Programmable Middleware for the Dynamic Deployment of Services and Protocols in Ad Hoc Networks

S.Gouveris, S.Sivavakeesar, G.Pavlou, A.Malatras
Centre for Communication Systems Research, Dept. of Electronic Engineering, University of Surrey, Guildford, Surrey GU2 7XH, UK
{S.Gouveris, S.Sivavakeesar,G.Pavlou, A.Malatras}@eim.surrey.ac.uk

Abstract

Mobile ad hoc networks (MANETs) are characterized by their heterogeneity and the diverse capabilities of the nodes forming the network. Almost any device equipped with a wireless network interface can join an ad hoc network. In such an environment it is difficult to deploy common services without a common understanding among the participating nodes and their capabilities, in terms of processing power, battery life, expected residence time and also in terms of already installed and operational software. This paper presents a middleware-based programmable infrastructure that allows the nodes of a mobile ad hoc network to download and activate required protocol and service software dynamically. This enables the alignment of the nodes' capabilities so that common services and protocols are used among heterogeneous ad hoc network nodes. We first present the middleware functionality and architecture and then evaluate it through both testbed experimentation and simulation. Our evaluation shows that the proposed approach and architecture perform relatively well, with overall convergence time growing linearly with the number of nodes and without any adverse effects because of increasing node density.

Keywords

Programmability, Middleware, Ad hoc Network Management

1. Introduction

The concept of mobile ad hoc networks (MANETs) has recently received significant attention due to the increasing popularity of tetherless computing and the rapid growth of wireless networking. In ad hoc networks, the mobile nodes (MNs) are free to move randomly and organise themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Typically this kind of network operates in a standalone fashion or may be connected to an infrastructure-based network through a gateway, e.g. wireless LAN access point, base station, etc. Conventional wireless networks require as prerequisite some form of fixed network

0-7803-9087-3/05/$20.00 ©2005 IEEE

infrastructure and centralised administration for their operation. In contrast, since MANETs are self-creating, self-organising and self-administrating, individual MNs of the network are responsible for dynamically discovering other nodes they can communicate with. This way of dynamically creating a network often requires the ability to rapidly create, deploy and manage services and protocols in response to user demands.

There has been no proper previous research on deploying application-level services or routing protocols dynamically in MANETs, but such aspect is important in ad hoc networks. For example, while routing and Quality of Service (QoS) conform to standardised frameworks and protocols in fixed IP networks, there are many potential solutions for ad hoc networks that depend also on the characteristics of the particular ad hoc network, e.g. topology volatility, characteristics of radio links, capabilities of nodes, etc. Given the multitude of potential solutions that may be environment-dependent, programmability is of paramount importance to allow mobile nodes to be enhanced on the fly with the required capabilities in the ad hoc environment. In addition, application servers may migrate to more powerful devices that have the required capabilities while less powerful devices may outsource computing tasks (cyber foraging). Programmability is possible through recent advances in distributed systems technologies and transportable "execute-anywhere" software. In this paper we present a novel programmable middleware that is capable to support cooperation, adaptability and alignment with respect to the required basic capabilities and additional services of the devices that form an ad hoc network, allowing a degree of self-management to be achieved.

A key aspect of MANETs, given their fluidity, is the possibility to adapt to their environment through context-awareness for many co-operation and co-ordination scenarios. The sheer amount of context information necessary for relevant adaptation places an important burden on the network, as potentially large amounts of data from diverse sources need be managed. This requires an infrastructure for sensing, collecting, and providing context information to applications [14][15]. The proposed programmable middleware takes this into consideration and takes an elementary step to provide such an infrastructure. In this paper though we mostly concentrate on the programmable middleware aspects that support self-management.

The rest of the paper is organised as follows. Section 2 provides basic background, on network programmability. The proposed middleware functionality and architecture are described in section 3. Section 4 presents the evaluation of the proposed approach, first through testbed experimentation and then through simulation. Finally, section 5 presents our conclusions and future work.

2. Background on Network Programmability

As already mentioned, programmability in ad hoc networks is of paramount importance given the multitude of potential solutions for routing, QoS support and other application services. There are various different approaches to achieve programmability. Active control packets may carry code to be evaluated in routers [1]; this approach has also been used for active routing in ad hoc networks [5].

Mobile agents may be used in full mobility scenarios, carrying code and state to manipulate different network nodes, or in a constrained mobility mode [6] as a more flexible means for the management by delegation approach [7]; in the latter, code is uploaded and executed in network nodes through "elastic management agents", augmenting the node functionality. Programmability is also possible through the provision of suitable management interfaces that allow code to be uploaded to network nodes and activated in a controllable fashion. This approach has been first adopted in the Xbind framework, targeting the quick and flexible introduction of new telecommunication services in programmable network infrastructures [8]. It has given rise to the IEEE P1520 initiative for Programmable Network Interfaces [9]. The Mobiware approach has relied on the Xbind approach, modifying and extending it for cellular networks [10]. Other related approaches have been the Tempest framework, targeting the creation of virtual partitions in ATM networks [11] and the Phoenix framework, targeting new network services on reprogrammable router processors [12]. It should be finally mentioned that while there exists research work on network programmability, there has been no attempt to apply it to ad hoc networks in the manner proposed here.

3. Programmable Ad Hoc Middleware Functionality and Architecture

Our programmable platform follows a lightweight approach to achieve programmability through the use of loadable plugins. The latter are blocks of code that can be uploaded and executed on the ad hoc network nodes i.e. terminodes, in order to perform specific operations. This can be an installation of a new routing protocol, extensions to an existing one or any other function that the MANET could benefit from. In order to decide on a particular plugin to be used for a given scenario, a plugin election should take place utilising the current contextual information. This involves advertisements from every MN that can provide suitable plugins for a given election. A predefined election algorithm should be used and identify a plugin to install globally. The latter should then be distributed throughout the MANET in a peer-to-peer fashion. Following the installation of the elected plugin, the mobile nodes should activate, i.e. execute, the plugin. In the following sub-sections we describe the functionality and architecture of the middleware platform, highlighting important design decisions and presenting the relevant reasoning.

3.1 Centralized vs. Distributed Management

A key consideration in an ad hoc network is the management approach to be deployed. In a centralized approach, the whole MANET is grouped into clusters, each electing a local leader or cluster head (CH). CHs then cooperate and elect a global leader or network head (NH). Key management decisions, such as triggering and coordinating the plugin election process, are taken by the NH. This hierarchical approach is similar to that of routing protocols, e.g. OSPF, and scales well, limiting interactions within a cluster or among cluster heads. It also allows operation in a controlled distributed fashion, when decisions are taken not only by the NH but through cooperation and

"voting" among the CHs. A diametrically different approach is a fully distributed one, in which all the terminodes have "equal rights" and determine collectively any management decisions to be taken. This approach requires more complex cooperation protocols and may not scale for large networks with 100's of nodes.

For our intended use of aligning dynamically the capabilities of terminodes through programmability, we have opted for the centralised approach, employing CHs and a NH. This approach is chosen due to its inherent desirable features outlined below: i) the leader or NH can impose a uniform management approach over the ad hoc network formed, ii) it is easy for a leader, at any point in time, to make a decision as to the selection of the most optimal protocol to be deployed relatively quickly; this holds both at bootstrap time or when the already deployed protocol/service becomes inappropriate for the current context, and iii) this clustering approach is more scalable for a larger network. The decentralised approach has the key drawback that the voting mechanism can be time-consuming. This is especially lengthy when in our scenario the already deployed protocol becomes inappropriate for the current context. The difficulty arises because there is no central node (NH) or few nodes (CHs) responsible for triggering the re-deployment process. More importantly, the decentralised approach would lead to a dangerous situation where any node may trigger this process unnecessarily.

The CHs/NH election process is based on contextual information and can take into account location, capabilities such as processing power, memory, battery life, etc., expected residence time and possibly owner's privileges. For example, the CH/NH needs to be a relatively central node in the cluster or network respectively while there is no point in having a cluster head with high probability to move radically away from its current location soon. There is a lot of work in the literature on cluster formation and cluster head election, we have also proposed relevant algorithms. Stable cluster formation is important, and hence the CH election heuristic can be similar to the one we proposed in [1] for a longer-term large-scale MANETs or similar to the one we proposed in [2] for more spontaneous MANETs. In [1] and [2], we compared our cluster head election heuristic with other similar approaches, and showed that our heuristics result in stable cluster formation in comparison with others. It should also be noted that a deputy cluster head or network head is also elected, so that there is immediate "functionality fallback" should a CH/NH leave.

3.2 Middleware Communication and Components

We have chosen to use the lightweight XML-RPC [16] protocol as the basis of communication between terminodes running the programmable platform. XML-RPC can be considered as a subset of the SOAP protocol, unburdened from unnecessary complexity. Like all remote call approaches, it allows software running on different operating systems and hardware architectures to communicate through remote procedure calls (RPCs). XML-RPC uses the HTTP protocol as transport and XML encodings for the RPC protocol itself. We chose an XML-based approach because we also use XML to represent contextual data in terminodes and this achieves easy integration. We could have possibly chosen Web Services, but this approach would have certainly been more heavyweight. In addition, Web Services, in the same fashion

with distributed object technologies such as CORBA, necessitate object advertisement and discovery functionality, which is not required in our platform that relies on simple message passing modeled through RPCs. Given our recent performance evaluation of XML and other management approaches [3], we believe that XML-RPC provides a useful blend of functionality and performance.

In addition, in order to make the platform even less demanding on processing power and portable to mid-range and small devices, we used the Java 2 Micro Edition (J2ME) virtual machine. The latter requires a much smaller memory footprint than the standard or enterprise edition, but at the same time it is optimized for the processing power and I/O capabilities of specific categories of devices. We used the Connected Device Configuration (CDC) framework instead of the limited (CLDC) one, as the latter lacks the required support of advanced operations.

The programmable platform can be divided into two middleware modules according to relevant functionality, as shown in Fig. 1: the Cluster (or Network) Head and the Terminode modules. Depending on a node's current status, middleware functionality switches between Cluster/Network Head and Terminode mode respectively. For the time being, we have implemented and validated Cluster Head but not Network Head functionality, since our experiments were conducted in a small testbed with 4 terminodes – we plan to implement cluster head cooperation and network head functionality in the future.

The Cluster Head components deal with the management of the programmable Terminodes regarding programmable plugin election and distribution, as well as (re) configuration. The key components of the Cluster Head module are:

- Plugin Election and Distribution: it is responsible for the initiation and coordination of the election process. It implements the plugin election algorithm and is responsible for the initiation, distribution and activation of the elected plugins across the ad hoc network.
- Plugin Configuration: it is responsible for the (re-)configuration of installed plugins across the ad hoc network. It is also able to modify on-the-fly key parameters of active plugins, if the latter support this functionality.

The Terminode components deal with the management of the programmable plugins regarding plugin advertisement, installation, storage, distribution and configuration. The key components of the Terminode module are:

- Plugin Management: is responsible for advertising available plugins to the Plugin Election component, listening for requests from the latter for distributing a specific plugin, communicating with its peer nodes for the purpose of exchanging network plugins, as well as for installing, executing, reconfiguring and terminating the operation of a particular plugin.
- Plugin Repository (PR): is responsible for storing and exposing available plugins to the Plugin Management component when required. The latter should be able to extract and advertise to the Plugin Election component the characteristics of the stored plugins for the purpose of plugin election. The Plugin Management component offers all the necessary operations for storing and deleting plugins, as well as searching for a plugin by name or type.

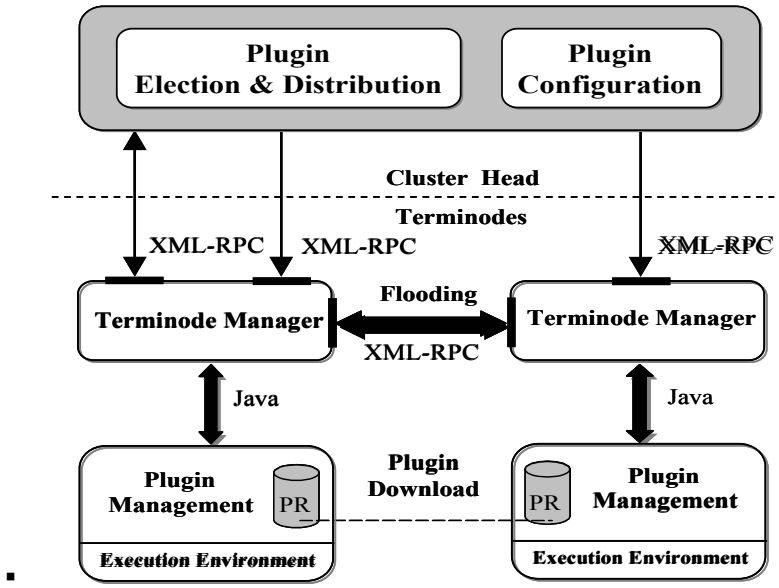

Figure 1: Middleware Platform Architecture

3.3 System Operation

We describe here the complete system operation, from election triggering to plugin activation, considering an ad hoc network that consists of a single cluster. According to the current context or human user command, the CH triggers the election process by contacting Plugin Election and providing the type of the plugin currently required by the MANET. The Plugin Election object contacts all the terminodes of that cluster in order to request advertisements of candidate plugins. Each member node performs a lookup in its plugin repository for one or more suitable plugins that can satisfy the requirements of the election process and replies to the CH Plugin Election with the characteristics of the retrieved plugins. The CH Plugin Election executes the election algorithm and decides on the most suitable plugin, based on the current context, e.g. by assigning different weights to criteria such as CPU time, memory required, etc. Following the actual plugin election, the CH contacts terminodes that already possess the elected plugin and instructs them to distribute it across the cluster. Plugin distribution is carried out in a peer-to-peer fashion, with the "owning" nodes flooding the plugin to their peers, and so on. Prior to any plugin exchange, each node probes its peers in case they already have acquired the plugin, so as to avoid unnecessary retransmissions. For the actual transmission, the terminode currently distributing the plugin contacts the neighbor MN and passes all the characteristics of the elected plugin, followed by the actual transfer of the plugin's execution code. When a new plugin has been successfully installed, the node sends a notification to the Plugin Election of the CH. At this point, the plugin is installed and available to be activated. The CH Plugin Election object, after receiving installation notifications from all the

member nodes or after a predefined timeout period, it floods an activation message across the cluster to instruct the member nodes to execute the elected plugin. Each member MN should then perform a lookup in its Repository for the plugin, in order to obtain its reference in the local file system and consequently execute it in user space.

3.4 Loadable Plugins

A programmable plugin is a dynamically loadable object or module that can be installed, removed, activated and configured on-the-fly in order to extend the functionality of a node. A loadable plugin is a Java object that implements a generic interface with methods supporting common necessary functionality. Such programmable plugins can be loaded and activated dynamically into the operating system's user space at run-time. They can be instantiated as often as required, while it is possible to have several instances of the same plugin with different configurations. We should note here that the plugins execute only in user space, so a new or extended ad hoc routing protocol should operate in user space. This implies a performance limitation for plugins that implement network device functionality but it is too difficult and dangerous to achieve programmability at kernel level. The key reason behind the selection of Java for implementing the programmable plugins is platform independence. This is required in a diverse ad hoc environment, as is also the case in our ad hoc network testbed, which consists of laptops and a PDA with different computing architectures. It would have been possible to cater for plugins in compiled languages, e.g. C/C++, but this would complicate the system and would require many versions available for each plugin, one for each node operating system / hardware architecture combination present.

Plugins expose two programmable interfaces for the purpose of configuration and monitoring. This first interface is used to configure and alter various aspects of the plugin functionality at run-time. On the other hand, through the monitoring interface plugins can provide some information regarding their status and possibly various useful statistics collected from their operation. For each programmable plugin there are several characteristics to be considered, such as the CPU cycles required for its operation, the run-time memory size required, and the plugin physical size which governs the distribution overhead. Each characteristic is assigned a unique and ordered identifier in order to be comparable with other candidate plugins during the election process. Due to the dynamic nature of plugins' installation and reconfiguration, a security mechanism for authorization and certification would be required in order to ensure stability in an untrusted environment – this aspect is however outside the scope of our current work.

3.5 Plugin Election and Distribution

As already explained, an election procedure takes place among the member nodes. The coordinator of the election phase is the cluster head, whose functionality switches to Cluster Head mode. At this point, each terminode, whose functionality switches to Terminode, collects the characteristics of the plugins that fit the current requirement as advertised by the CH and sends them to the Cluster Head in a unicast fashion. The CH decides on the most suitable plugin for the current context by executing the

election algorithm. The plugin election is based on a deterministic selection process, where each characteristic of a plugin has a comparable identifier. The election mechanism uses a simple cost algorithm based on weights assigned to selection criteria. Each criterion is assigned a mathematical weight based on its importance to the election process, while each plugin characteristic has a unique identifier, ordered from 0 to 10. The election algorithm used is given by equation (1). It is possible to adjust the criteria weights, based on the current context and on constraints the nodes might impose. The plugins characteristics that are considered during the election process are the CPU utilization, memory size required and the plugin size.

$$f(x) = \sum_{i=1}^{n} w_i \times A_i(x), \text{ x is the specific plugin} \quad (1)$$

where:, $w_i \in [0,1]$, $\sum w_i = 1$ and $A_i \in [0,10], \forall i \in [1,n]$

Following the plugin election process, it is required to distribute the elected plugin by requesting the nodes that possess it to distribute it to their immediate neighbors that don't have it. This technique minimizes the network traffic and convergence time to achieve complete flooding of a particular plugin. The actual plugin transfer is achieved using the Trivial FTP (TFTP) protocol. Compared to FTP, the former is less complex and demanding on network resources. TFTP has no user authentication, which spares time and traffic in a trusted environment, but most importantly it uses only one connection contrary to FTP that required two connections, one for control and one for data traffic.

A key aspect of plugin flooding is that a terminode should not receive the same plugin twice. This is deliberately prevented as the plugin size might be considerable and prove costly to the network. In order to prevent this from happening, the transmitting node will have to first probe its peer if it has already acquired the plugin and then only transmit it. The plugin distribution includes apart from the actual plugin transfer, the transmission of the plugin characteristics.

4. Platform Evaluation

4.1 Testbed Experimentation

For the purpose of evaluating the programmable platform in a real ad hoc network, we deployed the former in our ad hoc testbed. This consists of three laptops and one personal digital assistant, running the Linux Debian operating system. The ad hoc testbed supports the 802.11b wireless standard for all the required communication between nodes. Packet routing is achieved using the AODV-UU, user space routing daemon by Uppsala University, which implements the Ad hoc On-Demand Distance Vector (AODV) routing protocol [13]. In order to create custom network topologies without having to place the nodes far from each other, we used a MAC address based filtering tool to simulate an "out of reach" situation. In this case, nodes discard incoming packets from predefined source MAC addresses, as for example AODV

"hello" messages from specific nodes [13]. Several network topologies were used and test cases were carried out in order to validate and evaluate the ad hoc programmable platform. The scenarios implemented were different static network topologies and a case where a link between two nodes breaks and AODV has to construct an alternative route on-the-fly – this verified that the ad hoc routing protocol worked as expected.

The initial measurement taken was the response time and traffic generated by a single XML-RPC method call. The selected method was the "advertisePlugin", which is invoked by the Cluster Head to the Terminodes for requesting advertisements for candidate plugins. The selected method call carries as argument the "advertisePlugin" string and has no return type. The traffic generated for that method was measured to be 1211 bytes and the response time was 6.5 milliseconds when the call was made between two laptops and 23 ms when it was made between a laptop and the PDA. As expected, the PDA exhibits a much slower processing time and therefore the parsing of XML messages needs considerable more time than on the laptops. The latter appears to be around 3.5 times faster.

The next set of measurements that were recorded, involved the complete system run for different topologies of the ad hoc network. For the first test case, the topology shown in Fig. 2 was realized. The CH in this case is manually selected to be node "ares", as it is powerful and is a single hop away from every other node. During the plugin advertisement procedure, all four nodes will have a number of plugins to advertise but the plugin with the highest identifier is deliberately selected to be the one that belongs to node "zeus". The elected plugin is a simple echo server and its size is 1450 bytes. For a complete system execution as described in 3.3, the measurements taken were the overall time and the traffic generated by AODV, the interaction between Terminodes and the Cluster head modules and the transfer of the elected plugin from the owner to all other member nodes within the cluster. The time taken for a complete system operation within the considered cluster, from the request for plugin election to the activation of the elected plugin, was measured to be 2.56 secs over a number of samples. If "ares" had the plugin, which is a more central node, the convergence time is 2.40 secs. During this period, the overall AODV traffic observed across the whole network was 528 bytes, which mainly includes "hello" messages. The traffic when the Cluster Head module is involved includes the request for plugin election (1084 bytes), which initiates the system, the advertisements of candidate plugins by the Terminodes module (3475 bytes), the request to the elected plugin owner to distribute the plugin (1095 bytes) and the plugin installation confirmation by the Terminode modules (2096 bytes). Overall, the Cluster Head related traffic was recorded to be 7750 bytes. On the other hand, there is the Terminode related traffic, which includes the communication between member nodes only. This was recorded to be 15114 bytes and includes the flooding messages for requesting the plugin advertisement (3633 bytes), the method for querying an adjacent Terminode module if it has the plugin under distribution (4392 bytes), the distribution of the elected plugin and its characteristics (3429 bytes) and the flooding messages for requesting the plugin activation (3660 bytes). Finally, there is the plugin distribution traffic, caused by the TFTP file transfers. In the specific test case, three

plugin transfers are required and the overall traffic generated was measured to be 5253 bytes.

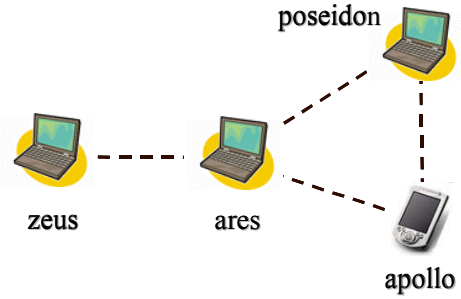

Figure 2: Test Case 1 TestBed configuration

For the second test case, the requirement was to examine the ability of the programmable platform to follow the sudden changes in the topology of the ad hoc network. The original topology of this scenario was the same as the previous case, but with the difference this time that during the plugin election process, the link between nodes "ares" and "Poseidon" was deliberately broken to form a "chain-like" topology. That would mean that all packets from "ares" to "poseidon" and vice versa would have to be routed via "apollo". It was found that this function is well supported by the AODV-UU daemon and was indeed effortless. The same holds for the programmable platform, which successfully completed its operation regardless of the sudden topology change. The time taken for a complete system run in this case is 2.71 sec, which is 150 ms longer than the first case. This was expected as AODV had to discover a new route and any packets going from "Poseidon" to the CH and vice versa would have to do an extra hop. Due to the fact that the AODV daemon had to construct a new route, the AODV traffic was also increased to 576 bytes. On the other hand, the Cluster Head related traffic remained the same (7750 bytes), as it is mainly dependent on the number of nodes in the network and the ones owning plugins. The Terminode related traffic however, which is heavily depended on the network topology and the number of neighbors of every node, was reduced to 14016 bytes. This is because the traffic caused by the message flooding in this chain-like topology, had one strict route. Finally, the plugin distribution traffic remained the same (5253 bytes), as again there is one plugin owner (zeus).

Key results to note from the experiments are the following. First, the overall amount of control traffic required is in the order of few Kbs per node and it is actually more for the terminodes than for the cluster head. In the next section we validate through simulation that the control traffic per node does not increase with the number of nodes, which guarantees scalability. In addition, the transmission traffic required for a 1.45 Kb plugin is 3.43 Kb because its characteristics are also flooded to the adjacent node through an RPC. The plugin size has obviously no impact on the control traffic, but for a 64 Kb plugin, which is a typical size of applications in Java Microedition, the transmission traffic becomes 66 Kb.

Another important result is that for a network of 4 nodes, the overall convergence

time is 2.56 secs with two hops and 2.71 secs with three hops of uploading a 1.45 Kb plugin. Increasing the plugin size to 64 Kb brings those times close to 3.5 and 4 secs respectively, mainly because of the additional transmission time. This means that the average latency per node is approximately 1 sec for a 64 Kb plugin in a 4 node network. In the next section we validate through simulation that this value increases almost linearly with the number of nodes, which again guarantees controlled convergence times. For example, for a network of 20 nodes the average latency is 2 secs per node, which means 40 secs overall convergence time (Fig. 4).

4.2 Simulation

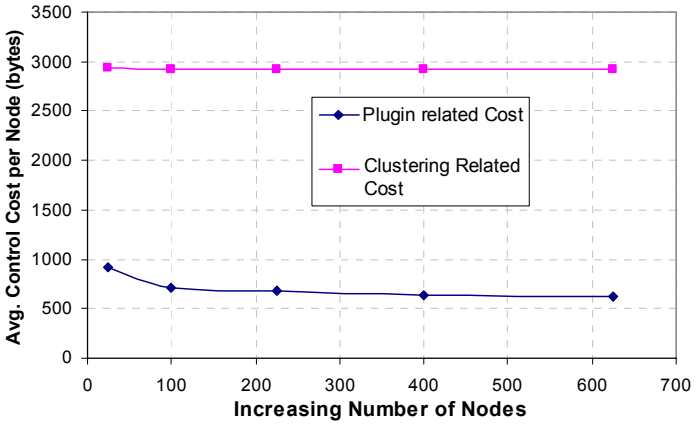

Figure 3: Average control traffic incurred per node as part of clustering and plugin deployment processes as a function of increasing node count.

While testbed experimentation gave us an initial idea about the relative performance, traffic overhead and convergence time of our system, as well as values for key elemental interactions, simulation allows us to experiment with much larger node populations and topologies and assess in much more detail the performance of our approach. The simulation works attempts to investigate the performance of our plugin election process in terms of the average control traffic incurred per node and the overall convergence time involved for the complete plugin deployment process. We performed our simulations using the GloMoSim simulation package in which we implemented the associativity-based CH election heuristic of [2] – note that in the testbed experiments we hard-assigned the CH role - and the already described plugin election process. The transmission range of each node is set to 100 m, and the link capacity takes a value of 2 Mbps (worst-case scenario). The simulations were performed for a stationary MANET and the simulation parameters are similar to those of [2], with key values used as measured in the testbed experiments. AODV was used in the simulations. Due to space limitations, we showed only the assessment of the scalability of our plugin election scheme in terms of increasing node-count only in this paper.

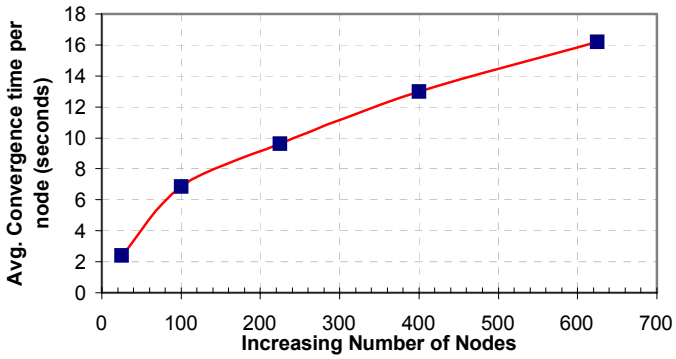

Figure 4: Average convergence time involved for the plugin deployment process as a function of increasing node count.

In order to assess the effect of increasing network size on the clustering and plugin election schemes, the terrain-area is also increased with an increase in the number of nodes, so that the average node-density is kept constant in the first set of simulations. The number of nodes in this case is varied from 25, 100, 225, 400 and 625. The terrain-area size is varied so that the average node degree remains the same and accordingly 200X200 m^2, 400X400 m^2, 600X600 m^2, 800X800 m^2 and 1000X1000 m^2 are selected for each scenario. Fig.3 shows the average control traffic incurred per node during the clustering as well as plugin election processes as a function of increasing number of nodes. The control traffic of CH election process is actually the traffic involved due to HELLO packet transmissions, and the control traffic of our plugin election process is the total traffic involved for the whole plugin deployment process. As it can be inferred from Fig. 3, the average control traffic per node does not depend on the increasing node count, and hence both the clustering and plugin election schemes are scalable. Fig. 4 depicts the average convergence time involved per node for a complete plugin deployment process as a function of increasing nodes. The convergence time involved is actually the time the plugin process takes from the point when a terminode receives the plugin election trigger message from the CH until it finally receives the plugin activation message. As it can be seen from Fig. 4, the convergence time increases almost linearly with the node count. This is somehow expected as we assume that the number of plugin "owner" nodes also increases proportionally with network size. Although these nodes are randomly distributed, they appear to be reasonably well distributed as the network grows in size, avoiding "empty" areas, hence the almost linear convergence time increase.

5. Conclusions and Future Work

In this paper we presented a programmable middleware platform that can align the capabilities of the nodes of an ad hoc network through the use of loadable plugins. This is crucial in a heterogeneous environment if a common communication infrastructure is to be deployed that could achieve, for instance, quality of service

based communication across the ad hoc network. For the platform communication we used the lightweight XML-RPC message-oriented protocol, where relevant management information is encoded in XML and transferred over HTTP. The platform was implemented in the Java 2 Microedition programming language, which can cater for small to medium devices, such as PDAs.

Our initial performance evaluation seems encouraging, with a few seconds required for convergence in a small network and linear increase with node count. In addition, high node density does not seem to have adverse effects until a threshold, above which it results in increased 802.11 collisions and performance deterioration, which is expected. Given the fact that we have adopted Java-based plugins for platform independence and XML-RPC for communication due to the easy integration with XML-formatted data, the overall performance seems encouraging.

We have adopted a centralized management approach, with cluster heads administering geographical clusters and one of them nominated as network head, administering the whole network. The approach could be centralized, with the network head taking all decisions, or partly distributed, with management decisions reached through collaboration among cluster heads. For the time being, the platform focuses in a single cluster only, with the cluster head being also the network head. Cluster-to-cluster interaction will be investigated in the future.

Furthermore, given the peer to peer nature of the platform, it is valid to state that the whole ad hoc network could be in risk if a loadable plugin was an engineered computer virus. In this case, a secure mechanism for verifying the advertised plugins would be a requirement in untrusted networks. Last but not least, the plugin election and distribution process could benefit from additional contextual information, gathered from the terminodes and their behavior in the mobile ad hoc network. We plan to work towards this direction in the future.

ACKNOWLEDGMENT

The work presented in this paper was carried out in the context of the Programmable Ad hoc Networks (PAN) EPSRC project – GR/S02129/01.

References

[1] S. Sivavakeesar, and G. Pavlou, Stable clustering through mobility prediction for large-scale multihop intelligent ad hoc networks, Proc. of the Wireless Communications and Networking Conference (WCNC 2004), vol. 3, Mar. 2004, pp. 1488 – 1493.

[2] S. Sivavakeesar, and G. Pavlou, Associativity-Based Stable Cluster Formation in Mobile Ad Hoc Networks, to appear in the proc. of the IEEE Consumer Communications & Networking Conference (CCNC' 2005), Nevada, USA, Jan. 2005.

[3] G. Pavlou, P. Flegkas, S. Gouveris, A. Liotta, On Management Technologies and the Potential of Web Services, IEEE Communications, special issue on XML-based Management of Networks and Services, Vol. 42, No. 7, pp. 58-66, IEEE, July 2004.

[4] D. Tennenhouse, J. Smith, D. Sincoskie, D. Wetherall, G. Minden, A Survey of Active Network Research, IEEE Communications, Vol. 35, No. 1, pp. 80-86, January 1997.
[5] C. Tschudin, H. Lundgren, H. Gulbrandsen, Active Routing for Ad hoc Networks, IEEE Communications, Vol. 38, No. 4, April 2000.
[6] C. Bohoris, A. Liotta, G. Pavlou, Evaluation of Constrained Mobility for Programmability in Network Management, Proc. of the 11th IEEE/IFIP Int. Workshop on Distributed Systems: Operations and Management (DSOM'00), Austin, Texas, USA, A. Ambler, pp. 243-257, Springer, December 2000.
[7] G. Goldszmidt, Y. Yemini, Evaluating Management Decisions via Delegation, Proc. of IEEE Integrated Network Management III, pp. 247-257, Elsevier, 1993.
[8] A. Lazar, Programming Telecommunication Networks, IEEE Network, Vol. 11, No. 5, pp. 8-18, Sep.-Oct. 1997.
[9] J. Biswas, A. Lazar, J.-F. Huard, K. Lim, S. Mahjoub, L.-F. Pau, M. Suzuki, S. Torstensson, W. Wang, S. Weinstein, The IEEE P1520 Standards Initiative for Programmable Network Interfaces, IEEE Communications, Vol. 36, No. 10, October 1998.
[10] O. Angin, A.T. Campbell, M. Kounavis, R. Liao, The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking, IEEE Personal Communications, Vol. 5, No. 4, August 1998.
[11] J.E. van der Merwe, S. Rooney, I. Leslie, S. Crosby, The Tempest: A Practical Framework for Network Programmability, IEEE Network, Vol. 12, No. 3, May-June 1998.
[12] D. Putzolu, S. Bakshi, S. Yadav, R. Yavatkar, The Phoenix Framework: A Practical Architecture for Programmable Networks, IEEE Communications, Vol. 38, No. 3, March 2000.
[13] C. E. Perkins, E. M. Belding-Royer, and S.R. Das, Ad hoc On-Demand Distance Vector (AODV) Routing, draft-ietf-manet-aodv-13.txt.
[14] D.Mandato, E.Kovacs, F.Hohl, and H.A-Alikhani, CAMP: A context-aware mobile portal, IEEE Communications Magazine, no. 1, Jan. 2002, pp. 90-97.
[15] B.N.Schilit, D.M.Hilbert, and J.Trevor, "Context-aware communication", IEEE Wireless Communications, no. 5, Oct. 2002, pp. 46-54.
[16] XML-RPC specifications web site, http://www.xmlrpc.com/spec.