Integrating Overlay Protocols for Providing Autonomic Services in Mobile Ad-hoc Networks

Panagiotis Gouvas†, IEICE Student member, Anastasios Zafeiropoulos††, IEICE Student member, Athanassios Liakopoulos†, Gregoris Mentzas† and Nikolas Mitrou††

SUMMARY Next generation network characteristics increase the complexity in the design and provision of advanced services, making inappropriate the selection of traditional approaches. Future networks are becoming larger in scale, more dynamic and more heterogeneous. In order to cope with these requirements, services are expected to adapt to environmental conditions and require minimum human intervention. In this paper a new model for providing autonomous and decentralized services is proposed, especially focusing on mobile ad hoc networks (MANETs). Using a newly proposed four-layered approach, service development may be realized independently from the underlying physical network. In a reference implementation, it is demonstrated that it is possible to set up an overlay network that hides any network changes from the service layer. Multiple mechanisms have been adapted in order to efficiently –in terms of message exchanges and convergence time– operate over an ad hoc environment. Finally, it is demonstrated that a specific service could operate over a dynamic network with multiple failures.

key words: Autonomic services, MANETs, adapted T-MAN, DSR, p2p, overlay networks.

1. Introduction

The trend towards the Future Internet is the transition to networks that provide highly distributed, pervasive and communication-intensive services. These services pose new requirements on the underlying network infrastructure [1-2] that derive from the need to manage dynamic, heterogeneous and complex networks and to enable seamless transition from current to future networks. In order to cope with these requirements, such services will be expected to (i) be aware of the context and the environment in which they operate, (ii) self-configure and self-adapt according to the network conditions that they sense and (iii) require minimum feedback from the end-user and no explicit human intervention.

The incorporation of autonomic principles for the transition from a service agnostic Internet to a service- and self-aware Internet is necessary [3]. Autonomicity is required in order to accomplish management of dynamic, heterogeneous and complex networks, where each network entity needs to be able to take network optimization decisions. The monitoring components of an autonomic network should be continuously adapted, in a flexible manner, to an ever changing network infrastructure without ignoring the goals and the constraints that the administrator has set for the network as a whole.

These challenges can be addressed through the creation and maintenance of overlay networks [5]. Overlay networks create a well-defined virtual topology above the underlying network infrastructure, where autonomic functionalities can be implemented and advanced services may be provided. However, new methodologies for implementing and operating overlays are needed. In particular new mechanisms are required that permit overlays to structure their topology, define their routing scheme, and manage their resources independently [4-5]. The transition to overlay networking and the incorporation of autonomic characteristics in next generation networks is most crucial in ad-hoc and mesh networks since they present specific challenges, especially due to the following characteristics: (i) dynamic topology (links are established / torn down frequently), (ii) unreliable operation (nodes may be disconnected at any time), (iii) complexity (lack of centralized control or network hierarchy) and (iv) loose control of nodes (roles in the overlay network are automatically assigned).

Peer-to-peer (p2p) networks constitute the best candidate for addressing most of these challenges (unreliable operation, complexity, loose control of nodes) [23]. In p2p networks a dynamic assignment of tasks among peers is realized and the provided services are based on their direct cooperation. Peer nodes share the available resources, provide redundancy due to distribution and replication of data and thus impose minimal requirements to the infrastructure. Ad-hoc deployment increases the self-organization level of the network and permits its operation also in an unstable environment with loosely connected nodes. Furthermore, despite the apparent chaos of periodic random changes to the membership of the network, p2p networks provide provable guarantees about performance.

However, the adoption of current p2p techniques necessitates the existence of a fixed network topology [16, 10]. With the term “fixed”, we refer to wired and wireless networks that have formulated their physical
topology and do not present any topology changes. Over a formulated physical topology, specific p2p functions can be applied and a wide range of services can be deployed. This is not the case in MANETs. Each device in a MANET is free to move independently in any direction, and thus frequently change its connectivity to other devices.

In this paper an approach is proposed for autonomic provision of services in MANETs. Focus is given on issues such as topology formulation, adaptation to topology changes, scalability and stability. Gossiping and exhaustive tree-based techniques for topology formulation are compared and specific adaptations to the routing protocols are proposed. The described approach is generic enough and can be used for the design and implementation of autonomic p2p services, from network monitoring and management to chatting and social networking applications.

The paper is organized as follows; section two briefly presents the current work in the field of autonomic services in MANETs. Section three describes the basic components and the design principles of our approach. Section four details specific implementation issues while section five describes the evaluation scenario and the simulation results. Finally, section six concludes the paper with a short summary of our work and a discussion of open issues and future work.

2. Related Work

2.1 Transition to autonomic service-oriented networks

The management of a dynamic, heterogeneous and complex network may become more robust and stable, provided that each network node is able to take local or network optimization decisions without strict centralized control [3]. Autonomic capabilities have to be integrated to network monitoring algorithms in order to overcome the inefficiencies of traditional network management systems usually based on centralized architectures. In an autonomic environment, the network nodes themselves are able to detect, diagnose and repair failures, and adapt their behavior according to generic network policies [6]. Several approaches have been proposed for autonomic network architectures, such as 4D [7], CONMan [8], a Knowledge Plane for the Internet [9] and FOCALE [6].

Our approach, as presented in the following sections, adheres principles from the GANA architecture [11]. In GANA, autonomicity is realized through the design of control loops operating within network nodes/devices and the network as a whole. In our approach, self-* functionalities (e.g. self-organization, self-optimisation, self-healing) are designed based on interactions among the existing autonomic entities. Interactions among nodes in the same networking neighborhood have been defined while no centralized entities or human intervention is necessary for the provision of services. However, the explicit mapping of our approach to the GANA architecture is out of the scope of this paper.

Current efforts indicate that the design of systems that combine the concepts of self-organization and decentralization can be realized through the creation of overlay networks over the underlying network infrastructure [12]. Overlay networks are a very successful paradigm that enables both the construction and the evolution to the future service oriented networks. In [4] it is stated that in order to address Future Internet challenges new methodologies for implementing and operating overlays are needed with possible re-layering of the existing IP protocol layering. In [3] a new autonomic management architectural model that facilitates the transition from a service agnostic Internet to service aware network by creating a self-managing virtual resource overlay is described. Finally, in [13] an approach is presented for exploiting the autonomic functionalities of peer-to-peer based overlay technology to form an autonomic service control for next generation networks.

2.2 Use of p2p protocols and DHTs in MANETs

The optimal way to manage resources automatically in MANETs is through the adoption of techniques that are applied in p2p protocols. This is due to the reason that p2p and MANETs present many similarities that can be exploited, such as decentralization, self-organization, and dynamically changing set of nodes [14-15-16-17]. Furthermore, characteristics, such as robustness and reliability, provided by the delegation of tasks to several nodes, make p2p techniques a perfect fit for content dissemination in MANETs.

P2p protocols rely on decentralized structures, such as Distributed Hash Tables (DHTs). A DHT is a structure that is collaboratively built by all participating nodes of a network and provides a lookup service for resources that are published by these nodes. Resources are utilized transparently and data is stored/retrieved seamlessly in the network. Several existing approaches try to improve service discovery performance and efficient dissemination of content by deploying the concept of DHT in MANETs [17-18].

Limited work has been done for the overlay topology creation and maintenance in MANETs [19-20]. Although many reference implementations exist for p2p protocols -such as Chord [21], Pastry [22], and many others [23]-, these (protocols) operate under the assumption that a fixed network topology is already established, on top of which the overlay network is created and maintained. P2p protocols allow a user to store and retrieve key-value pairs in the network seamlessly without knowing where the actual data is stored. The task of overlay topology
construction/maintenance is undertaken by low level mechanisms which in most of the cases are centralized or semi-centralized. Indicatively, such mechanisms are used in the Gnutella network [24], in which topology creation may be achieved by using a pre-defined address-list of working nodes included within a compliant client or by using web caches of known nodes, a.k.a. Gnutella web caches. Similarly, Chord pre-assumes that nodes are ordered in a ring and are aware of their successor and predecessor in the overlay ring topology. Chord also relies on underlying mechanisms for the overlay network bootstrapping [21]. In conclusion, p2p protocols are able to react to topology changes (and automatically re-assign key-value pairs) but are not responsible to create and maintain the overlay topology.

Due to the dynamic nature of the MANETs, centralized or static mechanisms cannot be used for the bootstrapping of an overlay network. Thus, in the proposed approach we handle issues related with topology formulation and maintenance in MANETs, where the existence of a stable network topology is not considered as granted.

2.3 Routing protocols and topology formulation mechanisms in MANETs.

Given the existence of a bootstrapped network, several mechanisms have been proposed for the maintenance of the overlay topology in MANETs. These mechanisms vary from gossiping techniques [25] to exhaustive techniques [26]. The common characteristic of these mechanisms is that they pre-assume guaranteed communication among the network nodes. However, their principles are completely different. Gossiping techniques attempt to identify the relative position of one node in the overlay topology by consulting adjacent nodes in the overlay topology. For these techniques the key indicator is convergence i.e. how many messages are required so as all nodes know their relative position to the overlay topology. Alternatively, exhaustive techniques attempt to pass through all nodes periodically in order to identify their relative position in the overlay topology.

Since the topology formulation mechanism prerequisite the communication capability among the network nodes, very crucial is the selection of the routing protocol that will be applied in the MANET. In MANETs there are no dedicated routers. Multi-hop communication is established among neighboring nodes and packets are transmitted from a source node to a destination node. Several routing protocols have been proposed that perform variously depending on the type of traffic, the number of nodes, the rate of mobility, etc.

These protocols can be classified in three categories: (i) proactive routing protocols where the routes to all the destination are determined at the start up, and maintained by using a periodic route update process, (ii) reactive routing protocols where routes are determined when they are required according to a route discovery process and (iii) hybrid routing protocols that combine both proactive and reactive protocol properties to come up with a better routing scheme. According to the network topology and the network characteristics, the most suitable routing protocol may be selected [27-28-29]. Reactive routing protocols are more suitable for MANETs due to their ability to cope with rapidly changing topologies. However, they behave worse than proactive routing protocols in terms of scalability. In our approach, a reactive routing protocol is selected but specific modifications are proposed in order to handle scalability issues.

3. Autonomic approach for services provision

3.1 Generic principles

The proposed layered-approach aims at the provision of a generic framework that will facilitate the design and development of autonomic and decentralized services in MANETs (see Figure 1). The introduction of the different layers of the proposed approach is necessary due to the need to address the following challenges: a) efficiently utilize available network resources in a dynamic environment, b) provide services independently from the underlying topology, c) ensure reliability of services in case of network topology changes and d) reduce the management complexity and increase flexibility to application developers. In order to address these challenges, autonomic functionalities have to be incorporated. The following self-* properties have been defined [6] and should be supported by an autonomic system: self-configuration, self-optimization, self-awareness and self-healing.

![Fig. 1 Autonomic Services in MANETs](image-url)
Existing protocols that satisfy partially the challenges described above were considered during the design of the proposed approach. There is no existing work on how to combine existing protocols for achieving autonomic service provisioning and how different protocols could interact using predefined interfaces.

Taking into account these considerations, the proposed approach is focusing on a) defining concrete layering for enabling autonomic service provisioning in MANETs, b) specifying the discrete functionality of each layer and the interfaces between them and c) resolving conflicts between existing protocols, specifically in the field of the overlay topology construction.

The creation and maintenance of an overlay topology that logically interconnects all the participating nodes in the physical network is critical in our approach. Any node that connects to the ad-hoc network has to join to the overlay network. The overlay network is formulated during the topology stabilization phase in an autonomic manner and hides any details of the underlying physical infrastructure, e.g. link establishment or torn down, node failures, node mobility, etc. In case of multiple changes in the physical topology, the overlay network is able to adapt quickly to the new environment (re-stabilization). Furthermore, recovery from failures can be easily succeeded based on information that is available in the network. All these tasks are realized without the intervention of the network administrator.

After the overlay network is established, participating nodes are able to store and retrieve data using typical p2p protocols. Every node that wishes to store a key-value pair, or query a value based on a key, can achieve it by using a Distributed Hash Table (DHT) [18] that operates on-top of the overlay topology. In a similar way, several applications can be built taking under consideration the existence of a high level API put(key, value) and get(key) that would interact with a DHT protocol that operates on-top of a non-reliable MANET.

Provided services are designed based on the assumption of collaboration and dissemination of information among the participating nodes. These services can be fully decentralized as data and functionality is allocated in different nodes at the overlay network. Some functions may be delegated to more than one node for higher reliability. In case of changes or failures, roles may be re-assigned autonomously and performance guarantees may be assured for the services provision.

### 3.2 Description of the four-layered approach

We propose a four-layered scheme based on the functionality requirements imposed by the provided services and the underlying physical networking environment. As shown in Figure 2, the following **four layers** are defined; i) **Neighbor-to-Neighbor layer**, ii) **Routing layer**, iii) **Topology Maintenance layer**, and iv) **DHT layer**. Each layer has a discrete role, implements different mechanisms and specifies its messages types. The proposed layered approach is independent from the selection of p2p protocols, topology formulation mechanisms and routing protocols. Therefore, any combination of different protocols may be selected and proper adaptations may be proposed.

![Four-Layered Approach](image-url)

The **Neighbor-to-Neighbor layer** is responsible for delivering an upper-layer frame from a neighbor to another neighbor. No information from the upper layer is necessary for the delivery. Two types of messages are used; i) **MAC_SEND** in order to achieve one way frame delivery from neighbor X to neighbor Y and ii) **MAC_ACK** in order to achieve acknowledgment for successful message delivery from neighbor Y back to neighbor X. Also, this layer is responsible for maintaining (i.e. initializing and keeping up-to-date) the routing cache of the **Routing layer** since when neighbor-to-neighbor links are created or destroyed the related routing information has to be updated.

The **Routing layer** is responsible for delivering an upper-layer frame from a node X to another node Z. It is assumed that node X is not aware how node Z can be reached. The layer is also agnostic of the reason that node X wants to communicate with node Z. This layer relies on routing protocol for frame forwarding across the network. As we stated in section 2.3, in case of MANETs it is suggested the use of a re-active routing protocol.

The **Topology Maintenance layer** is responsible for formulating a virtual topology of the participating nodes. In our case the desired topology is a ring (imposed by the use of Chord). Consequently, this layer undertakes the task of identifying the relative position of each node in the overlay topology without being based in centralized or semi-centralized techniques.

The **DHT layer** is responsible for maintaining a distributed hash table that is bootstrapped over the stabilized overlay topology. For this purpose any existing p2p protocol may be used. These protocols are (semi or fully) decentralized and in addition to storage and retrieval functionality- may succeed load balancing,
reduce bandwidth consumption and improve data reliability across the network.
The following interfaces have been defined for the communication among the different layers:

- The Neighbor-to-Neighbor layer provides to the Routing layer routing information for existing neighbors that is stored in the routing cache of each node, through the validateRoutingCaches() function. The Neighbor-to-Neighbor layer provides also medium-level acknowledgments to the Routing layer for neighbor-to-neighbor communication, through the transfer_Packet() function.
- The Routing layer provides routing functionality to upper layers through the routePacket() function. Additionally, it exposes topology information derived directly from the routing caches to the Topology Maintenance layer, through the getRoutingInfo() function. It is up to the Topology Maintenance layer to utilize this information for optimizing its mechanisms or not.
- The Topology Maintenance layer provides information to the DHT layer regarding the relative position of a node in the overlay network (e.g. the predecessor and successor in case of a ring topology) through the getRelativePosition() function. In case of changes in the network topology, stabilization procedures take place in both layers. The Topology_Stabilize() function is used for re-ordering the overlay topology (e.g. ring in our case) and triggers the DHT_Stabilize() function that is used for the re-assignment of key-value pairs that are assigned in the overlay network nodes.

The following interfaces have been defined for the communication among the different layers:

- The Neighbor-to-Neighbor layer provides to the Routing layer routing information for existing neighbors that is stored in the routing cache of each node, through the validateRoutingCaches() function. The Neighbor-to-Neighbor layer provides also medium-level acknowledgments to the Routing layer for neighbor-to-neighbor communication, through the transfer_Packet() function.
- The Routing layer provides routing functionality to upper layers through the routePacket() function. Additionally, it exposes topology information derived directly from the routing caches to the Topology Maintenance layer, through the getRoutingInfo() function. It is up to the Topology Maintenance layer to utilize this information for optimizing its mechanisms or not.
- The Topology Maintenance layer provides information to the DHT layer regarding the relative position of a node in the overlay network (e.g. the predecessor and successor in case of a ring topology) through the getRelativePosition() function. In case of changes in the network topology, stabilization procedures take place in both layers. The Topology_Stabilize() function is used for re-ordering the overlay topology (e.g. ring in our case) and triggers the DHT_Stabilize() function that is used for the re-assignment of key-value pairs that are assigned in the overlay network nodes.

In Figure 3, a snapshot of the physical network topology (solid lines) and the logical overlay topology (dashed lines with arrows) is depicted. Initially, node 3 does not exist in the network and the key-value pairs have already been assigned to the network nodes by applications that run on the existing nodes (i.e. applications that use DHT). Then, node 3 is physically connected with node 1 and node 4 and the corresponding overlay topology is updated. It is the responsibility of Topology Maintenance layer to find the successor for each node. However, it is not the Topology Maintenance layer’s responsibility to re-assign key-values according to the DHT’s assignment algorithm. The Topology Maintenance layer must inform the DHT layer that the relative position for the node in the overlay topology (e.g. ring in case of Chord) has changed. Then it is up to DHT layer to reassign key-value pairs. This re-assignment will be addressed as DHT re-stabilization while the updated knowledge for the relative position in the overlay topology is called Topology stabilization.

4. Implementation Details

In this section, a reference implementation of the proposed approach is described in detail. Specific mechanisms and protocols are selected for each independent layer. In the DHT layer we have selected Chord [26] as a p2p protocol, in the Topology Maintenance layer we follow gossiping and exhaustive tree-based techniques, in the Routing layer we have adapted the Data Source Routing (DSR) [30] protocol and in the Neighbor-to-Neighbor layer communication is established through a simulation environment.

It is important to note that, in addition to adopting specific techniques and mechanisms, several adaptations - that were considered useful for our approach - are provided. Specifically we describe in detail how gossiping techniques for topology formulation and the DSR routing protocol can facilitate each other. These adaptations are stated in detail in the following subsections.

4.1. Adapted DSR Routing Protocol

Prior to analyzing the overlay topology algorithms that have been implemented we briefly refer to the reactive routing protocol that has been adopted and customized. The Data Source Routing (DSR) [30] is chosen for our reference implementation. Firstly, we describe the basic characteristics of DSR and the implementation details and secondly the proposed adaptations that are correlated with the Topology Maintenance layer.

DSR is relied in two mechanisms; a) Route Identification and b) Route Maintenance. The protocol if fully reactive; hence every time one node wants to communicate with another node, it initiates a route request mechanism. Note that it is not DSR’s responsibility to know why one node wants to communicate with another node. In general, this is upper layer’s responsibility (Topology Maintenance layer in our case).

The Route Maintenance mechanism is used for route identification (see Figure 4). A specific message, called DSR Route_Request, is broadcasted. Each DSR Route_Request contains a request-id that is given by the initiator and the desired-destination node. This information remains intact all across the flooding procedure. The message also contains a header that logs the routing path of a route-request. Thus, a route-request message that is initiated by one node ends up to many route-request messages with the same request-id but with
different headers (because different pathways are followed). Each node that receives a DSR Route Request can perform three tasks:

· If a node is the destination-node it creates a route-reply message, called DSR Route Reply. This message, in parallel with the request, contains a reply-id (which is directly correlated with the request-id) and a header (which is actually the reversed route-request header), that remain intact until the reply reaches its destination. In case that a route-request reaches a node that knows a-priori the route to the destination node, it appends to the header the remaining route and creates a DSR Route Reply.

· If a node knows nothing about the destination node, it just forwards the DSR Route Request to its neighbors (excluding the neighbor that initially sent the request). In both replying and forwarding cases the node adds the request-id to a ‘served’ list. The ‘served’ list is a temporary list of all the request/response ids that are served by a specific node, after the processing of a DSR Route Request (the same applies in route-replies).

· If a node has the request-id of the DSR Route Request in its ‘served’ list, the node does nothing at all. This is very critical for avoiding loops.

When a DSR Route Reply reaches its destination, the node that initially initiated the route-request is ‘informed’ for the route to a specific node. This information is stored in the nodes’ routing cache. According to our implementation each node maintains a primary routing cache and a secondary routing cache. The primary routing cache contains only one route for every known node and specifically the route which is considered to be optimum (for our implementation optimum means shortest but in general optimum may refer to other network parameters such as low latency etc.). The secondary routing cache contains multiple alternative routes without loops for known nodes. In case that a broken link is identified (through DSR Route Error solicitation mechanism) both routing caches are revalidated. If an invalid route -that contains this link- is identified in the primary routing cache, the route is removed and substituted by a possible alternative valid route, requested from the secondary routing cache.

As explained above, the Routing layer is necessary so as two nodes can exchange an upper layer frame. In order to do so, a message called DSR Message Transfer is used. When a route to a destination is known then a DSR Message Transfer is initiated. This message contains a packet-id (which is directly correlated with the sender’s address), the destination node, the route that must be followed in the MANET in order to reach the destination and a flag that informs the destination node whether it should respond with a confirmation (acknowledgement). Finally, the DSR Message Transfer contains encapsulated upper layer data (topology formulation, DHT etc.).

When a DSR Message Transfer has to be transmitted from node A to node B and from node B to node C, the intermediate nodes are responsible to complete the transfer. Practically, if the DSR Message Transfer packet has reached node B and the header of the DSR Message Transfer is A->B->C, assuming that node B has direct connection with node C (because in the past a route_reply informed node A about that) and the connection between node B and node C is no longer available, then it is in the responsibility of node B to identify a new route to node C. Identification means usage of another route (consulting the routing cache) or initiation from scratch of a new route-request.

Whenever during a DSR Message Transfer a broken link is identified, then a message called DSR Route Error is initiated by the node that declares the broken link. The DSR Route Error contains a route-error-id and information about the broken link. This message is flooded all across the MANET. When a node receives a DSR Route Error it automatically removes from the primary and secondary cache all the occurrences of routes that contain this link. Afterwards each node re-evaluates its primary and secondary cache in order for the optimum (shortest) paths to be pushed in the primary cache.

At this point, we have to emphasize in some adaptations that we propose. The first adaptation regards the DSR Route Reply messages. Each DSR Route Reply contains a valid up-to-date path that is routed from the destination back to the route-request initiator, information that is extremely valuable also for the intermediate nodes. In our implementation all intermediate nodes overhear the route-replies that pass through them and enrich their routing cache. The second adaptation regards the introduction of the Hopes-To-Live
(HTL) parameter. As we already mentioned, the route-request mechanism uses the request-id in order to prevent infinite loops. The HTL parameter is added as a field in the DSR Route_Request. Every time a route-request is forwarded then the HTL is increased by one. When the maximum HTL is reached the node does not forward the DSR Route_Request messages any more.

The idea behind the HTL is the following; it is useless for a DSR Route_Request to be forwarded more times than the diameter (in hops) of the network. As a diameter we define the maximum non cyclic route that can be accomplished. Of course, the problem for a MANET is that it does not have a stable diameter and even if the nodes are stationary the diameter has to be collaboratively inferred by a respective protocol. In our implementation we take into account the worst case as we state in the following paragraphs.

We assume that we have a network where each node has approximately \( N_d \) neighbors (\( d \) refers to Network density). Without loss of generality we can represent this network as a tree with tree-degree \( T_d = N_d - 1 \). As someone can notice from Figure 5, the worst case scenario for a route request is to go from one leaf to the root of the tree and back to another leaf; i.e. twice the tree depth. Consequently the maximum HTL that a route request must have in order to find one node is tightly bound to the depth of the virtual tree that represents our network.

Consequently the virtual tree depth is correlated with the network size for several network degrees. As we are going to see in the simulation results the introduction of the maximum HTL in the DSR, results in radical reduction of the DSR Route_Request messages. It is important to note that, in order to estimate the current depth of the network, based on the network size and degree, averaging techniques may be applied [33]. In these techniques, each node interacts with its neighbours in order to calculate the mean value of a parameter and convergence is succeeded after a small number of iterations.

![Fig. 5 Tree-based representation](image)

**4.2 Adapted T-MAN Protocol**

Up to now we have examined the Routing layer that is the cornerstone regarding message exchanges for the upper layers. Such an upper layer is the Topology Maintenance layer. The topology formulation mechanism is responsible for identifying the relative position of a node among all the network nodes. This is essential for providing DHT functionality. As stated earlier, the challenge is to apply a topology formulation algorithm for topology creation and maintenance to a MANET environment.

One of the most prominent algorithms that have been proposed for general purpose topology formulation is T-MAN [32]. The T-MAN approach is a gossip-based approach where each node refines its view about its relative position in the overlay based on the ‘knowledge’ of the ‘closest’ nodes that exist in a buffer. Each node that participates in a network uses a ranking function to evaluate its distance from another node. The evaluation parameters can vary. Some indicative examples are the network address, proximity, latency etc. The usage of different ranking function results in the creation of different topologies.

In our implementation we are based on the creation of an overlay ring topology. In order to simplify our experiments we use as evaluation parameter the network address and as a distance function: \( d(a, b) = \min(N - |a - b|, |a - b|) \).
For example, we consider the topology where we have a network with 5 nodes and we assume that the network addresses of the participating nodes are 1, 2, 3, 4 and 5 respectively (see Figure 7(a)). For the 3rd node, nodes 2 and 4 have score 1, while nodes 1 and 5 have score 2. In order to succeed successor identification as fast as possible, T-MAN applies a gossiping technique. Specifically, each node maintains a view with the nodes that are, up to a specific time, known and scored. Periodically, each node communicates with the most ‘closer’ node, it solicits its view and requests the current view of the ‘closer’ node (see Figure 7(b)). After this mutual exchange, nodes re-evaluate their views ((see Figures 7(c) and 7(d)). This iterative procedure leads to extremely fast convergence (or stabilization), where convergence refers to the state that each node knows its successor and any further exchange of messages leads to no further refinement of the views (see Figure 7(e)).

Table 1 T-MAN parallel threads

<table>
<thead>
<tr>
<th>active thread</th>
<th>passive thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>do forever {</td>
<td>do forever {</td>
</tr>
<tr>
<td>Node_To_Sentp ← selectCloserNode()</td>
<td>buffer ← merge(view, {myNodeDescriptor})</td>
</tr>
<tr>
<td>buffer ← merge(view, {myNodeDescriptor})</td>
<td>buffer ← merge(view, {myNodeDescriptor})</td>
</tr>
<tr>
<td>send buffer to Node_To_Sentp</td>
<td>send buffer to Senderq</td>
</tr>
<tr>
<td>view ← Reevaluate(buffer)</td>
<td>view ← Reevaluate(buffer)</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

T-MAN is relied on two parallel threads as it is shown on Table 1. Since T-MAN is proposed for routed networks, the only variable parameters that exist in the above Table are the size of the view and the size of the buffer. The operation ‘send buffer to Node_To_Sentp’ of the active thread and ‘send buffer to Senderq’ of the passive thread in our layered approach is equivalent of initiating a DSR_Message_Transfer between these nodes. As mentioned above, a DSR_Message_Transfer consults the routing cache to find out if a route to the destination exists. If not, the route-request mechanism is initiated. Method ‘merge’ returns a list that contains all the elements that exist in the sub-lists that are passed as arguments. Finally, method ‘Reevaluate’ applies the scoring function to all the elements of the argument-list and returns only the first n occurrences where n is the size of the view.

Table 2 Adapted T-MAN parallel threads

<table>
<thead>
<tr>
<th>active thread</th>
<th>passive thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>do forever {</td>
<td>do forever {</td>
</tr>
<tr>
<td>buffer ← merge(view, {myNodeDescriptor})</td>
<td>buffer ← merge(view, {myNodeDescriptor})</td>
</tr>
<tr>
<td>routingachaelist← ExtractNodesFromCache()</td>
<td>DSRMessageTransferToSenderq</td>
</tr>
<tr>
<td>buffer ← merge(view, routingachaelist)</td>
<td>buffer ← merge(view, {myNodeDescriptor})</td>
</tr>
<tr>
<td>for (i=0;i&lt;gossipfactor;i++)</td>
<td>view ← Reevaluate (buf cr)</td>
</tr>
<tr>
<td>Node_To_Sentp ← selectCloserNode()</td>
<td>}</td>
</tr>
<tr>
<td>DSRMessageTransfer to Node_To_Sentp</td>
<td></td>
</tr>
<tr>
<td>receive bufferq from Node_To_Sentp</td>
<td></td>
</tr>
<tr>
<td>buffer ← merge(bufferq, view)</td>
<td></td>
</tr>
<tr>
<td>view ← Reevaluate(buffer)</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

Taking under consideration that DSR is continuously ‘working’ and contains up-to-date routing information, an adaptation (adapted T-MAN) of the initial T-MAN is proposed as it is shown in Table 2. In the adapted T-MAN approach there are the following major changes:

· Before the active thread sends its view to the closest node(s) it consults the DSR cache. Consulting means actually that all nodes that exist in the primary and secondary cache are scored (using the Ring-scoring function in our case). This ‘consultation’ results in major reduction of the T-MAN messages that have to be exchanged (before the ring is stabilized).

· Send-to-node is substituted by DSR_Message_Transfer as explained above.

· Per each ‘cycle’, instead of sending one message to the closest node that is in the node’s current view, we send multiple messages (as defined by the parameter gossip factor). Multiple messages facilitate the fast dissemination of information regarding the network topology, something that is very important in case of dynamic networks. It is obvious that the gossip factor can be between one and buffer size. Proper selection of this option may accelerate the convergence while keeping the routing overhead low.

4.3 Exhaustive tree-based approach

In order to have a comparative view of T-MAN we have also implemented an exhaustive tree-based approach (see Figure 8). According to this approach each node of the network is responsible to find its successor by using an excessive flooding mechanism. Each node in the network solicits a Find_Successor_Request message after assigning a request-id to that message (see Figure 9a and 9b). This message flows across the nodes using a Neighbor-to-Neighbor communication scheme (and not DSR node-to-node message transfer scheme as it
After sending the message a ‘served’ list is enriched in each node with the request id. In this way each node keeps track of the request-ids that have been forwarded by the node. Before the Find_Successor_Request is forwarded to the next neighbor, each node enriches its header with all the nodes that are ‘visible’ as neighbors to that node. Consequently when the next node receives the Find_Successor_Request, it has to check if all of its neighbors exist in the header of the request. If not, it enriches the header and forwards the message to the neighbors that did not exist in the header (see Figure 9c and 9d). The critical part of the process is that if one node receives a Find_Successor_Request and the request-id is not in the served list and the header contains all of its current neighbors, then this node is considered to be a ‘virtual leaf’ of a tree that has as root node the initiator of the Find_Successor_Request (see Figure 9e).

The number of Find_Successor_Responses in order to succeed stabilization of the overlay ring topology is the sum of the Find_Successor_Responses that receives each node in the tree. The root node receives $T_d^k$ responses (the number of the virtual leafs). In the first level of the tree there are $T_d$ children where each one receives $T_d^k$ responses. In the second level of the tree there are $T_d^2$ children where each one receives $T_d^k$ responses. In the (k-1) level of the tree there are $T_d^{k-1}$ children where each node receives $T_d^k$ responses. Finally, in the k level of the tree there are $T_d^k$ children where each node receives $T_d^k*(T_d^k-1)$ responses. Thus, the number of Find_Successor_Responses that are required for stabilization is:

$$\text{Routing} - \text{Cost} = \frac{(N_d-1)(N_d-1)(N_d-1)(N_d-1)}{N_d-2}$$

$$\Rightarrow \text{Routing} - \text{Cost} = \frac{(N_d-1)^{2(k+1)}-N_d+1}{N_d-2}$$ (3)

4.4 Chord p2p Protocol

In order to maintain the DHT on top of the stabilized ring topology, we have selected Chord [21]. Chord is a simple but powerful protocol, which solves the problem of efficient data storage and retrieval. It is an efficient distributed lookup system based on consistent hashing. Its only operation is to map a key to a responsible node. Each node maintains routing information about O(log N) other nodes, and lookups are feasible via O(log N) messages. Therefore, Chord scales well with a number of nodes and, thus, it can be applicable to large systems. Chord continues to function correctly even if the system undergoes major changes or if the routing information is partially correct [21]. Chord does not implement services directly but rather provides a flexible, high-performance lookup primitive, upon which such functionality may be efficiently layered. Its design philosophy is to separate the lookup problem from additional functionality. By layering additional features on top of a core lookup service, overall systems will gain robustness and scalability [22]. The DHT layer uses three types of messages; i) PUT(k,v) in order to associate the value v with the key k, ii) GET(k) in order to retrieve the value that is associated with the key k and, iii) STABILISE in order to move parts of the keys that are stored in the local cache to another node due to changes in the ring (refer also to section 3.2).

5. Experimental Evaluation

In this section we investigate the behavior of the proposed implementation, as it is described in Section 4. Each layer that was previously presented -except Neighbor-to-Neighbor layer- was implemented separately in PeerSim [31]. We assessed the performance
of each layer in terms of messages exchanged, generated errors and convergence capability in diverse network sizes. Each simulation was executed three times and average values were considered in our analysis.

5.1 Topology and Simulation Results

In the next scenarios, multiple nodes are simultaneously activated in the simulated ad hoc network. Key parameters of the topology are the number of nodes $(N)$ (size of the network) and the average number of neighbors $(d)$ (density degree). In all simulations, any operational node automatically recognizes its neighbors and joins the overlay network. Each node keeps state related to its successor in the ring topology. In case of node failures, established connections are torn down and re-stabilization procedures take place.

Initially, we compare the performance of the proposed adaptations in terms of total number of messages exchanged until the overlay ring topology is formulated. Simulations are taken for practically sized and degree (density) networks. The Hops-To-Live parameter in DSR protocol was set to infinite and optimum respectively.

Fig. 10 Adapted DSR vs DSR messages

In Figure 10, the performance of the proposed approach with the use of the adapted DSR compared to the pure DSR is depicted. It is evident that the use of pure DSR underperforms significantly compared to the adapted DSR. This is reasonable, since the exploitation of routing cache information –generated by DSR– in adapted T-MAN makes easier the identification of the successor nodes in topology formulation phase and thus requires less messages.

At this point, it is important to note that the selection of T-MAN instead of adapted T-MAN was not considered a priori as a viable solution, since T-MAN is not able to converge faster than the adapted T-MAN. This is due to the fact that T-MAN does not consult the existing routing caches of the nodes and therefore more cycles are required for overlay topology stabilization [32].

In Figure 11, we compare the performance of adapted T-MAN and exhaustive tree-based approach. It can be shown that the number of messages for topology formulation in the tree-based approach is significant larger compared to the adapted T-MAN. This is due to the fact that in the tree-based approach, exhaustive neighbor to neighbor communication takes place and also that in the adapted T-MAN approach the information that is available in the nodes’ routing caches is exploited.

Fig. 11 Adapted T-MAN vs tree-based messages

Fig. 12 Adapted T-MAN performance with HTL infinite

These simulation results led us to focus on the adapted T-MAN algorithm and further investigate the impact of other parameters, such as the routing HTL parameter or the network degree in the overlay topology formulation and maintenance process. In Figure 12 we can observe that as the number of nodes in the network increases, the number of adapted T-MAN messages also linearly increases. In addition, fewer messages are required for topology formulation in dense networks (high degree) as each node exchanges more precise gossiping messages. Precision is achieved due to the scoring of larger number of neighbors. Finally, the number of messages remains almost stable in large degree networks.
In Figure 13, the effect of the \textit{Hops-To-Live (HTL)} parameter to the total number of adapted T-MAN messages is depicted. In case the HTL parameter is set to the optimal value according to the Equation 2 in section 4.1, the number of T-MAN messages is slightly increased compared to the previous case. As expected, by limiting the HTL parameter we reduce the flooding of routing messages across the network -to the absolute minimum for addressing routing needs- and, thus, smaller routing caches in each node are exploited by adapted T-MAN stabilization phase.

Furthermore, we aim to observe the amount of DSR's route-request messages that is exchanged until the overlay ring topology is formulated. In Figure 14 and 15, the HTL parameter is set to infinite and to optimum respectively. As shown in both figures, more route-request messages are generated as the size of the network increases. This is expected as DSR messages are generated in response to T-MAN messages, which are analogous to the number of nodes (Figure 12 & 13). Furthermore, sparse networks require more routing messages than dense networks, especially as the number of nodes increase due to smaller number of entries generated via neighbor-exchanges. The adapted T-MAN affects the flooding of routing messages in the network and, thus, the routing overhead is significantly reduced by approximately an order of magnitude.

Thus, we conclude that the effect of the HTL parameter is extremely important because, when the parameter is set to its theoretical optimum value, adapted T-MAN messages were slightly increased while in parallel the routing overhead was reduced by an order of magnitude. Figure 16 depicts the number of adapted T-MAN messages for different network sizes and variable density. An important observation is that the number of messages converges quickly for different network density degrees. This observation could potentially be exploited in building clustering algorithms, where the network density could be used to estimate the best clustering size.

5.2 Indicative Service - Visualization

Upon stabilization of the overlay ring, a visualization service is provided. Each node stores topology data that is available through information acquired from neighbor discovery messages and also from information existing in the routing cache of each node. All nodes use a predefined key (e.g. “network_topology”) in order to
identify the responsible node for the storage of the visualization data. The visualization cost, i.e., the total number of messages exchanged for the visualization data to be stored, is calculated.

Figure 17 depicts the number of adapted T-MAN and DHT_PUT messages for constant and variable network density. In the first case, the network expands without any change in its density while in the later case the density is proportional increased as the network sizes increases. We notice that the number of DHT_PUT messages required for the provision of the visualization service, is slightly affected by the network size and density. It is important to note that in case that the proposed adaptations in the T-MAN and DSR algorithm were not applied, the visualization cost would be significantly higher, as already shown in Figure 10.

In the last set of simulations, we investigated whether the visualization service is robust to multiple node failures. Under the unfavorable condition that 20% - 30% of network nodes become simultaneously non-operational and for various network sizes, we measure the number of DHT put(key, value) calls that generate DSR Route Error messages for any broken link identified. Such messages are flooded across the network in order to update the routing cache for future requests (see Section 4.1). As shown in Figure 18, the majority of the put(key, value) requests are successfully completed without any delay (error). For example, for a 40-node network and 30% node failure, 17 messages were successful while another 10 failed. Even if a put(key, value) request initially fails due to routing inconsistencies, the requests will be eventually be successful after the routing tables are updated. Therefore, there is only a time penalty for the service delivery after a major network failure. It could be argued, therefore, that the visualization service remains operational even when a significant portion of nodes (and attached links) fails. Similar trends should be noticed for other distributed services, which have to be verified with further simulation experiments.

6. Conclusions and Future Work

In this paper we presented a new paradigm for service development and operation in mobile ad hoc networks. We initially proposed a four-layer approach for setting up an overlay network, over which services are deployed. Each layer may be independently realized with existing peer-to-peer and routing protocols. Topology formulation, maintenance and routing challenges in MANETs are addressed by suggested adaptation of existing algorithms, such as T-MAN and DSR. It could be argued that the overlay network hides from the service layer any physical network topology changes and, thus, service development is significantly simplified.

Using extended simulations, we showed that the adapted algorithms quickly converge while the number of messages is kept low. In a reference implementation, we showed that a distributed network service could efficiently run over a dynamic network. We argue that other services may also be implemented based on our approach. In our future work, these initial conclusions will guide the development of a more detailed prototype, possible taking in to consideration functionality of IPv6 protocols.

Acknowledgments

This publication is based on work partially performed in the framework of the European Commission ICT/FP7 project EFIPSANS (www.efipsans.org).

References


Athanassios Liakopoulos received the Dipl.-Ing. degree in Electrical and Computer Engineering from the NTUA, Greece in 1996, MSc with Distinction in Telematics from the Electrical Engineering Department in University of Surrey (UniS) in 1998 and PhD in Electrical & Computer Engineering from the NTUA in 2005. Since September 2000, he joined the GRNET S.A. where he is Coordinator of Networking and Computing Infrastructure.

Dr. Gregoris Mentzas is Professor of Management Information Systems at the School of Electrical and Computer Engineering of the NTUA and Director of the Information Management Unit (IMU), a multidisciplinary research unit at the University. His area of expertise is information technology management and his research concerns the integration of knowledge management, semantic web and e-service technologies.

Prof Nikolas Mitrou's research interests are in the areas of digital communication systems & networks and networked multimedia in all range of studies: design, implementation, modelling, performance evaluation and optimization. Since 1988 Prof. Mitrou has been actively involved in many RACE, ACTS and ESPRIT projects and he was the coordinator of one of them (AC235, WATT). He is a member of the IEEE, member of the IFIP WG 6.3 and member of the Technical Chamber of Greece.