hoc network. The location service is based on a highly fault tolerant, self-scaling architecture called *hypercubic location service* or HLS. For the position based routing sublayer, the *Yao-graph* is used, which contains approximate power minimal routing paths.
In order to describe the maintenance of the network and the routing protocol, the existence of a *Media Access* or MAC layer, e.g. IEEE 802.11, is assumed to resolve interference problems. Furthermore, it is assumed that the mobile hosts use directed radio so that they can increase the transmission power (in discrete steps) in a direction until at least one node eventually reacts by sending an acknowledge signal including its current geographic position.

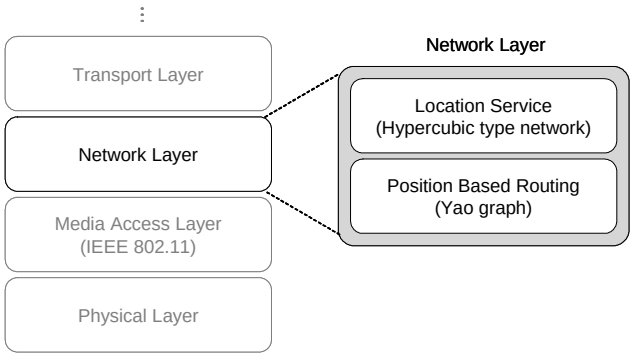


**Fig. 1.** Layered communication architecture

### 2.1 Hypercubic Location Service

The hypercubic location service is presented in [1] and keeps most of the features of GLS [11], but also improves GLS in several ways. Here are the main aspects:

- It needs very little assumption on how the density varies across geographic subregions.
- It provides a method for very fast network topology recovery even after a failure of several mobile nodes—a task that requires reactive routing in GLS.
- It provides very strong fault tolerance, still keeping degrees as low as in the GLS.
- The network delivers a packet with a probability $1–n^{-c}$ even in the case when a constant fraction of the nodes crash at the same time.
- It survives a systematic destruction of devices within a region.
- It includes the self-scaling feature of the network.
- The location service forms a sublayer or the network layer, thus providing the flexibility of combining HLS with an arbitrary position based routing strategy.
- It keeps the administrative traffic close to that of [11] with a slight loss in the locality of communication, which can be fixed by a multi-layer solution.

Conceptually, HLS forms a sublayer of the network layer and consists of a dynamic and fault tolerant *hypercubic network* [10] with random IDs of the mobile devices identifying the nodes. Random ID distribution can be easily achieved by either hashing the node's IP or other physical address, or hard-wiring a random sequence into the device. The nodes are organized in a hypercubic network with respect to their

random ID. In such networks neighbor IDs can be obtained by simple bit shift and flip operations of IDs. Each node in the network maintains location information of its hypercubic neighbors by periodically sending its own location to those neighbors. When the location of a node is known, a position based routing strategy can be used to pass messages to that node.

When a node *u* wants to send data to another node *v*, first *u* has to detect the location of node *v*. This happens as follows: The random ID of *v* can be obtained from the random ID of *u* by performing a certain sequence of elementary bit operations. This sequence defines a path in the hypercubic network from *u* to *v* [10]. Node *u* sends a so-called *location request* for *v* on this path which contains the ID of *u* and *v* as well as the location of *u*.

Since the hypercubic neighbors are storing up-to-date location information from each other, the location request will be sent to the next node on the hypercube path using a certain position based routing. When the location request reaches *v*, then *v* sends its own location to *u* by using position based routing. After *u* has received the position of *v* it sends the data to *v* using the position based routing, as well.

As shown in [1], the random distribution of mobile node ID's ensures two key feature of the HLS:

- The location operations and the storage requirements are balanced across all the nodes.
- Since the IDs are independent of the geographic location or any other sensible network property, HLS resists faults of almost all nodes in a certain region, or a systematical demolition of a constant fraction of nodes.[1]

### 2.2 Position Based Routing

In order to route the message between two consecutive nodes of the hypercube path, any position based routing can be used, and, in particular, any appropriate geometric graph, such as spanner graphs [4], can be used as underlying network topology with any suitable routing strategy on that graph. In [1] the so-called *Yao-graph* [17] is proposed as network topology, because the following properties.

- Each node has a constant number of outgoing edges as illustrated in Fig. 2.
- The graph can be computed and maintained efficiently in a distributed manner (see [16] for example), allowing fast recovery from faults.
- It is a so-called power spanner [12] that allow low power routing and/or a position based routing where each node can determine the next node of the a path only knowing the geographic position of the destination node.

[1] The location service can be adapted to the required scale of fault tolerance. The more storage per node is allowed for storing geographic positions of other nodes the higher is the scale of fault tolerance that can be reached. HLS is planned for resisting even against the simultaneous fault of a constant fraction of the nodes. In that case a Packet will find the destination with a polynomial probability of $1-n^{-c}$ for a given constant $c$ (see [1] for further details).

The Yao-graph $G_Y(V)=(V,E_Y)$ for a set $V$ of $n$ points in the plane is defined as follows (see Fig. 2a for an example). Let $k$ be an integer and $\theta = 2\pi / k$. Let $h_i$, $i = 0,...,k-1$, be the half-line coincident with the rotated positive $x$-axis around the origo by an angle $\theta$, and let $c_i$ be the cone between $h_i$ and $h_{i+1 \bmod k}$. For a point $p \in \mathbb{R}^2$ let $c_i(p)$, $i = 0,...,k-1$, be the translated cone $c_i$ whose apex is at $p$. For each $p \in V$, let $\alpha_p$ be an angle, $0 \le \alpha_p < \theta$, and let $c_i'(p)$ be the rotated cone $c_i(p)$ around $p$ by an angle $\alpha_p$. We call the cones $c_i'(p)$ the *sectors* of $p$. For $p \in V$ and $c_i'(p)$, $i = 0,...,k-1$, let $n_i(p) \in S$ be the Euclidean nearest neighbor of $p$ in $c_i'(p)$, if exists, and connect $p$ to $n_i(p)$ by a directed edge (see Fig. 2b), i.e. $E_Y = \{ (p, n_i(p) : 0 \le i < k, c_i'(p) \cap V \neq \emptyset \}$.

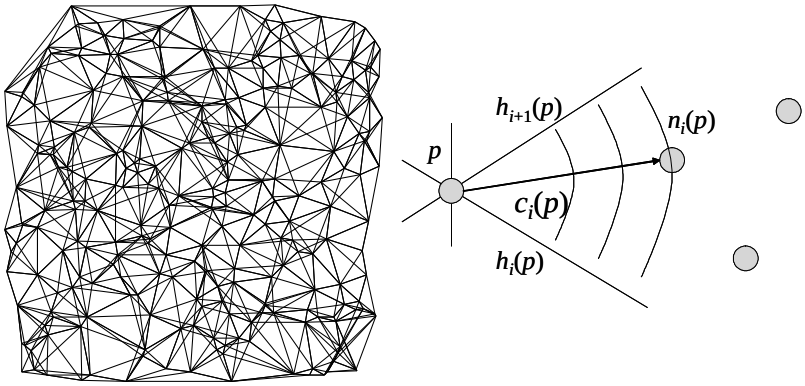


**Fig. 2a.** Yao graph for $k = 6$ with 256 nodes distributed uniformly at random within a square
**Fig. 2b.** Sector $c_i(p)$ with nearest neighbor $n_i(p)$

We remark that the Yao-graph neighbors can be determined efficiently in a distributed and power efficient manner, as described in [16] for a variation of the Yao-graph. This distributed construction assumes that the mobile nodes use directed radio and they can increase the transmission power (in discrete steps) in the sectors until at least one node hears them and sends an acknowledge signal which also can contain its geographic position. Then the node only has to choose the closest one among the responding nodes in each sector. Here we assume for simplicity that each node is within the maximum transmission range of each other node.

## 3 Abstract Network Model

Consider a distributed communication architecture consisting of some finite number of mobile hosts interconnected through a wireless communication network. The global topology is formed by viewing the mobile hosts as *nodes* of the network. Each node has the ability to act as communication endpoint and as router at the same time. The network is ad hoc and as such changes its topology frequently and without prior notice. Nodes operate concurrently and interact with each other by asynchronously sending and receiving *packets* of variable length.

Conceptually, we distinguish two basically different views of communication, commonly referred to as the *horizontal* and the *vertical* view of communication. Any direct communication between peer protocol entities (i.e., entities residing within the same layer) at different nodes is called horizontal communication. This form of communication is only virtual and effectively realized by the lower level network layers. Therefore, the MAC layer renders a corresponding service to the network layer. This service is accessible through well defined interfaces, called *service access points*. Similarly, the network layer renders its service to the next higher layer, the transport layer. Communication between different layers of the same node is called vertical communication. This form of communication is restricted to adjoining layers.

### 3.1 Concurrency and Real Time

We describe the concurrent and reactive behavior of network protocol entities by an asynchronous computation model in the form of a DASM. Autonomously operating protocol entities are *agents* interacting with each other, and also with the external world, by reading and writing shared locations of global machine states. The underlying semantic model regulates such interactions so that potential conflicts are resolved according to the definition of *partially ordered runs* [7].

Additionally, certain real time aspects need to be taken into account as well. Real time behavior imposes additional constraints on DASM runs ensuring that the agents react instantaneously [8]. To model the timing behavior faithfully, we introduce a notion of *local* system time in combination with mechanisms for handling timeout events. Our semantic model of time resembles those defined, e.g., in [3],[5],[8], except that we deliberately avoid a notion of global system time.

We construct our network model based on an intuitive level of understanding with a degree of detail and precision oriented towards practical needs in applied systems engineering. As such, our model serves as a basis for development and experimental validation. For a rigorous mathematical definition of the underlying computation model, we refer to the original literature on the theory of ASMs [7]. For a tutorial introduction to the DASM paradigm of systems modeling see also [6].

### 3.2 Communication Infrastructure

The total number *N* of nodes is a parameter of the communication network. Each node has a coordinate identifying its geographic *position* on the plane. As nodes may (and usually do) move, their positions change over time. The current position of a node is always known and the node can access this information locally through an interface to some location system, for instance, such as a global positioning system or GPS.

```
domain NODE = {n1, ..., nN}
domain POSITION, monitored pos : NODE → POSITION
```

Every node has a unique ID or *address* within a fixed global address space. We assume a random distribution of the node IDs. This distribution is ensured by mapping some device specific address, e.g. such as Internet host names, IP addresses, or MAC

addresses, to the node IDs using a strong hash function (not further specified here). Consider the resulting addresses as given by a static mapping from nodes to some abstract domain of addresses.[2]

```
domain ADDRESS, address : NODE → ADDRESS
```

We model the network layer by identifying the underlying protocol entities with corresponding DASM agents. Each node has two autonomously operating agents, one for the distributed location service and one for the position based routing. These agents execute the program *DistributedLocationService* and *PositionBasedRouting* as stated by a unary dynamic function *program* defined on agents. Thus, the domain *AGENT* is formed by the disjoint union of two sets of agents to which we refer as *DLS* and *PBR*. (Note that the function *program* is undefined on all those elements of the base set not representing agents.)

```
domain AGENT = DLS ∪ PBR, node : AGENT → NODE
```

In any given state of the network, the status of a node is either *switched on* (active), *switched off* (passive), or *undefined* (crashed). This status can change dynamically at run time depending on actions and events in the operational environment into which the network is embedded. For crashed nodes we assume that they do not interact with any other nodes. Consequently, they do not have any observable behavior and, as such, are not considered as agents of the DASM. Nonetheless, crashed nodes may recover becoming active or passive again. Recovery of crashed nodes is completely under the control of the external world and not further addressed here.

```
domain STATUS = { switched_on, switched_off }

monitored status : NODE → STATUS
```

For the representation of the local system time as associated with the individual mobile hosts, we introduce a monitored function *now* defined on nodes. Time values are given by a linearly ordered domain *TIME*. Intuitively, *now* represents the time as measured by some local clock. We do not state any constraints on the accuracy for measuring the local time with regard to the overall global system time.

```
domain TIME, now : NODE → TIME
```

### 3.3 Abstract Communication Model

Nodes send and receive packets consisting of two basically distinct parts, a packet *header* and the actual *payload*, as illustrated in Fig. 4. The header has a fixed structure identifying the receiver node, the sender node, and the packet type. For representing the information on the receiver and the sender within a packet header, we introduce a dynamic domain of abstract *node references*, called *NREF*.

```
domain PACKET

domain NREF, sndr,rcvr : PACKET → NREF
```

[2] By not identifying nodes with their addresses, we gain additional flexibility allowing us, for instance, to model the dynamic assignment of addresses to nodes (e.g., as done in [5] for TCP/IP networks), although we do not exploit this option in the model as presented here.

With every node reference we associate a node address and a geographic position.

```
address : NREF → ADDRESS, position : NREF → POSITION
```

With respect to the meaning that a packet has, one can distinguish three basically different types of packets, namely: (1) detection packets, which are meaningful for the position based routing only; (2) discovery packets, which are meaningful for the distributed location service only; and (3) data packets. For detection packets, we further distinguish between *neighbor requests* and *neighbor replies*. Similarly, for discovery packets, we distinguish between *location request*s and *location replies*.

```
domain DETECTION = {neighborRequest,neighborReply}
domain DISCOVERY = {locationRequest,locationReply}
type : PACKET → PACKETTYPE = DATAPACKET ∪ DETECTION ∪ DISCOVERY
```

We abstractly represent data as associated with a packet's payload using some not further specified domain *DATA*. Information encoded in the data part of detection packets and discovery packets is accessed by means of partial functions defined on packets (as will be explained). In the protocol description we do actually not consider the interpretation of data as associated with data packets any further.

```
domain DATA
```

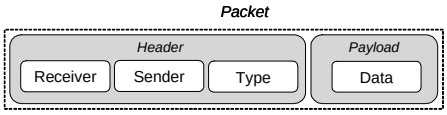


**Fig. 3.** Logical structure of packets for the communication between protocol entities

According to the vertical view of communication, adjoining layers directly interact with each other through service access points. The network layer interacts with the MAC layer and the transport layer. Similarly, the position based routing, or *PBR*, interacts with the distributed location service, or *DLS*. Such interactions are restricted to the operations as provided by the *service primitives* of a service access point.

We use here two kinds of service primitives: (1) forwarding a single packet to the next high/lower layer; (2) receiving a single packet from the next high/lower layer.

PACKET_to_⟨*Layer*⟩(⟨*Packet*⟩[{,⟨*Parameter*⟩}*])

The tag ⟨*Layer*⟩ refers to an adjoining layer, ⟨*Packet*⟩ to the packet to be forwarded, and ⟨*Parameter*⟩ to additional information, such as the position of the destination node. We assume that the sender information is automatically attached to the packet header.

At the receiving layer, the above operation causes a *packet event* through which the receiving layer obtains the packet.

PACKET_from_⟨*Layer*⟩(⟨*Packet*⟩)

The tag ⟨*Layer*⟩ refers to the sending layer, and ⟨*Packet*⟩ to the packet being received. We formalize a packet event using a monitored Boolean-valued function. In any given state, the value of this function is *true* only in the presence of a packet event, and *false* otherwise. Whenever a packet event occurs, the formal parameter ⟨*Packet*⟩ is bound to an actual parameter identifying the packet being forwarded.

## 4 Position Based Routing

With every node we associate some fixed number *k* of *sectors*, where *k* is a parameter of the network. Directed communication within a given sector *i*, $0 \le i \le k-1$, is limited to those neighbors with a geographic position within the sector *i*. For every sector of a given node, the nearest (reachable) neighbor, if any, is identified locally through a dynamic mapping from node sectors to corresponding node references.

```
domain SECTOR = {0 .. k–1}
neighbor : NODE → (SECTOR → NREF)
```

Finite Euclidian distances on the plane are represented as positive real numbers from a linearly ordered domain *DISTANCE*. We introduce a binary operation *distance* on pairs of coordinates for calculating finite distances.

```
domain DISTANCE
distance : POSITION × POSITION → DISTANCE
```

### 4.1 Detection of Nearest Neighbors

The search for the nearest neighbors of a given node splits into *k* independent search operations, one for each of the *k* node sectors. That is, we must ensure that the search for a nearest neighbor within sector *i* does not interfere with the search for a nearest neighbor within another sector *j*, for $i \ne j$ and $1 \le i,j < k$. Therefore, nearest neighbor request packets also specify the related sector ID so that the responses from neighbors then can use this ID to refer to a particular sector of the requesting node. Thus, every reachable neighbor can be uniquely assigned to one of the *k* sectors.

The sector information is encoded into the data part of the detection packets by means of a partial dynamic function *sector* defined on packets.

```
sector : PACKET → SECTOR
```

Nodes periodically update the information about their neighbors as the position of the nodes change dynamically and also because neighbors may become temporarily or permanently unreachable. The frequency of updates depends on various parameters, for instance, such as the relative speed of the neighbors and the node itself and the probability of node crashes. We abstract here from specific characteristics assuming the presence of externally controlled *update events* that trigger the start of detection cycles as indicated by the following function.[3]

```
monitored UpdateNeighborEvent : AGENT × SECTOR → BOOL
```

Intermediate results need to be stored temporarily until a detection cycle is completed. We therefore introduce an additional dynamic mapping from sectors to nodes.

```
newNeighbor : NODE → (SECTOR → NREF)
```

[3] In any given state, the Boolean–valued function *UpdateNeighborEvent* has the value *false* on all those locations for which no detection cycle needs to be invoked.

The detection of neighbors is a time critical operation since node positions may changes over time. Nevertheless, responses from neighbors may be subject to delays caused by retransmissions at lower layers and the fact that a neighbor may not always react instantaneously. Therefore, a timeout mechanism ensures that late responses are ignored. Every neighbor request also carries a *time stamp* encoded into the data part. The time stamp is then copied into the related reply. For accessing the time stamp, we introduce the following partial function on packets.

```
time : PACKET → TIME
```

Each sector of a node has its own timer operating under control of the PBR agent of that node. The time value of a timer specifies its expiration time. Initially, and after each reset operation, it has the distinguished value "∞" which is greater than all other time values. A timer with time value *t*, $now \leq t < \infty$, is considered to be active.

```
timer : AGENT × SECTOR → TIME ∪ {∞}
```

The actual time frame for detection cycles indeed is a parameter of the particular ad hoc network and depends on various application specific system characteristics. We represent this constant by a distinguished element *duration* from a static domain of finite time intervals that are given as positive real numbers.

```
domain DURATION, duration : DURATION
```

We can now define the rules for nearest neighbor detection that are executed by PBR agents as part of the *PositionBasedRouting* program defined in Sect. 4.2. In the below rules, *ag* refers to a DASM agent from *PBR*, $PBR \subset AGENT$.

```
NearestNeighborDetection=
  let node = ag.node, position = ag.node.pos, time = ag.node.now in
    forall s in SECTOR
      if UpdateNeighborEvent(ag,s) then // start new detection cycle
        IssueNeighborRequest(s,position,time)
        newNeighbor(node)(s):= undef
        timer(ag,s):= now(node) + duration // set timer
      if now(node) ≥ timer(ag,s) then // detection cycle timeout
        if newNeigbor(node)(s) ∈ NREF then // assign final result
          neighbor(node)(s):= newNeighbor(node)(s)
        timer(ag,s):= ∞ // reset timer
```

In order to generate a detection request, the position based routing creates a new detection request packet containing the relevant information, namely: the node address and geographical position, the sector ID, and a time stamp. By means of the MAC layer, this packet then is sent to nodes within the specified sector. For brevity, the formal definition of this operation is not given here.

On receiving a detection request, a detection reply packet is returned to the sender. The request and the reply are logically linked by copying the sector information and the time stamp of the request packet into the reply packet (see the below rule).

```
HandleNeighborRequest(p:PACKET) =
  extend PACKET with q
    extend NREF with r,s
      q.rcvr:= r
      q.sndr:= s
      r.address:= p.sndr.address, r.position:= p.sndr.position
      s.address:= ag.node.address, s.position:= ag.node.pos
      q.type:= neighborReply
      q.time:= p.time // copy time stamp from request packet
      q.sector:= p.sector // copy sector information from request packet
      Packet_to_MAC(q,p.sndr.position)
```

Finally, detection replies need to be handled as well. The selection of neighbors depends on the distance to a responding neighbor as well as on the distance to the nearest neighbor detected so far. This operation is specified by the below rule.

```
HandleNeighborReply(p:PACKET) =
  let s = p.sector, node = ag.node, pos = ag.node.pos
    if p.time ≥ ag.node.now – duration then
      // check distance against intermediately stored nearest neighbor
      let α = pos, β = p.sndr.position, γ = position(newNeigbor(node)(s))
        if distance(α,β) < distance(α,γ) then
          newNeigbor(node)(s):= p.sndr
```

### 4.2 Packet Routing

When receiving some packet from the MAC layer, the position based routing decides whether the packet can be delivered locally, or it needs to be forwarded to a remote destination. Delivering a packet locally means that the packet is handed over to the distributed location service running on the local node. Forwarding a packet means to send it to the nearest neighbor within the sector that matches with the position of the final destination node.

Depending on the position of the destination relative to the position of the local node, the respective nearest neighbor is computed by means of the following function.

```
computeNeighbor: NODE × POSITION → NREF
```

Below we define the model of the position based routing layer through the DASM program *PositionBasedRouting*. In addition to performing the actual routing task, this model also includes the control of nearest neighbor detection cycles and the handling of detection requests and detection replies. In the below program, *ag* refers to a DASM agent from *PBR* executing this program.

```
PositionBasedRouting=
  if status(ag.node) = switched_on
    NearestNeighborDetection // run detection cycle in parallel
    if Packet_from_MAC(p:PACKET) then
      if p.type = neighborRequest then
        HandleNeighborRequest(p)
      else
        if p.type = neighborReply then
          HandleNeighborReply(p)
        else // packets other than detection packets
          if ag.node.address = p.rcvr.address then
            Packet_to_DLS(p) // deliver packet locally
          else // forward packet to remote destination
            if computeNeighbor(ag.node,p.rcvr.position) ∈ NREF then
              let v = computeNeighbor(ag.node,p.rcvr.position)
                Packet_to_MAC(p,v.position) // perform next hop
    if Packet_from_DLS(p:PACKET) then
      Packet_to_MAC(p,p.sndr.position)
  else // node is passive
    skip
```

## 5 Distributed Location Service

With each node we associate a set of direct neighbors in the dynamic hypercube (see Sect. 2.1) as identified by a dynamic function *hypercubicNeighbors* on nodes.

```
hypercubicNeighbors : NODE → NREF-set
```

For a given node and a given destination address, a hypercubic neighbor on the way to the final destination is computed using the following dynamic function.

```
nextHypercubicNeighbor : NODE × ADDRESS → NREF
```

The definition of this function is based on the definition of another dynamic function, called *computeNextID*, which performs the actual address calculation on the dynamic hypercube. For brevity, we refer to [1] for the definition of this address calculation.

```
computeNextID : ADDRESS × ADDRESS → ADDRESS

nextHypercubicNeighbor(u, a) = v ∈ NREF: v ∈ hypercubicNeighbors(u)
                             ∧ v.address = computeNextID(u.address, a)
```

Consider some sequence of consecutive packets received, one by one, from the transport layer. The first packet of every such sequence requires special treatment.

This packet needs to be stored locally to first determine the position of the destination node before sending the packet. Accordingly, we assume that the first packet always can be recognized as such using a monitored Boolean-valued function on packets.

```
monitored Firstpacket : PACKET → BOOL, firstpacket : DLS → PACKET
```

A location reply to the location service always matches a pending location request. Thus, the contained position information of the sender is added to a waiting first packet, which then eventually can be sent via the position based routing.

When receiving a location request, the location service checks the address of the final destination. This address is encoded into the data part of the discovery packet. We therefore introduce a partial dynamic function *address* defined on packets.

```
address : PACKET → ADDRESS
```

If the final destination address does not match with the local node, we calculate the next hypercubic neighbor on the way to the final destination and forward the request.

```
HandleDiscoveryPacket(p:PACKET) =
  if p.type = locationReply then // extract position information
    ag.firstpacket.rcvr.position:= p.sndr.position
    Packet_to_PBR(ag.firstpacket)
  if p.type = locationRequest then
    if p.address = ag.node.addres then // generate location reply
      HandleLocationReply(p)
    else // compute next hypercubic neighbor and forward packet
      let v = nextHypercubicNeighbor(ag.node,p.address)
        p.rcvr:= v // specify next hop in the hypercube
        Packet_to_PBR(p)
```

We can now define the model of the distributed location service through the below DASM program *DistributedLocationService* (where *ag* refers to a *DLS* agent).

```
DistributedLocationService=
  if Packet_from_Transport(p:PACKET) then
    if Firstpacket(p) then // discover destination position first
      DiscoverDestinationPosition(p.rcvr.address)
      ag.firstpacket:= p // intermediately store first packet
    else // destination position is already known
      Packet_to_PBR(p),
      p.rcvr.position:= ag.firstpacket.rcvr.position
  if Packet_from_PBR(p:PACKET) then
    if p.type ∈ DISCOVERY then
      HandleDiscoveryPacket(p)
    else Packet_to_Transport(p)
```

```
DiscoverDestinationPosition(a:ADDRESS) =
  let v = nextHypercubicNeighbor(ag.node,a)
    extend PACKET with q // create location request packet
      extend NREF with r,s
        q.rcvr:= r, r.address:= v.address, r.position:= v.position
        q.sndr:= s, s.address:= ag.node.address, s.position:= ag.node.pos
        q.type:= locationRequest
        q.address:= a
      Packet_to_PBR(q)
```

## 6 Concluding Remarks

We present a DASM model of the network layer protocol for geographic ad hoc routing based on the hypercubic location service defined in [1]. Conceptually, the network layer splits into two separate sublayers, one for the location service and one for the position based routing. Our model captures the core functionality of both sublayers exposing the key properties of the network layer protocol. The particular focus is on the overall interoperability of protocol entities rather than on the details of their internal realization. We formalize the basic algorithmic aspects with a degree of detail and precision that goes far beyond the informal, pseudocode like description of fundamental operational aspects as presented in [1].

The abstract operational view allows us to represent a relatively complex protocol in a concise and coherent way so that one can analyze the resulting behavior. Not yet included is the reorganization of the dynamic hypercube as required for the elimination and insertion of nodes in order to ensure fault tolerance of the distributed location service. This is feature defined in [1], and we do not see any principle difficulty for extending our model accordingly. Also not included are the operations to be performed when a node is switched on or switched off.

**Acknowledgements**

Our sincere appreciation to Peter Päppinghaus, Siemens Munich, and Margus Veanes, Microsoft Research Redmond, for their fairly detailed and valuable comments on a preliminary version of this paper.

## References

[1] A. Benczúr and T. Lukovszki. A Degree *O*(*log log n*) Fault Tolerant Distributed Location Service for Geographic Ad-Hoc Routing. Technical Report, TR-RI-02-231, Heinz Nixdorf Institute, Paderborn, Germany, June 2002

[2] D. E. Comer. Internetworking with TCP/IP, *Principles, Protocols, and Architectures*. Prentice Hall, 2000

[3] R. Eschbach, U. Glässer, R. Gotzhein, M. von Löwis and A. Prinz. Formal Definition of SDL-2000. *Journal of Universal Computer Science*, 11 (7): 1025-1050, Springer Pub. Co., 2001

[4] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric Spanner for Routing in Mobile Networks. In *ACM Symposium on Mobile Ad Hoc Networking and Computing* (MOBIHOC'01), pages 45-55, 2001

[5] U. Glässer, Y. Gurevich and M. Veanes. High-Level Executable Specification of the Universal Plug and Play Architecture. In Proc. *35th Hawaii International Conference on System Sciences*, Software Technology, IEEE Press, 2002

[6] U. Glässer, Y. Gurevich and M. Veanes. An Abstract Communication Model. Microsoft Research Technical Report, MSR-TR-2002-55, May 2002

[7] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger (Ed.), *Specification and Validation Methods*, Oxford University Press, 1995, 9-36

[8] Y. Gurevich and J. Huggins. The Railroad Crossing Problem: An Experiment with Instantaneous Actions and Immediate Reactions. In H. Kleine Büning (Ed.), *Computer Science Logic*, LNCS 1092, 266-290, Springer, 1995

[9] D. B. Johnson and D. A. Malz: Dynamic Source Routing in Ad Hoc Wireless Networks. Mobile Computing, Vol.: 353, Kluwer Academic Publishers, 1996.

[10] T. Leighton. Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes. Morgan Kaufmann, 1992.

[11] J. Li et al. A Scalable Location Service for Geographic Ad Hoc Routing. ACM International Conference on Mobile Computing and Networking (MobiCom'00), 120-130, 2000.

[12] X.-Y. Li and P.-J. Wan and Y. Wang. Power Efficient and Sparse Spanner for Wireless Ad Hoc Networks. IEEE International Conference on Computer Communications and Networks (ICCCN'01), 2001.

[13] M. Mauve, J. Widmer and H. Hartenstein. A Survey on Position-Based Routing in Mobile Ad-Hoc Networks. IEEE Network, 15(6): 30--39, 2001

[14] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. IEEE INFOCOM, 1405-1413, 1997

[15] C. E. Perkins and P. Bhagwar. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. ACM SIGCOMM'94, 234-244, 1994

[16] R. Wattenhofer et al. Distributed Topology Control for Power Efficient Operaion in Wireless Multihop Ad Hoc Networks, IEEE INFOCOM, 2001

[17] A. C. Yao. On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems. SIAM Journal on Comp., 11(4): 721-736, 1982