TinyPolicy: A Distributed Policy Based Management Framework for Wireless Sensor Networks

Nidal Qwasmi
University of Ontario Institute of Technology
Oshawa, Canada
Nidal.Qwasmi@uoit.ca

Ramiro Liscano
University of Ontario Institute of Technology
Oshawa, Canada
rliscano@ieee.org

***Abstract*—Policy-Based Management Systems (PBMS) are becoming a critical component of Wireless Sensor Networks (WSNs). Due to hardware resource constraints, policy-based management applications on WSNs can store only a limited number of policies in the local memory of a sensor node and must recycle them when additional policies are required. To tackle these challenges, a new distributed policy-based management framework named TinyPolicy has been developed, which can store, locate, access, and execute policies in a WSN. This framework uses a P2P policy storage and deployment mechanism, named PolicyP2P, which is designed to take advantage of available memory in the network and replication thus resulting in a policy system that is more robust against node failure, and single points of failure.**

***Index Terms*— Distributed Policy-Based Management, Wireless Sensor Networks, Peer-To-Peer.**

I. INTRODUCTION

Wireless Sensor Networks (WSNs) are becoming pervasive in our daily life, finding their way into such fields as environmental, medical, and military. Each WSN contains a number of sensors that are responsible for monitoring one or more events. A WSN usually works in a heterogeneous environment where sensors are incompatible with different hardware and software standards and from different manufacturers. Even though certain types of sensors may overcome some of these problems, this usually proves complex and costly. To overcome some of these challenges and to conceal the complexity of the underlying network devices from the human operator, researchers have considered Policy-Based Management (PBM) platforms a viable solution [1], [2], [3].

Many of the existing or proposed policy-based WSN platforms rely on a local policy repository on the sensor node to access any required policy [4], [5]. This type of architecture raises concerns related to the network dynamism and robustness since it creates node silos that can only communicate with the network gateway and cannot share resources with other nodes in the network. Moreover, this architecture creates an administrative overhead during the deployment of policies, because the administrator needs to explicitly know the address of the targeted node and how to create an exact replica of a defective node.

Our new framework, called TinyPolicy, can avoid this additional overhead by deploying the new policy to a hosted node based on a P2P architecture and the targeted node can access the new policy from the hosted node when it is required. The framework also maintains replicas of the policies in order to mitigate node failures.

II. RELATED WORK

The Madeira project [6] is a research project to develop solutions to Next Generation Networks (NGN) challenges. That project uses a fully distributed policy-based network management framework, which exploits the peer-to-peer paradigm. Madeira developed an overlay mesh network of distributed management elements. Each management element will be responsible for managing a subset of the network independently from other subsets of the network. The approach adapted by the Madeira project is similar to that in this paper, in that both use the policy-based management concept supported by an overlay network structure, except that our work is one of the few distributed policy frameworks to run on wireless sensor networks.

VanderHorn et al. [7] introduced the Cognitive Network Management System (CNMS). CNMS is a research initiative for complex Mobile Ad hoc Networks (MANETs). It provides a real-time policy-based management framework that aims to mitigate the need for centralized network management, provide automated management by providing reasoning and enforcing mechanisms for network resources, reduce human intervention, and increase network reliability. The authors achieve these objectives by utilizing a lightweight policy-based framework, which is able to adapt at runtime to unpredictable network conditions by creating and enforcing new learned policies. A learned policy is a new policy created by a cognitive node to mitigate unpredictable network conditions. Learned policies can be distributed to other nodes to manage similar network conditions.

Lee et al. [8] investigate a policy coordination approach to sensor node reprogramming. The two known methods for reprogramming are manual and over-the-air [9], [10], [11]. In manual reprogramming, the sensor node code is updated through physical access to the node. This has proven to be tedious and time-consuming. In over-the-air reprogramming, the code is disseminated over the air to all sensor nodes in the WSN. The drawbacks of this method are network congestion and energy depletion.

The proposed approach creates profiles (policies) that reduce the negative impact of different node reprogramming cases in WSNs, but it still suffers from the inherent issue with over-the-air reprogramming like restarting the node and code size. Our work reduces the overhead of over-the-air reprogramming by specifying the node behavior through policy programming, which requires significantly less transmission of data compared to full code reprogramming and no restarting of the node.

III. TinyPolicy: A Distributed Policy Framework

TinyPolicy is a policy-based management framework for WSNs in which policies are distributed among the memory of the nodes in the WSN. It also replicates multiple copies of a policy in the WSN in order to mitigate node failures in the network. It is designed to execute on low memory devices and consumes about 29.3 Kbytes of memory.

The framework for TinyPolicy consists of four main software components as shown in Fig. 1. The main four software components are Monitor (C1), Local Policy Decision Point (LPDP) (C2), Policy Enforcement Point (PEP) (C3), and PolicyP2P (C4) and their respective repositories.

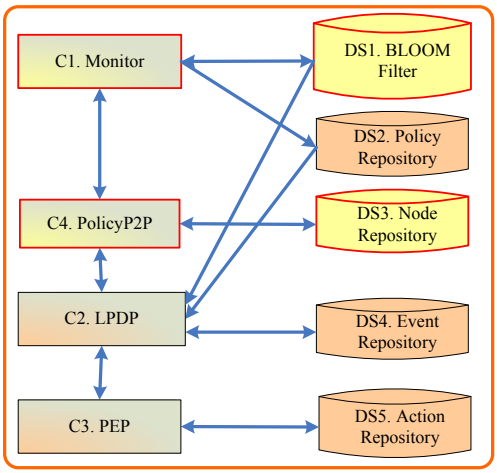

Fig. 1. The TinyPolicy Framework

Monitor: The software component responsible for acquiring and keeping track of policies used on the network. It uses a Bloom filter to reduce the lookup time for policies in the network. It is also responsible for pushing policies to the target nodes and maintaining replicas of these policies.

Local Policy Decision Point (LPDP) and Policy Enforcement Point (PEP): These components are responsible for executing and enforcing any local policies.

PolicyP2P: This component is responsible for maintaining a reference to the location of different policies within the sensor network. As the name implies, a P2P structure is used in the sensor network as a policy repository index.

Details of PolicyP2P are not presented in this paper as we focus primarily on the Monitor that coordinates the creation, modification, deletion, execution, and retention of policies.

IV. Policy Management in TinyPolicy

A. *Policy creation.*

TinyPolicy is based on the Finger2 platform [4] so the polices leverage the same compact structure that is used in Finger2 but the actual structure of the policies is not relevant for this paper.

The new policy creation process starts by using a policy user interface on a computer that is connected to the Root node of the WSN. The Root node of the WSN is defined as the highest-level node in the PolicyP2P hierarchical addressing structure. Typically this node is the gateway node for the WSN and has substantial power and memory.

The steps for creating a policy are illustrated in Fig. 2. After the policy is created using the Policy GUI, the Root node uses the PolicyP2P software component to determine a host node address for the policy. It does this by constructing the policy key (consisting of the concatenation of Node ID, Event ID, and sequence number) and then hashing the policy key to fit within address space of the sensor network. The Monitor of the Root node will them push that policy onto the node with the longest matching node address to the hashed policy key.

It also updates the local Bloom filter array and broadcasts the array to the rest of the nodes in the WSN. The Bloom filter is used as a mechanism to determine the existence of policy in the network. If the policy exists, it will be listed on the Bloom filter.

B. *Policy modification and deletion.*

The policy modification and deletion process is illustrated in Fig. 3. The process starts by checking if the policy exists in the local repository. If the policy does not exist in the local repository, the process is directed to the policy creation process as described in section IV.*A*.

If the policy is an existing policy, the process checks if the operation type is either deletion or modification. If the operation type is deletion, the Root deletes the policy from its repository (typically this is a backup policy) and broadcasts the deletion request to the rest of the WSN. Any other nodes with that policy remove the targeted policy from their local repositories.

Similar to section IV.A the Bloom filter array is updated and broadcast to the network.

For a policy modification, the policy creation authority retrieves the policy from the Root's local repository and the user can modify the policy. After the policy modification operation is completed, the Root broadcasts a deletion request to the other nodes, which remove the policy from their local repositories. The purpose of broadcasting the deletion request is to make sure that only one version of the modified policy exists in the WSN. Finally, the Root sends

the modified policy to the host node. There is no need to perform any changes on the Bloom filter array since the Bloom filter is only used to keep track of the existence of a policy and not its content.

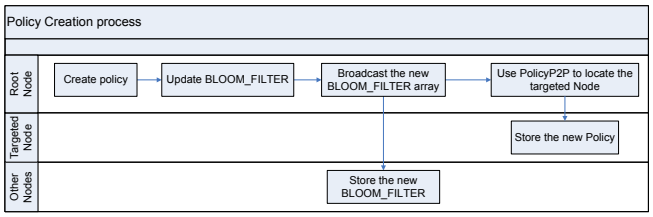

Fig. 2. Policy Creation process

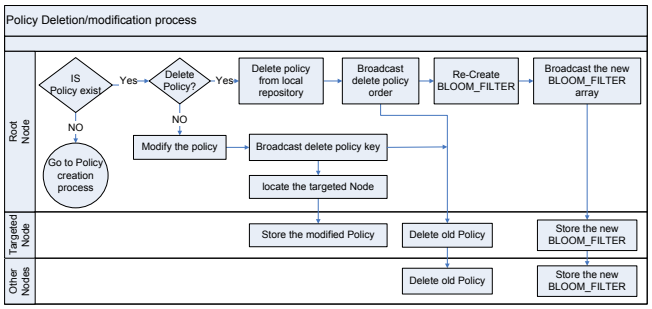

Fig. 3. Policy modification and deletion process

C. *Policy execution.*

The policy execution process is shown in Fig. 4. Each policy is associated with an event on the sensor node. The policy execution process of the associated policy starts when the sensor node triggers the associated event.

The algorithm then moves to check if the policy exists in the local policy repository. If the policy exists then two tasks are executed. The first task determines if there is more than one policy (multiple policies or a chain of policies) associated with this event. The algorithm examines that by incrementing the sequence number and submitting a new task for policy execution with the new policy key. The second task enforces the policy by evaluating the condition in the policy and applying the required action if it is a valid policy.

If the policy is not found in the local policy repository, the process will check the Bloom filter to validate the existence of the policy within the WSN. If the Bloom filter test is negative then no further action is required and the execution is stopped. However, if the Bloom filter is positive then PolicyP2P calculates the remote host node address, after which the Policy Execution Process sends a policy request to obtain the policy from the host node. If the host node provides the required policy then the process posts a new task for policy lookup with an increment to the sequence number to verify whether it is a single policy or multiple policies. After that, the algorithm enforces the acquired policy.

If the host node fails to provide the required policy the local node sends the request to the Root node as the ultimate "backup" node. If the Root node does not provide the required policy then the local node stops the execution and ends the process, because the policy does not exist.

As discussed previously, the local node might receive policies from remote nodes. In such cases, the local node would store the policies in the local node policy repository for future uses, based on the discretion of the policy-retention algorithm described in section IV.D

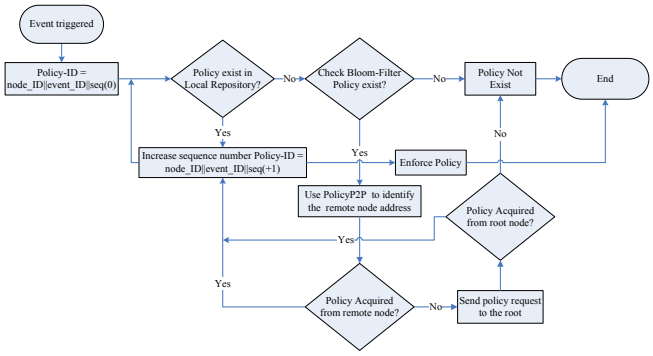

Fig. 4. Policy execution process

D. *Policy retention algorithm*

Fig. 5 shows the detailed steps of the policy retention algorithm. The purpose of this algorithm is to maintain the frequently used policies in the local policy repository. Every time the node receives a request to load a policy, this algorithm is triggered to check if the repository is full. If the repository is not full then no action is necessary. However, if the policy repository is full then the algorithm searches for a foreign policy that has the lowest frequently used rate. (Foreign policy is defined as a policy that has been hosted in the current node based on the discretion of the PolicyP2P algorithm.) The targeted policy is then replaced with the new policy.

V. IMPLEMENTATION OF TINYPOLICY

TinyPolicy was implemented in the nesC programming language [12] and executes on TinyOS version 2.1.0 operating system [13]. All experiments were carried out using the TOSSIM simulation software [13].

The following screen dumps show a typical policy execution in TinyPolicy based on the triggering of the "Timer" event (event id 0x06) on sensors 0x1000. The root node address is 0x0000. At this point it has been determined that the policy does not reside on the local node (0x1000).

In Fig. 6, the Bloom filter check is positive signifying that the policy exists in the network ttherefore, node 0x1000 requests the missing policy from a remote host node. In this example the targeted address for the remote node is node 0x0000, because the policy ID of 0x0fdb is hosted at node 0x0000 since it is the closest matching address to 0x0fdb. When node 0x0000 receives the request, it fetches the policy from its local policy repository and sends the requested policy to node 0x1000 as shown in the bottom section of Fig. 6.

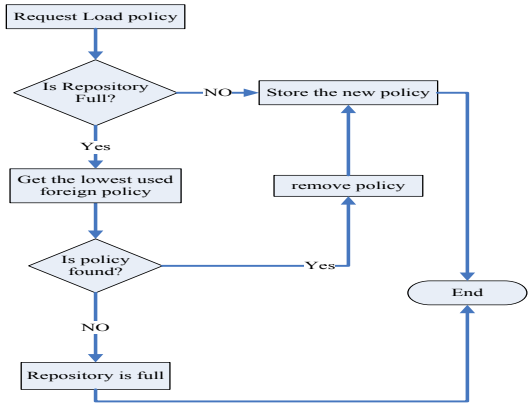

Fig. 5. Policy retention process

After the requested policy arrives at the targeted node (node 0x1000), the node checks its local policy repository to load the new policy if it does not exist, or overwrites it if it already exists. Following the loading of the new policy, the node triggers the applicable event to execute the new policy as shown in Fig. 7. The new policy is then evaluated as shown in Fig. 7.

VI. CONCLUSION

Policy-based systems for WSN are expected to play an important role in the Internet of Things (IoT), due to their great ability to abstract hardware complexity from a system's users. Policy-based management will help WSNs resolve the challenging issues of governing and controlling embedded devices.

A new distributed policy framework for WSNs named TinyPolicy was successfully created and tested. The new framework supports many new features, such as the dynamic distribution of policies among other sensor nodes using a P2P architecture, decentralized policy-based management, and policy replication. Another benefit of TinyPolicy is that it will dynamically manage the location of these policies by pushing the most widely used policies onto the target node as opposed to solely leaving them in the central Root node or on the other nodes in the WSN.

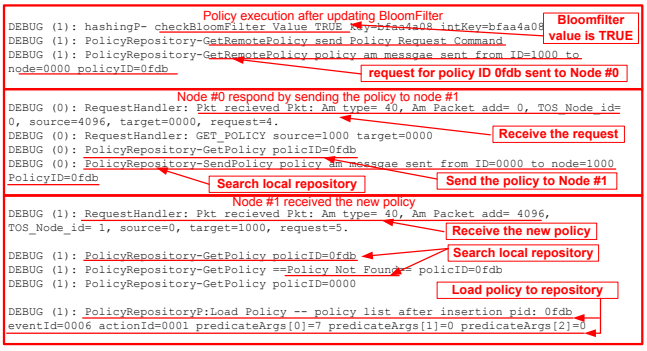

Fig. 6. Policy execution - Looking up and receiving a distributed policy

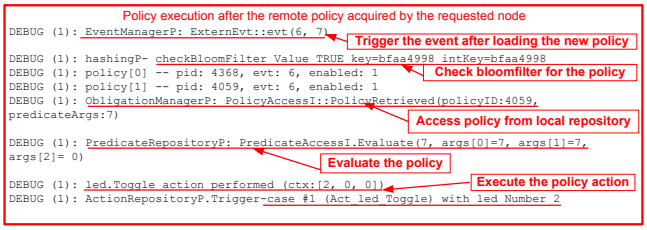

Fig. 7. Policy execution – Locally executing the policy.

ACKNOWLEDGEMENT

The authors would like to acknowledge the financial support of National Science and Engineering Research Council of Canada.

REFERENCES

[1] N. Matthys and W. Joosen, "Towards policy-based management of sensor networks," *Proc. 3rd Int. Work. Middlew. Sens. networks - MidSens '08*, pp. 13–18, 2008.

[2] W. Han, Z. Fang, and L. T. Yang, "Collaborative Policy Administration," *IEEE Trans. PARALLEL Distrib. Syst.*, vol. 25, no. 2, pp. 498–507, 2014.

[3] J. Strassner, *Policy-based Network Management: Solutions for the Next Generation*. San Francisco: Morgan Kaufmann, 2004.

[4] Y. Zhu, S. L. Keoh, and M. Sloman, "Finger: An efficient policy system for body sensor networks," in *5th IEEE International Conference on Mobile Ad-Hoc and Sensor Systems, MASS 2008*, 2008, pp. 428–433.

[5] T. Bourdenas, M. Sloman, and E. C. Lupu, "Self-healing for pervasive computing systems," *Archit. dependable Syst. VII. Springer Berlin Heidelb.*, vol. VII, pp. 1–25, 2010.

[6] R. Marin, J. Vivero, P. Leitner, M. Zach, and C. Fahy, "A Distributed Policy Based Solution in a Fault Management Scenario," in *Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE*, 2006, pp. 1–5.

[7] N. VanderHorn, B. Haan, M. Carvalho, and C. Perez, "Distributed policy learning for the Cognitive Network Management System," in *2010 - Milcom 2010 Military Communications Conference*, 2010, pp. 435–440.

[8] S. H. Lee, L. Choi, Y. Nah, S. Hong, and J.-A. Jun, "Policy-Based Reprogramming for Wireless Sensor Networks," *2010 13th IEEE Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput. Work.*, pp. 194–203, 2010.

[9] J. Gutiérrez, J. F. Villa-Medina, A. Nieto-Garibay, and M. Á. Porta-Gándara, "Automated Irrigation System Using a Wireless Sensor Network and GPRS Module," *IEEE Trans. Instrum. Meas.*, vol. 63, no. 1, pp. 166–176, 2014.

[10] S. Movassaghi, S. Member, M. Abolhasan, and S. Member, "Wireless Body Area Networks : A Survey," *Commun. Surv. Tutorials, IEEE*, vol. pp, no. 99, pp. 1–29, 2014.

[11] T. Naumowicz, R. Freeman, H. Kirk, B. Dean, M. Calsyn, A. Liers, A. Braendle, T. Guilford, and J. Schiller, "Wireless Sensor Network for Habitat Monitoring on Skomer Island," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference*, 2010, pp. 882–889.

[12] P. Levis, *TinyOS / nesC Programming Reference Manual*. 2009.

[13] TinyOS, "TinyOS." [Online]. Available: http://www.tinyos.net/. [Accessed: 02-Feb-2014].