A Rule-based Distributed System for 
Self-optimization of Constrained Devices

Javier Baliosian, Jorge Visca, Eduardo Grampín, Leonardo Vidal, Martín Giachino
School of Engineering, University of the Republic, Montevideo, Uruguay.
Email: {javierba, jvisca, grampin, lvidal, giachino}@fing.edu.uy

Abstract—During the last years there has been a strong research effort on the autonomic communications and self-management paradigms. Following this impulse, the academic community and the industry have proposed several architectures and techniques to allow network devices to make their own configuration decisions. Those proposals often include resource-expensive technologies such as complex inference machines, ontological modeling and probabilistic prediction that may not be suitable for the most pervasive and inexpensive network-enabled devices. This paper addresses such facet of the autonomic systems introducing RAN. This system aims to be a complete rule-based, distributed system specially designed and implemented to enable autonomic behavior on very constrained devices, such as domestic wireless routers with resources as low as 16 MB of RAM and 4 MB of storage memory. The RAN system was developed to serve the objectives of Rural Ambient Networks, a project that targets the so called Digital Divide deploying low-cost wireless mesh infrastructure in rural communities. In this context, RAN, in autonomic and distributed manners, optimizes the network configuration to minimize the monetary cost that the community has to pay for using the IT infrastructure. Finally, this work presents an evaluation of RAN that shows how it makes possible to perform sophisticated optimization decisions with a very small overhead in terms of CPU and memory.

I. INTRODUCTION

The industry and the academic community have been making strong research efforts on the autonomic communications and self-management paradigms and they have produced several architectures and techniques aimed at allowing network devices to make their own configuration decisions.

Those proposals often include technologies such as generic inference machines, ontological modeling and probabilistic prediction that, with some exceptions such as [1], have been developed for resource-generous devices. However, the common assumption that, for example, memory or storage space are virtually free and therefore disregarded, may not correspond with the reality of some commodity devices such as domestic wireless routers or with the economy of some communities. In one hand, driven by an intense competition, the cost of domestic wireless routers has dropped to prices that are affordable in developing regions making of them an excellent opportunity to deploy low-cost communications infrastructures for communities suffering of the so called Digital Divide. In the other hand, the hardware resources of those devices have also been trimmed down in order to be economically viable for the vendors. This situation imposes severe constraints to the software that can be run in those inexpensive devices and therefore, specific techniques for embedded systems must be used.

The Rural Ambient Networks project (RAN) [2] was designed to improve connectivity of remote rural communities in the less expensive possible manner. The RAN project deployed a mobile computing infrastructure based on low-cost wireless routers that supports the production of community-based cooperatives. It can also provide voice and text communications between remote sites, and it may promote cultural initiatives like local radio or news services. These user applications are based in a network infrastructure founded over the Ambient Networks (AN) [3] concepts, taking into account the geographical characteristics of rural areas, the economic situation of its members and the cultural profile of rural communities. The basic supporting technologies for the RAN environment are Mesh Networking, introduced by the IETF Mobile Ad-hoc Networking (MANET) Working Group V, the IEEE 802.11 set of wireless LAN standards (usually known as Wi-Fi), GSM/GPRS and other evolving mobile communications standards such as 3G.

To serve the RAN project, we developed a self-managing system that is what will be the focus of this paper and that is named RAN after the project. The RAN system aims to be a complete rule-based, distributed system specially designed and implemented, using embedded-systems techniques, to enable autonomic behavior on very constrained devices, such as the domestic wireless routers mentioned before, with resources as low as 16 Mb of memory. In this context, the task of RAN is, in autonomic and distributed manners, to optimize the network configuration in order to minimize the cost for the community.
of using Internet services.

The structure of the paper is as follows. The overall design of the system is presented in Section II. Later, in Section III, we shortly describe the characteristics of our in-field prototype deployment and the main hardware characteristics and constraints that drive the implementation presented in Section IV. Finally, in Section V, the paper presents an evaluation of the RAN system that shows how it is possible to perform sophisticated optimization decisions with a very small overhead in terms of CPU and memory.

II. RAN SYSTEM DESIGN

RAN combines centralized, hierarchical and fully-distributed management structures to address different tasks with the most appropriate approach. At the same time, RAN tries to maximize the distribution of tasks over the nodes. Only the tasks that inherently require a centralized organization, such as global optimizations, or those tasks which perform better on a weakly distributed structure, are carried out using hierarchical structures. Other management tasks such as local optimizations, follow a fully distributed approach.

In its most distributed aspect, the system is mainly composed by independent units or nodes that work as peers deployed at each managed network device. Those nodes, as in the well-known policy-based architecture described in [4], are composed by a Policy Decision Point (PDP), an Enforcement Point (EP) and an agent that monitors the state of the device and the behavior of the network. The PDP follows the event-condition-action (ECA) paradigm and the notifications sent by the monitoring agent of any of the nodes of the network is the source of events for the PDP of the same or any other node. All the communications between nodes, including control messages, notifications and request for actions are carried out by a content-routed asynchronous communication bus that organizes the nodes in a hierarchical overlay. Finally, a centralized management station is in charge of the edition, optimization and distribution of rules into the nodes.

Within this framework, policies are implemented as condition-action rules where the condition may be the occurrence of some event (e.g., an alarm or a service request) or certain network state, and the action is the desired response to that condition event.

The reconfiguration actions decided by the PDP may also involve other network elements, in which case the node may be the manager of the activity and delegate responsibility for some actions to other nodes.

In the following section, we describe with more detail the implementation of each of these components. The PDP and the EP while independent components are collectively named LuPA, and the monitoring agent RMoon.

III. PROTOTYPE PLATFORM

In the context of the RAN project, a fully working prototype deployment was made in a rural community of Uruguay (see [5] for more details). This prototype is a mesh ad hoc network with eleven wireless routers Linksys WRT54gl v1.1. These routers are designed as consumer-level devices that the vendor has trimmed down as much as possible. In particular, these routers have only 16 MB of RAM and 4 MB of storage memory. For the prototype, the original Linksys operative system has been substituted by OpenWRT Kamikaze 7.06 [6], a linux distribution for embedded devices. The routing on the mesh is made with OLSR [7] running inside the routers. To conform the mesh as a communications backbone for the community, the wireless routers were mounted inside weather-proof cases, added with Hyperlink 2.4 GHz 12 dBi Omnidirectional Antennas and fixed to electricity poles around the town (see Figure 2).

This setup imposes several constraints to the RAN system. The isolation and the lack of local people with networks knowledge is a strong motivation for self-management. However, the constrained resources of the chosen devices complicates the implementation of such autonomic behavior. This situation drove the design and implementation decisions described in the following section.

IV. IMPLEMENTATION

The first prototype developed for the RAN project, presented in [8], was developed using Java for the platform, operative system-dependent scripts for the enforcement agents and Witmate Java Logic/Rule Engine [9] as the core of the PDP. Due to the constraints of the devices that were finally used for the deployment of RAN, a new design and implementation was needed. The second implementation of the RAN system has been made using Lua [10]. This language runs on a virtual machine with a very small footprint, is fast and runs on many operative systems including some such as...
OpenWRT. This characteristic is particularly useful to our purpose of using small wireless routers. Below we present a short description of the RAN system components. Its source code and more details about the software can be found in [5].

A. RnR: Router-based Notification Router

As stated before, all the communications between the entities of the RAN system (e.g., PDP, EP, etc.) are made asynchronously by means of a distributed notification service named RnR (Router-based Notification Router). RnR implements a simple content-based notification service, which distributes messages from publishers to subscribers. The RnR design is inspired by existent notification services such as Siena [11], but its many implementation specificities are a consequence of the restricted resources of the small target devices.

The messages transported by RnR are made of a set of key-value pairs and are forwarded from one router of the service to another in accordance with filtering rules applied on the key-value pairs of the message itself. The routers of the service are organized as an overlay with an acyclic connectivity graph (a spanning tree of the network connectivity graph).

At this stage of development, the overlay’s topology is computed offline and configured at the routers centrally. An automatic overlay-construction protocol is to be developed as future work. Additionally, the matching process between arriving notifications and the subscriptions maintained by the RnR router is currently made in a simple sequential manner. This makes the process of matching algorithms with better scalability and their implementation is part of the future work.

In the context of the autonomic system being presented in this paper, the RnR is deployed as a supporting communications service that carries information about events of the system and asynchronous commands about actions that must be performed. However, RnR is a completely independent communications bus that can be used for any kind of communications between constrained devices. An evaluation of its performance is presented in Section V.

All the RnR communications are through TCP sockets, using a simple protocol with four control messages. The same protocol is used between clients and routers and between routers. In our setup, there is an instance of the RnR router running on every node that is part of the system, including clients. This makes the system homogeneous from the communications point of view.

The RnR protocol has four messages:

a) Subscription: when an entity of the system, such as a PDP or an RnR router, wants to subscribe to some kind of notifications it has to send a subscription message like the one depicted bellow.

```
SUBSCRIBE
subscription_id=srid
subscription_id=snid123
FILTER
antribute = this text
anotherattribute < 20
END
```

The message has a header with basic information about the message itself. That is: the identifier of the subscriber needed later to forward the notifications, and the identifier of the subscription, needed for processes such as unsubscriptions.

b) Notification: a notification is a message that either (i) informs about the occurrence of an event at some entity of the system, or (ii) carries a request for the enforcement of an action. Any client (publisher) can issue them or subscribe to receive those notifications of its interest. A sample notification could be:

```
NOTIFICATION
source=srid
notification_id = notid234
timestamp= 1234
antribute = this text
anotherattribute = 5.0
END
```

c) Control messages: there are two messages with a supporting role: HELLO and HELLO_REPLY. They are used during the connection phase to interchange identities between clients and routers and between routers.

B. RMoon

RMoon is a service that monitors the state of a device, and triggers “trap” notifications when certain conditions occur. Those notifications are delivered by RnR as described in the previous section. Therefore, RMoon is the main supply of events for the PDPs distributed over the network.

In order to start monitoring a certain variable with RMoon at a given node, a client has to send an action message with the command `watch_mib` to the monitoring node specifying: the attribute to watch, a threshold value and a comparison operator. Additionally, when using an inequality operator (“>”) or “(<”) a hysteresis value can be provided. A client of RMoon

\[1\] The name RMoon is a pun with the well known RMON (Remote Network MONitoring) and the Portuguese meaning of Lua (moon).
can be any entity of the RAN system, at the same node or at any other node of the network.

As a descriptive example, the message bellow sets up a monitor that will trigger a trap when the free RAM drops below a threshold:

```
NOTIFICATION
notification_id=notif1243
source=obsid
timestamp=1234
target=source_rmoon
message_type=action
command=watch_mib
mib=free_ram
op=<
value=40
hysteresis=1.5
END
```

The target of the Rmoon notifications is always the entity that originally had set up the monitor, therefore, when an entity sets up a monitor, it is automatically subscribed for the monitor’s output. Nevertheless, any other node can subscribe to the traps issued by any monitor and receive them whether they where targeted to it or not.

A typical trap message generated by the RMoon service looks like:

```
NOTIFICATION
notification_id=notif1243
source=nodeid_rmoon
timestamp=2345
target=obsid
message_type=trap
watcher_id=watch123
mib=free_ram
value=36.5
END
```

In the current implementation, watchable variables include free RAM, CPU load, and several network configuration parameters.

C. LuPA

The Lua Policy Agent (LuPA) groups the policy-related services or functionalities. It includes a PDP and an EP which can be installed together or separately, according to the role of the node. Each service connects to the RnR bus with his own identifier and all the local or remote communications between these services and with RMoon are made through the RnR bus.

1) Policy Decision Point (PDP): This component is the responsible of making decisions at each node and, as stated before, the policy rules that it executes follow the event-condition-action paradigm. The PDP knows about the occurrence of events at any place of the distributed system through notifications delivered by the RnR, it checks the condition of the rules with the help of the local or remote EPs, and produces actions in the form of notification messages that are delivered by the RnR bus with the attribute message_type = action. Those actions are to be executed by local or remote EPs depending on its target attribute.

The ensemble of local rules is modeled using a special kind of finite state machine, presented in [12], called a **Finite State Transducer extended with Tautness Functions and Identities** (TFFST). These machines are graphs with two labels on each edge, one expressing an input symbol and another specifying an output symbol. For a simple if-then rule:

```
if |jitter| > 20 ms then
   re-route Video-class connection;
```

this may be expressed as a transducer that receives information about jitter for one connection and, depending on this value, produces a certain re-routing action. This is illustrated in Figure 4. The TFFST has only one edge, its input label represents the type of event the rule is expecting (i.e., jitter) and the condition that this must fulfill (i.e., the jitter value) and the output label the output action that must be performed.

The inference process using this model exhibits good performance for policy evaluation. It is also oriented to the resolution of policy conflicts as each condition and action has an associated Tautness Function which can be applied to select between two conflicting policies. Tautness Functions are generally based on criteria, such as specifcness of the policy condition or cost of the policy action, and provide a means of evaluating the appropriateness and effectiveness of applying a given policy. In fact, the concept of a tautness function for conflict resolution provides a very flexible means of influencing policy choices in the PDP. (see [12] for more details)

The translation from the high-level policy rules into the finite state transducers implementing those rules inside the PDP, is currently made by a programmer. Despite the fact that this translation process is assisted by the algorithms presented in [12], it is still a complex and work-intensive task.

2) Enforcement Point (EP): The EP is an ordinary enforcement point, as it is described in [13], providing functionality to configure a node. The EP subscribes in the RnR for messages with message_type = action that are directed to its node, and implements several actions to modify device specific settings such as routing tables, interface parameters and tunnel features or doing higher level operations such as starting programs or setting RMoon monitors as described in IV-B. It also provides a get_mib action, which allows to query the node on its internal state. It’s possible to query on, for example, the CPU load, free RAM, or whether a given program is running.

The EP will answer with a notification with message_type = response, with the success status of

```
if |jitter| > 20 ms then
   re-route Video-class connection;
```

![Fig. 4. Correspondence between if-then constituents and FST elements.](image-url)
the operation, and the resulting value if the command was a query. At this point of development, all the configurations and queries are implemented running shell scripts.

V. EVALUATION

We evaluated the functional characteristics and some scalability and performance metrics of the RAN system on an emulated environment. In this section we present the results of those experiments. They are designed to demonstrate that RAN performs properly the tasks for which it was developed on a very constrictive running environment.

A. Emulation Environment

For the evaluation of the RAN system, we have developed an emulation platform named EmulaRAN. In this platform, the wireless routers are emulated by a set of VMWare [14] virtual machines running OpenWRT [6], the same operative system used in the in-field prototype. The virtual machines are built using the OpenWRT Kamikaze SDK for the i386 platform. The amount of RAM assigned to the virtual machine is 16Mb, the same amount of RAM available in the Linksys WRT54gl routers.

The wireless mesh network is emulated using iptables packet filtering [15]. For this purpose, all emulated OpenWRT nodes are setup in a single IP subnetwork; the nodes are assumed to be deployed in a flat, square field were all the nodes are in line-of-sight of each other. The dimensions of the field vary for each experiment. The radio visibility between any two nodes is computed considering a fixed transmission range and the distance between them. When two nodes are too far away to be visible, a DROP rule is set in the iptables chains of both nodes in order to avoid IP traffic between them. At its current stage, EmulaRAN does not emulates losses or any other quality-related characteristics of a wireless network. More details and source code for EmulaRAN can be found in [5].

**Fig. 5. Evaluation Setup**

EmulaRAN allows to connect a real Linksys router to the emulated network via a wireless interface of the hosting computer. This setup allows mixed experiments with part of the network emulated and part real.

B. Scenario

The experiments are designed following a scenario based on the objectives of the RAN project. In this scenario a mesh of domestic routers provides Internet services to a community using two 1Gbps/128kbps ADSL gateways that are installed at two different locations of the mesh. The billing for one of the ADSL connections is traffic-based and for the other is time-based, as in two commercial packages that are offered by a local ISP.

In this context, the less expensive option for applications with short peaks of traffic (e.g., a file download) is to route their traffic through the time-based gateway. However, for other applications, such as VoIP clients, may result less expensive to use the traffic-based gateway. The RAN system has to turn on and off the ADSL connections and manipulate the IP routes at each node with the purpose of reducing the total amount of money paid by the community for using Internet services.

As depicted in Figure 6(a), the traffic-based bill in this scenario is flat-rate up to 20GB of traffic per month, then it is lineal on the amount of traffic transmitted or received up to a maximum price (about US＄67) and then the connection is flat rate again with no limits of usage. The time-based package follows the same idea (see Figure 7(b)) but considering how long the ADSL connection is up.

C. Rules

To achieve the objective described before, different sets of rules are deployed at the nodes of the mesh depending on their

**Fig. 6. Per Traffic (a) and Per Time (b) Cost of ADSL Connections.**
role. The basic idea is that those rules vary a “control price” that the nodes have to “pay” for using each gateway (measured in arbitrary units). Each node chooses the gateway that is less expensive. Varying those control prices, the RAN system directs the traffic of different nodes to different gateways in order to optimize and balance the use of the two ADSL connections. Some more complex rules may be required to perform this task properly, but the small set below was enough to demonstrate the functionality of the system.

The rules below (1 and 2) are deployed in the traffic-based gateway (GW\textsubscript{T}). They are in charge of turning on the time-based gateway (GW\textsubscript{H}) when the relation between the traffic forwarded by the node (T) and a given period time H is too high.

Rule 1: if is\_high(T/H) and increasing(T/H) then turn\_on(GW\textsubscript{H})
Rule 2: if is\_high(T/H) and steady(T/H) then turn\_on(GW\textsubscript{H})

The rules below (3-11) are deployed in the time-based gateway. These rules change the control price of using the time-based gateway to avoid its congestion (rules 5 to 11) and turn it off when the relation T/H is too low (rules 3-4).

Rule 3: if is\_low(T/H) and steady(T/H) then turn\_off(GW\textsubscript{H})
Rule 4: if is\_low(T/H) and decreasing(T/H) then turn\_off(GW\textsubscript{H})
Rule 5: if is\_low(T/H) and increasing(T/H) then decrease\_price\_fast
Rule 6: if is\_medium(T/H) and steady(T/H) then decrease\_price\_slow
Rule 7: if is\_medium(T/H) and decreasing(T/H) then decrease\_price\_fast
Rule 8: if is\_medium(T/H) and increasing(T/H) then do\_nothing (no need to write this rule)
Rule 9: if is\_high(T/H) and steady(T/H) then increase\_price\_slow
Rule 10: if is\_high(T/H) and decreasing(T/H) then decrease\_price\_slow
Rule 11: if is\_high(T/H) and increasing(T/H) then increase\_price\_fast

Finally, the Rule 12 is deployed at all the wireless routers of the mesh. This rule sets the default gateway of the local router depending on the control price given by each gateway.

Rule 12: if new\_price\textsubscript{gwX} and price\textsubscript{gwX} < current\_price\textsubscript{defaultGW} then setGW\textsubscript{gwX}

D. Experiments

The objective of this experiment is to show that the system and the set of rules presented above in Section V-C perform the expected task: to reduce the monetary cost that a certain community has to pay for using Internet services.

The setup of this experiment follows the scenario described in Section V-B. It was set up on a PC with a Core 2 Duo processor of 1.66GHz and 4GB of RAM running EmulaRAN, plus one real Linksys WRT54gl router connected to the PC via the wireless interface.
On the PC, EmulaRAN runs the application server and the two application clients (see Figure 5) with Ubuntu 6.06 and 128MB of RAM, one for the “VoIP client” and the other for the “Bulk-transfer client”. Also running on EmulaRAN there are three virtual OpenWRT routers with 16MB of RAM each, one runs the traffic-based gateway and the other two play the role of the two mesh routers connected to the clients. The real Linksys router plays the role of the time-based gateway. The up-stream bandwidth of both emulated ADSL connections is limited to 128kbps and the down-stream to 1Gbps. The four routers, three virtual and one real, have the RAN system installed in them. The CPU and memory loads, depicted in Figures 7(f) and 7(e), are measured in the real Linksys router.

The synthetic traffic produced by the clients is emulated with MGEN [16]. There is one MGEN source on each client and a sink in the application server. One VoIP call is modelled as a CBR flow with 14.4 kb/s (voice encoded with the G.729 codec and using cRTP headers over ethernet). The bulk transference is emulated with a CBR flow of 96kbps.

For this experiment, we assumed that the community has just spent all the flat-rate traffic allowed by the ADSL contracts mentioned in Section V-B. Therefore, the cost of using each ADSL connection follows the cost defined by the graph sections between H1 and H2 in Figure 6(b) (approx. USD 0.02 per minute) and between T1 and T2 in Figure 6(a) (approx. USD 0.02 per MB).

The execution of the experiment starts with the traffic-based gateway activated and the time-based gateway deactivated and with its control price set to 200 units. The execution continue as follows. Firstly, all the participating nodes boot-up and the policies are distributed to the nodes. This can be seen as the first peak in the CPU load of the real Linksys router in Figure 7(f) and as several peaks in the traffic forwarded by the traffic-based gateway in Figure 7(a). Ten seconds later, the VoIP client starts generating up-stream traffic. Because the relation $T/H$ is low (less than 33%), no rule is triggered and nothing is done by RAN. Fifty seconds later, the bulk-trafic client starts generating an up-stream traffic that floods the traffic-based gateway (see Figure 7(a)). This triggers the Rule 1 and the time-based gateway is activated. Then, the control price offered by the time-based gateway is diminished drastically (to about 40 arbitrary units) to attract traffic to it (see Figure 7(d)). Therefore, Rule 12 works at all the nodes and the traffic of both clients is deviated to the time-based gateway (see Figure 6(b)). From then on, the control price offered by the time-based gateway increases too slowly to be visible in Figure 7(d). At the second 140, the bulk traffic is stopped. This situation triggers Rule 4 at the time-based gateway, which makes this gateway to stop forwarding traffic around second 170. Additionally it sets its control price to the maximum possible again (200 arbitrary units). Then, the VoIP traffic is forwarded again via the traffic-based gateway.

During the 210s of the experiment, a total of 3388.7 KBytes are forwarded through both gateways. Of that total, 1024.32 KBytes are forwarded via the traffic-based gateway and the rest through the time-based, which is active during 99s. This amounts to USD 0.053 to be paid by the community. However, same amount of traffic forwarded only through the traffic-based gateway, costs USD 0.066, and forwarded only for the time-based gateway, costs also USD 0.070.

Is worth noticing the evolution of the percentage of CPU consumed by the RAN system (LuPA and RnR) in Figure 7(f) and its the memory load in Figure 7(e). The CPU consumed by the combination of both agents is never beyond 40%, and it is almost all the time under 5%. This lets plenty of CPU resources for the main task of the router, to forward IP packets.

Although the figures in this experiment are low, the experiment manage to show how the RAN system works, and that it is able to achieve its objective consuming very few resources.

VI. FUTURE WORK

More experimentation is needed in order to test the properties of RAN. In particular the in-field prototype, that is already operational, will be used for similar experiments to those shown in Section V-D. Real, uncontrolled traffic will be transported by the mesh and real technology-related problems will probably appear. Additionally, new experiments must be designed to test the scalability of the system on the number of policies driving the nodes, and the network overhead caused for a high number of participating entities.

At this point of development, the RAN system has implemented many of its functionalities. However, there are many fronts in which the system needs further work. Currently the RnR overlay mentioned in Section IV-A is computed offline and enforced by centralized configuration on each node. Future work must include a process for the automatic construction and maintenance of that overlay, letting the nodes to go in and out of the system without disrupting its normal operation and without human intervention.

As stated before, the translation from the high level policy rules such as those seen in Section V-C into the finite state transducers implementing those rules inside the PDP, is currently made by a programmer. An automatic translation process is foreseen following techniques based on Metalua [17].

In Section IV-A we mentioned that the matching process is too simple and its order is $O(n \times m)$ on the size ($n$) of the notifications and the number ($m$) of subscriptions. Further work will implement better matching techniques based on search trees and hashes, which we expect will improve significantly the performance of the overall system.

In this paper we have not addressed several security issues that will inevitably arise when a system allows that some nodes modify the configuration of others. Future work has to address issues such as malicious nodes and privacy.

Finally, the current Lua code implementing RAN is functional but it was developed as a prototype. Therefore, that code must be largely optimized and debugged. We will continue working in this research line and, therefore, we will work on those code improvements.
VII. CONCLUSIONS

In this paper we present the RAN system. This system was developed to serve the objectives of Rural Ambient Networks, a project that targets the so called Digital Divide deploying low-cost wireless mesh infrastructure in rural communities. In particular, this implies to work with extremely constrained devices. We have shown that the RAN system, in autonomic and distributed manners, configures and dynamically re-configures the network devices to reduce the monetary cost that the community has to pay for using Internet services.

The main contribution of this paper is to show that it is possible to perform such a complex task with devices with very few computational resources.

It is worth noticing that all the experiments depicted in this paper where performed with the code implemented to run in the real target devices using the actual WRT54gI platform or virtual machines with the same OS and memory constraints. As stated in Section VI, there is still much work to do, however, as we have shown, the code developed until now comprises all the services needed for the operation of a self-configuring distributed system. Finally, we believe that the work presented in this paper may be of value not only for the particular scenario presented but also for any other distributed system for constrained devices such as sensor networks.

REFERENCES


