ANT 2012 Graduate Symposium

Distributed Policy-Based Management for Wireless Sensor Networks

Nidal Qwasmi, Ramiro Liscano

University of Ontario Institute of Technology
2000 Simcoe Street North
Oshawa, Ontario, Canada, L1H 7K4
Nidal.Qwasmi@uoit.ca Ramiro.Liscano@uoit.ca,

Abstract

Due to hardware resource limitations in Wireless Sensor Network (WSN), devices on WSN may hold up to 20 policies at any given time [1]. This number may not be sufficient at all times and has a huge impact on restricting the management capabilities and tasks that can be performed on the device as well as the whole WSN. The design choice of an existing policy-based WSN platform causes the policy engine to execute policies serially [2]; therefore, when multiple policies are triggered by an event, the order of the execution is not persistent [2]. This phenomena causes instability and unpredictability in the system because it may cause different policies’ orders to be executed every time the same event is triggered. The architecture of many existing or proposed policy-based WSN platforms relies on a local policy repository on the node to access any required policy [1] [2] [3]. This architecture choice raises many issues, mainly exposing the users to serious difficulties since they have to store policies on the targeted node only, creating serious administrative difficulties. The goal of this research is to create a new framework for distributed policy-based management for WSNs to overcome the existing policy-based WSN platform limitations.

1. Problem to be addressed by the research

Due to hardware resource limitations in WSN, devices on WSN may hold up to 20 policies at any given time [1]. This number may not be sufficient at all times; therefore, a dynamic deployment of the policies is necessary to utilize the node resources and accurately execute the required policies. These limitations have a huge impact on restricting the management capabilities and number of tasks that can be performed on the device and on the WSN as a whole. The design choice of an existing policy-based WSN platform...
causes the policy engine to execute policies serially [2]. Therefore, when multiple policies are triggered by an event, the order of the execution is not persistent [2]. This phenomena causes instability and unpredictability in the system because it may cause different policies’ orders to be executed every time the same event is triggered. The architecture of many existing or proposed policy-based WSN platforms relies on a local policy repository on the node to access any required policy [1] [2] [3]. This architecture choice raises many issues, mainly exposing the users to serious difficulties since they have to store policies on the targeted node only, creating serious administrative difficulties.

2. Research Hypothesis

The goal of this research is to create a new framework for distributed policy management for WSNs. The expectation is that this new distributed policy framework will meet the following objectives when compared to a non-distributed policy framework such as Finger/Finger2 [1] [2] [3]:

1. Extend the WSN management capabilities.
2. Conceal the complexity of administrating policy distribution processes from the users.
3. Ensure consistent execution ordering in the case of multi-policy execution to preserve consistency and persistence.

3. Research Contributions

1. Extend the WSN management capabilities
   Due to the nature of limited resources on the sensor node memory as discussed by Zhu et al [1], it is quite possible for a policy-based WSN network to have more policies than the sensor node capacity. The number of policies in the WSN is directly connected to the number of constraints that can be created on the WSN, which logically equal the number of functions that can be performed on the WSN. Therefore, the more policies the WSN can accommodate the more management functions (constraints) that users of the WSN can create.

2. Conceal the complexity of administrating policy distribution processes from the users
   As shown in [1] [2] [3], the architecture of many existing or proposed policy-based WSN platforms relies on a local policy repository on the node to access any required policy. This architecture choice raises many issues, mainly exposing the users to serious difficulties since they have to store policies on the targeted node only, creating serious administrative difficulties.

3. Ensuring consistent execution ordering
   In the case of multiple policies being triggered by an event, the sensor’s policy engine executes policies serially; however the execution is not persistent [2]. This phenomena causes instability and unpredictability in the system when the multiple policies are triggered by an event since it may cause different policies’ orders to be executed every time an event is triggered. This research intends to create a novel process to guarantee the policy execution sequence and prioritize each policy within a group of policies based on its triggered event(s).

4. Research approach and summary

In conducting this research work the existing policy-based management platform named Finger/Finger2 [1] [2] [3] will be used as a starting point for the new platform supporting distributed policy management. This research intends to build distributed policy frameworks for sensor networks based on the following components:
4.1. **System architecture:** Our proposed system framework consists of four main software components as shown in Figure 1. The main four software components are Monitor, Local Policy Decision Point (LPDP), Policy Enforcement Point (PEP), and finally a group of integrated Pastry applications which include TinyPastry, TinyPAST and TinySCRIBE. Moreover the proposed framework will include six data repositories.

4.2. **Software Components:** As shown in Figure 1, the main software components of our framework are the following:

- **Monitor:** a software component responsible for monitoring and updating BLOOM Filter value on the sensor network as well as the Local sensor node. The second responsibility of the Monitor component is to acquire any necessary policy from any other remote sensor node based on a request from TinyPastry. Monitor will also watch the most frequently used policies in the local sensor node and store them on the Local Policy Repository.

- **Local Policy Decision Point (LPDP):** a software component responsible of making local decisions based on applicable policies which to be enforced by the PEP. The LPDP decision is based on policies stored in the local policy repository or acquired by TinyPastry component. LPDP will first try to get the policy from the local policy repository, if the policy is not exist the LPDP check the BLOOM Filter to validate the existence of the policy within the sensor network. Based on the process outcome, LPDP decide wither to pass the request to TinyPastry or declare the policy as not exist.

- **Policy Enforcement Point (PEP):** PEP is a software component that enforces the policy decision (Action) as provided by LPDP.

- **TinyPastry:** a software component responsible of maintaining the location of different policies within the sensor network. In some cases where the actual policy is not exists in the local repository,
the TinyPastry will issue a request to the Monitor software component to acquire the targeted policy from a remote node.

4.2.5. TinyPAST: a software component builds on top of TinyPastry, it will be responsible of replicating the local policies to a different multiple remote nodes. The benefit of having TinyPAST in our framework is to increase the system persistence and overcome the nature in sensor network of nodes leaving the network with no previous warning.

4.2.6. TinySCRIBE: a software component builds on top of TinyPastry; it will be responsible of creating, participating, communicating and maintaining the necessary Topics (Events) to the local node. The benefit of having TinySCRIBE in our framework is to be able to create more complex policy cases where different events on different remote nodes may collaborate together through series of policies execution to achieve a desired results.

4.3. Data Repositories: The proposed framework will include six data repositories to support the system operations, and can be described as the following:

4.3.1. BLOOM Filter: the main objective of BLOOM_FILTER is to inquire about whether an element is a member of given set or not. This data structure will be utilized by an algorithm which was first developed in 1970 by Burton H. Bloom [4]. The purpose of BLOOM_Filter in our proposed framework is to provide assurance on whether a policy exists on the sensor network or not. This process prevents any unnecessary policy inquiry transaction to the sensor network which will results in faster decision processing and preservation of sensor node energy.

4.3.2. Policy Repository: a data structure to store the policy contents. The repository will have limited capacity and will be able to hold pre determined number of policies. Monitor software component will update the Policy Repository based on TinyPastry discretion or by monitoring the most frequently used policy.

4.3.3. Leafs Table: a data structure to store close-by sensor node address and Keys. This table will be used by TinyPastry to forward the request within the sensor network.

4.3.4. Routing Table: a data structure use to store faraway sensor node information. This table will be used by TinyPastry to forward the request within the sensor network.

4.3.5. Event List: a data structure to store all possible events for the local sensor node. It can be populated at the run or compile time.

4.3.6. Action List: a data structure to store all possible actions for the local sensor node. It can be populated at the run or compile time.

4.4. Policy Management Processes: The main processes covered are policy creation, policy deletion/modification and finally is policy execution. The steps for policy creation process are shown in Figure 2. Deletion and modification process steps are shown in Figure 3. Finally, policy execution processes are shown in Figure 4
Figure 2 Policy creation process

Figure 3 Policy execution processes

Figure 4 Deletion and modification process

References


