IEEE International Symposium on Policies for Distributed Systems and Networks

Policies for Self Tuning Home Networks

Dimosthenis Pediaditakis, Leonardo Mostarda, Changyu Dong, and Naranker Dulay
Department of Computing
Imperial College London, UK SW7 2AZ
{dpediadi, lmostard, cd04, nd}@doc.ic.ac.uk

Abstract

A home network (HN) is usually managed by a user who does not possess knowledge and skills required to perform management tasks. When abnormalities are detected, it is desirable to let the network tune itself under the direction of certain policies. However, self tuning tasks usually require coordination between several network components and most of the network management policies can only specify local tasks. In this paper, we propose a state machine based policy framework to address the problem of fault and performance management in the context of HN. Policies can be specified for complex management tasks as global state machines which incorporate global system behaviour monitoring and reactions. We demonstrate the policy framework through a case study in which policies are specified for dynamic selection of frequency channel in order to improve wireless link quality in the presence of RF interference.

1. Introduction

Home networks (HN) are becoming more complicated. A typical home of the past decade had a single computer with a modem. Nowadays, a typical home network features increasing numbers of devices, desktops, laptops, PDAs, smart phones networked with a wireless broadband router. It is not a trivial task to deploy such networks and it often requires bespoke configuration in order to work properly. This makes the management of home network complex. In addition, home networks have to be installed, used and managed by non-expert users, who often know little or nothing about the technicalities of home networks. The consequences are that home networks are usually not configured to an optimal state and it is difficult for home users to resolve networking problems.

Ideally, a home network should be able to configure itself automatically with minimum user involvement. Research efforts have been performed towards this direction. However they have mainly focused on automatically configuring network layer settings and service discovery. Little work has been done on automatic performance tuning and fault recovery. Moreover, most existing approaches perform simple actions that involve only a few (if not just one) devices and do not consider tasks which may require changing settings on several devices simultaneously.

In the present paper, we propose a policy-based solution to the aforementioned problem of home network management. We define policies in terms of global state machines for the target network. The network is monitored and the monitoring data is used to trigger reconfiguration and to provide feedback. We use a case study to demonstrate the proposed policy-based solution. More specifically, in section 2 we present a test-case scenario. In section 3 we present a systematic mechanism to define global system management behavior, using state machine based policies. In section 4 we present related work. In section 5 we conclude and discuss our future work plans.

2. Scenario description

A quite common problem-scenario for wireless home networks involves a user experiencing poor network performance. In many cases, degradations in performance are due to bad-quality wireless links between the home devices and the wireless access-point(s) of the network.

Bad link quality can be the result of so-called “co-channel interference” (or simply interference from now and on) which is caused by nearby stations operating on the same frequency. Interference either adds delays due to backing-off (when carrier sense operation can perceive it) or even worse, it causes frequent collisions which result in a high frame loss rate.

Home networks are usually deployed without little or no planning. When multiple networks are deployed in the same area, most of the time, will result in chaotic network topologies [1]. The main characteristic of chaotic wireless network deployments is the presence of extensive and intense interference. However,

978-0-7695-3742-9/09 $25.00 © 2009 Crown Copyright
DOI 10.1109/POLICY.2009.30

popular wireless technologies like IEEE 802.11 b/g [2] provide several channels for a given operating frequency band, which allows the simultaneous operation of more than one co-existing network. Despite the fact that the total number of channels is 13, stations can only use every fourth or fifth channel without overlap. Selecting the best channel to operate in a dynamic way is a problem without a straightforward solution, especially in the context of home networks where the use of expensive channel-measurement equipment is not possible.

In this work we basically focus on a particular case-scenario, where we assume a home network consisting of a number of devices that are equipped with wireless interfaces of compatible technologies (e.g. WLAN/WiFi). All devices are managed and interconnected via a wireless access point (AP). As described above, home WLANs are deployed in an arbitrary way and as a result, they suffer from serious contention and delays due to co-channel interference. We will aim to provide a policy-based approach for switching frequency channel in order to limit the interference levels which result in better quality for wireless links. It is quite important to understand that such a channel selection scheme has to be dynamic since interference varies over time and space, and home networks are also dynamic by nature. In addition, we usually cannot assume that links are symmetric in terms of quality. The need to go beyond the single-pair link quality management is quite evident. Channel section decisions should be taken in a distributed fashion and be based on distributed information from multiple nodes across the network.

3. Specifying global State Machine based Policies (SMP)

A global state machine describes a set of actions that can be performed in response to certain distributed events under certain conditions. The state machine is given in terms of sets of components which provide the system functionality. Components are themselves specified using an Interface Description Language (IDL). Given these specifications we use the Goanna platform [3] to automatically generate a distributed state machine implementation.

3.1. The state machine based language

In this section we briefly describe our state machine based language. In order to simplify the presentation we refer to our case study.

A global state machine declaration starts with a state machine signature, that is the keywords *global* and *fsm* followed by the state machine name and its formal parameter list. Each formal parameter is declared as a *set* which groups together component instances of the same type. For instance, in Figure 1 we define a state machine named *changeChannel,* implementing our channel tuning policy. It takes as an input the set of all clients and the set of routers (composed exactly by one instance in our case study). After the state machine signature, a list of event-state-condition-action rules follow, to define the state machine transitions. Each rule specifies which action can be performed when an event has been observed and some condition is true.

```
global fsm changeChannel(set routerSet,set clientSet){
  on c in clientSet to * where c sees bad_link
    0-1: {} → {}
    1-1: {not_persistent()} → {update()}
    1-2: {persistent()} → {
      signal changeChannel() to s in routerSet;
    }
    2-1: {not_persistent()} → {update()}
  on timeout(n)
    1-0: {} → {}
    2-0: {} → {}
}
```

Figure 1: The channel selection policy definition

Events can be component events or timeout events. A component event is consequence of a component service invocation. Four component invocation events can be observed: (i) invoke service and (ii) receive service reply for clients, (iii) receive invocation and (iv) reply to invocation for servers. We can also use the symbol * to denote unknown component types. For example, in the state machine of Figure 1 the event *on c in clientSet to * where c sees bad_link* is generated when a client device in the *clientSet* observes a low quality of its local link.

A timeout event specifies an integer *t*. This event is not a consequence of component interactions but the state machine implementation itself triggers it when (within the amount of time *t*) no component event has been successfully accepted. Referring back to our case study, a timeout can be accepted in state 1 when no bad links are observed.

For a given event *e*, the state machine can define a list of state-condition-actions. A state-condition-action is of the form q_s-q_t*: { condition }* → *{ action }* where q_s and q_t are the starting and the destination states while condition and action are a predicate and a piece of code respectively. Let us suppose that an event *e* is observed and the current state of the global state machine is *q* (with $q = q_s$). If in addition the *condition* is true, then, the *action* can be executed and the current state is set to the destination one q_t (i.e., the event has been successfully accepted). It is worth mentioning that the first state listed in the state machine definition is assumed to be the starting one. For instance 0 is the starting state of the state. The machine matches one event at a time. When an event can be accepted the state is changed accordingly otherwise the event is rejected. Referring to the case study, the rule: *on c in clientSet to * where c sees bad_link 1-1: {not_persistent()} -> {update()}* can be applied when the state machine is in state 1, a bad link is observed on a client and the *bad_link* observation is not persistent i.e., it is observed for a small amount of times for a given period of time.

3.2. Component definition

A component definition specifies two main parts: component provided/required services and a component object.

3.2.1. The component services

Component services are specified through a CORBA-like [4] IDL. This allows the description of a component as a set of services. Each service can be either asynchronous (labelled with "*async*") or synchronous and can be either required or provided (specified through the keywords *required* and *provided*). Nonetheless, a service has a return type and an optional list of formal parameters. In Figure 2 we describe a client device component providing a service *bad_link*.

```
component = client {
  services {
    async provide void bad_link();
  }
  attributes { integer counter=0; }
  methods {
    void update(){.......}
    boolean persistent(){...}
    boolean notPersistent(){...}
  }
}
```

Figure 2: The *client* Component definition

```
component = router {
  ...
  methods {
    external void changeChannel(){.......}
  }
}
```

Figure 3: The *router* Component definition

3.2.2. The component object

The component object is used to support the state machine specification. This information contains attributes and methods. Attributes can be used inside methods and state machine conditions/actions. The Methods are written using a C-like platform independent language and can refer to variables declared in the attributes section, call methods, component services and third-party libraries. A method can also call *external* remote services and methods defined on other component definitions and give rise to component events. For instance, the client declaration specifies an attribute *count* used by the procedure *update*() to count the number of bad links. The same variable is used by the predicate *persistent* and *notPersistent* to signal persistent and not persistent bad link conditions. The router component declares a method *changeChannel()* that can be invoked from methods of different component types.

Two primitives can be used for remote calls: (i) *signal call to instance* and (ii) *signal call to set*. The keyword *signal* is always followed by a method/service call (in our case *call*) while the keywords *to* can be followed by a component instance or a set. By using this primitive the method signals the platform to execute a remote method/service call. In case (i) the platform contacts the component instance and performs the call while in (ii) the platform contacts all component instances reachable inside the set and performs the call on them. The primitive *signal* returns a positive number when at least one instance was found. We emphasize that when the signal call returns the control to the method, the related method call has not been necessarily performed but instead, it may have been scheduled to be executed later on. For instance the signal *changeChannel() to s in routerSet* signal procedure is used to call the method *changeChannel* on the router definition in order to switch channel.

3.2.3. Component sets

```
  set clientSet where (clientSet.type==client)&&(
       clientSet.host==ALL);
  set routerSet where (routerSet.type==router)&&(
       routerSet.host=="192.168.0.1")
```

Figure 4: Sample *Set* definitions

Components can be grouped together to form sets. We have developed a definition language that allows the specification of sets based on component types and hosts that they are deployed on. While a *set* definition is unique it can have multiple instances each related to a different host. Moreover, *set* definitions can be categorized either as *local* or *global* ones. A local set is composed of component instances residing in the same host while a global one includes instances scattered over several hosts.

In Figure 4 we illustrate an example of two set definitions: *clientSet* and *routerSet*. The set *routerSet* is composed of all components of the type router instantiated on the host "192.168.0.1 ". This definition specifies a local set since all the component instances it includes must belong to the same host and moreover, it has a unique instance. The set *clientSet* is composed of all *client* components deployed in all hosts (i.e., a global set with a unique instance). The global state machine is decomposed into a fully distributed implementation. The reader is referred to [3] for details on the distribution process.

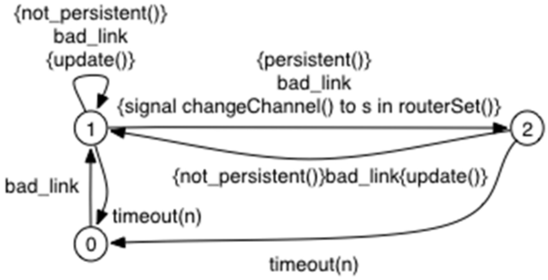

Figure 5: The FSM

The schematic representation of the proposed global state machine based policy is depicted in Figure 5. The state machine is in state 0 (healthy state) when the home network is working properly and no *bad_link* events have been observed for a while. The global state machine is in state 1 (alert state) when a *bad_link* event has been observed by some device. While the FSM is in state 1, any further *bad_link* events are simply logged as long as they do not become persistent and the FSM transits to state 2. Nonetheless, when no *bad_link* events are observed over a time duration *n*, a *timeout* event changes the state to 0. The global state machine is in state 2 (testing state) when the router has changed channel. In this state, further *bad_links* events change the state to 1 while if no such events are received for a given amount of time, a *timeout* changes the state back to 0. We emphasize that when an event cannot be accepted in a certain state it is discarded.

4. Related Work

Performance and fault self-management of networks has attracted a lot of research interest over the last decade. A recent trend is to make use of policies for defining global management behaviours [5]. The use of state machines for policy specifications is relatively new approach [3] [6]. In [7] Baliosian et al. use finite state transducer based policies for the self-configuration of wireless networks but their work focuses on the detection and resolution of conflicts. The advantage of our approach is that it enables the specification of non component / device specific policies, allowing the definition of global management behaviours for the network, seen as a whole. There is relatively little work that does this, especially in the context of home networks. Finally, there is a number of dynamic channel selection studies like [8] and [9]. These are quite low-level and do not allow cooperation between nodes in the network.

5. Conclusions and future work

In this paper we have introduced a novel policy framework for self-tuning home networks. Policies are specified using a state machine-based approach. In particular, a state machine can relate events scattered over several distributed components. Events can trigger actions which are further used to tune network parameters. For future work we are planning to implement the approach on a real network. We are also exploring the definition of state machines that can change their rules in order to dynamically adapt to the context. Finally, we plan to extend our framework to support multiple interacting state machines.

Acknowledgements

This research was supported by UK EPSRC research grants EP/F06446/1 (Homework) and EP/D076633/1 (UbiVal). We would also like to thank the members of the Policy Research Group for their continued support.

References

[1] A. Akella, G. Judd, S. Seshan, and P. Steenkiste. Self-management in Chaotic Wireless Deployments: Extended Version. Wireless Networks (WINET), 2006

[2] IEEE 802.11 Local and Metropolitan Area Networks: Wireless LAN Medium Access Control (MAC) and Physical (PHY) Specifications, ISO/IEC 8802-11:1999(E).

[3] L. Mostarda and N. Dulay. GOANNA: State machine monitors for sensor systems. http://www.doc.ic.ac.uk/~lmostard/tool/goanna.php

[4] W. A. Ruh, T. Herron, P. Klinker, and W. Ruh. IIOP Complete: Understanding CORBA and Middleware Interoperability. Addison Wesley, 2000.

[5] Kowtha, S. and Jiang, X. 2006. An N-State Driven Policy-Based Network Management to Control End-End Network Behaviors. In Proceedings of the 7th IEEE international Workshop on Policies For Distributed Systems and Networks, 5-7 June 2006

[6] Baliosian, J.; Serrat, J., "Finite state transducers for policy evaluation and conflict resolution," Fifth IEEE International Workshop on policies for Distributed Systems and Networks, June 2004.

[7] Baliosian, J.; Oliver, H.; Devitt, A.; Sailhan, F.; Salamanca, E.; Danev, B.; Parr, G., "Self-configuration for radio access networks," Seventh IEEE International Workshop on policies for Distributed Systems and Networks, June 2006

[8] N. Ahmed and S. Keshav. SMARTA: A Self-Managing Architecture for Thin Access Points. ACM CoNEXT, 2006

[9] B. Aznar, R. Kays, W. Endemann, O. Hundt and C. Schilling, Dynamic characteristics of wireless LAN channels for multimedia home networks. IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2007), 2007.