NASCENT: Network Layer Service for Vicinity Ad-hoc Groups

Jun Luo Jean-Pierre Hubaux
School of Computer and Communication Sciences
EPFL (Swiss Federal Institute of Technology at Lausanne), CH-1015 Lausanne, Switzerland
Email: {jun.luo, jean-pierre.hubaux}@epfl.ch

***Abstract*— Many envisioned applications of ad hoc networks involve only small-scale networks that we term *Vicinity Ad-hoc Groups* (VAGs). Distributed coordination services, instead of pairwise communications, are the primary requirements of VAGs. Existing designs for distributed services apply either a layered structure or a vertical integration. While the former contributes to design simplicity, the latter improves runtime efficiency. In this paper, we argue that, since distributed services require group-oriented communications, our NASCENT approach can achieve *both* design simplicity and runtime efficiency in VAGs. NASCENT is a network layer service dedicated for VAGs. It provides a light-weight membership service along with a routing structure for message passing, and it supports the concurrent execution of various distributed algorithms. NASCENT is also tailored to cope with the transiency of VAGs. We demonstrate how smoothly distributed algorithms can be built on top of NASCENT. With a complexity-based analysis, we also show that NASCENT greatly improves the runtime efficiency of these distributed algorithms. Finally, through simulations with *ns*-2, we confirm the ability of NASCENT to support the envisioned VAG applications.**

***Index Terms*— Ad Hoc Networks, Vicinity Ad-hoc Groups, Network Layer Services, Distributed Algorithms.**

I. Introduction

Distributed services are often required in ad hoc networks, because centralized services relying on individual nodes are not dependable enough. In particular, certain applications require primarily distributed services to coordinate the collective actions of nodes in small-scale networks, whereas they barely need pairwise connections to support stream traffic. We term the networks implied by such applications *Vicinity Ad-hoc Groups* (VAGs), in order to emphasize their small network scale and group-oriented communication paradigms. Because of these peculiarities, protocol re-engineering appears to be necessary.

Currently, protocols for supporting distributed services in ad hoc networks are usually built upon routing protocols (e.g., AODV [1] and DSR [2]). Examples include overlay multicast (e.g., [3], [4], [5]), membership service (e.g., [6]), and resource discovery and management (e.g., [7], [8]). This layered approach, inherited from the Internet architecture, aims at reducing the protocol design complexity; as unicast routing provides the upper layer with a fully connected graph, virtually any distributed algorithm can be built on top. However, this approach can be inefficient for applications where a fully connected graph is an overkill. An alternative approach [9], [10], [11], [12] suggests that *mobility aware* distributed algorithms be built directly upon the MAC layer. Although protocols designed with this vertical integration approach are efficient at runtime, the design complexity is increased because, without a layered structure, low-layer protocols have to be developed separately for individual services.

We believe that a trade-off between these two approaches can achieve the *best of both worlds* in VAGs. Since communications in VAGs are group-oriented, it is possible to design a network layer service that includes common enabling mechanisms (e.g., broadcast message routing and membership management), and to leave only distinct requirements fulfilled by upper layer algorithms. The design complexity of this approach is similar to that of the layered approach, but the runtime efficiency of resulting protocols can be improved to the level of the vertical integration approach. In this paper, we investigate the design of such a dedicated service and propose NASCENT (Network lAyer ServiCE for viciNiTy ad-hoc groups) as a solution. Our contributions are: (i) a general network layer service to support various distributed services, (ii) an embedded membership service requiring only localized operations, (iii) algorithms to initialize a logical networking structure among an emerging VAG, and (iv) algorithms to recompose the structure upon the VAGs' merging. We present examples of distributed algorithms that are easily developed by using the services of NASCENT. In addition, we apply a complexity-based analysis to show the reduced runtime complexity of these algorithms. Finally, we perform simulations with *ns*-2; the results prove the ability of NASCENT to support the targeted applications.

The rest of this paper is organized as follows: Section II explains our motivation for this work. Section III states the problem and the network model. Section IV presents our NASCENT service. Section V gives examples of distributed algorithms built upon NASCENT and proves the efficiency gains over existing approaches. Simulation results are provided in Section VI. Finally, Section VII concludes the paper. We skip the literature survey because related topics are discussed where they are needed.

The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. (`http://www.terminodes.org`)

II. Motivations

In this section, we first envision several applications. Then we identify the common requirements of these applications. The outcome of the case studies then leads to our motivations for the protocol design.

Three examples of envisioned applications that have immediate needs for both ad hoc networking and distributed coordination services are:

a) **Wireless multi-player gaming**: Wireless gaming devices (e.g., Nokia N-Gage™ game deck) either require players to pay bills (with GPRS) or limit their numbers or mutual distances (with Bluetooth). Devices based on IEEE 802.11-like wireless MAC are more flexible, but distributed coordinations become necessary in the absence of a centralized control. Fig. 1(a) shows a gaming scenario where players are contending for a special role (the *queen*).

b) **Cooperative robotics** [13]: Using teamed robots for unmanned explorations and rescue operations becomes increasingly tempting. Some ongoing research (e.g. [14], [15]) is focusing on coordinating robots with wireless communication networks. Fig. 1(b) shows an example of exploration robots making an agreement on the direction of movement.

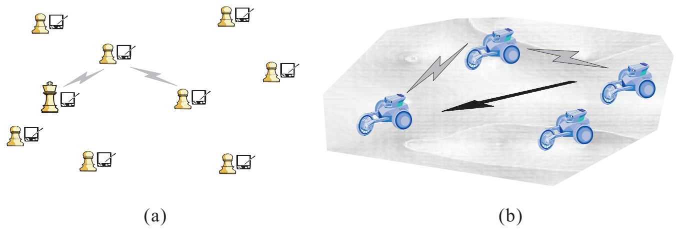


Fig. 1. Application examples: (a) wireless multi-player gaming, and (b) exploration robots.

c) **Cooperative driving system** [16]: Vehicles running through highway entrances and crossroads have to be scheduled to share the resources (i.e., these critical points), in order to avoid collisions. Distributed algorithms relying on inter-vehicle communications could be a conflict resolution method performed by the vehicles themselves.

The aforementioned applications share several characteristics. First, the scale of the network is small (either geographically or logically, or both). This is easy to see because, for example, people in one city do not play a wireless game with people in another city (in which case they can play games over the Internet). Secondly, these networks tend to be transient. For example, players join and leave a wireless game spontaneously; and task dedicated groups of robots might change when finishing a job and starting another. The network topology also changes with time due to node mobility. Thirdly, broadcast (with reliability requirements in most cases) is the dominating communication paradigm within the network. Nodes in such a network have the same goals (performing specific tasks or sharing certain resources), while the nodes are inherently distributed with each one having only local information. This situation calls for network-wide reliable communications to coordinate the behavior of nodes.

Following the aforesaid discussions, we term such a small network VAG (i.e., *Vicinity Ad-hoc Group*) to depict these peculiarities. We also summarize our motivations as follows:

- A dedicated network layer service could be more efficient than unicast routing protocols (which support point-to-point stream traffic) in supporting various distributed algorithms, considering the broadcast nature of communications in a VAG.
- This service should also take care of membership management, because distributed algorithms usually have the same need of membership information, in spite of different intentions.
- The transiency of VAGs requires the service to have the ability of prompt initialization and to cope with frequent topology and membership changes. It also suggests that a global membership tracking would be highly inefficient.

III. Goals and Models

Given all the aforementioned incentives, we now formally define the problem we want to solve and the considered environments.

We consider small-scale ad hoc networks (or VAGs), where the network diameter is about 3 or 4 hops and group communication is the dominating communication paradigm. Our goal is to develop a group-oriented network layer service, other than a unicast routing protocol, to support distributed services (in particular, distributed algorithms such as reliable broadcast and resource allocation) in VAGs.

We assume that every node in a VAG has a unique address $addr$.[1] Nodes may experience only recoverable failures (i.e., crashing). All communications between nodes are assumed to rely on the underlying MAC protocol, so that only nodes within the transmission range of each other can communicate directly and are thus termed *neighbor*s. Each communication link is assumed to be bidirectional and FIFO.[2] A link is reliable for unicast (denoted by a $\textsc{Send}()$ primitive),[3] but it is unreliable for broadcast (denoted by a $\textsc{BCast}()$ primitive). The network is modeled as an undirected graph $\mathcal{G} = (V, E)$, with nodes as vertices and an edge existing only between neighbors. The graph changes dynamically and is not necessarily connected. We further assume that the members of a VAG follow group-based movements (e.g., [17]); each member is aware of the mobility group defined by its VAG application (which is not the case for [17]). This implies that partitions of a VAG seldom occur and that most partitions are transient.

[1]Usually, an *identifier* is used to designate a node. However, as we will see later, the identifier (or id) of a node is used by protocols to dynamically indicate the status of the node. Therefore, we use *address* for node identification.

[2]Our protocols refrain from using any unidirectional link.

[3]IEEE 802.11, for example, provides reliable link layer data transmission using the well-known Stop&Wait ARQ.

IV. NASCENT: Group-Oriented Network Layer Service

We present NASCENT in this section. We first over-view its architecture, then we briefly justify our design considerations. The protocol is detailed afterward.

A. Overview of NASCENT

As shown in Fig. 2(a), NASCENT is a service to replace routing protocols in the protocol stack for VAG members. It acts as a building block of the group-oriented communications in a VAG, and provides support to distributed algorithms such as *Mutual Exclusion* (MX, a special case of resource allocation), *Leader Election* (LE), *Reliable Broadcast* (RB), etc. The protocol architecture contrasts clearly with the layered structure based on unicast routing in Fig 2(b) and the vertical integration approach in Fig 2(c) (introduced in Section I).

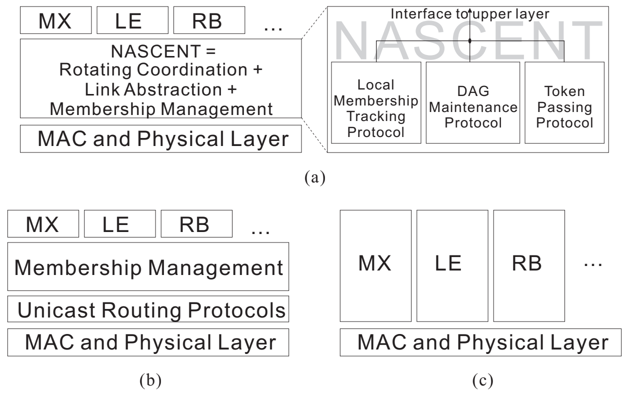


Fig. 2. Comparisons between the approach of NASCENT (a) and two existing designs of distributed algorithms (b)&(c).

Fig. 2(a) also shows the components of NASCENT. The *Local Membership Tracking Protocol* (LMTP) keeps track of the members within 2 hops by exchanging lists of neighbors with neighbors. The *DAG Maintenance Protocol* (DMP) transforms the undirected graph $\mathcal{G}$ into a token-oriented DAG (*directed acyclic graph* with its sink holding a token) and maintains the DAG when $\mathcal{G}$ is undergoing any changes. The combination of LMTP and DMP acts as the membership service of NASCENT, and DMP also facilitates the (broadcast) message routing. The *Token Passing Protocol* (TPP) causes a *unique* token to visit every member periodically and reshapes the DAG upon a token transfer. The token holder becomes a centralized control of its VAG temporarily, which is a key to building the upper layer algorithms.

B. Design Rationale

We hereby address the main decisions about our design.

Why local membership tracking instead of global?: Making global agreements on membership views is highly inefficient in a VAG, because the network topology and membership change frequently and because possible node failures are often transient.

Why apply a DAG?: A logical structure provides paths to route messages. It also binds all local views together and this results in a functional membership service. DAG is better than other structures (e.g., ring and tree) because it provides enough redundancy to cope with the transiency of VAGs.

Why circulate a token?: The token passing, sometimes called a rotating coordinator paradigm, has long been regarded as an efficient way to support (ordered) reliable broadcast [18], [19] and mutual exclusion [20], [10]. We further observe that a circulated token can be a basis of virtually any distributed algorithm.

C. Protocol Details

We describe NASCENT in detail in this section. Each subsection is devoted to a protocol.

1) Local Membership Tracking Protocol (LMTP): Each member of a VAG periodically broadcasts a *beacon message* ($bmsg$) including the ids of its neighbors. A member, upon receiving the message, infuses the ids into its *local view* ($lview$). An id is removed from the local view if no $bmsg$ from that member is received in τ_b consecutive beacon periods. Note that the id of a member is not exactly its address $addr$ but a tuple including the address. The format of an id and the group identifier gid are introduced in Section IV-C.2 (where they are used by DMP). Fig. 3 illustrates the exchange

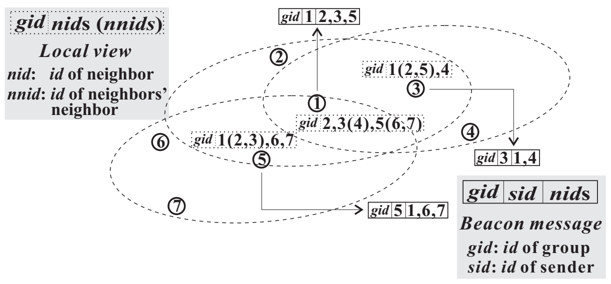


Fig. 3. Beacon-based local membership tracking ($addr$s instead of ids are shown to save spaces).

of $bmsg$s, as well as the **format** of a $bmsg$ and $lview$. Through this information exchange, each member is aware of other members within 2 hops and also synchronizes with its neighbors. The GBCast primitive (explained in Section V-A.2) takes advantage of this information to reduce its costs.

2) DAG Maintenance Protocol (DMP): We propose new algorithms to initialize a token-oriented DAG in a newly formed VAG and to recompose the DAG upon the VAGs' merging. We let the id of a member i (i.e., $addr = i$) be a triple $[\alpha_i, \beta_i, i]$, where α and β are two intergers. These ids, ranked lexicographically,[4] form a total order sequence. By varying the values of α and β in its id, a member can change its rank within the sequence without modifying its physical address. Member i considers a communication link with member j as an *outgoing* edge if $id_i > id_j$, otherwise the

[4] $id_i > id_j \Leftrightarrow$
$\alpha_i > \alpha_j \vee (\alpha_i = \alpha_j \wedge \beta_i > \beta_j) \vee (\alpha_i = \alpha_j \wedge \beta_i = \beta_j \wedge i > j)$

link is an *incoming* edge. As a result, the communication graph $\mathcal{G}$ is transformed into a DAG. The idea of associating a triple with each node and defining logical directions for all links based on these triples (originally described in [21]) allows for dynamical manipulation of network structure.

Initialization: Initially, the id of member i is set to $[0, 0, i]$, so the DAG may have more than one sink. In the example of Fig. 5(a), members 1,2, and 3 are all sinks. Since we let a sink hold a token, this situation does not guarantee the uniqueness of the token in a VAG. In order to remove those superfluous sinks (e.g., members 2,3), each member adjusts its id according to the incoming $bmsg$s. Algorithms describing these adjustments are shown in Fig. 4. Each member initializes

```
1: procedure INIT($id_i$)
2:     $lview_i.gid \leftarrow id_i$; $init_i \leftarrow$ true

3: upon RECV($bmsg$) do
4:     if ($bmsg.gid < lview_i.gid$) $\wedge$ $init_i$ then
5:         $lview_i.gid \leftarrow bmsg.gid$
6:         $id_i.\beta \leftarrow bmsg.sid.\beta + 1$
```

Fig. 4. Initialization at member i

the gid in $lview$ (refer to Fig. 3 for details) with its id (line 2). Upon receiving a $bmsg$, each member checks if there exists a gid that is smaller than the one it knows (line 4). The lower gid is then taken and the id is adjusted properly (lines 5–6). Fig. 5(b) shows the situation after each member broadcasts the first $bmsg$ following the initialization of its $lview.gid$. At some point in time, gids of different sinks will arrive at the same members. These members form a *collision region*, shown in Fig. 5(c). Members belonging to a collision region choose the sink with a lower gid as the token holder; the following $bmsg$s will confirm the outgoing edge to the token holder and reverse the edge to other sinks. This procedure continues until all superfluous sinks are removed, which terminates the algorithm and results in a token-oriented DAG (with member 1 being the **initial** token holder), as shown in Fig. 5(d).

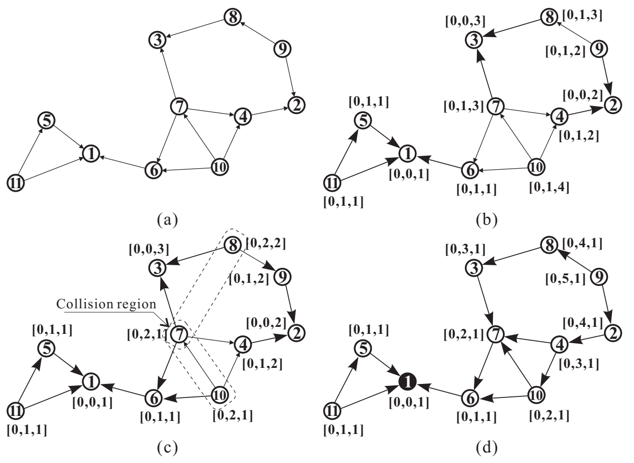


Fig. 5. Initialization of a token-oriented DAG. The gid and id of each member is put in a compact way, such that the first two elements are α and β of an id and the third element is the $addr$ of a gid.

We prove the correctness of this algorithm by showing that the following two properties of the algorithm hold.

Property 1: The algorithm eventually terminates.

Proof: The $lview.gid$ of each member will be set to the smallest **initial** id in the network ($[0, 0, 1]$ in Fig. 5) after some time. The algorithm terminates because of the conditional statement in line 4. ■

Property 2: The resulting DAG has only one sink.

Proof: Assume, in contradiction, that, after the algorithm terminates, there is an extra sink whose initial id is not the smallest one. Since the algorithm has terminated, the $lview.gid$ of this member is set to the smallest initial id in the network. Considering that the initial $lview.gid$ is larger than the final one, the algorithm must reach line 6 once exactly before the algorithm terminates, which implies that it has an outgoing link. The link will not be reversed after that, because the $lview.gid$ at the other end of this link has already been the smallest initial id (i.e., the algorithm has terminated), a contradiction. ■

A timer is also set to force the algorithm to stop in case of long converge time. The timeout value of this timer is set according to the time complexity of the algorithm (refer to Section V-B for details). The accuracy of this value is not quite relevant because, even if the algorithm is stopped too early due to an improper value, the resulting token collision is solved by keeping the token with the smallest gid and removing all others.

Other operations: With proper extensions, the algorithm of Fig. 4 also handles group *merging*, member *join* and *leave*. Note that group merging is similar to group initialization since both have to solve the collision of multiple sinks; we refer to [22] for detailed explanations. To cope with topology changes due to node mobility, we reuse the *partial reversal method* described in [21]. It is known that the algorithm in [21] becomes unstable in partitions that disconnect from the token holder. Partition detection mechanisms (e.g., TORA [23]) can be one solution. We will describe an alternative solution based on a timer in Section IV-C.3.

3) Token Passing Protocol (TPP): TPP circulates a *token* within a VAG. The token is a message that contains certain system states. A token holder acquires and modifies these states, and thus acts as a temporary coordinator, which facilitates various distributed coordinations. We propose two algorithms, TPP-Q and TPP-R, for circulating the token. In TPP-Q, the token circulation is performed by a distributed Queueing system. This system guarantees that the token repetitively visits each member with a period linear with n. TPP-R applies a more heuristic method based only on local Recency information, which brings less overhead but comes with a larger worst-case upper bound of the circulation period. Note that, in addition to token circulation, TPP also guarantees that only a unique sink of the DAG holds the token, by reshaping the DAG upon a token transfer. We provide only intuitive ideas of TPP in this section and refer to the APPENDIX for the corresponding pseudocodes.

Token Circulation with a Queueing System (TPP-Q): As a follow-up of Fig. 5, Fig. 6 provides an example of one token circulation period with TPP-Q. Each VAG member is equipped with a queue. Upon completing initialization, each member, except the token holder, enqueues its own request of the token and also forwards a request to a member with minimum id in its local view. A request contains the requester's address $saddr$

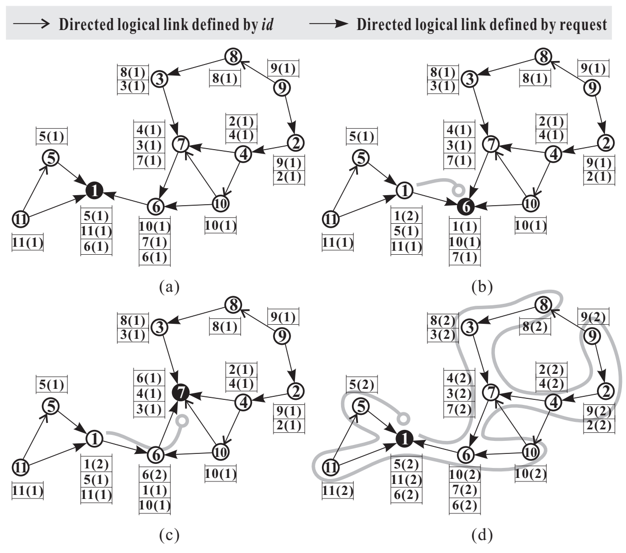

Directed logical link defined by id
Directed logical link defined by request
(a)
(b)
(c)
(d)

Fig. 6. One cycle of token circulation by TPP-Q. Each request in the queue is represented by $saddr(epoch)$, and the queue is FIFO with its header at the bottom.

and a counter $epoch$ that denotes which cycle the request is for. Fig. 6(a) shows that the distributed queueing system actually builds a rooted spanning tree. Then, as exhibited by Fig. 6(b),(c),(d), the token will travel along the tree edges. They also show that the $epoch$ of every queued request is stepped up by one after the token has visited all members. As a consequence, the token will repeat the same trajectory that it follows in the first cycle and thus visit each member periodically. We distinguish the case where a member is **visited** by the token from the case where a member **receives** the token. A member is considered to be visited only if its request is the first in the queue when it receives the token.

Token Circulation with Recency Information (TPP-R): TPP-R requires each VAG member to keep an array that records the time when the token visits each neighbor recently. A member that **receives** the token is **visited** by it only if the member has the least recency record among all neighbors, otherwise it forwards the token to a neighbor with the least recency record. The recency information is piggybacked with the token to inform a member about the records of its neighbors. Although TPP-R usually has a smaller token circulation period than that of TPP-Q (because shortcuts can be exploited to avoid backtrackings), the worst-case upper bound of this period is much larger (refer to Section VI-D for an example). A similar scheme is also described in [24].

For both cases of TPP-Q and TPP-R, a timer is set after a token holder releases the token, in order to detect group partitions. A timeout leads to a new initialization phase. If partitions really happen, all members that are disconnected from the token holder will enter the initialization phase and thus regenerate a token for each connected partition. If it is a false positive, the member can either rejoin the previous VAG or merge with the VAG. Since false positives make a VAG unstable, setting the timeout value is crucial for reducing the probability of false positives. The following property of TPP-Q facilitates this value setting.

Property 3: The time for a token to visit all VAG members in one cycle is upper bounded by $\hat{T} = D_{max}nT_t + nT_s$ if the VAG remains static during that time period. Here D_{max} is the degree of the undirected graph $\mathcal{G}$, n is the cardinality of the VAG, T_t and T_s are the one way **t**ransmission time and the time for the token to **s**ojourn at a visited member, respectively.

Proof: The distributed queue maintained by TPP-Q creates a rooted spanning tree $\mathcal{T} = (V, E_\mathcal{T})$, where $E_\mathcal{T}$ is the set of all tree edges. The exact time to visit all vertices in $\mathcal{T}$ is $2|E_\mathcal{T}|T_t + |V|T_s$, where $|E_\mathcal{T}| < \frac{1}{2}D_{max}n$ and $|V| = n$. Note that the upper bound is tight if $T_t \ll T_s$. ■

Although TPP-R has a larger worst-case upper bound, $\hat{T}$ works well in most cases. Therefore, we set the timeout value to $\hat{T}$, and the parameters can either be adaptively changed on-line or be estimated off-line (e.g., the maximum number of roles in a game is known to every player). The way to make on-line adaptations or off-line estimations depends on specific applications, so NASCENT provides the upper layer with an interface to set $\hat{T}$.

V. Distributed Algorithms on top of NASCENT

In this section, we sketch the implementation of distributed algorithms on top of NASCENT and we also perform a complexity-based analysis on NASCENT and these algorithms. Our goal is to demonstrate how NASCENT simplifies the design of distributed algorithms and improves their runtime efficiency.

A. Algorithm Descriptions

We briefly describe several distributed algorithms built upon NASCENT. It is straightforward to see that the design of these algorithms is simplified thanks to the use of services provided by NASCENT.

1) Resource Allocation: We consider a general resource allocation problem, where several instances of a single system resource exist and only one member can access one instance each time. We distinguish between *local multi-instance* and *distributed multi-instance* resource allocation. In the former case, all instances of the resource are geographically close to each other. The latter case is more broad.

The local multi-instance resource allocation can be solved by simply extending the mutual exclusion algorithm: a token holder, instead of allowing only itself to access the resource, also grants permission to its neighbors for accessing other instances of the resource. We hereafter focus on the distributed multi-instance resource allocation. Let the token carry m sub_tk that represent m instances of a certain resource. Fig. 7

```
1: upon ReqRes() do
2:     $req\_res_i \leftarrow$ true

1: upon RelRes() do
2:     $req\_res_i \leftarrow$ false

1: upon Recv($token$) do
2:     if $req\_res_i \wedge (\exists i \in [1, m]$ s.t. $token.sub\_tk[i].id =$ null$)$ then
3:         $token.sub\_tk[i].id \leftarrow id_i$
           /* now access the resource and release the token */
4:     if $\overline{req\_res_i} \wedge (\exists i \in [1, m]$ s.t. $token.sub\_tk[i].id = id_i)$ then
5:         $token.sub\_tk[i].id \leftarrow$ null
           /* now release the token */
```

Fig. 7. Request and release a resource instance at member i

shows an algorithm for allocating these instances. The point is that only members that catch a sub_tk can access one instance of the resource. In addition, the token can also carry a queue that orders the resource requests to ensure fairness.

2) (Ordered) Reliable Broadcast: A reliable broadcast ensures that a message broadcast by a member is delivered by all correct members [25]. Our solution built upon NASCENT is similar to what is proposed in [18]. As shown in Fig. 8, a

```
1: procedure RBCast($msg$)
2:     $SMsgBuffer_i \leftarrow SMsgBuffer_i \cup \{msg\}$

3: upon Recv($token$) from $id_j$ do
4:     if $SMsgBuffer_i \neq \emptyset$ then
5:         for all $msg \in SMsgBuffer_i$ do
6:             GBCast($msg$)
7:             $token.view[id_i][msg.mid].recv \leftarrow$ true
8:             $SMsgBuffer_i \leftarrow SMsgBuffer_i / \{msg\}$
9:     for all $msg$ s.t. $token.view[id][msg.mid].recv \neq$ true do
10:        if $msg \in RMsgBuffer$ then
11:            $token.view[id_i][msg.mid].recv \leftarrow$ true
12:        else
13:            request $msg$ from $id_j$
14:    if $\nexists nid$ s.t. $token.view[nid][msg.mid].recv =$ false then
15:        Deliver($msg$)    /* to the upper layer */

16: upon Recv($msg$) from $id_j$ (sent by BCast) do
17:     if $msg \notin RMsgBuffer_i$ then
18:         $RMsgBuffer_i \leftarrow RMsgBuffer_i \cup \{msg\}$
19:         GBCast($msg$)
20:     $bcast\_view[msg.mid] \leftarrow bcast\_view[msg.mid] \cup \{id_j\}$

21: procedure GBCast($msg$)
22:     if $\exists nid \in lview$ s.t. $nid > id_i \wedge$
        $nid \notin \bigcup_k id_k.lview, \forall id_k \in bcast\_view[msg.mid]$ then
23:         BCast($msg$)    /* see Section III */
```

Fig. 8. Reliable broadcast at member i

member broadcasts a message only when holding the token (lines 1–8), by invoking the GBCast primitive (lines 21–23). The token contains a record about message receptions. A message is delivered to the upper layer only if all members have received it (lines 9–15), and a missed message is requested from the previous token holder. A member stores a broadcast message received for the first time and rebroadcasts it, also by the GBCast; the member also memorizes the sender of each received message (lines 16–20).

The GBCast primitive leverages on the token-oriented DAG and on the neighborhood information in $lview$ to reduce collisions in the MAC layer and gains reliability and efficiency over flooding. It is also more robust than a tree-based multicast protocol, since the VAG members are linked by a (directed) mesh. The reliability of GBCast is further enhanced as only one member (the token holder) can initiate a group-wide message dissemination. As a consequence, negative acknowledgements (line 13) are rarely sent by a member, resulting in an efficient reliable broadcast algorithm. Urgent messages from a non-token holder can be sent through proxy-broadcasts. This is possible because the DAG supports unicasts from all members to the token holder. The protocol implicitly guarantees FIFO order and causal order [25], whereas a slightly modified version that requires each member to deliver messages according to their order in the token solves the total order broadcast problem (i.e., all correct members deliver messages in the same order).

3) Leader Election: Solutions to the leader election problem become trivial if both mutual exclusion and reliable broadcast are solved. As defined in [9], leader election needs to ensure that a group whose topology remains static sufficiently long will eventually have exactly one leader. Our solution is the following: a leader candidate who acquires the token before other candidates declares itself to be the leader; it then broadcasts its $addr$ with the RBCast primitive (Fig. 8). An alternative way is to put its id into the token, thus trading latency for reduced communication costs.

B. Complexity Comparisons

In this section, we defend our claim that NASCENT improves the runtime efficiency of the described algorithms. For this purpose, we show that the complexity of maintaining NASCENT is of the same magnitude as unicast routing protocols, and that the complexity of running distributed algorithms upon NASCENT is greatly reduced. Note that the former complexity is measured against one network perturbation (i.e., node or link failures) and the latter is evaluated with respect to one specific operation.

We make use of *time complexity* (TC) and *communication complexity* (CC) to quantify the performance of protocols. TC is defined as the number of steps needed to perform a protocol operation, and CC is the number of messages needed to perform a protocol operation; we refer to [26] for detailed definitions of these two terms and related synchrony assumptions. In Table I, we compare the performance of several

	DSDV	AODV; DSR	GB; TORA	NASCENT
TC$_i$	O(d)	O$(2d)$	O$(2d)$	O$(2d)$
TC$_f$	O(d)	O$(2d)$	O(l); O$(2d)$	O(l)
CC$_i$	O(n)	O$(2n)$	O$(2n)$	O$(2n + n)^*$
CC$_f$	O(n)	O$(2n)$	O(x); O$(2x)$	O$(2x + x)$

n = Number of node in the network
d = Network diameter
x = Number of nodes affected by a topological change
l = Diameter of the affected network segment
* CC of DAG maintenance + CC of token request forwarding
Note that the second part appears only if TPP-Q is used.

TABLE I
Performance comparisons between routing protocols and NASCENT.

routing protocols (DSDV [27], AODV [1], DSR [2], GB [21], and TORA [23]) with NASCENT. Note that DSDV is used as a representative of table driven protocols. The complexity computations of routing protocols are borrowed from [28], [29]. Each protocol is evaluated in two situations, namely *initialization* and *postfailure*, distinguished by the subscript of performance terms (i and f, respectively). The comparisons show that the postfailure complexity of NASCENT is comparable to those of routing protocols, whereas the initialization complexity is slightly higher. However, directly comparing table driven protocols (DSDV and NASCENT) with on-demand protocols (AODV, DSR, TORA) is unfair, since the complexity of an on-demand protocol is evaluated for each route. In the case of group-oriented communications (where one node needs routing paths to every other node), on-demand protocols would incur a much higher complexity than table driven protocols because, in the worst case, an actual CC is the one shown in Table I times $n(n-1)/2$. Furthermore, given a topological change, GB and TORA incur a much larger value of x than NASCENT does, because GB and TORA maintain up to n DAGs whereas NASCENT needs only one.

The comparisons in Table I do not prove that NASCENT is better than DSDV for supporting distributed services in VAGs. However, if we look at distributed algorithms that make use of the services provided by these two protocols, the superiority of NASCENT becomes clear. In Table II, we use reliable broadcast as an example to show the complexity of building distributed algorithms on top of DSDV and NASCENT. The

	DSDV+HT_RB	DSDV+RB	NASCENT+RB
TC	$\mathrm{O}(n)$	$\mathrm{O}(n)$	$\mathrm{O}(n)$
CC	$\mathrm{O}(n)^*$	$\mathrm{O}(3n+x)^\dagger + 1^*$	$\mathrm{O}(3n)^\ddagger + 1^*$

HT_RB: the reliable broadcast protocol describe in [25]
RB: the reliable broadcast protocol describe in Section V-A.2
* Broadcast † Multi-hop unicast ‡ Single-hop unicast

TABLE II
PERFORMANCE COMPARISONS BETWEEN RELIABLE BROADCAST PROTOCOLS BUILT UPON DSDV AND NASCENT.

algorithm proposed in [25] (we refer to it as HT_RB to credit the authors) is a representative of non-token-based solutions. It can be roughly described as "each member, upon first receiving a message, broadcasts it to other members". This explains why it has a complexity of $\mathrm{O}(n)$ for both TC and CC. However, it is much more expensive than a token-base approach, considering that each message is broadcast within the group. If only the number of messages is counted, the token-based algorithm we proposed in Section V-A.2 has quite similar CCs when running on top of DSDV and NASCENT, i.e., 1 broadcast message (Fig. 8 line 6) and $3n$ unicast messages in the worst case (1 token passing message, 1 negative ack and 1 response for each member), whereas the algorithm has to maintain a DAG with x extra messages if it is based on DSDV. However, a unicast message in the case of DSDV is transmitted through multi-hop routing and DSDV does not provide an efficient way to do broadcasting. Therefore, the algorithm based on NASCENT is much more efficient thanks to the single-hop unicastings and the collision avoidance GBCAST (Fig. 8 lines 21–23). Note that although the messages used to pass the token are a part of NASCENT, they are counted in Table II since these messages are involved in operations (e.g., broadcasts) instead of coping with network perturbations.

Actually, there are two other network layer services that would compete with NASCENT: flooding and multicast (e.g., MAODV [30], ODMRP [31]). The complexity of flooding is comparable to NASCENT+RB in the case of broadcast (but without any reliability guarantee), but 2 flooding messages and n (multi-hop) unicast messages are needed to achieve 1 mutual exclusion [32]. With NASCENT, up to n mutual exclusions can be achieved with n (single-hop) unicast messages (which correspond to the cost of circulating the token). Multicast protocols perform broadcasts within a subset of network nodes, whose goal is different from VAG requirements. They are less efficient than flooding for network-wide broadcast, and they still have the aforementioned drawbacks of flooding: i.e., lack of reliability guarantee and efficient support to other distributed algorithms.

VI. SIMULATIONS

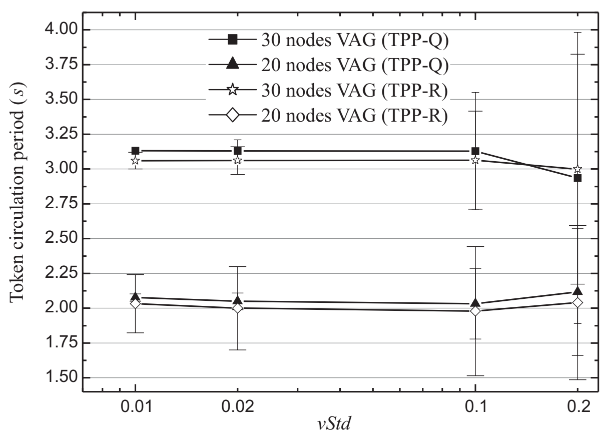
Having provided complexity-based analysis, we use simulations to verify the ability of NASCENT to support the envisioned applications.

A. Simulation Setup

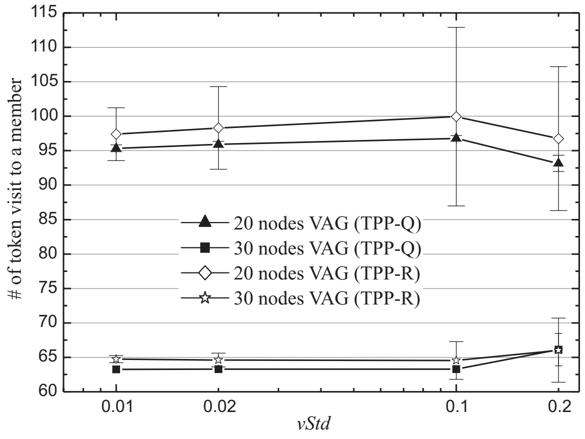
We take *ns-2.26* as the simulation platform. We use IEEE 802.11 with 2Mbps transmission rate as the wireless MAC but reduce the nominal range from 250m to 100m, in order to make our simulations more realistic. A two-ray ground reflection model is used for radio propagation.

We simulate VAGs with 20 and 30 nodes in a square area of 1km^2, during 200 seconds of simulated time. Initially, members of a VAG are randomly distributed within a region of 250m$\times$250m such that they are in the "vicinity" of each other. The movement pattern is defined by a group mobility model similar to [17] where the following process is repeated[5]: a VAG chooses a *group speed* uniformly distributed between zero and a maximum value as well as a random direction, then each member chooses a velocity following a 2-D normal distribution parameterized by the group speed and a standard deviation and begins to move for a certain time period. Upon timing out, the VAG remains static for some pause time. Both moving time and pause time are described by a uniform distribution between zero and a maximum value. Note that assuming a random moving time instead of an arbitrary destination partially solves the problem of decreasing average speed pointed out by [33], which allows us to have a short simulation time (e.g., 200s) without any warming-up period. In this mobility model, the only parameter that has a significant impact on the performance of NASCENT is the standard deviation of individual member velocities ($vStd$ hereafter). Therefore, we fix the maximum values of group speed, moving

[5]This model is not compatible with some of the cooperative driving scenarios described in Section II. Dedicated traffic modeling should be applied for detailed investigations on these cases, so we leave it as future work.



(a)



(b)

Fig. 9. The distributions of (a) token circulation period and (b) number of token visits to a member under different $vStd$s, with $T_s = 100$ms.

time, and pause time to 20m/s, 50s, and 10s, respectively, and test NASCENT only under different $vStd$s.

We set the beacon period to 200ms and test NASCENT under $T_s = 100$ms and 10ms (defined in Section IV-C.3). We also assume a 18-byte beacon message and a 50-byte token message. Members perform NASCENT initialization within the first 2 seconds, then the member with the smallest id (i.e., the sink) generates a token and starts to circulate it. Each simulation is carried out 10 times with different scenario files.

B. Stability of Token Circulation

In a static network, we say that the token circulation is stable if the token visits every member in a cycle and the period of a cycle remains constant. According to *Property* 3, NASCENT meets this requirement. However, no protocol can be qualified with this criterion in networks with a dynamic topology and membership, notably because the token cannot visit a member temporarily broken from the network in the current cycle. Therefore, we take an alternative criterion: a token circulation is stable if (i) the distribution of the circulation period has a small variance and (ii) the token visits each member infinitively often. The stability of token circulation under different $vStd$s is a major performance index of NASCENT, because a stable token circulation guarantees the correctness of upper layer distributed algorithms and indicates an effective membership tracking.

Fig. 9(a) and (b) show the verification of the conditions (i) and (ii), respectively. The value of $vStd$ in the figure is the ratio between a standard deviation and a mean value (the group speed for the polar distance of the velocity and π for the polar angle). Fig. 9(a) shows distributions of the circulation period, and Fig. 9(b) presents distributions of the number of token visits to a member (a way to check condition (ii) within limited simulation time). Each distribution is characterized by a mean value and a standard deviation. The mean values of the circulation period are all quite close to nT_s, which matches the prediction by PROPERTY 3 ($D_{max}nT_t$ can be neglected if $T_t \ll T_s$). In the quasi-static scenarios with $vStd = 0.01$ (e.g., a group of wireless game players in a moving train), the token is stably circulated for both 20 and 30 node VAGs, since the standard deviations of the circulation period and the number of token visits to a member are relatively small (mainly due to the fluctuation of wireless links). The stability is modestly degraded in scenarios with $vStd = 0.02, 0.1, \text{and } 0.2$ (real life examples could be teamed robots).

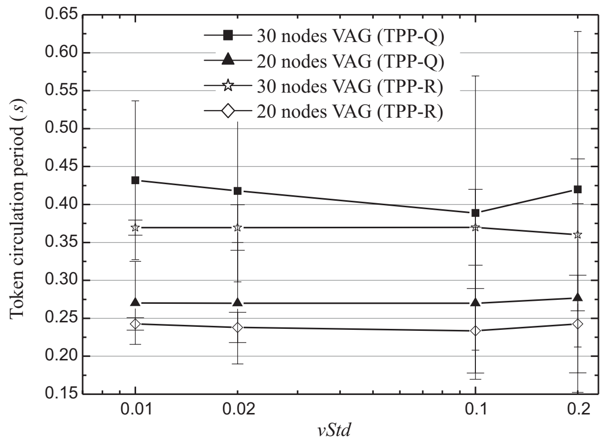
C. Circulating Token with Smaller T_s

Fig. 9(a) shows that the token circulation period is in the order of seconds with $T_s = 100$ms (i.e., the token stays 100ms at each member in a cycle). Fig. 10 shows that, with $T_s = 10$ms, NASCENT provides a circulation period of several hundred milliseconds. In practice, applications need a certain amount of time to process backlogged operations upon acquiring the token. Therefore, the value of T_s (and thus the circulation period) is defined according to the processing capacity of given devices.

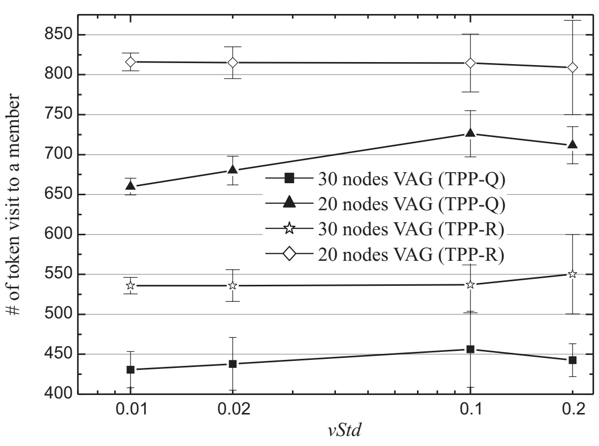
For wireless multi-player gaming, $T_s = 10$ms is a reasonable value, because the device needs only to exchange some state information with the token. Therefore, the resulting circulation period is short enough to prevent impatient players from giving up the game. Cooperative robotics need a relatively long T_s since the behaviors of a token holder could involve mechanical movements. Fortunately, a circulation period in the order of seconds is tolerable, again due to the low-speed mechanical movements involved in the applications.

D. Comparing TPP-Q with TPP-R

Both TPP-Q and TPP-R perform well in most cases. However, we have observed in simulations that there exist some special situations where these algorithms fail to guarantee the stability of token circulation. For example, in Fig. 11(a), member a cannot pass the token to member b due to a link breakage, but b is not aware of this until the breakage is detected by LMTP after τ_b beacon periods. TPP-Q, in this case, leads to a virtual partition since members that have sent requests to b will not receive the token in the current cycle although the network

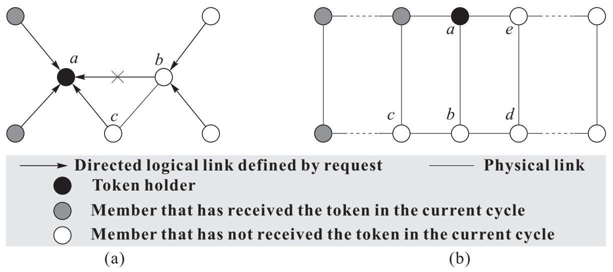


(a)



(b)

Fig. 10. The distributions of (a) token circulation period and (b) number of token visits to a member under different $vStd$s, with T_s = 10ms.



a
b
c
e
d
Directed logical link defined by request
Physical link
Token holder
Member that has received the token in the current cycle
Member that has not received the token in the current cycle
(a)
(b)

Fig. 11. Special cases where (a) TPP-Q and (b) TPP-R fail to guarantee the stability of token circulation.

is connected. A worst-case scenario for TPP-R is shown in Fig. 11(b). Assume that $addr$ is used to break a tie, the token will follow the path $a \rightarrow b \rightarrow c \rightarrow \cdots$. Therefore, the token will not visit members to the right side of a before it revisits all members to the left side of a. The significance of the effect of the virtual partition shown in Fig. 11(a) depends on the values of T_s and the beacon period. If they are comparable or T_s is larger, the effect becomes less significant and TPP-Q outperforms TPP-R, otherwise TPP-R wins. This conclusion can be observed in Fig. 9 and 10. The conclusion also suggests different application domains (defined by required T_s and beacon period) for TPP-Q and TPP-R.

VII. Conclusion

In this paper, we focus on small-scale ad hoc networks (or VAGs) that involve mainly group-oriented communications. Our NASCENT aims at replacing routing protocols for VAGs applications; it integrates a directed acyclic graph (DAG) with token circulation to concurrently support various distributed services. While the DAG is used to route messages to and from a token holder and to bind VAG members together, the circulated token grants temporary control to its holders. The two components greatly facilitate the implementation of distributed algorithms based on NASCENT. The light-weight membership service embedded in NASCENT requires only localized operations. NASCENT also includes new algorithms to initialize a token-oriented DAG among an emerging VAG and to recompose the DAG upon the VAGs' merging. These are all crucial functions for coping with the transiency of VAGs.

We have given examples of building distributed algorithms on top of NASCENT and perform both complexity-based analysis and simulations. We show that the complexity of maintaining NASCENT is of the same magnitude as unicast routing protocols, but that NASCENT greatly reduces the complexity of building and running distributed algorithms, compared with unicast and broadcast/multicast routing protocols as well as flooding. Further simulations prove the feasibility of NASCENT; it circulates a token stably and timely even when networks undergo topology changes.

In terms of future work, we expect to gather more data on the performance of NASCENT in different environments with the implementation and field tests; this will further encourage the deployment of NASCENT.

VIII. Acknowledgements

The authors wish to thank the anonymous reviewers for their valuable feedback for this work. The authors would also like to thank Mark Felegyhazi, Naouel Ben Salem, Rachid Guerraoui, Jim Pugh, and Nitin H. Vaidya for several instructive comments.

References

[1] C. Perkins, E. Belding-Royer, and S. Das, *Ad hoc On-Demand Distance Vector (AODV) Routing*, 2003, rFC 3561 (draft standard). IETF.

[2] D. Johnson, D. Maltz, and Y.-C. Hu, *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)*, April 2003, internet-Draft, draft-ietf-manet-dsr-09.txt.

[3] L. Ji and M. Corson, "Differential Destination Multicast - a MANET Multicast Routing Protocol for Small Groups," in *Proc. of IEEE INFOCOM'01*, 2001.

[4] J. Luo, P. Eugster, and J.-P. Hubaux, "Route Driven Gossip: Probabilistic Reliable Multicast in Ad Hoc Networks," in *Proc. of IEEE INFOCOM'03*, 2003.

[5] C. Gui and P. Mohapatra, "Scalable Multicasting in Mobile Ad Hoc Networks," in *Proc. of IEEE INFOCOM'04*, 2004.

[6] G.-C. Roman, Q. Huang, and A. Hazemi, "Consistent Group Membership in Ad Hoc Networks," in *Proc. of IEEE/ACM ISCE'01*, 2001.

[7] Z. Haas and B. Liang, “Ad Hoc Mobility Management with Uniform Quorum Systems,” *IEEE/ACM Trans. on Networking*, vol. 7, no. 2, pp. 228–240, 1999.

[8] J. Liu, K. Sohraby, Q. Zhang, B. Li, and W. Zhu, “Resource Discovery in Mobile Ad Hoc Networks,” in *Ad Hoc Wireless Networks*, M. Ilyas, Ed. CRC Press, 2003, ch. 26.

[9] N. Malpani, J. Welch, and N. Vaidya, “Leader Election Algorithms for Mobile Ad Hoc Networks,” in *Proc. of ACM DIAL-M’00*, 2000.

[10] J. Walter, J. Welch, and N. Vaidya, “A Mutual Exclusion Algorithm for Ad Hoc Mobile Networks,” *Wireless Networks*, vol. 7, no. 6, pp. 585–600, 2001.

[11] U. Kozat and L. Tassiulas, “Network Layer Support for Service Discovery in Mobile Ad Hoc Networks,” in *Proc. of IEEE INFOCOM’03*, 2003.

[12] C. Carter, S. Yi, P. Ratanchandani, and R. Kravets, “Manycast: Exploring the Space between Anycast and Multicast in Ad Hoc Networks,” in *Proc. of ACM MobiCom’03*, 2003.

[13] X. Défago, “Distributed Computing on the Move: From mobile computing to cooperative robotics and nanorobotics,” in *Proc. of ACM POMC’01*, 2001.

[14] J. Redi, “Wireless Networking for Mobile Robots,” in *Invited talk for Winlab/Berkeley FOCUS’99*, 1999.

[15] C. Jones and M. J. Matarić, “Communication in Multi-Robot Coordination,” USC CRES-04-001, Tech. Rep., 2004, http://cres.usc.edu/pubdb_html/files_upload/355.ps.

[16] D. Reichardt, M. Miglietta, L. Moretti, P. Morsink, and W. Schulz, “CarTALK 2000 - Safe and Comfortable Driving Based Upon Inter-Vehicle-Communication,” in *Proc. of IEEE IV’02*, 2002, http://www.cartalk2000.net.

[17] K. Wang and B. Li, “Efficient and Guaranteed Service Coverage in Partitionable Mobile Ad-hoc Networks,” in *Proc. of IEEE INFOCOM’02*, 2002.

[18] N. Maxemchuck and D. Shur, “An Internet Multicast System for the Stock Market,” *ACM Trans. on Computer Systems*, vol. 19, no. 3, pp. 384–412, 2001.

[19] S. Dolev, E. Schiller, and J. Welch, “Random Walk for Self-Stabilizing Group Communication in Ad-Hoc Networks,” in *Proc. of IEEE SRDS’02*, 2002.

[20] S. Nishio, K. Li, and E. Manning, “A Resilient Mutual Exclusion Algorithm for Computer Networks,” *IEEE Trans. on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 244–355, 1990.

[21] E. Gafni and D. Bertsekas, “Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology,” *IEEE Trans. on Communications*, vol. 29, no. 1, pp. 11–18, 1981.

[22] J. Luo and J.-P. Hubaux, “NASCENT: Network Layer Service for Vicinity Ad-hoc Groups,” EPFL, Tech. Rep. IC/2004/55, 2004, http://ic2.epfl.ch/publications/documents/IC_TECH_REPORT_200455.pdf.

[23] V. Park and M. Corson, “A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks,” in *Proc. of IEEE INFOCOM’97*, 1997.

[24] N. Malpani, N. H. Vaidya, and J. L. Welch, “Distributed Token Circulation in Mobile Ad Hoc Networks,” in *Proc. of IEEE ICNP’01*, 2001.

[25] V. Hadzilacos and S. Toueg, “Fault-Tolerant Broadcasts and Related Problems,” in *Distributed Systems*, 2nd ed. Addison-Wesley, 1993, ch. 5, pp. 97–145.

[26] J. Garcia-Lunes-Aceves, “Loop-Free Routing Using Diffusing Computations,” *IEEE/ACM Trans. on Networking*, vol. 1, no. 1, pp. 130–141, 1993.

[27] C. Perkins and P. Bhagwat, “Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers,” in *Proc. of ACM SIGCOMM’94*, 1994.

[28] M. Corson and A. Ephremides, “A Distributed Routing Algorithm for Mobile Wireless Networks,” *Wireless Networks*, vol. 1, no. 1, pp. 61–81, 1995.

[29] E. Royer and C.-K. Toh, “A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks,” *IEEE Personal Communications*, vol. 6, no. 2, pp. 46–55, 1999.

[30] E. Royer and C. Perkins, “Multicast Operation of the Ad-hoc On-demand Distance Vector Routing Protocol,” in *Proc. of ACM MobiCom’99*, 1999.

[31] S.-J. Lee, W. Su, and M. Gerla, “On-Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks,” *ACM/Kluwer Mobile Networks and Applications*, vol. 7, no. 6, pp. 441–453, 2002.

[32] S. Nesargi and R. Prakash, “MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network,” in *Proc. of IEEE INFOCOM’02*, 2002.

[33] J. Yoon, M. Liu, and B. Noble, “Random Waypoint Considered Harmful,” in *Proc. of IEEE INFOCOM’03*, 2003.

APPENDIX
PSEUDOCODES FOR TOKEN PASSING PROTOCOL (TPP)

We report only the pseudocodes of TPP-Q in Fig. 12 but refer to [22] for detailed algorithm descriptions of TPP (including both TPP-Q and TPP-R).

name	functions
epoch	A counter that logs which cycle of token circulation the member is in.
reqsent	A pointer that directs to another member to which a token request has been sent.
$\mathcal{Q}$	A queue that stores token requests ($token_req$s) from neighbors.
tokenheld	A flag that designates whether the member holds the token or not.

```
upon ViewChg do
  if $|\mathcal{Q}_i| \neq \emptyset$ then
    if $tokenheld_i \wedge (token.gid > lview_i.gid)$ then
      $tokenheld_i \leftarrow$ false; Store($token$)
      FwdReq(ture)
    else
      $priority \leftarrow \min_{req \in \mathcal{Q}_i}\{req.epoch\}$
      for all $nid \notin lview_i$ and
          all $nid \in lview_i$ s.t. $nid < id_i$ do
        Delete($\mathcal{Q}_i, nid.addr$)
      if $id_i > \min_{nid \in lview_i}\{nid\}$ then
        if $(id_i < reqsent_i) \vee (reqsent_i \notin lview_i)$ then
          FwdReq(true)
        else if $priority \neq \min_{req \in \mathcal{Q}_i}\{req.epoch\}$ then
          FwdReq(false)
      else
        $reqsent_i \leftarrow id_i$

procedure FwdReq($redirect$)
  if $redirect$ then
    $reqsent_i \leftarrow \min_{nid \in lview_i}\{nid\}$
  $token\_req.saddr \leftarrow addr_i$
  $token\_req.epoch \leftarrow \min_{req \in \mathcal{Q}_i}\{req.epoch\}$
  Send($token\_req$) to $reqsent_i.addr$

upon Recv($token\_req$) from $id_j$ do
  if $(|\mathcal{Q}_i| \neq \emptyset) \wedge (id_i < id_j)$ then
    $priority \leftarrow \min_{req \in \mathcal{Q}_i}\{req.epoch\}$
    Enqueue($\mathcal{Q}_i, token\_req$)
    if $priority \neq \min_{req \in \mathcal{Q}_i}\{req.epoch\}$ then
      FwdReq(false)
    if $\nexists nid \in lview_i$ s.t. $nid = id_j$ then
      Insert($lview_i, id_j$)                /* incur ViewChg */

upon Recv($token$) from $id_j$ do
  if IsStoreToken() then
    MergeToken($token$)
  $reqsent_i \leftarrow$ getIDinView($lview_i$, Dequeue($\mathcal{Q}_i$))
  if $reqsent_i = id_i$ then
    $tokenheld \leftarrow$ true
    Deliver($token$)                  /* to the upper layer */
  else
    Send($token$) to $reqsent_i.addr$; FwdReq(false)
  if $\nexists nid \in lview_i$ s.t. $nid = id_j$ then
    Insert($lview_i, id_j$)               /* incur ViewChg */

upon Release($token$) do
  $reqsent_i \leftarrow$ getIDinView($lview_i$, Dequeue($\mathcal{Q}_i$))
  Send($token$) to $reqsent_i.addr$
  $tokenheld_i \leftarrow$ false; $epoch_i \leftarrow epoch_i + 1$
  $req.saddr \leftarrow addr_i$; $req.epoch \leftarrow epoch_i$
  Enqueue($\mathcal{Q}_i, req$)
  FwdReq(false)
  $token\_timer_i \leftarrow 0$

upon Timeout($token\_timer_i$, $\tau_{part}$) do
  $init_i \leftarrow$ true; $init\_timer_i \leftarrow 0$
  $lview_i.gid.\beta \leftarrow lview_i.gid.\beta - 1$
  $lview_i.gid.addr \leftarrow addr_i$
```

Fig. 12. Response to different events in TPP of member i