LUND UNIVERSITY

Networking Media Abstraction, Device Discovery, and Routing for the Pervasive Middleware PalCom

Ergawy, Amr

2016

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Ergawy, A. (2016). *Networking Media Abstraction, Device Discovery, and Routing for the Pervasive Middleware PalCom*.

Total number of authors:
1

Creative Commons License:
Unspecified

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Networking Media Abstraction, Device Discovery, and Routing for the Pervasive Middleware PalCom

Amr Ergawy

Doctoral Dissertation, 2016

Department of Computer Science
Lund University

ISSN 1404-1219
ISBN 978-91-7623-960-5 (printed)
ISBN 978-91-7623-961-2 (electronic version)
Dissertation 51, 2016
LU-CS-DISS: 2016-03

Department of Computer Science
Lund University
Box 118
SE-221 00 Lund
Sweden

Email: amr.ergawy@cs.lth.se
WWW: http://www.cs.lth.se/amr_ergawy

Printed in Sweden by Tryckeriet i E-huset, Lund.

© 2016 Amr Ergawy

Abstract

PalCom is a pervasive middleware that can be used to assemble services provided by networked devices into configurations, called assemblies, for specific use cases by the user. In this dissertation, we present the development of a networking media abstraction framework for PalCom that abstracts different network interfaces in a PalCom device to upper layers of PalCom. The media abstraction framework is documented in paper I. Over the media abstraction layer, we define a device discovery mechanism that enables a PalCom device to discover other devices on its local networks, where it has network interfaces, as well as across interconnected networks. The device discovery mechanism is documented in paper II. On top of the device discovery layer, we implemented support for distance vector routing that enables routing data among discovered devices via the least cost routes. The routing layer is documented in paper III. In the last phase of our work, we refined our device discovery mechanism for PalCom to include a distributed synchronization algorithm that two PalCom nodes can utilize to re-sync their exchanged views of the network to overcome possible loss of device discovery and undiscovery notifications over unreliable channels. The synchronization algorithm is documented in paper IV.

Acknowledgments

I thank Prof. Boris Magnusson for giving me the chance to work on challenging tasks for the PalCom middleware. I also thank him for his help and valuable inputs during different stages of my work. I thank Prof. Görel Hedin for her support during different courses that were part of my studies and that helped as a background study for this dissertation.

I thank my colleague Mattias Nordahl for the valuable discussions and feedback as a PalCom-developer who used the components that I added to PalCom. I thank my colleague Björn A. Johnsson for his valuable inputs while testing PalCom against real world application scenarios.

I thank all administrative and technical support personnel at the CS department for their professional and friendly performance that helped me a lot to focus on my work.

I thank the Swedish research funding agencies SSF and VINNOVA that supported our group during my work. I thank the EU for my Erasmus Mundus MSc scholarship that definitely formed the foundation of the work in this dissertation. I thank Egypt for my free-of-charge education from 1st grade to BSc level.

Last but not least, I thank my wife Amira and my kids Reem and Ismail for their patience and support during the stressful times of my work on this dissertation and the system behind it. And before all, I thank my parents who made me what I am.

Table of Contents

Preface

This dissertation consists of two parts. The first part introduces the field of pervasive middleware and PalCom as a pervasive middleware. Then it summarizes the three layers that this dissertation has added to the PalCom stack, their design principles, and their evaluation. Finally, the first part of the dissertation discusses the planned future work.

The second part of this dissertation lists the four research papers that detail the discussion in the first part.

List of Included Papers

1. **Media Abstraction Framework for the Pervasive Middleware PalCom**,
   Amr Ergawy and Boris Magnusson,
   Published in Proceedings of the 2nd International Conference on Future Internet of Things and Cloud, FiCloud-2014, Barcelona, Spain, 2014. IEEE.

2. **Device Discovery for the PalCom Pervasive Middleware with Eliminated Cross-networks Periodic Heart-beat Messages**,
   Amr Ergawy and Boris Magnusson,
   Published in proceedings of the 5th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2014), Halifax, Nova Scotia, Canada, 2014, Procedia Computer Science, Volume 37, 2014, Pages 64-71.

3. **Distance Vector Routing over Device Discovery Forwarding Flows in the Pervasive Middleware PalCom**
   Amr Ergawy and Boris Magnusson,
   Published in Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies, ANT-2015, London, UK. Procedia Computer Science, volume 52, 2015, p. 153-160.

4. **Synchronizing device discovery on loss of update messages in the pervasive middleware PalCom**
   Amr Ergawy and Boris Magnusson,
   Published in Proceedings of the 11th International Conference on Future Networks and Communications (FNC 2016). Montreal, Canada, 2016.

Procedia Computer Science, Volume 94, 2016, Pages 347-352.

Contribution Statement

Prof. Boris Magnusson specified the requirements for the media abstraction layer, the state machines design of the device discovery mechanism in local-networks and cross-networks, and the requirements for the distance vector routing. Moreover, Prof. Boris Magnusson was continuously providing valuable comments for the design and implementation improvement. For the text in this dissertation, Prof. Boris Magnusson provided valuable reviewing comments.

Amr Ergawy designed, tested and implemented the media abstraction layer, the device discovery implementation, and the distance vector routing support. He also modified the design of the device discovery mechanism to align the discovery state of remotely discovered devices, via cross-networks discovery, to the discovery state of the router nodes that introduced these nodes.

Amr Ergawy designed the algorithm for synchronizing device discovery information between two neighbor PalCom nodes on the loss of device appearance/disappearance notifications over an unreliable channel. He also designed, implemented, validated, and tested the algorithm implementation.

For the included papers, Amr Ergawy is their main idea/structure designer and author while Prof. Boris Magnusson provided comments, modifications, and reviews. Amr Ergawy presented the first and the third papers while Prof. Boris Magnusson presented the second and the fourth papers.

Introduction

In section 1 of this introduction, we introduce the background of the PalCom pervasive middleware. In section 2 of this introduction, we summarize the background of the required communication in pervasive systems for service discovery and data exchange. Moreover, in section 3, we introduce PalCom, including its service discovery and composition features. In section 4, we introduce our problem statement while in section 5 we discuss our contributions. Continuing this introduction in sections 6, 7, and 8, we summarize the design principles, details, and evaluation of the networking media abstraction framework that we proposed for PalCom, our proposed device discovery mechanism for PalCom, and our solution for supporting distance vector routing in PalCom. Finally in section 9, we discuss our future work.

1 Background

PalCom is a *pervasive middleware* that was originally developed as an EU project between 2004 and 2007. We mean by pervasive middleware that PalCom aims to enable embedding continuously available computing in different devices that surround users in their living and working environments. This enables system developers and end users to combine the services of such devices into *assemblies* of useful usage scenarios. A more recent and practical alternative term to pervasive computing is the *Internet-of-Things*, *IoT*. David Fors's dissertation (1) details the original design, implementation, and possible usages of PalCom.

As part of the early development of PalCom, it was concluded that devices need to network via different types of networking interfaces, e.g. an IP interface or a Bluetooth interface. Such devices need to maintain their overlay PalCom network over whatever available underlying networks.

However, the original PalCom design did not provide a framework that enables developers to easily develop and integrate new network interfaces into PalCom. The first addressed problem in this dissertation is to design an architecture that enables PalCom developers to easily and quickly develop new abstraction objects, i.e. drivers, to support more network interfaces.

Another conclusion from the early development of PalCom was the need to enabled devices to autonomously detect the appearance and

disappearance of each other, in a timely manner. The original PalCom implementation uses a device-discovery mechanism that enables a device to periodically broadcast heart-beat messages via its various network interfaces. This approach enables other devices on the reachable local networks to discover and keep track of the availability of the sender device of the periodic heart-beat messages they receive.

As part of the original PalCom implementation, the device discovery mechanism was developed so that heart-beat periods are configurable per device. This configuration parameter can be used to optimize PalCom devices for a specific use case. Moreover, the original PalCom implementation provides the possibility of preventing a PalCom device form broadcasting heart-beat messages via a specific network interface as long as it broadcasts heart-beat-reply messages via that interface, where another device with a shorter heart-beat period initiates the heart-beats sequence. This key optimization in the original PalCom device benefits device discovery within local network.

However, the original PalCom implementation does not provide an optimal way to handle device discovery across multiple interconnected networks, i.e. cross-network device discovery. For example, if three local networks are connected in a line and there is only one device between each two consequent networks, then those two devices will forward periodic heart-beats between these networks. This is a network performance problem.

As a result, the second addressed problem in this dissertation is to improve the PalCom device discovery mechanism to replace the forwarding of cross-network heart-beats with once-sent device appearance and disappearance notifications. Moreover, the third problem that we address in this dissertation is to support shortest path routing over the resulting overlay network from the new device discovery mechanism. Finally, the fourth addressed problem in this dissertation is to design a synchronization mechanism that recovers devices discovery status after possible loss of once-sent notifications over unreliable channels.

2 Pervasive Computing and the Internet-of-Things

Pervasive computing is defined by Wieser's vision to embed computing technologies into different aspects of life so that user do not need to be aware of the existence of the computing platforms (2) (3).The vision of pervasive

computing was revised by different industry and research communities to identify challenges and approaches that may be used to address them (2). That revision was more oriented towards the technologies that we have today that differ from Wriser's time.

Even a decade after Weiser's vision, pervasive computing was still define as embedding computing devices and infrastructure in spaces where people live (4), producing smart spaces. This enables home and work places to sense different inputs and opens the door for an endless array of undefined use cases.

In its original version, Weiser's vision of pervasive computing ideally defined that users shall be minimally distracted by the computing infrastructures in their smart spaces (2). User may get distracted to fix minor issues and systems may learn for the user actions so that they can make automated decisions in the future (3) (4). However, it is not preferred to involve users in fixing networking issues in a pervasive environment that may contain a large number of interconnected devices (3).

In turn, enabling communication among a large number of interconnected devices in a pervasive environment was a main challenge to Wesier's research (2), where scalability was defined as the main challenge to enable interaction among devices. They focused on the density of connections that a device has to handle as its user approaches a pervasive environment.

During the years after Weiser's vision, many other technologies have evolved to support communication among interconnected devices (2) (3), including distributed computing, mobile computing, and sensor networks. These technologies provide foundations to communication support in pervasive computing.

Distributed computing provides the foundations of interconnecting devices whether they are in the same local network or in different remotely connected networks and regardless of the type of networking media of these devices (2) (3). These technological foundations were generic enough to be employed in pervasive systems.

The first of these foundation provided by distributed computing is defining layered network protocols and end-to-end quality of service (2) (3). It provided communication substrate to pervasive devices. Another technological foundation, of distributed computing, is fault tolerance techniques, e.g. atomic transaction, that enable a pervasive device to survive error-prone environments (2) (3). Distributed computing provides a third technological foundation for replicating control and execution (2) (3), which can be used for more reliability in pervasive systems. A fourth technological foundation of distributed computing is secure communication (2) (3), which

may be applied to pervasive computing with special considerations that match its error prone and ad-hoc nature.

For mobile computing, it supports communication in pervasive systems by providing several technological foundations that handle mobility management, variable network quality, constrained resources, and power consumption (2). Like mobile computing systems, pervasive environments assume that users move their hand-held devices among different networks (2) (3). That similarity makes the foundation of mobile computing a very close match to the needs of pervasive computing, considering the special requirement of pervasive computing.

For sensor networks, it provides the technological foundations for pervasive systems to sense and communicate important data about users and environments (4). Sensor networks devices have both the sensing and communication capabilities that provide the necessary building blocks for pervasive systems. Furthermore, sensor networks research provided important improvements in ad-hoc networking, system integration and real-time operating systems.

All of the above communication support is composed into pervasive systems using middleware systems and the underlying networks (3). Middleware systems interface different resources to pervasive applications, including networking resources (3). PalCom (5) (1) is an example of such a system.

3 Communication and Discovery in Pervasive Systems and the Pervasive Middleware PalCom

The communication support to pervasive systems is mainly important for service discovery, which is a networking protocol that enables a device to advertise its services to other devices in a network where it exists (6) (7). Also, it enables such a device to discover the services on other devices in the network. As devices in pervasive environments continuously join and leave networks, service discovery protocols must provide mechanism to register and unregister devices and their services so that devices can maintain enough information about each other.

A device that has just joined a network in a pervasive system needs to advertise its features and attributes that can be used by the service discovery

system to reply service discovery look-ups that may be initiated by other devices (6) (7). Matching services needs and requirements may employ querying searches or matching incoming service announcements to service needs.

3.1 Communication Support for Device and Service Discovery in Pervasive Systems

Service discovery protocols have to work in computing environments with no well-defined boundaries and the computing resources may vary over time from a network to another (6) (7). Also, service discovery protocols need to minimize the required configuration by automatically detecting devices and their services in a generic enough way, which enhance usability and reduce the amount of required development to handle different system setup and usage scenarios. However, service discovery still needs to maintain security and privacy.

a. Initial communication and networking media abstraction

Given the above goals of service discovery in pervasive systems, providing a media abstraction framework that enables service discovery protocols to discover devices and services via different types of network media is essential, which we provide by defining a media abstraction framework for the pervasive middleware PalCom (5) (1). Also, the above mentioned level of automaticity in service discovery needs support from an underlying device discovery mechanism that can be used by upper layers of a pervasive middleware, which we provide for PalCom by our proposed device discovery mechanism.

Moreover, service discovery protocols must specify an initial communication method that enables devices to discover and communicate with each other (6) (7). The most efficient way to provide such a facility is using unicast communication (6) (7), which enables the communication between two entities. However, unicast communication requires pre-configuration of the two communicating devices, which is a disadvantage. Alternatively, a service discovery protocol may use multicast as an initial communication method (6) (7). After using multicast for communication initialization, device may switch to unicast communication. However, multicast may also require a minimal amount of pre-configuration. A third alternative for an initial communication method is to use broadcasting (6) (7), which does not require pre-configuration but may flood the network with control packets in case of the inter-network broadcasting.

The above mentioned initial communication methods are highly coupled

with the underlying networking media and may require pre-configuration, e.g. for the unicast and multicast cases, and may flood the networks with control packets, e.g. for the case of broadcasting. Alternatively, for the PalCom pervasive middleware, we define the media abstraction framework that abstracts different networking media to our proposed device discovery mechanism, which provides an automatically configured initial communication method for upper layers in PalCom, including service discovery. In this way, the media abstraction framework decouples the initial communication method from the underlying networking protocols. Also, the device discovery mechanism limits the broadcasting of PalCom heart-beat messages, which are used as both device discovery initiators and keep-alive messages, within the boundaries of local networks.

b. Approaches of service discovery

Also, service discovery protocols must specify a method to refer to services and to exchange service attributes (6) (7) (8). Such a method can be template based, which uses a standard format to define service naming and attributes. That approach can be extended with a predefined set of commonly used services. In particular, service discovery and registration can be announcement based or query based (6) (7) (8). For an announcement based service discovery, all participating devices listen to service announcement via a specific communication channel. Alternatively, a query based system may reply to a service query by sending the required services information to the query sender.

To be able to exchange service information, service discovery protocols need to provide the required discovery infrastructure (6) (7) (8). A service discovery protocol may maintain a directory of all of the discovered devices and services. The system can use such a directory to handle service queries or to manage and process service announcements. Such a directory based solution can be used with a large number of interconnected device. Alternatively, a service discovery protocol may maintain a non-directory based solution, where all services on all devices process all received service queries and positively reply, with service information, when on matching the request. A non-directory based solution is more suitable for system with a limited number of devices.

c. Supporting device discovery

Regardless of the type of the service discovery infrastructure, the very first step of maintaining is to enable interconnected devices to discover each other. Our proposed device discovery mechanism for PalCom enables that very first step on local networks as well as across interconnected networks. Additionally, having the media abstraction framework under the device

discovery layer, the service discovery infrastructure in PalCom is enabled over different types of networking technologies.

After devices have exchanged service discovery information, services on devices must maintain their own state (6) (7) (8). A service may be assumed available for a specific time period since its discovery. Alternatively, a service may be periodically interrogated by the system for its availability. The availability of the devices that host services is the first step towards maintaining the availability of its services. Within a local network, our proposed device discovery mechanism for PalCom maintains device availability via heart-beat messages that are periodically sent by devices. For cross-network device availability, our proposed device discovery mechanism for PalCom depends on forwarding discovery events among the interconnected nodes.

Another perspective of service discovery in pervasive systems is the possibility to be limited to a specific discovery scope (6) (7) (8). That scope can be defined in terms of network topology, where the discovery process is limited to a given domain. However, such limitation may not suit pervasive systems that spread across different interconnected networks that form different domains. Our proposed device discovery for PalCom provides event based cross-network device discovery, which overcomes network based discovery scope limitations.

3.2 Device and Service Discovery in the Pervasive Middleware PalCom

PalCom is a pervasive middleware that uses heart-beat messages for device discovery and availability tracing (5) (1). On top of the network of discovered devices, PalCom can perform service discovery by exchanging service descriptions among these devices. PalCom employs the design principle of “human in the loop” (5) (1) that enables users to specify ad-hoc compositions of pervasive services into specific use cases. PalCom refers to such compositions as assemblies, as shown in Figure 1.

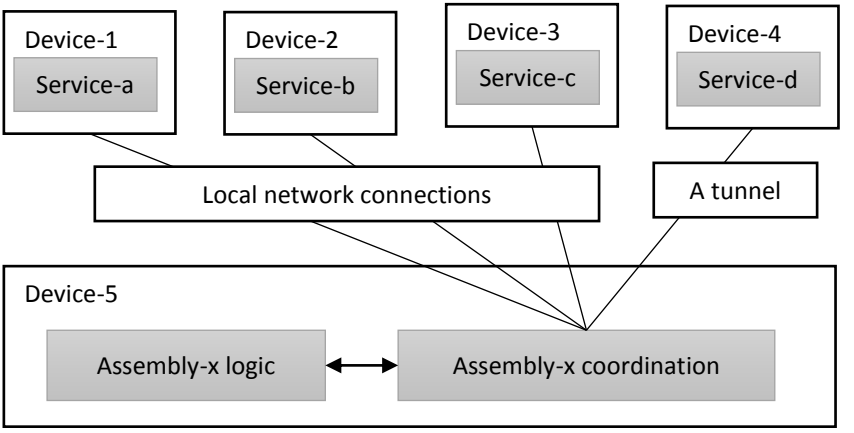

Figure 1 an example structure of user-defined PalCom services assembly.

However, PalCom aims to avoid overwhelming the users with an endless number of available services. Instead, it focuses the users' experience of fully automated service discovery on the available services in their local network environments (5) (1). For remotely available services, i.e. outside local networks, PalCom uses a tunneling mechanism that is explicitly configured by users. This approach supports the scalability of the service discovery protocol of PalCom.

In addition to device and service discovery, PalCom defines connection discovery (5) (1). In PalCom, connections define paths of communication among interacting services. Such feature improves the support of the "human in the loop" principle by introducing users' awareness of the communication paths that resulted from their ad-hoc service compositions, i.e. PalCom assemblies.

a. PalCom is a layered system

PalCom has a layered discovery stack (5) (1), which is shown in Figure 2. At the lowest level of this stack, the wire protocol layer abstracts different types of underlying networks. On the next level, the device discovery layer periodically announces the existence of devices via heart-beat messages that can be used for discovering and keep tracing of devices among each other. A third layer of service discovery manages the announcement of service availability and defines service descriptions. Finally, at the top most level, the service interaction layer defines how services may invoke each other and the formats of the commands, their parameters, and their responses.

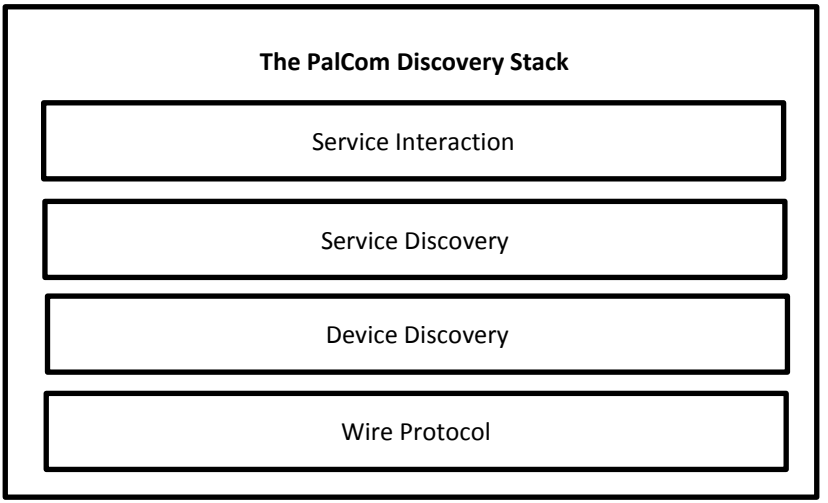

Figure 2 the layers of the discovery stack in PalCom.

b. The wire protocol layer

The wire protocol layer supports heterogonous pervasive environments by defining formats of payload messages that are conveyed over different underlying networks (5) (1). PalCom message formats are human readable and a message is composed of nodes that makes it flexible to modify by adding or removing nodes. The wire protocol defines the heart-beat mechanism that enables device discovery and tracing of each other. Broadcasting heart-beats is limited to device discovery while unicast is used for device communication, service discovery, and service interaction. Discovery messages, including heart-beat messages, convey device discovery status parameters that indicate its reboot number, configuration or services change number, and its functional readiness.

c. The device discovery layer

The discovery protocol maintains a distributed and replicated discovery registry among the interacting devices (5) (1). The heart-beat mechanism allows the configuration of the heart-beat period for each node and the node with the shortest period will become a heart-beat leader node within a local network (5) (1). In this way, nodes other than the leader node will continuously abort sending heart-beat messages and they will only reply to the received heart-beat messages from the leader with hear-beat-replies. This configuration limits the number of heart-beat messages within a local network during a single heart-beat period of the leader node to be equal to the number of nodes in that local network. Additionally, at the service level, the PalCom discovery protocol enables explicit status requests and replies among interacting services (5) (1).

d. The service layer

Additionally, using the PalCom discovery protocol, grammatically constructed descriptions of devices, services, and connections are exchanged and cached by interconnected PalCom nodes (5) (1). The change-number in the heart-beat messages represents the current version of the device, services, and connections descriptions on its source PalCom node. When that change-number is updated, nodes that cached device, services, and connections descriptions from its source node will need to execute descriptors request and responses sequences in order to refresh its cached discovery data.

e. A summary of PalCom features

To sum up main PalCom features (5) (1), firstly, it defines self-describing devices, services, and connections that exchange descriptions in terms of grammar based descriptors. This is different from the mentioned template based and predefined descriptions that we mentioned above in section 3.1. For the initial communication method, PalCom mainly employs broadcasting within local networks. Moreover, PalCom employs discovery announcements at the device level to enable device discovery and availability tracing. For the service discovery infrastructure, PalCom employs a non-directory based service registry structure. Considering the state of a PalCom device, it may be maintained via the heart-beat messages period, considering it as a keep-alive period. Naturally, the timely maintained device state overshadows the state of the services on that device.

For the feature of discovery scope, PalCom fully automates device discovery within local networks. Also, PalCom enables devices with network interfaces on multiple networks to function as routers among the devices within these interconnected local networks. Moreover, PaLCom enables users to configure PalCom tunnels to interconnect remotely located networks (5) (1). In turn, the interconnection of PalCom devices and services inherits the network topology. For usability, PalCom enables users to define assemblies that combine different services into specific use case scenarios by defining their roles and their interactions. For service invocation, PalCom utilizes the address of the service provider to enable services to communicate over the PalCom wire protocol so that PalCom assemblies can exchange their messages to perform application specific operations, which are realized as service invocations. After a service invocations ends, PalCom releases used resources. PalCom services do not hold computation resources whenever they are idle, not executing service requests. In other words, a PalCom service is assumed leased as long as it is serving requests or requesting services. For the service updates notification mechanism, only the

change number in the broadcasted heart-beat messages can convey the changes to service description on a device. However, the heart-beat messages broadcast is kept within the boundaries of local networks to optimize network traffic.

3.3 Example Application Scenarios of PalCom

In (9), PalCom is developed to enable users to easily develop PalCom services and assemble them into useful use cases, especially in the medial field. In particular, PalCom enabled health care devices can be combined into useful assemblies that integrate patients' data into more useful and informative perspectives. Such devices may provide services that vary from allowing patients to write their own notes to enabling healthcare personnel to graphically describe patients' pains.

Moreover, integrating household devices to serve specific user needs provide a number of other application scenarios where PalCom can be used in a flexible and efficient manner. In (10), a simple demo was developed to show the simplicity of using PalCom to program a water boiler to keep water at specific temperature level. However, more complicated scenarios and use cases can easily be defined and built by users using PalCom.

Another possible application of PalCom is the timely detection of losing contact among safety critical components of a robot. Such scenario should lead to the timely device undiscovery problem that we explain next.

4 Problem Statement

As mentioned in section 3.2, the lowest level of the PalCom discovery stack is the wire protocol that enables PalCom nodes to exchange messages among each other over different types of networking interfaces. Since the very first proposals of PalCom, the network interfaces on a device are abstracted to upper layers of PalCom by media abstraction objects, MAOs. For each supported network protocol, a MAO is implemented to wrap the different details of the network protocol including its message formats and communication mechanisms. A MAO is required to mainly provide interfaces to the send and broadcast procedures of a network protocol. For every network interface on a PalCom node, an instance of its protocol specific MAO is used to represent that network interface to the system.

a. The requirement of a media abstraction framework

However, in the initial designs and implementations of PalCom, there was

no MAO development and integration framework that enables MAO developers to easily and quickly develop a MAO for a new network protocol in PalCom. Instead, a MAO developer had to be aware of the internal details of PalCom upper layers and to find a customized way to integrate the instances of the newly developed MAO to those upper layers. This resulted in difficulties developing new MAOs and unstructured integration of these MAOs with upper layers of PalCom, which makes future system development and maintenance not feasible.

In turn, we identified the need to design and develop a media abstraction framework for PalCom that gives a starting point template for the MAO developers to start their MAO implementations from. This makes it much easier to develop and interface MAOs for newly supported network protocols in PalCom. However, as we explain in sections 6.1, 7.1, and 8.1, our first research question is how to design such framework given the vast number of network protocols that may exist in a pervasive environment. Also, we need to consider both the reliability and the ease of use of the framework.

b. The requirement of cross-network device discovery

In section 3.2 above, we mentioned that PalCom limits the fully automated device discovery process, using periodically broadcasted heart-beat messages, to the boundaries of local networks. Then, PalCom employs user-configured tunnels to connect devices on remote networks. In spite of such an approach optimizes the network traffic, it imposes a topological constraint on the fully automated device discovery, and of course service discovery.

To overcome such limitation, we identified that we need to develop an automated device discovery mechanism that enables both local and cross-networks device discovery. However, forwarding heart-beat messages across interconnected networks is not an option as this will result in flooding the network with discovery packets. Instead, as we explain in sections 6.2, 7.2, and 8.2, our second research question is how to limit the use of periodically broadcasted heart-beat messages within local networks by enabling PalCom nodes to timely forward their discovery events across interconnected networks.

c. The requirement of support distance vector routing

By introducing automated cross-networks discovery to PalCom, we also introduced the need for routing messages among interconnected nodes. We design and implement distance vector routing in PalCom according to the design principles and details in sections 6.3 and 7.3. This is where we address our third research question of how to minimally add the routing logic while preserving the correctness and the simplicity of the media abstraction layer and the device discovery layer.

d. The requirement of device discovery synchronization

Moreover, as a complement to the mentioned events based cross-network device discovery mechanism, we need a distributed synchronization algorithm that enables neighbor nodes on local networks to recover from possible loss of device appearance/disappearance notifications on unreliable channels. We explain the design principles and details of this algorithm in sections 6.4 and 7.4. This is where we address our fourth research question of how to add reliability to forwarding device discovery notifications.

We emphasize our research questions as follows:

1. How can we design a networking media abstraction framework considering a vast number of networking interfaces that can co-exist in a pervasive environment? Also, how can this framework be easy to use by PalCom developers and reliable against possible mistakes?
2. How can we handle device discovery and undiscovery in remote networks within a configurable time without flooding the network with heartbeats?
3. How can we support distance vector routing for an overlay network of PalCom devices that is built on top of the media abstraction framework and device discovery/undiscovery mechanism while preserving the simplicity and correctness of these two lower layers?
4. How can we overcome the possible loss of device discovery and undiscovery notifications over unreliable channels?

5 Contributions overview

PalCom (5) (1) is a pervasive middleware that can be used to assemble services provided by devices into configurations, called assemblies, for specific use cases by the user. Developers can write service descriptions for the capabilities of devices and PalCom can advertise these services to all devices running PalCom on the network. PalCom devices may exchange service descriptions and communication data via different network interfaces that connect theses device over different networks. Also, before being able to exchange service descriptions and data, PalCom devices need to discover each other within the boundaries of local networks as well as across different interconnected networks.

a. The media abstraction framework

In this dissertation, we provide a media abstraction framework for PalCom

that abstracts different network interfaces in a PalCom device to upper layers of PalCom. A network protocol is abstracted with a Media Abstraction Object, MAO. On a PalCom device, each enabled network interface is abstracted using an instance of the MAO of that protocol. The media abstraction framework enables developers to write and plug MAOs to enable the use of different network protocols on PalCom devices. Having multiple network protocols and interfaces on a PalCom node enables it to communicate with a broad range of devices. We design this framework according to a set of design principles that depend on the design principles of the abstracted network protocols (11). The design principles aim to separate the abstractions of network protocol, e.g. addresses and data, from its communication mechanisms, e.g. connection establishment and message transfer. The framework was implemented in Java, as part of the PalCom stack. It was also evaluated for the time overhead that it adds to the message transfer and the usability for implementing MAOs. This work resulted in paper I.

b. The device discovery mechanism

Over the media abstraction layer, we define a device discovery mechanism that enables a PalCom device to discover other devices on its local networks, where it has network interfaces, as well as across interconnected networks. The device discovery mechanism uses heart beat messages within local networks, enabling the discovery of devices within these networks, and also functions as keep alive messages among these devices. When a device discovers other devices via the network interface a given local network, it advertises these discovery events to devices on its other local networks. Devices on those local networks than can thus discover a remote device that appeared across interconnected networks. In turn, those devices may advertise cross-networks discovered devices to their local networks. Also, a PalCom device keeps track of the availability of cross-network discovered devices by monitoring forwarded discovery events from their neighbor nodes. Such events may indicate the appearance and disappearance of cross-network discovered devices. This eliminates the need to forward heart-beat messages across interconnected networks, saving network traffic and providing timely device un-discovery. Moreover, we refer to the devices on a local network that forwarded discovery events about remote-devices as router devices. The disappearance and appearance of a router device overshadows the discovery state of its advertised remote-devices. The device discovery mechanism was implemented in Java, as part of the PalCom stack. Also, it was analyzed and tested to prove its performance enhancement by eliminating cross-networks device discovery.

This work resulted in paper II.

c. Supporting distance vector routing

On top of the device discovery mechanism, we implemented support for distance vector routing that enables routing data among discovered devices via the least cost routes. We define the route cost with the number of hops. In contrary to on-demand ad-hoc routing protocols (12), distance vector routing in PalCom does not need to implement a separate route discovery mechanism. Also, in contrary to table-driven ah-hoc routing protocols (13), distance vector routing in PalCom does not need to implement a mechanism to exchange table updates. Instead, distance vector routing in PalCom makes use of the underlying device discovery mechanism that is used to maintain the entries in the routing tables using route cost updates that are conveyed in cross-networks discovery events. Moreover, the events driven cross-networks discovery in PalCom is more efficient than the techniques of flooding interconnected networks with control packets, which are used for different purposes in ad-hoc networking routing protocols (14) (15). From another perspective, distance vector routing in PalCom does not make use of the underlying MAC protocols to detect broken or failed links, which is a common measure in ad-hoc routing protocols (16) (17) (18). Instead, in PalCom, the media abstraction layer abstracts the underlying network protocols and broken links can cause the device discovery mechanism to undiscover known devices. We specified a set of design principles for the support of distance vector routing in PalCom. These principles ensure the state consistency among different layers of the PalCom stack, the separation of concerns among them, and minimizing the route dynamics. The support for distance vector routing in PalCom was implemented in Java, as part of the PalCom stack. Also, its functionality was evaluated against a set of network changing scenarios. This work resulted in paper III.

d. The device discovery synchronization mechanism

In the last phase of our work, we refined our device discovery mechanism for PalCom to be compatible with unreliable channels (19) (20) (21). In particular, since the events of device appearance/disappearance notification messages are once sent on the local network channel between two neighbor PalCom nodes, there is a possibility for losing such messages. Such situations lead to out-of-sync views of exchanged device discovery information. To solve this problem we defined our distributed synchronization algorithm that two PalCom node can utilize to re-sync their exchanged views of the network. The algorithm is sequence-number based and it uses the periodic heartbeat messages on the local networks to detect the out-of-sync situation and to ensure the reliability if the synchronization process itself. The

algorithm is modelled using UPPAAL (22) and verified against clear and simple correctness properties. Also the algorithm performance was evaluated using a simulated run of the system model in UPPAAL. Finally, we implemented the algorithm as part of PalCom and we tested its functionality using three virtual PCs connected via two virtual local networks. We used scripts to control the network interfaces of the nodes in the test to emulate channel failures. The algorithm is able to recover discovery information in case of losing update messages due to emulated channel failures. This work resulted in paper IV.

6 Design Principles

The design process of providing a networking Media Abstraction Framework, MAF, for a middleware system like PalCom needs to specify a set of principles or main guide lines that ensure the fault tolerance and resilience of the framework when it is used by developers to develop media abstraction objects, MAOs, that abstract and integrate underlying networking protocols to the system. I.e. the MAF needs to isolate errors or exceptions in a faulty implementation of a MAO from the rest of the integrated MAOs, and the framework needs to continue serving these MAOs when faults appear during run-time.

From another perspective, the design principles of the Media Abstraction Framework, MAF, shall consider the separation of the aspects of network protocols to make it clear for the MAO developer which of these aspects will be wrapped and hidden from upper PalCom layers and which of them will be abstracted to these layers. Likely, such design principles must clarify the concurrency assumptions of the framework and how the MAO developer shall handle internal concurrency. We discuss the design principles of the MAF in sections 6.1 and 7.1.

For a mechanism that enables device discovery in a middleware like PalCom, the design principles need to consider two very important aspects of pervasive systems, i.e. dynamism and heterogeneity (7) (23) (24). In a pervasive environment different devices with different network interfaces may interact while they continuously join and leave the pervasive environment. Not all of these devices need to be easy to configure every time they join or leave the network. Thus, a device discovery mechanism may depend on as much as possible of automated device configuration. In turn, such mechanism may make use of periodic broadcasting heart beats for both

discovery and keep-alive notification among the interaction devices.

However, from the perspectives of scalability and network traffic, using heart-beats broadcasting is only feasible within the limits of local networks and not for across networks device discovery. In section 6.2, we list our design principles a device discovery mechanism for PalCom that makes use of both heart-beats broadcasting and cross-network discovery events to build a substrate for service and data communication.

To build distance vector routing into PalCom, where we already implemented a media abstraction framework and a device discovery mechanism, we have to specify design principles that considers the relation among these components while keeping the already existing limits among them. Also, these design principles need to consider the scalability of handling the resulting changes in routes according to the received route cost updates.

In section 6.3, we list our design principles for adding distance vector routing to the PalCom stack aiming to maintain the consistency of the device discovery state among different components of the device discovery mechanism, while keeping these aspects separate. Also, we specify a design principle that ensures minimal changes in the advertised routes to the neighbor nodes of a router PalCom node on a received route cost update.

6.1 Design Principles of the Media Abstraction Framework

As discussed in the problem statement in section 4, PalCom requires a flexible way to easily add Media Abstraction Objects, MAOs, to support network interfaces that may appear in the future. We design a Media Abstraction Framework, MAFs, which enables the development of Media Abstraction Object, MAOs, that abstract different type of networking interfaces, and their protocols, to the upper PalCom layers. We specify the following design principles of this framework:

a. Abstracting the development framework of the Media Abstraction Objects, MAOs

We have to make sure that the development framework, where a MAO developer starts his implementation from, is very well separated from any protocol specific details. In other words, the MAO development framework needs to be composed of abstract classes that has no protocol specific mechanism implemented in any of its override-able or non-override-able methods.

In section 7.1 of the implementation details, we show that the

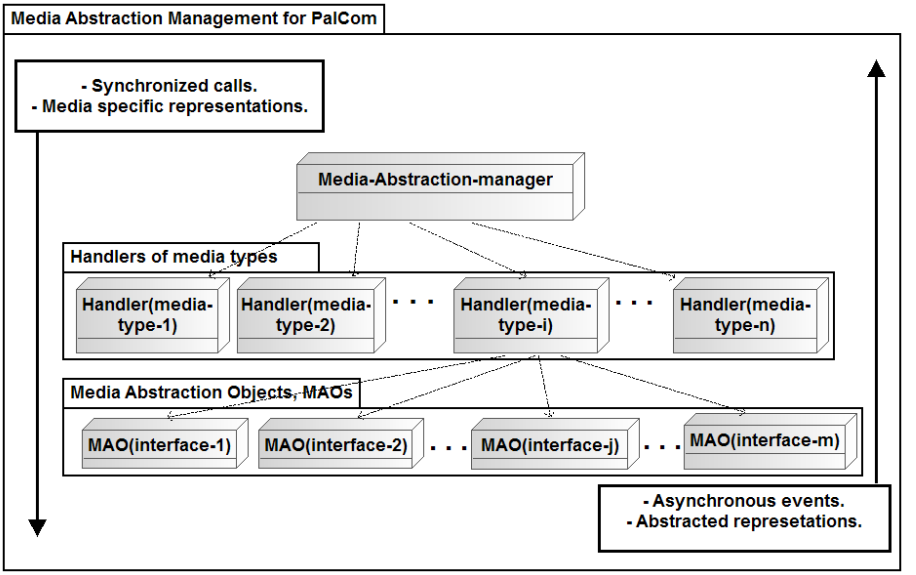

Figure 3 the design principles of the media abstraction framework.

methods of these abstract classes only represent very generic network abstraction of human-readable network addresses, human readable network interfaces names, human-readable device names, and byte array representations of exchanged data. These design principles make use of the separation of networking protocols mechanisms and abstraction that are discussed in (11).

b. Dynamic plugging of network interfaces

In Figure 3, we show the Media Abstraction Framework, MAF, in run-time. At the top of the framework, the Media Abstraction Manager, MAM, manages a number of Media-type handlers, which each represents a type of network interface and protocol. In turn, a media type handler manages a set of MAO objects for the enabled network interfaces on the PalCom device.

A user shall be able to enable or disable a network interface on a PalCom device to configure which networks the device can use and which devices it can communicate with. Since a network interface is represented by a MAO instance of its networking media type, then the framework must support the plugging and unplugging of these MAO instances into the PalCom middleware, which can be done on both user request and the detection of the availability of network interfaces.

In section 7.1 of the implementation details, we introduce the MAOPlug class that enables the plugging and the unplugging of MAO instances to the framework while handling errors and exceptions on starting and stopping these MAOs.

c. Fault tolerance

As we discussed above, the Media Abstraction Framework, MAF, enables the implementation and integration of network interfaces to the PalCom middleware. As a result the framework needs to be able to survive erroneous implementations of Media Abstraction Objects, MAOs that abstract network interfaces.

We detail the different activities of the MAF in section 7.1, of the implementation details. These activities include adding a media type to the system, plugging and managing a network interface to the frame work, and the managing and using of a plugged network interface.

d. Simplified and robust concurrency

Concurrency is a main aspect of the design and implementation of middleware like PalCom. We need to specify a simple and verifiable concurrency model that is used to implement the system. By clarifying such model to the developers of Media Abstraction Objects, MAOs, we help them to develop MAOs that are concurrency-consistent with the rest of the framework.

In Figure 3, we define the concurrency model in two opposite directions along the hierarchy levels of the Media Abstraction Framework. In particular, a higher level object can synchronize around a lower level object whenever it calls one of its methods. Oppositely, a lower level object can only pass a message or a notification as an asynchronous event.

6.2 Design Principles of the Device Discovery Mechanism

As discussed in the problem statement in section 4, PalCom is required to handle device discovery and undiscovery in a timely manner in order to support applications in fields like healthcare and robotics. We design a device discovery mechanism that enables a set of interconnected PalCom devices to discover the existence of each other. In initial PalCom designs (5) (1), only periodic heart-beats are used for both device discovery within the boundaries of local networks and across interconnected networks. This resulted in flooding the network with heart-beat messages, which is not a scalable

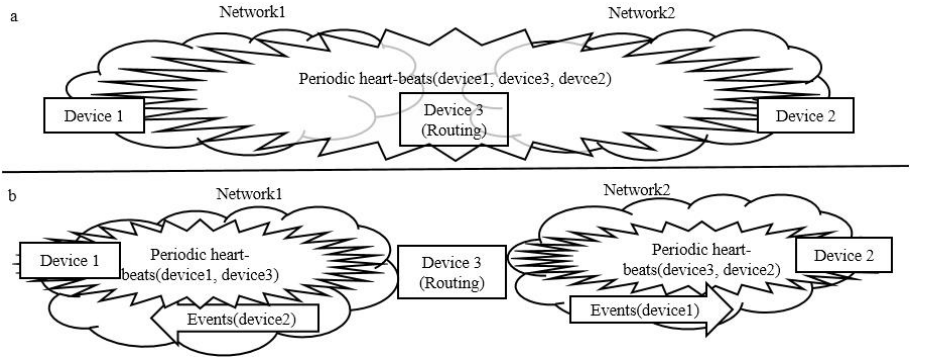

Figure 4 (a) cross-networks periodic heart-beats (b) local-networks heart-beating and cross-networks events.

solution. Such a solution is shown by Figure 4.

Alternatively, we design a device discovery mechanism that aims to support scalability by eliminating the use of cross-networks periodic heart-beats and using event-based device discovery events among interconnected devices. The main idea behind this approach is shown in Figure 4. We design this mechanism to realize two design principles that make use of device discovery to support service discovery and data communication. We specify these design principles as follows:

a. Maintaining an overlay network of interconnected PalCom devices

As we will explain in details in section 7.2, a PalCom node that runs our proposed device discovery mechanism maintains information about the discovered devices in a routing table. The collective view of these routing tables on the set of the interconnected PalCom devices forms an overlay network of these devices. This provides a communication substrate that can be used by upper layers of PalCom.

b. Utilizing the overlay routing information among devices for services discovery and data messaging

The above mentioned overlay network among PalCom devices can be utilized to enable them to exchange their service descriptions and data. Thus, we focus on providing an efficient mechanism of device discovery, which we explain its details in section 7.2.

6.3 Design Principles of Supporting Distance Vector Routing in PalCom

As discussed in the problem statement in section 4, PalCom devices need to find the lowest cost routes via their network to forward data among them. To support distance vector routing in PalCom, we may need to employ a cross-layer approach where different modifications at different layers of the PalCom stack are done. In particular, we need to enable exchanging route cost updates in the messages that are sent via the Media Abstraction Framework, MAF. Also, we need to enable the device discovery mechanism to make use of the received route cost updates to maintain the routing table on a PalCom device. In turn, different components of the routing layer, where device discovery is implemented, need to be updated to support distance vector routing. We make these updates and modifications according to the following design principles:

a. Discovery state consistency among system components

As we mentioned above, different components of the PalCom stack may need to be updated to support distance vector routing. It is important to ensure that these updates do not disturb the consistency among these components to reflect the state of discovered devices, based on the latest received heart-beat messages and discovery notifications.

As we explain in section 7.3, we ensure the satisfaction of this design principle by localizing the implementation of the logic that supports distance vector routing to the boundaries of a single component of the PalCom stack while implementing very small modifications to other components.

b. Separation of concerns among system components

In spite that we need to ensure state consistency among different PalCom stack components, which support distance vector routing in a cross-layer approach, we still need to keep the concerns of these components well separated. For example, components that function as data containers need to be clearly separated from components that implement device discovery logic.

c. Minimizing route dynamics on route cost updates

When a PalCom node receives a route cost update, e.g. via a notification message, it needs to update its routing table and notify its neighbor nodes about these route cost updates. From the perspective of the neighbor nodes of a PalCom node, the interfaces of that node work as introducers, i.e. receivers of route discovery notifications, or

advertisers, i.e. senders of route discovery notifications, of routes.

We need to ensure that the changes of the roles of the network interfaces, of a PalCom node, on route cost updates is kept to below a defined upper bound so that the changes to the advertised routes is kept as small as possible.

6.4 Design Principles of Synchronizing Device Discovery Information on Loss of Update Message over Unreliable Channels

As discussed in the problem statement in section 4, a PalCom device needs to overcome the possible loss of device discovery and undiscovery notifications over unreliable channels. As shown in Figure 5, on the drop of a once-sent discovery update message over an unreliable channel, the exchanged views of the PalCom devices world between two neighbor nodes becomes out-of-sync. We design a synchronization algorithm to recover from such a case. This algorithm is a reliability feature that we add to PalCom (20) (19) (21). We do not use acknowledge based discovery and undiscovery notifications because this will require an overhead of maintaining an

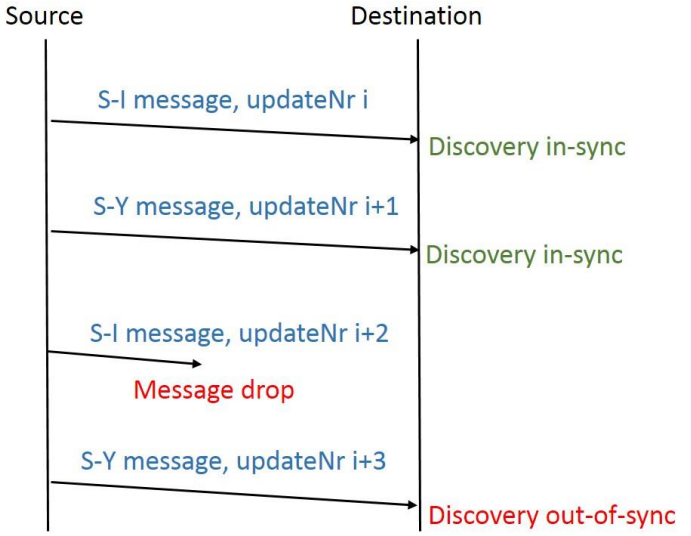

Figure 5 the drop of a discovery update message resulting in an out-of-sync situation

acknowledgement session for every sent notification.

From the perspective of design principles, we add this feature to PalCom as built-in architecture feature that enables each PalCom node to detect by itself that it is out-of-sync with a specific neighbor node. Then, that node shall explicitly ask for synchronization updates from its neighbor node. In particular, the synchronization destination drives the synchronization process. In this way our approach preserves the peer-to-peer nature of the PalCom ad-hoc network and does not use any form of central resources for synchronizing discovery information.

Moreover, both the out-of-sync detectability and the synchronization algorithm reliability are built on top of the periodic heartbeats that are exchanged within local networks. Those periodic heartbeats, as well as once sent notifications, will carry update numbers that can be used by the destination nodes to detect that they are out-of-sync. Also, on the loss of synchronization request/response messages, periodic heartbeats can be used to resume the synchronization process itself.

7 Design Details

In the system design of the media abstraction framework, the device discovery mechanism, and the support of distance vector routing, we follow the above mentioned design principles in order to achieve the desired system properties of functionality, fault tolerance, reliability and maintainability.

7.1 Design Details of the Media Abstraction Framework

As shown in Figure 6, the media abstraction framework itself is defined by the package, media_abstraction.framework. It defines four abstract classes that are required to be implemented for each network media type that we need to interface to PalCom. We call the result of such interfaces a Media Abstraction Object, MAO. The first of those classes is the AbstractMediaAbstractionObject, which mainly abstracts the sending, broadcasting, and receiving mechanisms of the networking protocol of the abstracted network interface.

The second abstract class is the AbstractPhysicalInterfacesEnumerator that can be used to implement an enumerator of network interfaces of the abstracted networking media-type on the local machine. Such enumerators may be implemented to periodically use system level calls to enumerate the

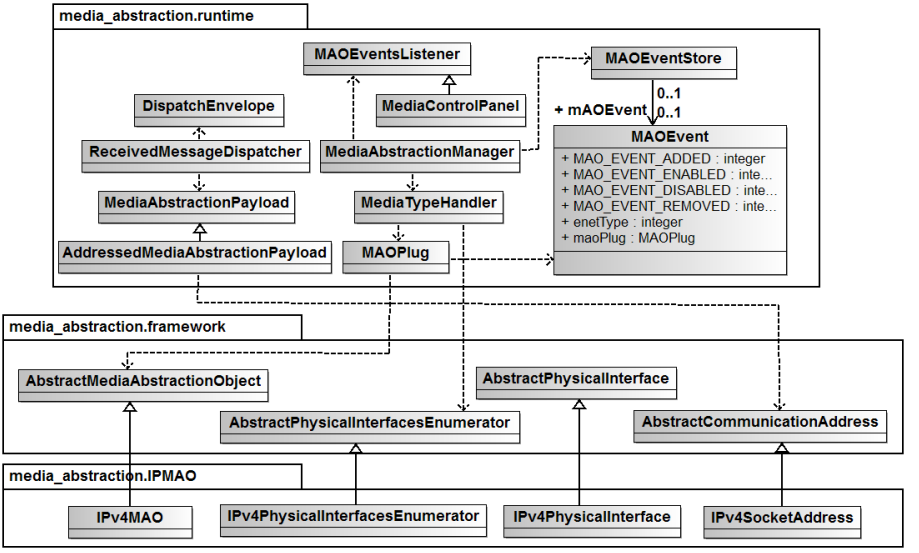

Figure 6 class diagram of the media abstraction framework.

interfaces on the local machine or it may be GUI-interfaced so that a user configures the networking interfaces that can be used by PalCom for the abstracted media type.

The third class of the media abstraction framework is the AbstractPhysicalInterface which simply represents a network interface on the local machine to upper PalCom layers. In particular, an implementation of that class must provide a human-readable representation of names of the abstracted network interfaces.

Similarly, the fourth abstract class of the framework, the AbstractCommunicationAddress, needs to provide a human-readable representation to networking addresses that are used by the the network protocol of the abstracted media type.

In Figure 6, we illustrate an example use of the media abstract framework to implement IPv4 in PalCom, namely the IPMAO. The IPv4MAO implements the AbstractMediaAbstractionObject and it provides send, broadcast and received methods that enable the corresponding operations on an IPv4 interface on the local host.

Also we see that IPv4PhysicalInterfacesEnumerator which implements the AbstractPhysicalInterfacesEnumerator by an implementation that periodically uses the operating systems specific calls to enumerate the list of network interfaces on the local host. This enumerated list of interfaces is

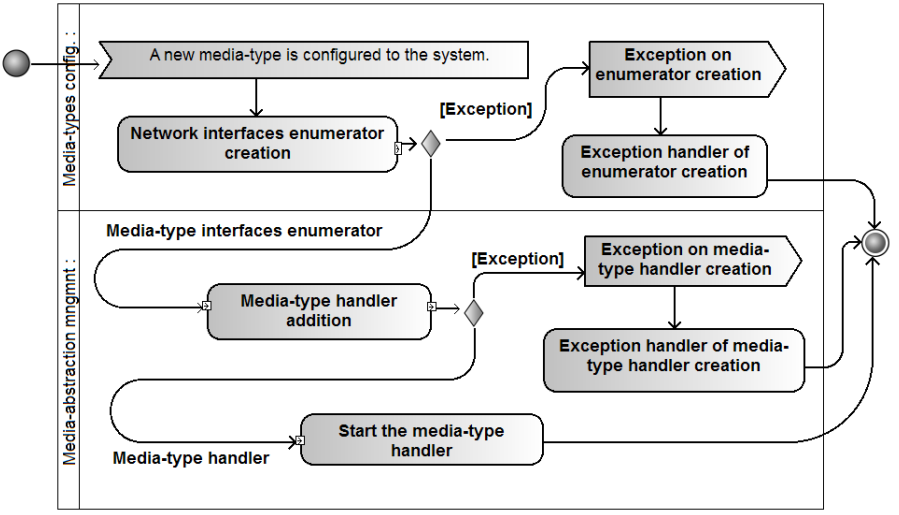

Figure 7 adding a media-type to the media-abstraction framework.

used by the media_abstraction.runtime as we will explain later in this section.

The third part of the example IPv4MAO is the IPv4PhysicalInterface, which implements the AbstractPhysicalInterface abstract class of the media abstraction framework. An IPv4 address is represented to upper PalCom layers in a human-readable format to represent routes via which devices are discovered and messages to and from them flow.

The last of the example IPv4MAO classes is the IPv4SocketAddress that implements the AbstractCommuincationAddress abstract class of the media abstraction framework. This class provides human-readable representation of IPv4 addresses that may represent the source, the destination, or the next hop of a PalCom message.

As illustrated in Figure 6, the media-types specific implementations of the media abstraction framework are integrated to the system by the media abstraction runt-time. While configuring the communication facilities when starting PalCom, we need to add an instance of the class media_abstraction.runtime.MediaTypeHandler to integrate the four classes that implement a media-type that we need to use in the system.

In particular, as we show in Figure 7, when adding a MediaTypeHandler instance of an integrated media-type, an instance of the implementation of the AbstractNetworkInterfacesEnumerator is created and passed to the MediaTypeHandler instance so that it can be used to enumerate the

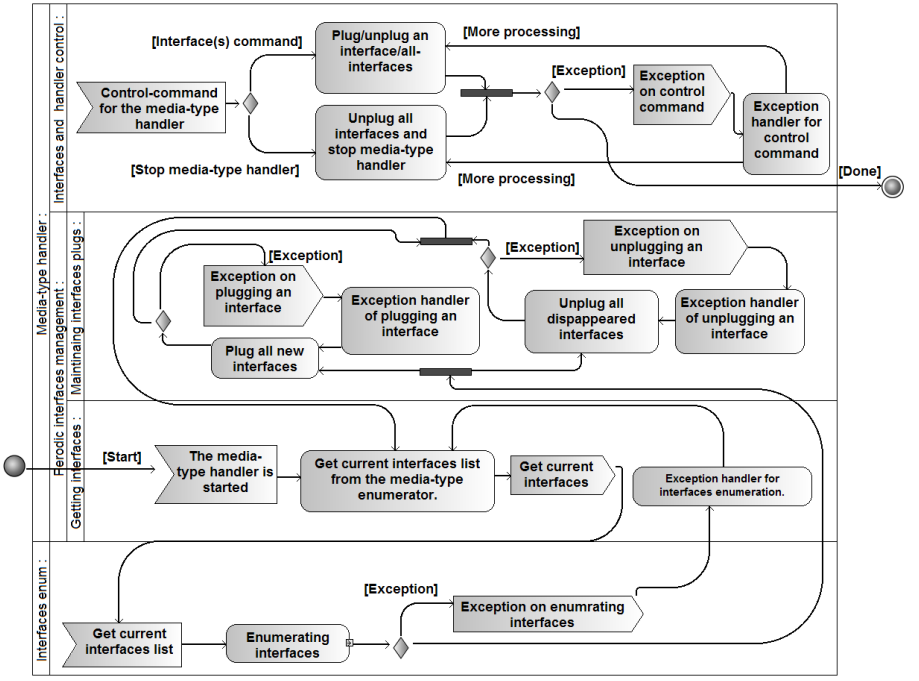

Figure 8 managing network interfaces of a media-type in the media abstraction framework.

interfaces on the system, and managing their usage as shown next in Figure 8.

As shown in Figure 8, once a MediaTypeHandler of an abstracted media type is added and started, it begins to periodically check the updates of the network interfaces list by using the media-type specific implementation of the AbstractPhysicalInterfacesEnumerator. For a newly detected network interface, according to the enumerated interfaces list, the MediaTypeHandler creates an instance of the MAOPlug class, illustrated in Figure 6, which represents this interface to the media abstraction run-time. Via the MAOPlug wrapper of a detected network interface, the MediaControlPanel GUI can process user requests for enabling or disabling the use that network interface.

On the creation of a MAOPlug wrapper for a newly detected network interface, the MediaTypeHandler directly turns its state to enabled, i.e. plugged. In turn, the MediaAbstractionManager, shown in Figure 6, distribute the MAO_Event_ADDED to interested component, e.g. the device discovery

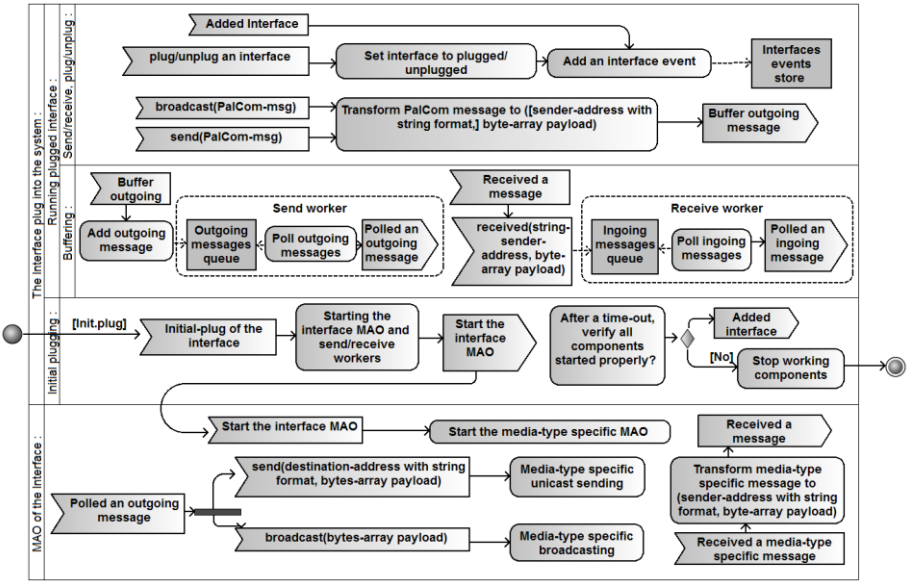

Figure 9 an overview of the functionality of a plugged network interface.

mechanism. On the other hand, when a network interface is undetected, i.e. disappears from the list of enumerated network interfaces of the media type, the MediaTypeHandler disables and removes the MAOPlug wrapper of that interface. In turn the MediaAbstractionManager distributes the MAO_EVENT_REMOVED to interested components, which are registered as listeners to such events. On disabling and enabling MAOPlugs in response to user requests from the MeidaControlPanel, as we explained above, the MediaAbstractionManager generates necessary events of MAO_EVENT_ENABLED or MAO_EVENT_DISABLED to registered listeners.

As summarized in Figure 9 and detailed in Figure 10, for an enabled MAOPlug of a network interface, there are two queues where messages are buffered, namely the ingoing messages queue and the outgoing message queue. Such buffering decouples the activity of the MAOPlug from the rest of the PalCom system and from the wrapped network interface.

On the recipient of a message, the network address of its source device is transformed to a human-readable format and its content is transformed into a byte-array. Then, this new representation of the received message is added to the ingoing messages queue. In turn, the receive worker polls such a

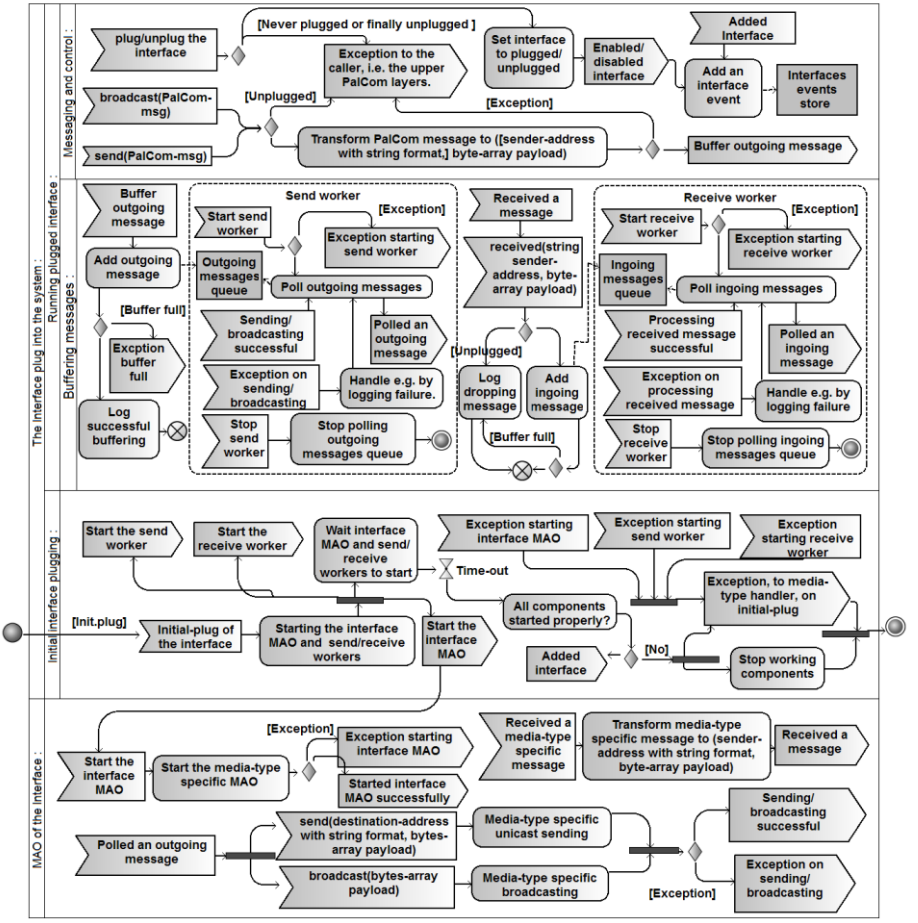

Figure 10 a detailed view of the functionality of a plugged network interface.

message from its queue buffer and pass it to the RecievedMessageDispatcher, shown in Figure 6, which parses it and distributes it to the proper component, e.g. a signaling message processor, or a data message processor.

On the request from upper PalCom layers to send or broadcast a message, a MAOPlug transforms the internal PalCom message object to the generic presentation of a byte-array and attach it with the human-readable format of the specific destination address, if any. Then, the message is put into the outgoing message queue of the MAOPlug for the outbound interface. In turn, the send worker of the MAOPlug polls the queued message and calls

the media specific send or broadcast operations, depending on the existence of an associated destination network interface. Accordingly, the media-type specific implementation of the AbstractMediaAbstractionObject transforms the message and the destination address, whether a broadcast or a unicast address, into a media specific format before sending or broadcasting via the outbound network interface.

7.2 Design Details of the Device Discovery Mechanism

As discussed in section 6.2, we aim to provide cross-network device discovery and undiscovery in PalCom without flooding local networks with heartbeat messages from their neighbor local networks. For the device discovery mechanism, we discriminate two types of routes to a device, namely a local route and remote/routing route. A local route to a device is a single hop route within the limits of a local network on which that device has a network interface. On a local node, a local route to a discovered device is defined with the pair:

(*The name of the network-interface on the local network that connects the local node and the discovered device,*

The networking address of the discovered node on the mentioned local network in the first element of this pair).

In contrast, a remote/routing route to a device is defined as a multiple-hop route that may go via at least one router node that span across the boundaries of multiple networks. On a local node, a remote/router route to a discovered device is defined with the triple:

(*The name of the network-interface on the local network that connects the local node to the next-hop router node that has introduced the discovered device*,

The networking address of the previously mentioned next-hop router node,

An integer valued short-ID that was assigned by the previously mentioned next-hop router node to the discovered node)

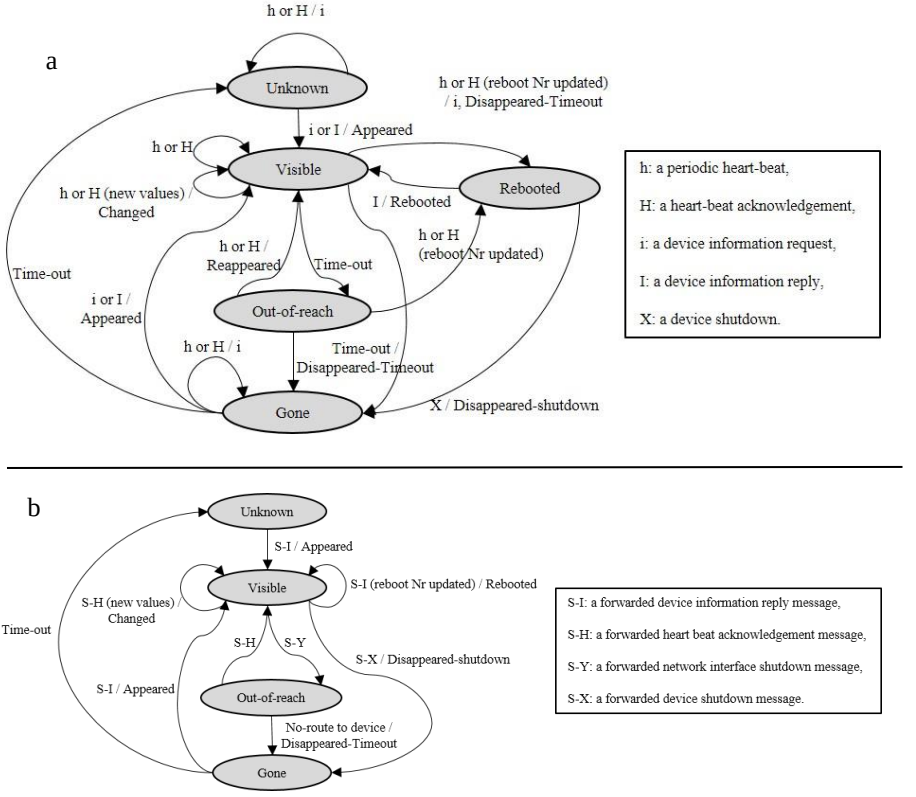

Figure 11 heart-beat based local-network device discovery vs. event based cross networks device discovery.

In Figure 11, we define the logic that we use to maintain the discovery state of a local route to a device on the local network. At its initial state, a local route is considered to be unknown. On a PalCom node, once a periodic heart-beat message is received, noted as “h/H”, or an information-request message, noted as “i”, is received via an unknown local-route to a device, this route switches to the visible state. Accordingly, the routing table is maintained with the information of the newly discovered route, and properly with the information of the newly discovered device if the device was not previously known via any other route. In all cases, a sequence of at least one information-request message, i.e. “i”, and one information-reply message, node as “I”, is exchanged to get the latest updates of the device parameters, which include the device reboot-number, its change-number, and its status-

information.

Once a local-route is discovered, its status can change between visible, rebooted, out-of-reach, and gone. A route state changes depending on the recipient of different types of discovery messages or when the local-route times-out in a specific state. The types of discovery messages and the configurable time-outs and their processing is illustrated in Figure 11.

From the perspective of the device discovery state on a local node, a device is declared appeared to upper PalCom layers, e.g. the service layer, once it is discovered via the first route. On the other hand, a device is declared disappeared once to upper PalCom layers when the only remaining route to it times-out in the gone state. In other than these two cases, a device remains as appeared as long as there is at least one route to that device that is currently in the visible state.

In Figure 11, we define the logic that we use to maintain the discovery state of a remote/routing route to a device on a remote/non-local network. At its initial state, a remote/routing route is considered unknown. On a PalCom node, once a forwarded information-replay message, noted as “S-I”, is received via an unknown remote/routing route to a device, this route is switched to the visible state. Such an S-I message is sent by a node that has a network interface on one of the local networks on which the receiver node has network interfaces.

From the perspective of the S-I receiver node, the source node of the S-I message is referred to as the introducer router node of the just discovered remote/routing route. In turn, the discovery of a remote/routing node is accepted only if the local-route to its introducer router node is in the visible state. Otherwise, an S-I message that is received via an un-reachable route is considered an error, and it is just dropped.

As shown in Figure 11, after the discovery of a remote/routing route, its state keeps changing between the values of visible, out-of-reach and gone in response to the recipient of different types of discovery messages or when the route times-out in one of the states, which is not possible when a remote/routing route is in the visible state. In particular, since the appearance of a remote/routing route is event based, as opposite to periodic heart-beat based, we do not time-out such route in the visible state. However, the disappearance of the local-route to the introducer router of a remote/routing route results in the disappearance of that route too, which we call route state alignment.

As we mentioned in the problem statement in section 4, one problem of depending on event based discovery state is the possibility of losing an event message, e.g. an S-I message, due to a temporary network failure. In section

6.4, we discussed the design principles of a synchronization algorithm to address this problem. We discuss the design details of this algorithm in section 7.4.

Another important aspect of maintaining an event based discovery state, of a remote/routing route, is the mechanism that a router node uses to forward discovery events about discovered routes, regardless of their types, to its neighbor nodes on the local networks on which it has networks interfaces. The most important consideration of such mechanism is to avoid looping forwarded discovery and undiscovery notifications. We employ a mechanism that we call Discovery Forwarding Flows, DFF. It is based on conventional internet protocols that address the looping problems (25).

In particular, on a PalCom node, for any available/visible route a DFF is defined as the pair:

(The introducer network-interface on the local node which the route was discovered via,

A set of network-interfaces on the local node through which discovery messages, about the route, are or to-be sent)

From the perspective of a discovered device, assuming that we sort its discovered routes according to the temporal order of their discovery, then at any moment in time a router node maintains only two DFF instances of that discovered device, which we define as follows:

DFF1 = (The network-interface on the local node via which the first route to that device was discovered,

The rest of network interfaces on the local node)

DFF2 = (The network-interface on the local node via which the second route to that device was discovered,

The network-interface on the local node via which the first route to that device was discovered)

Figure 12 shows an example set of DFFs. Of course, it is possible to update these definitions according to the criterion that we use to sort the discovered routes to a known device. E.g. we update these definition to support distance vector routing in PalCom, as we see in section 7.3.

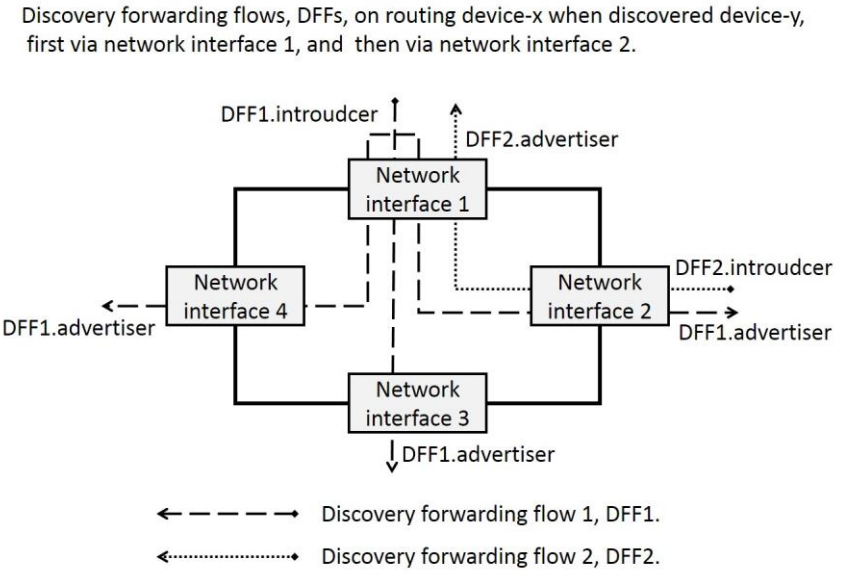

Figure 12 the discovery forwarding flows on device-x for the discovery and advertisement of device-y.

The advantage of using the DFFs to organize the forwarding of discovery messages is to avoid having loops by stopping the forwarding of a discovery event message about the availability of a specific device after all the interfaces of the local PalCom node, performing the role of a router, are covered by the two maintained DFFs for that device.

Moreover, it is possible that a router node discovers one or more routes to a discovered device via network interfaces other than the introducer interfaces of DFF1 and DFF2, which the router node maintains for that discovered device. In this case, the router maintains a temporally ordered list of these discovered introducer interfaces, from the oldest discovered to the newest discovered. We refer to this list and the *introducers-list*.

Once the router node receives an undiscovery notification about the discovered device via the introducer interface of DFF1, it moves the introducer of DFF2 to become the introducer of DFF1 and the first element in the *introducers-list* as to become the introducer of DFF2. To complete this change, the router node rearranges the advertiser interfaces accordingly and sends the necessary discovery and undiscovery notifications to the neighbor nodes. In case of no remaining introducers are known to the discovered device, the router node declares the device as undiscovered.

7.3 Design Details of Supporting Distance Vector Routing in PalCom

In section 6.3, we introduced the design principles of our solution to enable PalCom nodes to find the lowest cost routes for forwarding data messages among them. In the context of supporting distance vector routing in PalCom, we define the route-cost from a device to another as the number of hops between these two devices. Of course, the route-cost can be defined in terms of other criteria that may include link-quality attributes or even user preferences. However, for the simplicity, we describe our solution with the number of hops as the only criterion of a route-cost.

The first step of supporting distance vector routing is to extend the discovery notification messages, i.e. S-I and S-H messages in Figure 11, to convey route-cost updates. In PalCom, we construct and parse messages as a sequence of nodes, where each node represent a specific component of a PalCom message. We note a message as a dash separated sequence of nodes. Thus, in the previously mentioned S-I symbol, the S-node represents a forwarded message from a router node while the I-node represent a discovery-information-replay message with a full set of device parameters, which include its globally unique device-ID, its reboot number, its change number, and its status information that indicate its availability.

To convey route-cost updates in a PalCom discovery message, we insert a new C-node between the two nodes of the S-I and the S-H message. The C-node contains one field for the routing cost update. Such minimal update preserves the format and the processing of the discovery messages. Also, it does not change the format of any already existing PalCom message nodes.

The second step of supporting distance vector routing is to redefine the discovery forwarding flows, DFFs, of a known device in terms of route-cost updates of the discovered route instead of the temporal order of their discovery, as discussed in section 7.2. In particular, we maintain the two DFFs of a discovered device so that the network-interface of its lowest cost route is the introducer of DFF1 while the network-interface of its second lowest cost route is the introducer of DFF2. And the advertisers are updated accordingly. The discovery and undiscovery of more than two routes to a device is handled in the same way as we discussed above at the end of section 7.2.

7.4 Design Details of the Synchronization Algorithm of Device Discovery Information on Loss of Update Message over Unreliable Channels

In the problem definition, in section 4, we discussed that PalCom nodes need to overcome the possible loss of discovery and undiscovery notifications over unreliable channels. We introduced the design principles of a synchronization algorithm that addresses this problem in section 6.4. Our synchronization algorithm for device discovery in PalCom embedded update number fields into heartbeat and device appearance/disappearance update messages. A destination PalCom node can detect whether it is out-of-sync from a neighbor source node by comparing the latest update-number that it received from that node with the update-number in a received heartbeat or discovery update message. Then the destination node can initiate the synchronization process if required.

However, in the original device discovery algorithm, discussed above in section 7.2, a PalCom node can only remember the currently discovered

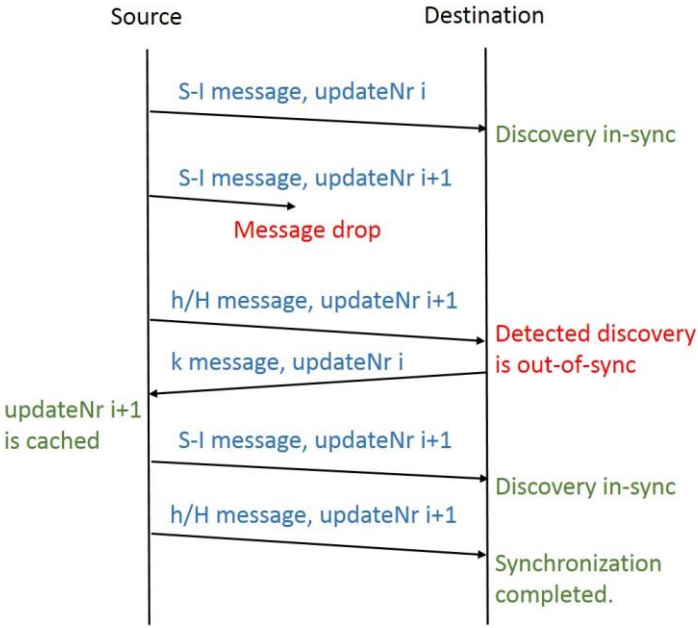

Figure 13 uninterrupted cached synchronization, i.e. no update request/response messages are lost during the synchronization process.

device. As a result, it will be required to transfer the entire list of discovered devices from the source to destination in order to verify that the destination does not have outdated entries for a device that is not known any more by the source. This is an expensive process that we need to avoid.

In order to minimize full synchronization rounds, a PalCom source node uses a limited size cache to remember full history of add and removed entries within a specific time period. An uninterrupted cached synchronization is shown in Figure 13. However, if the requested discovery information is beyond the cached continuous history, a full synchronization is performed as shown in Figure 14.

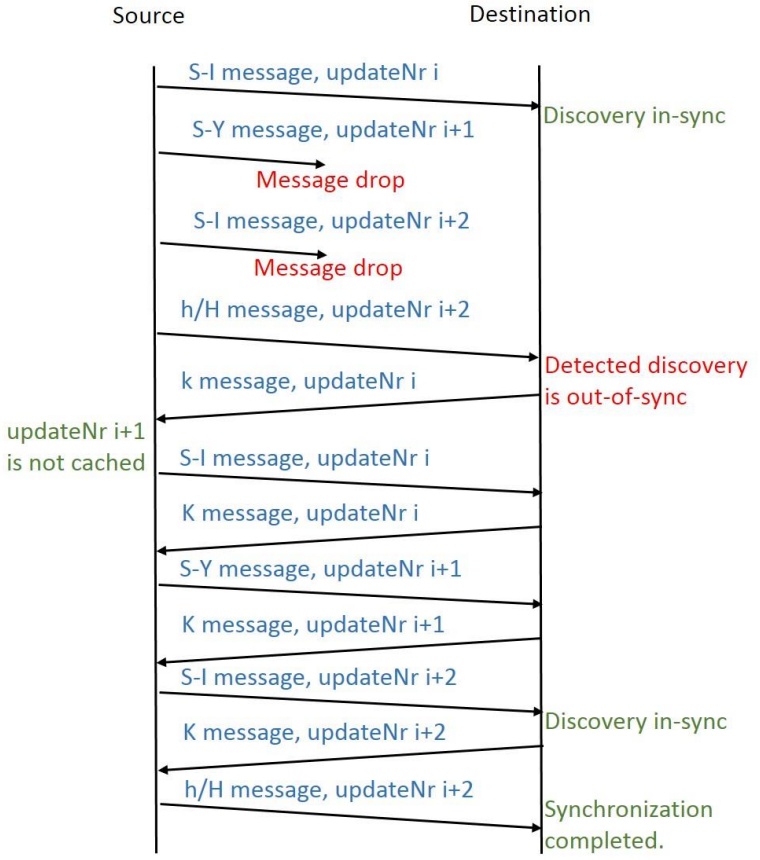

Figure 14 uninterrupted full synchronization, i.e. no update request/response messages are lost during the synchronization process.

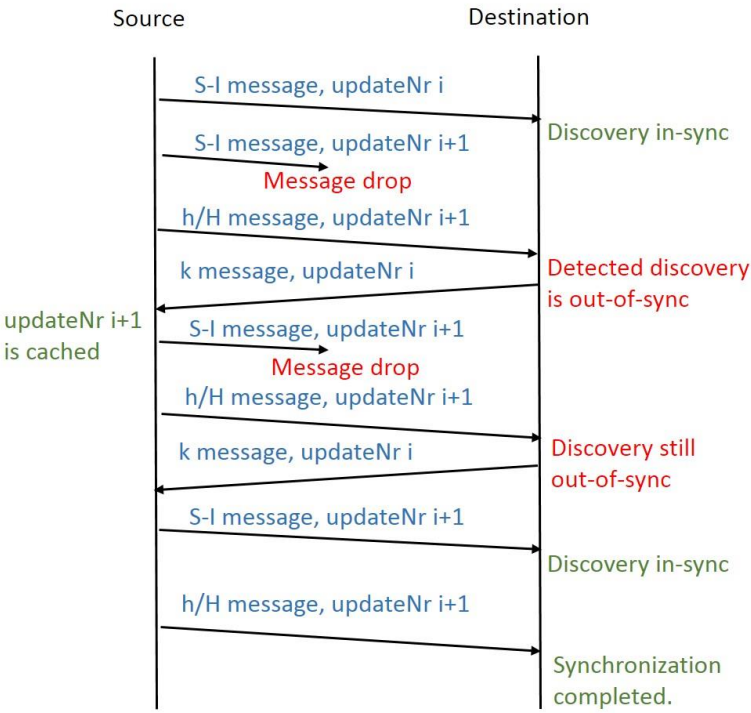

Figure 15 interrupted cached synchronization, i.e. some update request/response messages are lost during the synchronization process.

In both cached synchronization and full synchronization, there is a possibility of losing synchronization request/response message over unreliable channels. Such an interrupted synchronization is resumed starting from periodic heartbeat messages as shown in Figure 15.

8 Evaluation

Depending on the requirements and the designed features of the different system components, i.e. the media abstraction framework, the device discovery mechanism, and the support of distance vector routing, we design and execute different test approaches to evaluate their functionality and performance.

8.1 Evaluation of the Media Abstraction Framework

For the media abstraction framework, we evaluated two aspects, namely its usability, for integrating networking interfaces into PalCom, and its time overhead, when sending and receiving PalCom messages. For the framework usability, we used the framework to develop and integrate an IPv4 media abstraction object, which we called IPv4MAO. Also, other members of our research group used the framework to implement an IP tunneling MAO and an Android inter-process communication MAO. All implementations proved the flexibility and usability of the framework.

In code snippet 1, we show that it is a very small effort to integrate the entire framework into PalCom. In particular, on the start of the communication manager component of PalCom, it is only required to create an object of the RecievedMessageDispatcher, shown in Figure 6, and an object from the MediaAbstractionManager, also shown in Figure 6, and to connect both of them to each other and to the currently used RoutingLayer.

In code snippet 2, we can see that it take only one line to start the MediaAbstractionManager, of our proposed media abstraction framework, which in turn takes over the responsibility of starting and managing the media abstraction objects, MAOs that are implemented using the framework.

In code snippet 3, we show that adding the implementation of a new

```
// creating the received message dispatcher.
ReceivedMessageDispatcher receivedMessageDispatcher =
                new ReceivedMessageDispatcher();
((se.lth.cs.palcom.routing.RoutingLayer)routinglayer).
        setReceivedMessageDispatcher(
                receivedMessageDispatcher);

// creating and setting the media abstraction manager.
mediaAbstractionManager =
        new MediaAbstractionManager(
                receivedMessageDispatcher, true);
((se.lth.cs.palcom.routing.RoutingLayer)routinglayer).
        setMediaAbstractionManager(mediaAbstractionManager);
mediaAbstractionManager.addMAOEventsListener(
                (MAOEventsListener)routinglayer);
```

Code snippet 1: The integration of the Media Abstraction Manager to the Communication Manager of the PalCom.

```
success = mediaAbstractionManager.start();
if (success) addWP4IPv4MAO();
```

Code snippet 2: In the PalCom communication manager, starting of the Media Abstraction Manager and adding the IPv4 MAO.

networking technology specific media abstraction object, MAO, to the system is done using one line that results in creating and adding a MediaTypeHandler of the MAO. In turn, the MediaTypeHandler manages the creation and maintenance of different components of the MAO, i.e. its NetworkingInterfacesEnumerator and it MediaAbstractionObject itself, as discussed in section 7.1. In particular, the example in code snippet 4 adds an IPv4MediaType to the system.

To evaluate the message processing time-overhead of the designed media abstraction framework, we designed a test program that uses a dummy layer above the media abstraction layer that can create new outbound time-stamped test messages and echo inbound test messages as they are. The evaluation test uses one instance of the program to send out a sequence of test messages, i.e. the ping side, to another instance of the test program that only echoes the received test message, i.e. the pong side. The test setup is shown in Figure 16. When the ping-side creates a ping message, it embeds a trace-ID to that message so that it can identify that particular message when it is echoed by the pong-side.

On the ping-side, before the ping message is passed to the media abstraction framework, for sending it to the pong-side, a starting time-stamp is associated with the embedded trace-ID of the message. Also, a specified time-out is maintained to stop waiting for an echo of that particular message and to starting attempting to send another one.

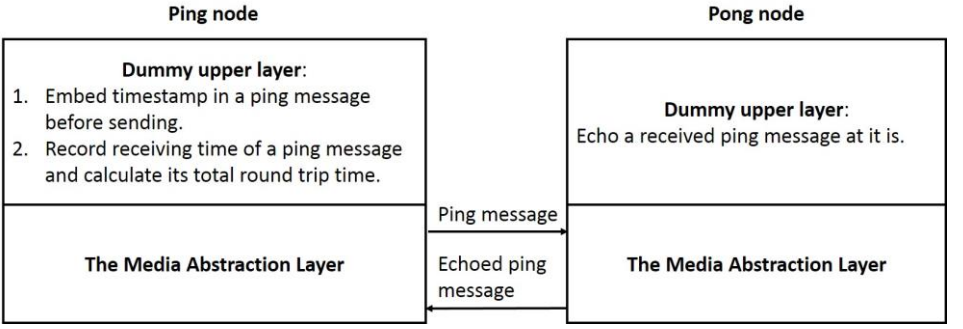

Figure 16 the test setup of the processing time overhead of the media abstraction layer using time stamped ping messages.

On the successful receiving of a ping message, the pong-side echoes the message as it is, with it embedded trace-ID. When the ping-side receives the echoed ping message, the media abstraction framework processes and hands it to the upper dummy layer where the ping-side records the time-stamp of this event and calculates its difference from the starting time stamp that is associated with the trace-ID of the received ping message. We do this for a sequence of such ping messages.

In Figure 17, we show the results of measuring the time overhead of using the media abstraction framework. In the context of these measurements, the round trip time of a message is the time it takes from the ping side to the pong side and back again to the ping side. In Figure 17, the circle-noted series contains more than 180 round trip times for our test messages. In the same figure, the x-noted series contains the same number of round trip times for ICMP ping messages that we send using the hrPING tool (26) to control their length to be the same as that of our test messages, i.e. 14 bytes a message.

The triangle-noted series in Figure 17 shows the result of subtracting the times in the x-noted series from their corresponding times in the circle-noted series. As a result, the values in the triangle-noted series represent the total time overhead where the media abstraction framework processed the test

```
mediaAbstractionManager.addMediaType(
                    new IPv4MediaType(deviceId));
```

Code snippet 3: In PalCom communication manager, adding a MAO to the Media Abstraction Manager is done by creating an instance of the MediaType class of that MAO.

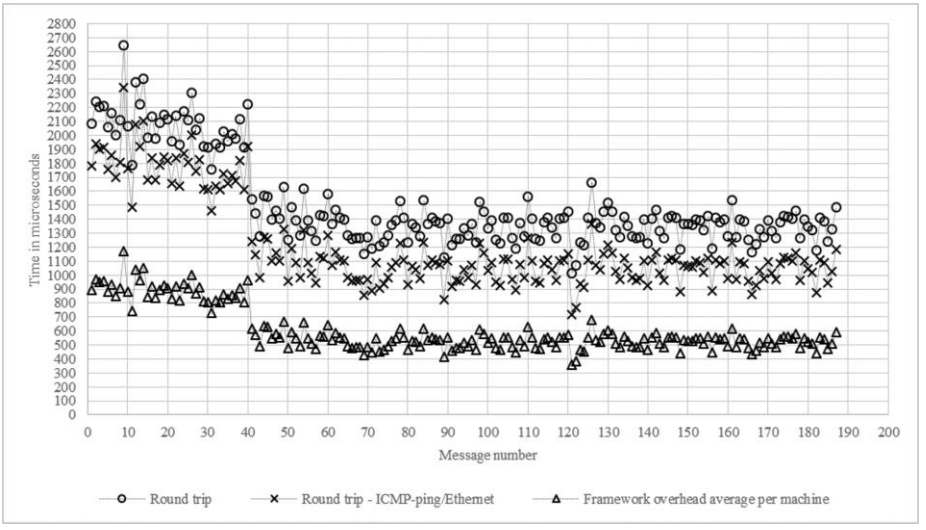

Figure 17 the time-overhead of processing a PalCom-ping-message is around 250 micro-seconds in each direction.

messages during their round trip. As we can see in Figure 17, the average round trip time overhead is about 500 micro-seconds. This value gives an average of around 250 micro-seconds of time overhead for either directions of the round trip of a test message. Note that, as shown in Figure 17, the first 40 messages seem to consume more than average times for both test messages and ICMP ping messages.

8.2 Evaluation of the Device Discovery Mechanism

The device discovery mechanism that we designed for PalCom has a number of key features that can be evaluated. One key feature is that it can timely discover all possible routes to a connected device. Another feature that we can test is that no discovery messages loop forever via different links. However, we test these two features as part of testing the support for distance vector routing, which we explained in section 7.3. In this section, we describe our evaluation of the effect of eliminating cross-networks periodic heart-beat messages from the device discovery mechanism. In Figure 18, we illustrate the test setup where a PalCom node functions as a router, connecting two other PalCom nodes. The test is designed in two rounds. In the first round we run, on all the three test machines, a version of PalCom that implements the simplistic approach of forwarding periodic hear-beat messages across the boundaries of local networks. In the second test round,

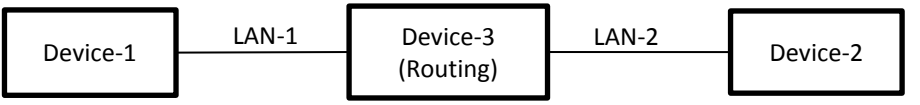

Figure 18 the test setup for evaluating the effect of eliminating cross-networks heart-beat messages from the device discovery mechanism.

on all the three test machines, we run a version of PalCom that implements the proposed device discovery mechanism that eliminates the cross-networks heart-beats.

In both rounds, the three devices discover each other without a problem. However, by recording the traffic of the heart-beat messages on the local network links, we can see that the exchanged number of these messages is reduced to half in the second round, where we used the proposed device discovery mechanism, compared to the first round, where we use the simplistic approach. As a conclusion, the events-based discovery notification provides a noticeable improvement to PalCom by avoiding to flood networks with heart-beat messages from their neighbor networks.

8.3 Evaluation of Supporting Distance Vector Routing in PalCom

For an evaluation of a feature like distance vector routing, we need a network of PalCom devices where different connections among these devices can be taken up and down using a test script. This is possible to do using Virtual PC on a Windows 7 machine.

We use the loopback interface on a Windows 7 machine as a virtual Local Area Network, LAN, among Virtual PCs by installing the Windows Virtual Network driver to the loopback interface itself. Then, by configuring IP addresses to Virtual PCs. Then, we can connect a Virtual PC to that loopback interface as one of its networks. Then by configuring proper network address and mask values we can create local and isolated networks among Virtual PCs. We can verify such network setup using normal ping commands among the connected virtual PCs.

Moreover, from the perspective of software development, it is possible to connect the host Windows 7 to these virtual networks in the same way as configuring normal network interface by adding a proper IP address and mask values to the loopback interface itself. This is useful for debugging and troubleshooting of the algorithms that we implement.

To evaluate our solution for supporting distance vector routing in PalCom,

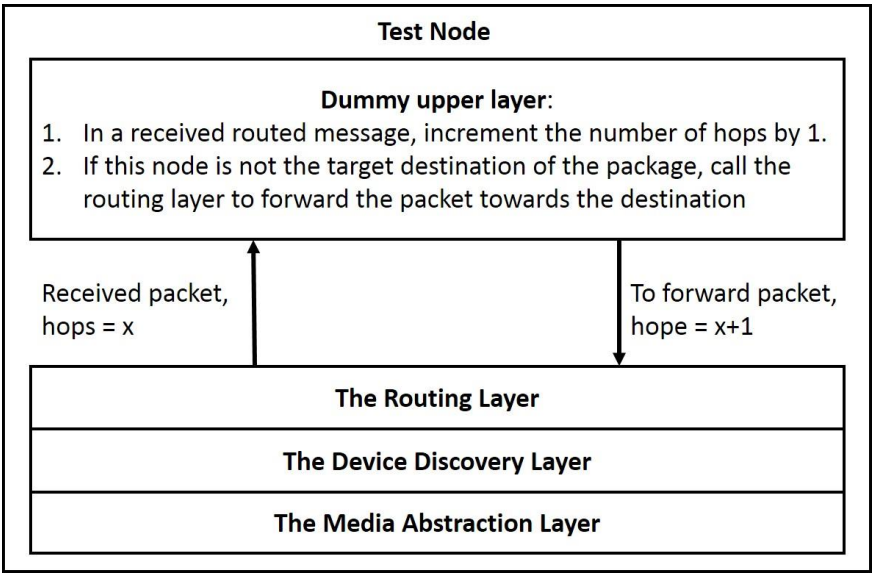

Figure 19 the internals of a test PalCom node in the distance vector routing test.

we define a test scenario where a source node sends a stream of test messages via available routes in a network of discovered PalCom devices. The test evaluates how the implemented distance vector routing over PalCom device discovery responds to changes in the connectivity of the network interfaces of the source node in terms of making correct and timely routing decisions of the test stream.

During the test, the source sends a sequence of numbered test messages to the destination and we record the arrival of these message on the destination side along with the number of hops that these messages have taken. At every node along the route that a test message takes, we increment an embedded counter in that message which represents the number of hops that it has taken until the current node, as shown in Figure 19. On the destination side, we log the sequence number and the number of traversed hops of received test messages.

To evaluate the response to changing network connectivity, we use an automated script to control disabling and enabling of networking interfaces on the source node. In particular, we used a network of Microsoft Virtual PC nodes connected via the above discussed virtual local network interfaces. We automated disabling and enabling the virtual network interfaces using a PowerShell script that uses the Windows Virtual PC COM interface. In Figure

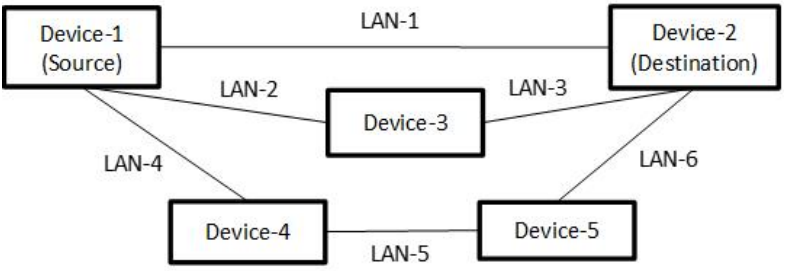

Figure 20 the test setup for evaluating the support of distance vector routing over discovery forwarding flows in PalCom.

21, we show the result of running three test rounds, where a test round is defined from the perspective of the source node, i.e. Device-1 in Figure 20, as:

1. Initially LAN-1, LAN-2, and LAN-4 are enabled.
2. Disable LAN-1 and wait for 15 seconds.
3. Disable LAN-2 and wait for 15 seconds.
4. Disable LAN-4 and wait for 15 seconds.
5. Enable LAN-4 and wait for 15 seconds.
6. Enable LAN-2 and wait for 15 seconds.
7. Enable LAN-1 and wait for 15 seconds.

The test results in Figure 21 show that the implemented distance vector

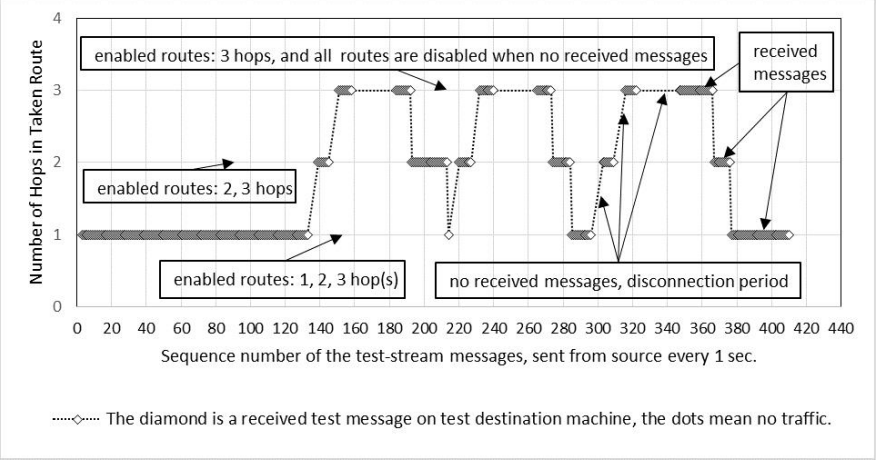

Figure 21 the test results for evaluating the support of distance vector routing over discovery forwarding flows in PalCom.

routing makes correct and timely routing decisions considering the resulting dynamics of device discovery as the network connectivity changes. Note that dotted lines mean no traffic while the source is trying to find a new alternative route.

8.4 Evaluation of the Synchronization Algorithm of Device Discovery Information on Loss of Update Message over Unreliable Channels

To evaluate our synchronization algorithm we used both model checking and simulation based evaluation. For the model checking evaluation we modeled our algorithm as a timed automata model using UPPAAL (22). We verified that our system is correct against a property that makes sure that a destination node receives all updates from the source after the completion of the synchronization process.

Moreover, we used the model in UPPAAL to study the algorithm performance showing that the larger the history cache of the discovery updates and the quicker the out-of-sync situation is detected, the fewer discovery update messages are exchanged among nodes, giving better performance.

On the other hand we used simulated network of virtual PCs, in the same way as shown above is section 8.3, to produce an out-of-sync situation, as shown in Figure 22, to prove the correctness of the algorithm implementation.

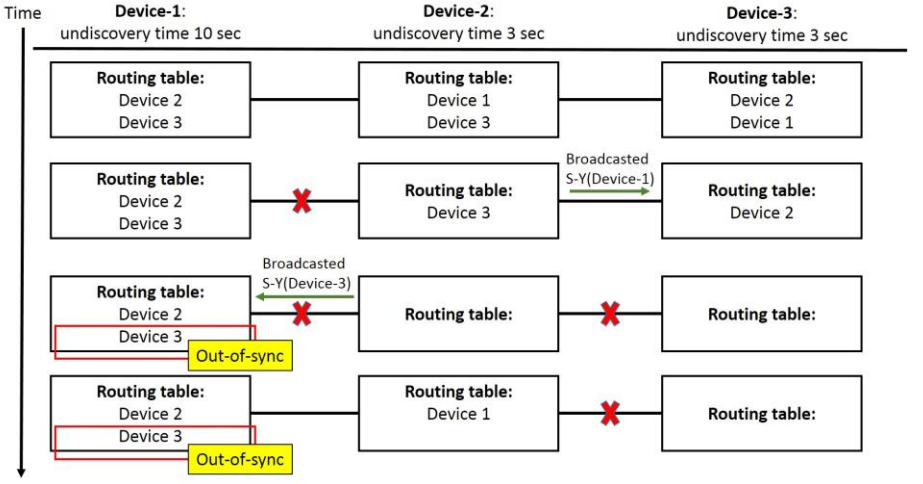

Figure 22 test setup for discovery out-of-sync situation.

And PalCom passed this test successfully.

9 Future Work

An important future work is revise the device discovery mechanism to handle an interesting case of looping discovery events that we detected during regression testing of that mechanism while setting up the test for evaluating the support for distance vector routing. In particular, considering the test setup in Figure 20, the problem appears in the following sequence:

1. When Device-2 discovers Device-3 for the first time, it may forward a discovery event to Device-5 and then to Device-4 and finally to Device-1, which may return that event to Device-2, which drops the event according to the mechanism of the Discovery Forwarding Flow, DFF, which explained in sections 6.2 and 7.2.
2. In case we disable Device-3, then Device-1 may forward a discovery event to Device-2 to notify that it has a remote/routing route to Device-3 via Device-4, which is the same discovery event that was previously dropped by Device-2, as we mentioned above. However, this time Device-2 does not know Device-3 anymore and there is no way in the device discovery mechanism to enable Device-2 to detect that this received discovery was originated from it before to advertised a recently disappeared local route to Device-3.
3. As a result, Device-2 increments the hop-counts in the discovery events and forwards the event via Device-5, Device-4, and Device-1 which returns it to Device-2 again. And since all of these devices increment the hop-count in the forwarded discovery event, Device-2 thinks that this is just a cost-update event from Device-1 about its remote/routing route to Device-3, so Device-2 increments the hop-count in the discovery event and forwards it to Device-4 and the forwarding loop continues for ever.

Till now, we have a simple solution for this problem using the gateway role of nodes like Device-1 and Device-2, with respect to Device-3. But, we may need to optimize this solution using less information in the discovery messages without long string values of gateway device IDs.

Another possible future work is to minimize broadcasts in a local network where no expected device is missing. In particular, we may add the capability

of configuring PalCom nodes to only send heart beat messages to specific devices on the network that we are interested in periodically checking their availability. Such approach shall minimize broadcast traffic in local networks

10 Conclusions

In this dissertation, we presented our work to fulfill our research questions that we listed in section 4. To address the first research question, we added to PalCom a networking media abstraction framework that developers can use to easily develop new media abstraction objects, MAOs for PalCom, which support different networking protocols. Our approach to design that framework was to refer to design principles of networking protocols to identify the main components of a network protocol which we represent in the framework as abstract classes. PalCom developers can implement these abstract classes to develop new MAOs. Moreover, the framework includes a runtime environment that can easily integrate a newly developed MAO to the system while enabling the plugging and unplugging of a specific network interface.

As part of the PalCom development by our research group, the media abstraction framework was successfully used in developing MAOs for IPv4, IP tunneling, Android inter-process communication, and even a virtual networking environment for testing and evaluation of complex scenarios. As part of the evaluation work in this dissertation, the framework time-overhead is found to be around 250 microsecond for a 14 bytes test message in either direction of communication. This overhead is acceptable depending on the application domain and there is a room for performance improvement, e.g. starting with optimizing logging facilities. The work on the media abstraction framework is detailed in this dissertation in sections 6.1, 7.1, and 8.1. It also documented in paper I.

To address our second research question in section 4, we added to PalCom an events based device discovery and undiscovery mechanism over routed networks in a timely manner without flooding the interconnected local networks with broadcast heart beats. The mechanism is composed of two state machines, one for device discovery and undiscovery within local networks and another for device discovery and undiscovery across interconnected networks.

We also refined the design of the mechanism to prevent looping discovery and undiscovery notifications and to align the state of discovered routes to

devices on remote networks to the state of discovered routes to their introducer router devices on local networks. The work on the discovery and undiscovery mechanism is detailed in sections 6.2, 7.2, and 8.2. It is also documented in paper II.

To address our third research question in section 4, we added distance vector routing logic to PalCom as part of the layer on top of the device discovery and undiscovery layer. The main goal was to preserve the modularity and simplicity of both of the two lower layers in the PalCom stack, namely the media abstraction layer and the device discovery and undiscovery layer. The evaluation shows that using this added logic, PalCom is able to choose lower cost routes from the set of available routes to a destination node. It can also respond to dynamically changing availability of routes. The distance vector routing that we added to PalCom is detailed in sections 6.3, 7.3, and 8.3. It is also documented in paper III.

To address our fourth research question in section 4, we refined our device discovery and undiscovery mechanism to include a synchronization mechanism to overcome possible loss of device discovery and undiscovery notifications over unreliable networking channels. The synchronization mechanism is based on tracking update numbers of the exchanged discovery and undiscovery messages. The model based evaluation of the synchronization mechanism shows that it correctly performs its function.

However, the synchronization overhead may be seen as a significant cost in the case of communicating over highly unreliable channels. So practically speaking, the synchronization mechanism provides a minimal recovery utility for occasional loss of discovery and undiscovery notifications. The work on the synchronization mechanism is detailed in section 6.4, 7.4, and 8.4. It is also documented in paper IV.

Moreover, we detailed possible future improvements to PalCom in section 9. These include introducing improved solutions to the problem of looping discovery notifications because of the disappearance of their originator router nodes. Currently we use a simple solution that refers to such originator nodes as gateways and enables PalCom nodes to break loops by dropping a notification message about a specific device that was originated at a specific gateway when it is received for the second time.

Another possible improvement is to provide PalCom with the ability to limit broadcasting heart beat messages by using unicast polling to specific devices on the network that we need to keep track of their availability. This concept is based on the fact that in some application setup we may not need to check the availability of some nodes as frequent as we need for other nodes.

References

1. Svensson Fors, David. *Assemblies of Pervaisve Services, PhD Thesis.* Lund : Dept. of Computer Science, Lund University, 2009.

2. *Pervasive computing: vision and challenges.* Satyanarayanan, M. 8, August 2001, Personal Communications, IEEE, pp. 10-17.

3. *Pervasive computing: a paradigm for the 21st century.* Saha, D. and Mukherjee, A. 36, March 2003, Computer, IEEE, pp. 25-31.

4. *Connecting the physical world with pervasive networks.* Estrin, D, et al. 1, Jan-Mar 2002, Pervasive Computing, IEEE, pp. 59-69.

5. *Ad-hoc Composition of Pervasive Services in the PalCom Architecture.* Svensson Fors, David , et al. London : s.n., 2009. Proceedings of the 2009 international conference on Pervasive services.

6. *Discovery systems in ubiquitous computing.* Edwards, W. K. 5(2), april-june 2006, Pervasive Computing, IEEE, pp. 70 - 77.

7. *Service discovery in pervasive computing environments.* Zhu, Fen, Mutka, Matt W. and Ni, Lionel M. 4(4), Oct.-Dec. 2005, Pervasive Computing, IEEE, pp. 81 - 90.

8. *Toward distributed service discovery in pervasive computing environments.* Chakraborty , D. , et al. 5(2), Feb. 2006, IEEE Transactions on Mobile Computing, pp. 97 - 112.

9. Johnsson, Björn A. *Where PalCom Meets the End-User: Enabling User Interaction with PalCom-based Systems, Licentiate Thesis.* Lund : Dept. of Computer Sceince, Lund University, 2014.

10. *Some like it hot: automating an electric kettle using PalCom.* Magnusson, Boris and Johnsson, Björn A. New York, NY, USA :

ACM, 2013. In Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication (UbiComp '13 Adjunct). pp. 63-66.

11. Sharp, Robin. *Principles Of Protocol Design.* Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2008.

12. *An Improvement of the Route Discovery Process in AODV for Ad Hoc Network.* Hu, Yongjun, Luo, Tao and Shen, Junliang. 2010. International Conference on Communications and Mobile Computing (CMC), 2010.

13. *A new survey of routing algorithms in ad hoc networks.* Ghazani , Seyed H. H. N., Lotf, Jalil J. and Alguliev, R. M. 2010. 2nd International Conference on Computer Engineering and Technology (ICCET), 2010.

14. *Efficient Route Discovery and Repair in Mobile Ad-hoc Networks.* Po-Jen, Chuang, Po-Hsun, Yen and Ting-Yi, Chu. Washington, DC, USA : s.n., 2012. IEEE 26th International Conference on Advanced Information Networking and Applications (AINA '12).

15. *NARD: Neighbor-assisted route discovery in MANETs.* Gomez, J., et al. 17, 2011, Wireless Networks, Vol. 8, pp. 1745-1761.
16. *A Novel Multiple Routes Discovery Scheme for Mobile Ad Hoc Networks.* Xie, Fang, et al. 2006. Asia-Pacific Conference on Communications, 2006. APCC '06.

17. *Improvements on DSDV in Mobile Ad Hoc Networks.* Liu, Ting and Liu, Kai. 2007. International Conference on Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007.

18. *Ad-hoc on-demand distance vector routing.* Perkins, C. E. and Royer, E. M. 1999. Second IEEE Workshop on Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99.

19. *A Survey on Fault-Tolerance in Distributed Network Systems.* Xiong, Naixue, et al. 2009. International Conference on Computational Science and Engineering, CSE '09. IEEE.

20. *A survey on reliability in distributed systems.* Ahmed, Waseem and Wu, Wei Yong. 8, 2013, Journal of Computer and System Sciences, Vol. 97, pp. 1243 – 1255.

21. *A Survey of Fault Management in Wireless Sensor Networks.* Paradis, Lilia and Han, Qi. 2, 2007, Journal of Network and Systems Management, Vol. 15, pp. 171-190.

22. *UPPAAL software and tutorials.* [Online] http://www.uppaal.org/.

23. *A Survey of Service Discovery Protocols in Multihop Mobile Ad Hoc Networks.* Mian, Adnan Noor, Baldoni, Roberto and Beraldi, Roberto. 1, 2009, IEEE Pervasive Computing, Vol. 8, pp. 66-74.

24. *Survey on Service discovery for Wireless Sensor Networks.* Anwar, Fatima Muhammad, Seung-Wha, Yoo and Ki-Hyung, Kim. Jeju Island, Korea (South) : s.n., 2010. Second International Conference on Ubiquitous and Future Networks (ICUFN), IEEE, 2010.

25. Hedrick, C. Routing Information Protocol, RFC 1058. [Online] June 1988. http://www.rfc-editor.org/rfc/rfc1058.txt.

26. hrPING. [Online] http://www.cfos.de/hrping-v506.zip.

27. *The effect of forgetting on the performance of a synchronizer.* Függer, Matthias, et al. 2015, Performance Evaluation, Vol. 93, pp. 1-16.

Paper I: Media Abstraction Framework for the Pervasive Middleware PalCom

Amr Ergawy and Boris Magnusson,

Published in Proceedings of the 2nd International Conference on Future Internet of Things and Cloud, FiCloud-2014, Barcelona, Spain, 2014. IEEE.

Abstract: Pervasive middleware systems are important for enabling the configuration and coordination of the services provided by devices in pervasive environments. PalCom is an example of such a system that aims to enable interaction among pervasive services over heterogeneous networks.

In this paper we discuss the need for providing networking-media independent messaging among pervasive devices, and identify the problem of designing a media abstraction framework for a pervasive middleware. We identify a number of design principles and features that need to be satisfied by such a framework. The most important of these principles is the separation of networking protocol aspects into abstraction representations, which are exposed in a unified format to upper layers of PalCom, and communication mechanisms which are hidden by media-specific implementations of our proposed media-abstraction framework for PalCom. Also, we explain our implementation and evaluation of that framework.

Keywords**:** pervasive middleware, networking media abstraction, software framework.

1 Introduction

PalCom is a pervasive middleware that employs the human-in-the-loop principle that enables users to compose systems as assemblies of available services on independent devices, which may communicate over local networks as well as inter-networks [1]. A PalCom instance runs of each of the interacting devices. Users can write PalCom assemblies that specify the roles of the services on these devices and their coordination in order to accomplish the task of the target application. A PalCom assembly separates the coordination information from services implementation on the provider devices. A PalCom assembly coordinates services that may not have been originally designed to work together. PalCom implements device and service discovery to support the composition and running of services assemblies. Device discovery is done via a heart-beat mechanism while devices offer self-describing services.

PalCom provides a set of key features [1]. Firstly, it separates the aspects of configuration and coordination from those of computation. Secondly, PalCom uses a distributed discovery mechanism spanning devices on different heterogeneous networks. Thirdly, PalCom defines its own interaction protocol among the coordinated services, making it independent from the underlying network protocols.

However, as PalCom was designed with the focus on assemblies of device services, it does not provide a media abstraction framework that abstracts different networking interfaces to the upper layer of the PalCom stack, i.e. the device discovery and routing layer. We use the term media in the same sense as in [2], referring to the means of accessing a communication or networking system. That is not to be confused with the elements that are being communicated, e.g. images, voice, or video. Considering that PalCom devices and services discovery and communication is done on heterogeneous networks, such media abstraction framework enables Media Independent Messaging, MIM, among such devices and services.

In this paper, we describe our design, implementation, and evaluation of a framework that enables the flexible addition and management of network interfaces, of different media types, to the PalCom pervasive middleware. We do this work in the context of extending the work in [3] to support cross-network discovery/undiscovery, with limited cross-network heartbeat, as well as cross-network message delivery in PalCom.

In particular, our problem can be stated as follows:

To design a framework that enables interfacing different types of networking media interfaces to PalCom upper layers.

In section 2, we illustrate the main challenge of designing such a media abstraction architecture, which is the vast diversity of networking technologies in pervasive systems. Then, in section 3, we review different approaches of designing media abstraction frameworks for pervasive systems. Afterwards, in section 4, we illustrate the design principles and the details of our proposed media abstraction framework for PalCom. And before testing and evaluation in section 5, we illustrate the implementation with a usage example of our proposed framework in section 6.

2 Diversity of Media Interfaces in Pervasive Systems

Media interfaces in pervasive systems are used to communicate data for a variety of application areas like healthcare [4] and vehicular ad-hoc networks [5]. In this section, we emphasize that the diversity of networking technologies is a challenge for designing a common media abstraction framework for a pervasive middleware like PalCom. For this purpose we review the networking technologies and protocols from the two example domains, of health care and vehicular communication.

2.1 Diversity of Media Interfaces in Healthcare

For the health care field [4], both patients and workers need to communicate and exchange data anywhere and anytime, requiring pervasive computing architectures. The applications range from remote-monitoring and ambient assistance of patients to improve hospital-centric workflows.

Health care applications utilize a very broad range of networking technologies and communication protocols [4], which can be categorized as follows:

1) Short range wireless networking, which is used to implement personal area networks that integrate patients' sensors into health care information systems. Mainly, such networks use Zigbee and Bluetooth technologies.

2) Radio frequency identification, RFID, technologies, which is mainly used for identification and tracking of people, e.g. patients or workers, and objects, e.g. tools and medicine packs.

3) Infrastructure networks that connect people outside hospitals and homes to health care information systems. That category includes both wireless local area networking, WLAN, technologies like IEEE 802.11 family and wide area networking, WAN, technologies like internet and even mobile networks.

Taking the first category as an example, different short range networking technologies and protocols differ in design considerations [6]. Some of these technologies are designed for low data rates, others are designed for power efficiency, while a third set is designed to ensure priority guarantees for a specific type of data traffic. These differences in design considerations lead to a large number of different details among different networking technologies and protocols, even if they are designed for the same application area.

2.2 Diversity of Media Interfaces in Vehicular Ad-hoc Networks

For the vehicular ad-hoc networks field [5], the advances in networking technologies supported computing intensive vehicles, which even extends to vehicle-to-vehicle and vehicle to infrastructure networking. Initially, vehicle networking is built around a heterogeneous set of networking technologies, including the WLAN IEEE 802.11 standard family and the WiMAX IEEE 802.6 standard family.

Additionally, dedicated vehicle networking standards are being proposed and developed [5]. Such protocols address design problems like routing in vehicle-to-vehicle networking and media access protocols for both vehicle-to-vehicle networking and on-board networking. These protocols are designed from the beginning with the application area specific considerations and challenges in mind [5]. These consideration include, continuous vehicles mobility, limited connections between vehicles and infrastructure networks, and the not fully adaptable solutions that are based on existing wireless technologies, like WLAN and WiMAX.

Additionally, a geographic dimension adds to the diversity of the designed dedicated vehicle-to-vehicle protocols [5]. Different standardization organizations all over the world have designed a wide variety of dedicated vehicle networking protocols based on the above mentioned design considerations. Again, this leads to a wide variety of the details of technologies and protocols for even this single application domain.

To sum up, the wide variety of design considerations and details of networking technologies and protocols, in different application domains of pervasive computing, introduces a challenge for designing a common media abstraction framework for a pervasive computing middleware like PalCom.

3 Previous Work: Service Oriented Abstraction vs. Media Independence Abstraction

Media abstraction was addressed in the areas of providing interoperability among different devices in pervasive systems [7] and seamless handover among different media in the area of mobility management [2] [8] [9].

3.1 Service Oriented Architecture based Media Abstraction

First, we review media abstraction for interoperability among devices and services in pervasive environments [7]. Service Oriented Architecture, SOA, based on Web Services technologies are used to abstract the services provided by devices, including their networking interfacing. In [7], an architecture of a universal service bus is designed to abstract services on devices to virtual web services using what they refer to as adapters.

For example, the services of devices communicating over Bluetooth are abstracted via a Bluetooth adapter, abstracting the Bluetooth communication to the universal service bus in terms of a virtual service. SOA based abstraction, using Web Services technologies, are criticized for the possibility of being resource demanding considering the resources limited devices in pervasive environments [9]. Our proposed media abstraction architecture minimizes the required extra resources by being a built-in component of PalCom, i.e. no additional framework is required for its integration.

3.2 Media Abstraction for Seamless Handover

Secondly, for seamless handover among different media in the area of mobility management [2] [8] [9], media abstraction is handled by providing a framework that can interface media-specific Service Access Points, SAPs, to upper layers of mobility management logic.

The Media Independence Handover, MIH, architecture described in [9], introduces a media abstraction architecture that shares two common features with the media abstraction architecture what we propose for PalCom, i.e. the dynamic nature of using plugging-in media specific SAPs, and the flexibility that developers can develop new media specific SAP implementations as needed. However the two architectures differ in the target of supporting media independence. In [9], the architecture focuses on Media Independent Handover, MIH, while our proposed architecture focuses on Media Independent Messaging, MIM.

4 Proposed Media Abstraction Framework for PalCom

From the perspective of a middleware software like PalCom, communication media are accessed using protocols. To design a media abstraction framework for PalCom, we may follow an approach where we survey most

used media protocols in pervasive environments, similar to the surveys in [4] [6] [5], aiming to design a media abstraction framework that abstracts their common and key features to the upper layer in PalCom. However, this approach may set non-flexible boundaries on the protocols that developers, who will use the proposed media abstraction architecture, may implement and plug into PalCom.

Alternatively, we chose to follow an approach where we argue that communication protocols share common design principles where their designers start from, then protocols differ in their details based on their specific requirements and design considerations. Protocol design principles are extensively discussed in [10]. By referring to that discussion, we argue that we can divide protocols aspects into two categories, i.e. mechanisms and representations.

Protocol mechanisms include connection establishment, different types of control, e.g. sequence and flow control, routing, and security. On the other hand, protocols representations include addressing formats, naming formats, and message encoding. We design media abstraction in PalCom to abstract representations to upper layers and to leave mechanisms to media specific implementations and upper layers of PalCom, e.g. supporting reliable sending at these layers.

4.1 Solution Basis and Approach

1) Solution Basis:

We classify aspects of networking protocols [10] of different media interfaces as follows:

a) Representations: devices names, networking addresses, routes definitions, and message encodings.

b) Mechanisms: including other aspects of a networking protocol, e.g. connection establishment and routing.

2) Solution Approach:

We propose a media abstraction architecture that:

a) Transforms networking protocols specific representations into a unified internal format for processing by upper layers of the PalCom stack.

b) Leaves networking-protocol specific mechanisms as internal implementation details of Media Abstraction Objects, MAOs, which abstract different media networking interfaces, as we explain next in section 4.2.

4.2 Design Principles and Features

We design our media abstraction architecture to utilize representations abstraction, to enable easy development and integration of media abstraction objects, MAOs, for different networking interfaces. Next, we detail our design principles under two main categories, namely utilizing representations abstraction and designing for flexible and robust framework.

1) Utilizing Representations Abstraction

In [10], networking protocol aspects that we refer to as protocol specific representations are defined as follows:

c) A name: a permanently associated identifier with an object.

d) An address: an identifier associated with the current object location.

e) A route: an identifier associated with a path to an object.

From these definitions, we argue that regardless of the protocol specific representations of these identifiers, they can be abstracted to PalCom upper layers as strings. For a name or an address, we abstract its protocol specific representation as its human readable string format, making it easy for both integration of its media type and for system implementers and maintainers.

For a route, we abstract its protocol specific representation as a pair of two human readable strings that together represent the next-hop from the local node to forward a message via that route. The two strings in that route abstraction pair are listed as follows:

a) The human readable format of the name of the local networking interface via which a message is to be forwarded on the route.

b) The second string is the human readable format of the address of the next-hop node on the route.

2) Designing for Flexibility and Robustness

Our proposed media abstraction framework aims to abstract networking interfaces of different media types as media abstraction objects, MAOs, which shall be easy to develop for newly PalCom supported networking media types as well as easy to integrate to PalCom upper layers.

Given the vast variety of existing and future networking technologies and protocols that are used in pervasive systems, as reviewed above in section 2, our framework design aims to provide the following features:

a) Abstracting development framework: we have to ensure that no media-type specific design principle or mechanism is built into our development framework where MAO developers start their implementation from.

In particular, we have to ensure that the development framework has no implementation details except of starting points, e.g. abstract methods, where a MAO developer can extend to provide the necessary representation abstraction elements that we discussed earlier in this section.

b) Dynamic plugging: we need to allow the possibility to plug and unplug media abstraction objects, MAOs, of different networking media interfaces to and from PalCom at any point in time while the system is running.

This is to accommodate the variety of networking media types, the changing number of available networking interfaces over-time while the system is running, and the possible changes of user preferences.

c) Fault tolerance: at different stages of plugging and using a media abstraction object, MAO, we need to ensure that internal MAO errors and failures are handled robustly by the media abstraction architecture, which shall continue functioning properly after handling or just smoothly tolerating such errors.

Also, we need to ensure that our framework provides enough buffering that decouples the upper layers of PalCom from MAO implementations. This is to avoid any faulty MAO implementations that would block upper layers for long times.

d) Simplified and Robust Concurrency: for a media abstraction framework, concurrency is an essential feature that enables the handling of multiple input/output operations to/from multiple networking interfaces, implemented as instances of different media abstraction objects, MAOs.

Therefore, we have to ensure that there is a simple concurrency model that is easy to understand by MAO developers. Such a model makes it easy to design, integrate, verify, and troubleshoot MAO implementations. More importantly, we need to ensure that our concurrency model eliminates the possibility of concurrency problems, e.g. dead-lock, when plugging an instance of a media abstraction object, MAO, to the system.

4.3 Framework Structure and Activities

In this section, we start by describing the layered structure of our media abstraction framework, mapping its different activities to the design principles and features that we described in the previous section. Then, we describe the details of the activities of each framework layer, explaining the actions and signals that realize the design principles and features.

1) Framework Structure vs. Design Principles and Features

As shown in Fig. 1, when the system is running, at the upper-most layer the Media Abstraction Manager, MAM, maintains a list of Media-Type Handlers, MTH, at the lower layer, which in turn maintains a list of Media Abstraction

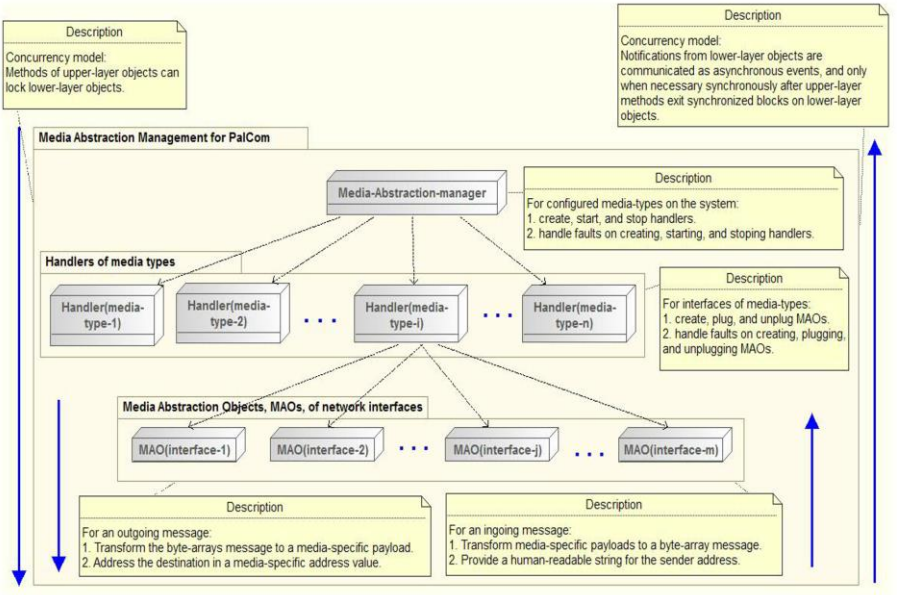

Figure 1. Media Abstraction Management for PalCom

Objects, MAO, for the network interfaces of their corresponding media-types. By “maintaining” here we mean creating the maintained elements, starting/plugging them, or stopping/unplugging them. We explain these operations in details later in this section, as part of the framework activities.

Continuing with the system structure, we map the design principles and features, discussed above in section 4.2. For utilizing the representation abstraction, as shown in Fig. 1, when a MAO receives a message in a media specific format, it decodes its payload to a byte array that forms a PalCom message. Similarly, the MAO supplies the address of the message sender to the upper PalCom layers with a human-readable string of that address. For an outgoing message, the MAO encodes the byte-array of a PalCom message as a payload for a media-specific message. In this case, the message is either a broadcast message or a unicast message, where the destination address is supplied by the upper PalCom layers to the MAO as a reference to a media-specific address.

As a fault tolerance measure, for both outgoing and ingoing messages, limited size buffering is used to decouple the MAO functions from the functions of the upper PalCom layers. On the buffer-full events of these buffers, proper measures are taken while keeping the properly functioning parties working. We explain the details of all of these mechanism below, as part of the system activities explanation.

For abstracting the development framework, this is more explained as part of the framework design, in section 6. But for dynamic plugging, as shown in Fig. 1, a media-type handler, MTH, of a given media-type continuously create, plug and unplug MAOs for the network interfaces, on the local system, of its media-type. For fault tolerance, all operations of dynamic plugging are checked at its different stages for possible faults due to implementation issues.

A more generic view of dynamic plugging and unplugging is the addition of new media-types to the system, by the media abstraction manager, MAM. In particular, a media-type can be configured to the system as a set of media-specific implementation elements, e.g. classes in the case of object oriented implementation of the framework. Then, for each configured media-type, the MAM creates and starts a media type handler that uses these implementation elements to dynamically plug and use the interfaces of this media to the system. For tolerating faults in the media-type implementations, the MAM handles possible faults at the different stages of using them. below, we explain the details of the MAM and the MTH mechanisms among other framework activities.

Finally, as shown in Fig. 1, the concurrency model has two simple rules. The first rule allows top-down method/procedure executions to lock used objects while the second rule restricts bottom-up message and signal flows to be communicated as only asynchronous events to avoid locking upper-layer objects. On a very limited scale, bottom-up message and signal flow can be done in synchronized blocks on an upper-layer object that must exist outside any synchronized block that is used by that upper-layer object.

2) Framework Activities

We explain the framework activities at the three layers of the framework structure. We start by explaining the activity of adding a media-type to the system by the MAM. Then, we explain the activity of adding and dynamically plugging network interfaces into the system, by MTHs. Finally, we provide the details of the functionality of plugged network interfaces.

a) Adding Media Types: New-media types can be configured to the system. One way to do this is to store the media specific implementation of a network interface enumerator, NIE, to a store that can be parsed by MAM to create an instance of that NIE. Another way is to programmatically create an instance of that media specific NIE and pass it to an explicit call of a configuration method/procedure of the MAM.

On the event of configuring a new media type to the system, the MAM creates a media type handler, MTH, which takes the NIE of the media-type as

a parameter. Finally the MAM attempts to start the MTH. On any fault event at any of these stages, the MAM takes necessary measures, at least by logging them and terminating the attempted operations properly.

b) Manageing Network Interfaces: On the start of the MTH, periodically, it uses the NIE to get the list of the current network interfaces of this media-type on the system. Then, for newly detected interfaces, the MTH starts to create MAOs and plug them into the system using MAOPlugs, which we explain next as part of the functionality of plugged network interfaces. For all disappeared interfaces, the MTH unplugs them.

In addition to the periodic maintenance of the network interfaces plugging and unplugging, control commands to the MTH, e.g. issued by the user interface can plug or unplug a specific MAO for a specific interface or even stopping the entire media-type by stopping the MTH, which in turn unplugs all MAOs for all interfaces. For all of MTH operations, possible faults, e.g. due to NIE or MAO faulty implementations, are handled at least by logging them or at most by properly terminating the MAOPlug and the MAO for the associated interface.

c) Functionality of plugged network interfaces: On the initial plugging of the interface, the MAOPlug attempts to start the MAO that abstracts the interface, the send worker, and the receive worker. All of these components are expected to start properly after a given time. In case of an exception during starting any of the components or the starting attempt times out, the initial plugging attempt is terminated gracefully. On the success of the initial plugging attempt, an “added interface” event is signaled to the Interfaces Events Store, IES. Also, the MAOPlug provides operations for plugging and unplugging the MAO, e.g. by the user interface, which also add corresponding events to the IES.

On the recipient of a media specific message, a plugged MAO decodes the payload of a received media specific message to a byte array that represents a PalCom message. Then, if the ingoing messages buffer has enough space, the message is buffered for further processing by the MAOPlug receive worker, which signals PalCom upper layers about the ingoing message and receive signals from them about either successful or unsuccessful processing of the message.

For outgoing messages, the PalCom upper layers can send or broadcast messages of the MAOPlug, passing the message in PalCom internal format. In turn, the MAOPlug transforms the message to a byte array and buffers it to the outgoing message queue if there is enough space. On polling an outgoing message, the MAOPlug send-worker calls the proper sending or

broadcasting method from the interface MAO, which encodes the byte array message into the payload of a media-specific message.

In addition to message buffering for decoupling the functionality of and interface MAOPlug and MAO from the PalCom upper layers, all faults from a MAO implementation or even from the framework are handled by at least logging them or at the worst case by terminating the execution of the MAOPlug. For all of the framework activities, the concurrency model, discussed above, applies across the three layers.

5 Framework implementation

We divided our framework into two layers, a run-time framework and a development framework for developing media abstraction objects, MAOs. We implemented our framework in Java, as an extension of the current PalCom implementation. As shown in Fig. 2, the runtime is implemented by the top-most package, media_abstraction.runtime, while the development framework is implemented by the media_abstraction.framework package.

All the development framework classes are abstract classes, with the abstract methods expected to be implemented by MAO developers while final methods are dedicated for error handling and interfacing with the runtime. As an example MAO implementation, we implemented the media_abstraction.IPMAO package, which have classes that implement a MAO for IPv4.

As illustrated in **Error! Reference source not found.** 2, in the runtime package, the MediaAbstractionManager class, the MediaTypeHandler class, the MAOPlug class, and the AbstactiMediaAbstractionObject class directly map to the components we show in Fig. 1. Moreover, the abstraction of a media-specific message content, which we discussed in section 4, is implemented as a MediaAbstractionPayload. That is a message payload that can be passed or received from a plugged MAO. Additionally, the abstraction of the network address is implemented as an AbstractCommunicationAddress, which is extended by the MAO for its own specific address format. To associate a source or a destination address with a message payload, the AddressedMediaAbstractionPayload class uses objects of the AbstractCommunicationAddress.

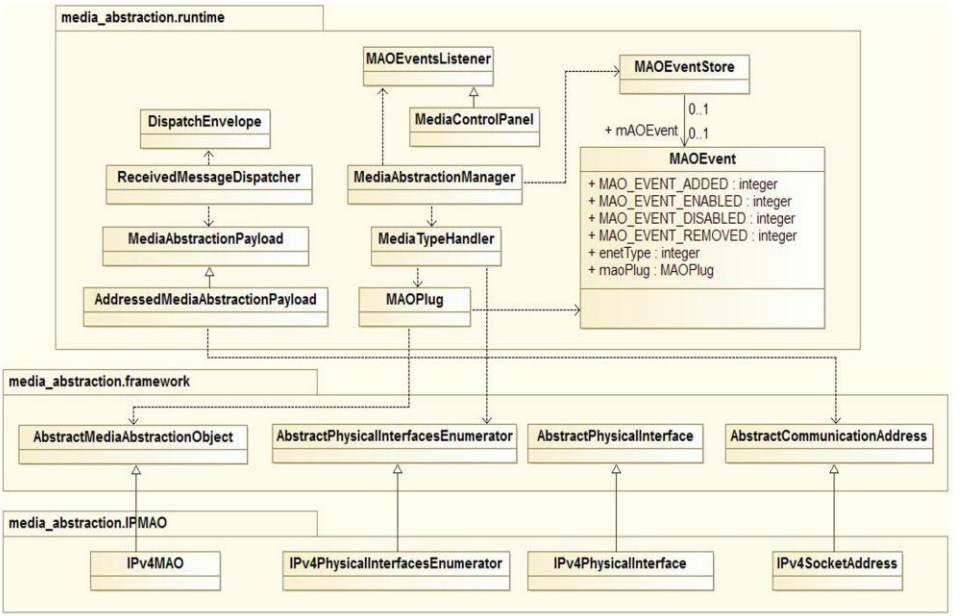

Figure 2. The Media Abstraction Framework

Another aspect that Fig. 2 illustrates is handling MAO events of different types, as listed by the constant values in the MAOEventClass. Such events can indicate an added MAO, a removed MAO, a disabled MAO, or an enabled MAO. With each MAOEvent object that a MAOPlug can pass about a MAO, the event type is indicated as well as the source MAOPlug. Such events are asynchronously communicated to the MediaAbstractionManager via a MAOEventsStore. The MediaAbstractionManager has to convey such events to objects of the MAOEvents listener, e.g. the MediaControlPanel.

6 Testing and Evaluation

In addition to conventional software unit and integration testing of our framework, we tested the usage of the framework by our being developed across-networks device discovery mechanism. Simulated devices that communicate concurrently were able to discover each other on top of our media abstraction framework and across multiple IPv4 networks.

Also, we considered evaluating the framework from two other aspects. The first is the required effort to integrate a MAO into the framework. The second is the upper-limit of the time overhead that our framework adds to send and

receive PalCom ping messages of a specific size compared to sending and receiving ICMP ping messages of the same size.

6.1 Integration Effort

To integrate the framework to PalCom we mainly need an object of the MediaAbstractionManager. Then, a programmer needs to start the MediaAbstractionManager and add different MAOs to the system. Thirdly, the example IPv4MAO is easily added to the system with just one method that take, as a parameter, an object of the IPv4MediaType, which implements an abstract bundling class in the MAO development framework called AbstractMediaType. It takes only one line to add a media-type to the system.

In addition to saving the name of the media type and creating a singleton of the MAO enumerator, the MAO implementation AbstractMediaType is responsible for creating objects of the MAO implementation of the AbstractMediaAbstractionObject as well as validating the type of AbstractCommunicationAddress objects against the type of the MAO implementation of the AbstractCommunicationAddress.

6.2 Messaging Time Overhead

We evaluated the time overhead of sending a PalCom ping message via our media-abstraction framework for a round-trip over an Ethernet LAN. For configurable and focused testing, we developed a testing framework that adds a dummy PalCom device stack on top of the tested media abstraction framework.

The test is done in a ping pong setup, where one node is configured to periodically ping another node that just echoes the ping message back. We refer to the first node as a ping node while we refer to the other as a pong node. The ping node is a dual core of 1.7 GHz each and 16 GB memory, while the pong node is a dual core of 2.6 GHz each and 8 GB memory.

The ping message contains a trace-ID that travels with it in the round-trip. The ping node uses the trace-ID to record two time stamps for a message:

1) On sending the message, the ping node records the time of the message entry to the sending method of the media abstraction framework.
2) On receiving the message, the ping node records the time of the message handing to the dummy processing logic above the framework.

In Fig. 3, the series with round-shaped elements represents the round trip times of more than 180 PalCom ping messages. Also in Fig. 3, the series with

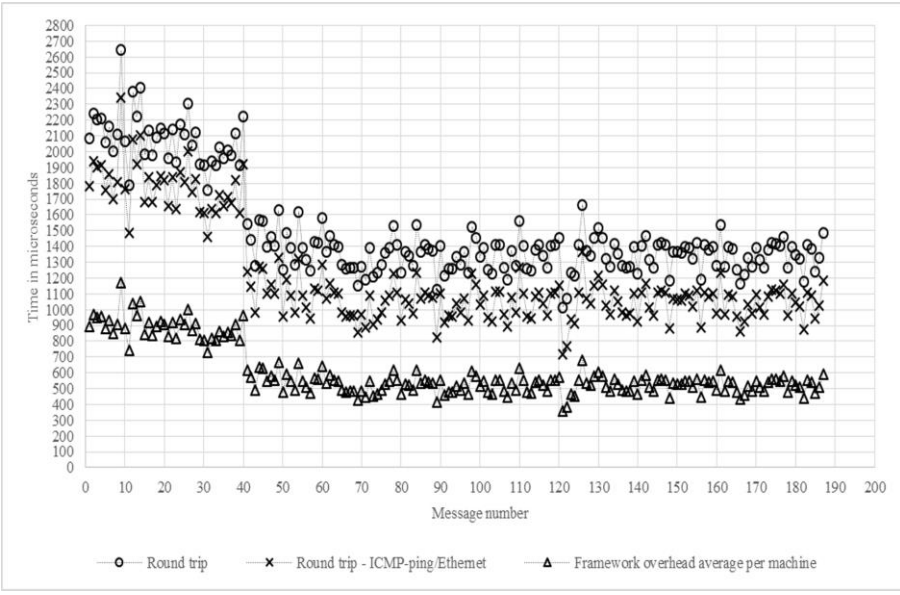

Figure 3. Framework processing time of a PalCom ping message over Ethernet LAN

x-shaped elements shows those round trip times minus the average round trip time of ICMP ping messages on the Ethernet link of the test setup, 300 microseconds, of the same size as PalCom ping messages. We set the size of the ICMP packet as an option in the ping tool we use. Finally, the series with triangle-shaped elements in Fig. 3 shows the subtracted round trip times divided by two. This series gives an estimation of the upper bound of the overhead processing time of PalCom ping message, in both directions, on each of the two machines in the test setup.

From Fig. 3, the processing time of a PalCom ping message via our framework decreases over time to almost stabilizing, after message number 40 in this test run, to be around 500 microseconds. The processing time includes both directions of the message. This result shall be interpreted in the context of the test setup. For realistic communication situation in PalCom, an overhead of 0.5 millisecond should be acceptable.

7 Conclusion and Future work

The vast diversity of networking in pervasive environments is a main challenge for designing a media abstraction framework for a pervasive middleware like PalCom. However, using such network technologies is

mainly done via network protocols which share common design principles. A previous work on media abstraction utilized web services technologies to abstract network adapters as services [7], which is resource-demanding with respect to pervasive devices [9]. Our proposed media abstraction framework aims to avoid this by being an integrated part of PalCom. Another previous work focuses on abstracting networking media among a specific family of network protocols for the purpose of handing-over communication among network interfaces [9]. Our proposed framework differs from that work by enabling media independent messaging across heterogeneous networks.

We identified the design principles for a media abstraction framework for PalCom by refering to the design principles of networking protocols, separating the protocol specific mechanisms from generic enough abstractions that are provided by these protocols, e.g. string representations of network addresses and byte arrays values of messages contents. That enables simplified integration of media abstraction objects, MAOs that integrate different network media to the PalCom. We evaluated both the MAOs integration effort and the framework performance, which shall be interpreted in the context of the application domains.

As future work, we aim at developing more MAOs for other media to identify more requirements for the framework, we also may investigate techniques to improve the performance of the system, if required.

Acknowledgment

This work was funded by The Swedish Fund for Strategic Research. We also gratefully thank the developers of the free tools:

1) Modelio: http://www.modelio.org/
2) hrPING: http://www.cfos.de/hrping-v506.zip

References

[1] D. Svensson Fors, B. Magnusson, S. Gestegård Robertz, G. Hedin and E. Nilsson-Nyman, "Ad-hoc Composition of Pervasive Services in the PalCom Architecture," in *Proceedings of the 2009 international conference on Pervasive services*, London, 2009.

[2] *802.21-2008 IEEE Standard For Local And Metropolitan Area Networks- Part 21: Media Independent Handover.*

[3] D. Svensson Fors, "Assemblies of Pervaisve Services, PhD Thesis," Dept. of Computer Science, Lund University, Lund, 2009.

[4] F. Delmastro, "Pervasive Communications In Healthcare," *Computer Communications,* no. 35.11, pp. 1284-1295, 2012.

[5] M. Booysen, S. Zeadally and G.-J. van Rooyen, "Survey Of Media Access Control Protocols For Vehicular Ad Hoc Networks," *IET Communications,* no. 5.11, pp. 1619-1631, 2011.

[6] Khan, Z. A et al., "A Comprehensive Survey Of MAC Protocols For Wireless Body Area Networks," in *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2012 Seventh International Conference on*, 2012.

[7] Y. Hyung-Jun and K.-C. Lee, "A Ubiquitous Web Services Framework For Interoperability In Pervasive Environments," *International Journal Of Multimedia & Ubiquitous Engineering,* 2012.

[8] Aguiar, R.L. et al., "A Framework For The Connectivity Of An Internet Of Things," in *Sensors, 2011 IEEE*, 2011.

[9] Aguiar, R. et al., "Mindit: A Framework For Media Independent Access To Things," *Computer Communications,* no. 35.15, pp. 1772-1785, 2012.

[10] R. Sharp, Principles Of Protocol Design, Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2008.

Paper II: Device discovery for the Pervasive Middleware PalCom with Eliminated Cross-networks Heart-beat Messages

Amr Ergawy and Boris Magnusson,

Published in proceedings of the 5th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2014), Halifax, Nova Scotia, Canada, 2014, Procedia Computer Science, Volume 37, 2014, Pages 64-71.

Abstract: Service discovery is central in pervasive middleware systems, built on top of the communication substrate using the more fundamental mechanisms for device discovery. In mobile pervasive systems devices come and go, and switch network frequently, demanding support for device discovery across heterogeneous networks. We present the design of a device discovery mechanism for the PalCom middleware that eliminates the need for cross-network periodic keep-alive messages while still supporting timely detection of missing devices, i.e. undiscovery. The design has been implemented and is evaluated against the simplistic approach of flooding the inter-connected networks with keep-alive messages.

Keywords: pervasive middleware; device discovery; network traffic optimization.

1 Introduction

Discovery mechanisms, for example Zero conf [1], have greatly simplified configuration of computer systems, such as finding a nearby printer. These mechanisms are frequently used in a local network, as supported by UDP [2] broadcasts, but not across networks. Palcom [3] is offering similar mechanisms across networks to configure general systems of Internet-of-Things covering both discovery, i.e. a device is available, and undiscovery i.e. finding out that a device is no longer available within a configurable time-out. The latter is relevant for example when monitoring patients in medical applications. Such mechanisms are implemented using broadcasting requests, for discovery, and exchanging periodic keep-alive messages, heartbeats, for detecting loss of devices, undiscovery.

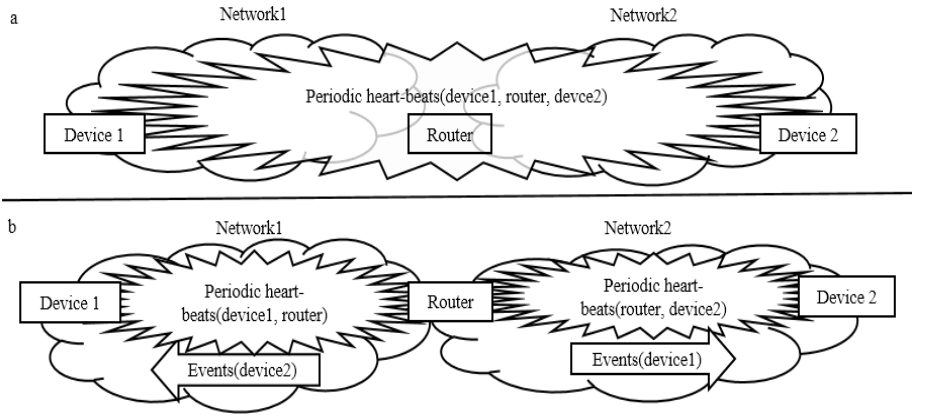

Fig. 1 (a) cross-networks periodic heart-beating (b) local-networks heart-beating and cross-networks events.

In Palcom there is support for discovery/undiscovery over heterogeneous, potentially large, networks where every device can act as a router of discovery information between networks it is attached to. In such cases a simplistic approach might result in a network flooded with keep-alive messages. In this paper we present a solution to this problem where broadcasting is only needed in the local networks, while maintaining the effect that loss of a device in one network can be acted upon in another network with a configurable and predictable delay. The idea behind our approach is to replace cross-network periodic heart-beat messages with discovery events notifications. This is shown in Fig. 1.

From the perspective of service discovery and data messaging, our proposed device discovery mechanism builds an overlay communication substrate [4] on router nodes that bridges the islands of discoverability [6] of different local networks. We discuss this perspective and its implications on designing a device discovery mechanism in section 2. In section 3, we detail our proposed device discovery mechanism and summarize the results of evaluating our cross-networks heart-beating optimized mechanism against the simplistic approach of cross-networks heart-beat flooding. Finally, in section 4, we present our conclusions that generalize our evaluation results and brief our future work.

2 Communication support for service discovery: challenges and design

In [4], a reference model to the pervasive computing middleware is introduced. It groups the functionalities of a pervasive middleware into three main areas: common services support, cross-layer support, and runtime support. Among the runtime support functionalities is the communication support. As we mentioned above, device discovery in PalCom provides a communication substrate for service discovery and data messaging. In turn, according to the model in [4], device discovery for PalCom can be classified as a communication support utility.

In order to understand the design considerations of such a utility, we survey the communication support for service discovery in pervasive computing [5]. Thus, we identify that dynamism and heterogeneity of pervasive environments, where devices with different communication interfaces continuously join and leave the network, are main challenges to communication. In turn, we refer to communication support for service discovery in ad-hoc and mobile networks [7] [8] to identify design options for handling dynamism in pervasive environments. Moreover, we discuss desired characteristics [5] that a communication support utility shall provide in order to handle heterogeneity in a pervasive environment.

2.1 Communication support vs. dynamism and heterogeneity

In pervasive computing environments [5], service discovery spans networks of different types. It may even integrate users by enabling them to assemble services into application scenarios. In turn, for service discovery, pervasive environments are more heterogeneous and dynamic than conventional distributed or mobile computing environments. The heterogeneity and dynamism are reflected in the design options for components of service discovery protocols [5]. Among the several components of a service discovery protocol is the initial communication method [5].

For the initial communication method, service discovery protocols mainly employ unicast, multicast, or broadcast messaging [5]. Unicast is efficient, but it requires configuration of network addresses prior to service discovery, which may not suite dynamism. In contrast, broadcast has the advantage of not requiring pre-configured addresses, but it is preferred to be limited to local networks. Alternatively, multicast can be an initial mechanism via which

unicast addresses are automatically configured, but it may still require a minimum of pre-configurations. Also, these approaches are bound to the network layer or the media-access layer, not suiting heterogeneity.

2.2 Designing device discovery for dynamism and heterogeneity

Mobility support is essential for service discovery in mobile and ad-hoc networks [7] [8]. Mobility is supported by periodic updates of service information. It is suggested to limit the diameter of advertising services [8]. To support dynamism in pervasive environments, it is suggested to utilize devices routing information to support service discovery [8]. From such suggestions, we specify two mobility support approaches as design principles for our proposed device discovery mechanism in PalCom, namely:

- Maintaining an overlay structure that improves service discovery.
- Utilizing the resulting routing information among devices for services interaction during application scenarios.

Compared to the approaches above in section 2.1, our device discovery mechanism provides an overlay routing substrate decoupling service discovery, and data messaging, from networks details. As we will explain in section 3, our mechanism builds this substrate using heartbeat broadcast messages that are limited to local networks and using discovery event advertisements for cross network discovery. In turn, we limit the effect of choosing an initial communication method to the building process of the routing substrate.

From the perspective of heterogeneity in pervasive environments, a communication support utility can be characterized with respect to the two dimensions of communication transport and discovery scope [6]. Some service discovery systems limit their communication support to reachable devices via the network-layer or the media-access-layer protocols. In contrast, by using an overlay substrate, our device discovery mechanism inter-links discovery scopes across different networking technologies. This is supported by our ongoing work to provide a framework to abstract different networking media. We detail our solution in section 3.

3 The proposed device discovery mechanism

On a given node, the device discovery mechanism maintains a routing table with the status and routing information of devices in the networks local to the device and in other networks that are reachable via router nodes, which are mentioned in section 1. We start by describing the structure and the constraints of the routing table. Then, we describe the device discovery state machine for local network discovery. Afterwards, we detail the discovery forwarding mechanism that is used by router nodes to forward device discovery events among the different networks. Then, we describe the device discovery state machine for cross networks discovery, which handles forwarded device discovery events. And before mentioning our evaluation results, we describe a state machine that aligns the discovery state of devices on non-local networks, i.e. remotely discovered, to the discovery state of their introducer router nodes on local networks.

3.1 Structure of the routing table

For a discovered device, the routing table contains information that includes: the globally unique ID of the device, a current-hop generated short-ID for the device, a reboot-number, and a change-number. As long as a discovered device is known, its short-ID may replace its longer device-ID in messages. For the device change-number, it indicates updates of device services and it is mainly interpreted and used by upper PalCom layers.

We define *a route to a device* as the following pair: (A network interface on the current hop node, the networking address of the next hop node).

In the routing table, route specific information includes; the route state, the last time the current-hop node received an indication of the route availability, and possibly a number that we refer to as remote-short-ID. Other than the implicit initial state of unknown, a route state can be visible, rebooted, out-of-reach, and gone. We explain more about the route states in section 3.2.

The existence of a remote-short-ID in the route information distinguishes two route types:

- A *local-route*: its destination node and the current hop node have interfaces on a particular local network.
- A *remote-route*: its destination node does not have a network interface on any of the local networks on which the current hop node has interfaces on. In this case, a *remote-short-ID* is the short-ID that was generated by the next-hop router node for the destination node.

The discovery logic of a local-route is explained in section 3.2. While the discovery of a remote-route is initiated by the discovery forwarding mechanism, explained in section 3.3. The remote-routes discovery logic is explained in section 3.4.

We impose the following constraints on the device information in the routing table:

- A device can have only one address on a local network.
- Two device are not allowed to have the same address on the same local network.
- The device parameters are independent of which route the information is received from.

Table 1 shows an example of the routing table structure.

Table 1. The structure of the routine table with example data.

Device-ID	Short-ID	Reboot-Nr	Change-Nr	Network interface	Address	Route state	Refresh time	Remote-short-ID
Device1	1	54321	54	IPv4-MAO-1	<IP-nr, port)	Visible	<Time2>	N/A
				BT-MAO-1	<BT-addr>	Out-of-reach	<Time1>	3
Device2	2	12345	6	X-MAO3	<Adr5>	Visible	<Time3>	2

3.2 Device discovery in local networks

Implicitly, a device is initially unknown to a node until at least a route is discovered to that device, via the proper sequence of discovery messages. In the context of a local network, such messages are handled are shown in Fig. 2. A node periodically broadcasts an h-message within the local networks that it has interfaces on. A node that receives an h-message from a device via a route in the visible state can reply with an H-message. In this way, nodes on a local network can track the availability of each other.

Otherwise, a node that receives an h-message from a device via a route in the implicit unknown state initiates a discovery sequence using device information request and reply messages, i-messages and I-messages respectively, which contain the globally unique device IDs of the interacting devices. On the success of the discovery sequence, according to the logic in Fig. 2, the conveying route of the h-message becomes a visible route to its sender. As shown in Fig. 2, a state transition may report the appearance,

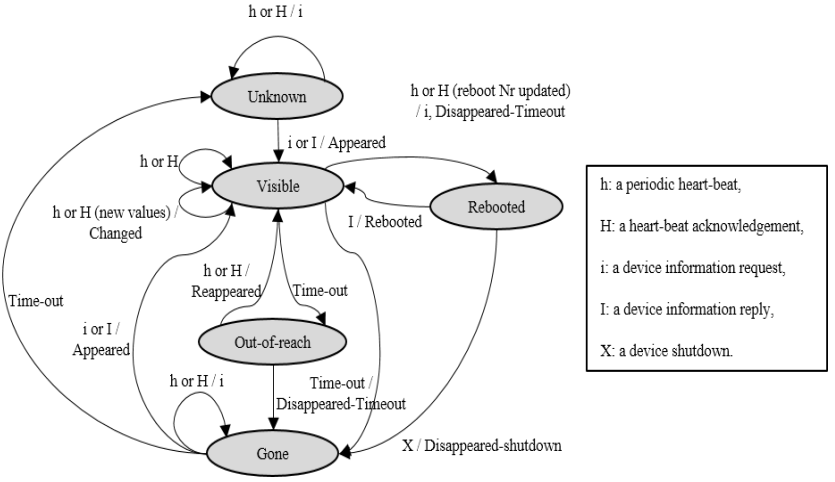

Fig. 2 Device discovery in local networks.

disappearance, or updates of a device via a route, which is processed by the discovery forwarding mechanism that we explain in section 3.3. State transitions happen either on configurable time-outs or on received messages. A received X-message indicates the shut-down of its source device, which results in setting the state of all the routes to that device as gone. In such a case, if no more discovery messages regarding that device are received from any interface during a configurable time-out, the device information is removed from the routing table. Another message type that is not shown in Fig. 2 is the interface-down message, i.e. the Y-message. A received Y-message indicates that its source node disabled the interface that the message was sent from. As a result, the state of the route to the sender device via that interface is set to gone. When all routes to a device are set to gone and subsequently become unknown, the device information is removed from the routing table after a time-out without receiving discovery messages regarding that device.

3.3 Discovery forwarding on a router node

A router node may have interfaces to multiple networks. It advertises device discovery events on one network to nodes on other networks. *A discovery advertisement message* has two parts. The first part is a routing-source node, S-node, which contains the short-ID that was given to the advertised device by the advertiser router node. The second part can be an H-message, an I-message, a Y-message, or an X-message, according to the

discovery forwarding logic, which is executed by router nodes to handle

Table 2. Handling discovery events.

Device discovery event on a route	Handling logic
Appeared	Maintain the device DFFs. If the interface of the discovered route becomes the introducer of DFF1 or DFF2, broadcast an S-I message via the advertisers of that DFF.
Changed	If the interface of the source route of the message that triggered the event is the introducer of DFF1 or DFF2, broadcast an S-H with the updates via the advertisers of that DFF.
DisappearedShutDown	All local and remote routes to the device are set to "Gone". If the interface of the source route of the message that triggered the event is the introducer of DFF1 or DFF2, broadcast an S-X via the advertisers of that DFF. Then, maintain the device DFFs. A local-route becomes "Visible" again on periodic update messages, processed as in Fig. 2. A remote-route becomes "Visible" again on non-periodic update messages, processed as in Fig. 3.
DisappearedTimeOut.	If the message that triggered the event conveyed a new reboot-Nr and its source rout is a local-route to the device, as in Fig. 2, all local routes to the device are set to "Rebooted" and all remote routes to it are set to "Out-of-reach". A local-route becomes "Visible" again on periodic update messages, processed as in Fig. 2. A remote-route becomes "Visible" again on non-periodic update messages, processed as in Fig. 3. On the time-out of the last local-route to the device in the "Rebooted" state, all the remote-routes to it are set to "Gone". In all cases, if the interface of the source route of the message that triggered the event is the introducer of DFF1 or DFF2, broadcast an S-Y via the advertisers of that DFF. Then, maintain the device DFFs.
Rebooted	If the interface of the discovered route becomes the introducer of DFF1 or DFF2, broadcast an S-I message via the advertisers of that DFF.

device appearance and disappearance reports, mentioned in section 3.2.

A router node forwards a discovery event advertisement via a *discovery forwarding flow, DFF*, which originates at the network interface of the route where the event originated, the *introducer*, and branches to a set of other network interfaces, the *advertiser*s. Thus a DFF is the pair (introducer, advertisers). On a router node, from the time of the discovery of the first route to a device until the time when all routes to it disappear, the discovery forwarding mechanism maintains only two DFFs for that device:

- *DFF1* = (the interface of the oldest discovered route to the device, the rest of interfaces on the router node).

- *DFF2* = (the interface of the 2nd oldest discovered route to the device, the introducer of DFF1).

Table 2 shows how the forwarding mechanism handles device discovery events that are triggered by the state machines in Fig. 2 and Fig. 3, explained in section 3.2 and section 3.4 respectively.

3.4 Handling routing loops

The communication substrate formed by discovery forwarding is formed of the set of DFF1 instances on all router nodes. A loop in that substrate may form when a router node forwards the same discovery event twice. To minimize such loops, the structure of DFFs prevents forwarding an event via its source interface, which is similar to the split-horizon mechanism of the routing information protocol, RIP [9]. Also, invalid routes are explicitly reported using S-Y and S-X messages, which is similar to route poisoning of RIP. In turn, a loop for a service discovery or data message is limited to be transient in a link between two router nodes while they exchange device discovery updates. Since a routing decision is made on hop-by-hop basis, a routing loop appears only when a node, B, decides to return a data message back to its direct neighbor source, A. The loop-initiator node, B, must have just lost the route that still seems valid to the source neighbor node, A. Eventually node A receives a message from node B to invalidate that route, terminating the transient loop.

3.5 Cross-networks device discovery

A node processes a received discovery advertisement message according to the logic in Fig. 3. The triggering of S-I, S-Y, and S-X messages is described in section 3.3. An S-H message is triggered by the discovery forwarding mechanism only on the event of updating the parameters of a remotely discovered device. Discovered routes by the logic in Fig. 3 are remote routes according to the definition in section 3.1. The state of a remote-route is affected by the state of the route, to the advertiser router device, that has the same network interface and networking address. Next, we explain the procedure that aligns the state of a remote route to it container remote route.

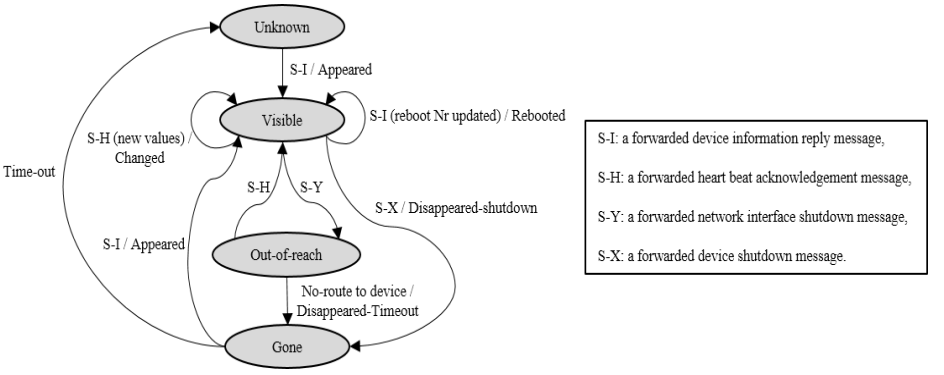

Fig. 3 Cross-networks device discovery. Notice the absence of the plain h/H messages compared to Fig. 2.

3.6 Aligning a remote-route to its introducer-local-route

On a node, a remote-route and its *introducer-local-route* have the same pair of (network interface on the current hop node, networking address of the next hop node). In other words, a remote-route was discovered based on forwarded discovery events that were received via its introducer-local-route. The availability of a remote-route is affected by the availability of its introducer-local-route. The *availability of a route* is described by two views:

- An internal view that is defined by the route state and maintained by state machines in Fig. 2 and Fig. 3.
- An external view that is defined by the route appearance and disappearance reports, and any subsequent forwarded discovery events, as described in section 3.3.

We align the availability of a remote-route to the availability of its introducer-local-route as shown in Fig. 4, following two rules:

- We align only the external availability view of a remote-route to the external availability view of its introducer-local-route. In particular, we align their appearance and disappearance reports, and any subsequent forwarded discovery events, and we never align their discovery states.
- We never discard a non-periodic discovery advertisement message about the status of a remote-route.

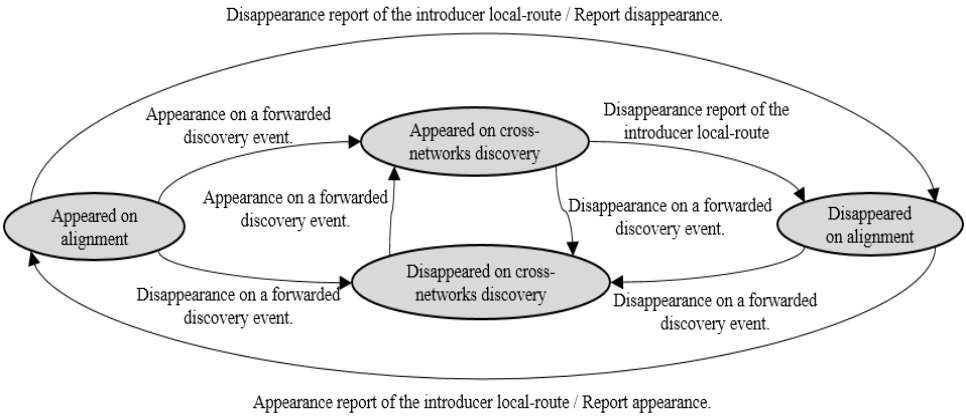

Fig. 4 Aligning remote-route to its introducer-local-route.

3.7 Implementation and evaluation

We evaluated an implementation of our proposed device discovery mechanism against an implementation of the simplistic approach, i.e. flooding heart-beat messages across networks. An analysis of the simplistic approach gives that the number of heart-beat messages seen by a $device_i$, N_i is:

$$N_i = \sum_{j \in K} n_{i,j} \qquad 1$$

Where $n_{i,j}$ is the number of devices on the $network_j$ disregarding $device_i$ and devices routing to it, summed over the entire set of interconnected PalCom networks, K. Notice that this expression do not cover situations with routing loops where the number of heart-beats seen can be even higher than the number of individual devices. In the proposed solution we calculate N_i as:

$$N_i = \sum_{j \in L} n_{i,j} \qquad 2$$

Where L is the set of networks that are local to $device_i$. By applying equation (1) to the simple scenario in Fig. 1 we get N_2=2, while by applying equation (2) to the same scenario we get N_2=1. In order to validate the analysis we made a matching setup. For each of the two cases, we recorded the heart-beat traffic on one of the networking interfaces, an IPv4 interface, of the router node. In this test setup, our optimized approach, Fig. 1 (b), was

measured to cut the heart-beat traffic in half when compared to the traffic in the simplistic approach, Fig. 1 (a), as expected.

4 Conclusions and future work

We proposed a device discovery mechanism for PalCom that uses discovery events notifications instead of flooding interconnected networks with periodic keep-alive messages. Also, our mechanism can detect device disappearance, i.e. undiscovery, in predictable configurable state time-outs. In the evaluation test setup, Fig. 1, our optimized approach reduces heart-beating traffic into half, as expected. Generally speaking, in the simplistic approach, a device processes heart-beating traffic from every single device in the set of interconnected networks. While in our approach, a device processes heart-beat traffic only from devices on its local network(s), while it processes discovery events notifications about devices on non-local networks from router nodes that belong to its local networks.

Currently our mechanism triggers events about all its visible devices to a just discovered node. We plan to optimize this mechanism by introducing a discovery information synchronization mechanism. Also, we plan other optimizations like selective polling of device using configurable heart-beats. Finally, we will evaluate our system against real-life use cases.

References

[1] Guttman E, Autoconfiguration for IP networking: enabling local communication, *Internet Computing, IEEE*, 2001, Vol. 5, No. 3, Pages 81-86.

[2] Postel J, User Datagram Protocol, STD 6, RFC 768, August 1980, available at http://www.rfc-editor.org/rfc/rfc768.txt.

[3] Svensson Fors D, Magnusson B, Gestegård Robertz S, Hedin G, Nilsson-Nyman E, Ad-hoc Composition of Pervasive Services in the PalCom Architecture, In: *Proceedings of the 2009 international conference on Pervasive services*, Pages 83-92. New York, NY, USA. ACM, 2009.

[4] Raychoudhury V, Cao J, Kumar M, Zhang D, Middleware for pervasive computing: A survey, *Pervasive and Mobile Computing*, 2013, Vol. 9, No. 2, Pages 177-200.

[5] Zhu F, Mutka MW, Ni LM, Service Discovery in Pervasive Computing Environments, *IEEE Pervasive Computing*, 2005, Vol. 4, No. 4, Pages 81-90.

[6] Edwards WK, Discovery Systems in Ubiquitous Computing, *IEEE Pervasive Computing*, 2006, Vol. 5, No. 2, Pages 70-77.

[7] Mian AN, Baldoni R, Beraldi R, A Survey of Service Discovery Protocols in Multihop Mobile Ad Hoc Networks. *IEEE Pervasive Computing*, 2009, Vol. 8, No. 1, Pages 66-74.

[8] Anwar FM, Seung-Wha Yoo, Ki-Hyung Kim, Survey on Service discovery for Wireless Sensor Networks. In: *Ubiquitous and Future Networks (ICUFN), 2010 Second International Conference on, IEEE*, Pages 17-21. Jeju Island, Korea (South). IEEE, 2010.
[9] Hedrick C, Routing Information Protocol, RFC 1058, June 1988, available at http://www.rfc-editor.org/rfc/rfc1058.txt.

Paper III: Supporting Distance Vector Routing over Device Discovery Flows in the Pervasive Middleware PalCom

Amr Ergawy and Boris Magnusson,

Published in Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies, ANT-2015, London, UK. Procedia Computer Science, volume 52, 2015, p. 153-160.

***Abstract**:* PalCom is a middleware system that provides different utilities to interacting devices pervasive environments. One of these utilities is device discovery that enables such devices to discover the existence of each and to establish an ad-hoc network among themselves. However, the device discovery mechanism does not provide a routing protocol for exchanging messages via the discovered ad-hoc network.

In this paper, we detail our solution to add support of distance vector routing to the PalCom stack. We build on top of our previous work of device discovery in PalCom, which uses periodic heart-beat messages for device discovery within a network and discovery event message for cross-networks device discovery. In turn, we avoid the need for route-queries, compared to on-demand ad-hoc routing protocols, and we avoid exchanging routing-table updates, compared to table-driven ad-hoc routing protocols. The proposed solution was implemented and integrated to the PalCom stack. Moreover, the solution is evaluated against a network with connections of continuously variable availability.

Keywords: distance vector routing; pervasive middleware; ad-hoc networking.

1 Introduction

PalCom is pervasive middleware that enables users to integrate the services different devices into assemblies of uses cases [1]. It provides a service discovery protocol that enables those devices to exchange their services descriptions. Using a device that can display such descriptions in a user friendly interface, a user can write, or compose, simple services assembly scripts that enable exchanging data and commands among those devices. We refer approach as a human or user in the loop system [1], where users are given more control on system configuration and control.

In [2], we defined the lowest level layer of the PalCom stack that enables third-party developers to write Media Abstraction Objects, MAOs, which abstract different types of network interfaces to upper PalCom layers. Moreover, in [3], we detailed a device discovery mechanism for PalCom that enables devices to discovery each other and to build an ad-hoc network among them. However, the supposed routing protocol among to a destination device in that network was to forward a message via the first discovered route, which is not a practical nor a realistic assumption. In turn, we need to add support of a more practical routing approach to PalCom, i.e. distance vector routing.

In this paper, we explain the design challenges, principles, options, decisions, and details of supporting distance vector routing in PalCom. We build our design on top of the device discovery mechanism that we proposed in [3]. That mechanism already provides an ad-hoc network among discovered devices, which saves a lot of signaling and control overhead for supporting distance vector routing compared to table-driven and on-demand ad-hoc routing protocols [4].

In section 2, we summarize our previous work on the PalCom stack to provide network media abstraction and device discovery. Also, we survey relevant routing protocols in ad-hoc networks to compare their required signaling and control overhead with the already provided device discovery information in the PalCom stack, which simplifies supporting distance vector routing in PalCom. Then, in section 3, we detail our design before we summarize its implementation and evaluation results in section 4. Finally we explain our conclusions and future work in section 5.

2 Previous Work

We build the support of distance vector routing in PalCom on top of our previous work of networking media abstraction and device discovery in PalCom [2][3]. Also, we compare the provided utilities by the device discovery mechanism to the required signaling and control overhead that is required by routing protocols in ad-hoc network, showing how the device discovery mechanism in PalCom simplifies the support of distance vector routing.

2.1 Device Discovery over Media Abstraction in PalCom

To make our design decisions for supporting distance vector routing in PalCom, we review the current state of *the PalCom stack*. At its lowest, we

define *the media abstraction layer* [2]. It enables the development and use of media abstraction objects, MAOs, that abstract different network protocols to PalCom upper layers. As a result, PalCom nodes can communicate over heterogeneous networks, which is a desired feature in pervasive systems.

On top of the media abstraction layer, we define the *device discovery layer* [3], which enables devices to discover each other and construct an overlay network that can be used by PalCom upper layer for service discovery and data messaging. As a result of the discovery process, each PalCom node maintains a *routing table* about the discovered devices. On a PalCom node, the next hop to a device is defined by *an introducer network interface* via which the device was discovered. In [3], trivial routing decisions are made to forward a message via the first discovered introducer interface to the destination device.

To support distance vector routing in PalCom, we need to extend the device discovery layer. In turn we review its components, which include *the discovery state machine*, *the discovery forwarding manager*, and *the discovery forwarding flows* [3]. The discovery state machine implements logic that enables a node to *discover devices on its local networks* as well as to process forwarded discovery events from devices on local networks about devices on other networks, i.e. *cross-networks device discovery*.

Within a local network, the discovery state machine, on a PalCom node, uses periodically broadcasted heart-beat messages from another node on the network to discover a *local-route,* defined by the pair of (network-interface, network-address), to that node and to keep track of its availability via that route [3]. On the event of appearance or disappearance of a device via a local route, that discovery state machine passes that event to the discovery forwarding manager, which notifies the discovery forwarding flows of the concerned device about that local-route discovery event.

From the perspective of a known device, the discovery forwarding flows arranges the network interfaces on a local PalCom node as *introducers* and *advertisers* of that device [3]. *A discovery forwarding flow, DFF,* is defined as a pair of the format (*an introducer interface, advertiser interfaces*). Device discovery events via the introducer interface are advertised over the advertiser interfaces of its discovery flow. At any point in time, a maximum of two discovery forwarding flows are maintained for a known device: *DFF1* that is defined as (the interface of the first discovered route to the device, the rest of network interfaces on the local node), and *DFF2* that is defined as (the interface of the second discovered route to the device, the interface of the first discovered route to the device).

PalCom discovery messages are composed of nodes [3]. A forwarded discovery starts with an S-node. Then it may include either a device status

node, an H-node, or a more extended device information node, an I-node. We indicate a PalCom message as a dash separated sequence of nodes. When a PalCom node receives forwarded S-H and S-I messages about a device that is not on its local networks, the discovery state machine on that node uses its cross-networks discovery logic to process these messages to discover and maintain a *remote-route* to that device. The local-route via which such S-H and S-I messages are received is called the *introducer-local-route* of that discovered remote-route.

The discovery state of a remote-route is overshadowed by the discovery state of its introducer-local-route [3]. The discovery forwarding manager implements the necessary logic that aligns the state of a remote-route to its introducer-local-route. That logic contributes to the decisions of forwarding discovery events to neighbor nodes, which are advertised by the discovery forwarding flows.

Next, we discuss the implications of the above mentioned details on supporting distance vector routing in PalCom.

2.2 Challenges and Design Options for Routing in Ad-hoc Networks vs. PalCom Stack Implications

In [3], we identified that devices in pervasive systems communicate over heterogeneous networks while they keep joining and leaving the pervasive environment, as users come and go. In turn, we referred to the communication support in mobile and ah-hoc networks to identify design challenges and options for different aspects of communication support for PalCom, as a pervasive middleware. Similarly, in the context of supporting distance vector routing in PalCom, we refer to the relevant protocols in mobile and ad-hoc networks [4].

From another perspective, we build our design on top of our previous work of device discovery [3] and networking media abstraction [2] in PalCom, which we summarized above in section 2. Those two layers impose their own assumptions on the resulting distance vector routing layer. This constraints our options for addressing identified design challenges.

One of the first challenges to distance vector routing protocols in ad-hoc networks is *route discovery* [4]. Handling this challenge resulted in categorizing these protocols into *table-driven protocols* and *on-demand protocols*. In table-driven protocols a node maintains a routing table with routes to every other node in the network [4]. This requires exchanging routing table updates among different nodes, which is criticized as an overhead for both communication and processing [5]. Examples of table-

driven protocols include Destination Sequenced Distance Vector Routing [6], DSDV, and its fault-tolerant improved version [7], I-DSDV.

In on-demand routing protocols, a sender node attempts to discover a route to a destination node only when there is a message to send [4]. Basically, a sender node broadcasts a route-request message that may be replied with the first intermediate node that has a route to the destination or may be replied by the destination itself. Different on-demand protocols use route-request and route-reply messages in different ways to set-up a route between a source node and a destination node. On-demand protocols are criticized for their route-setup latency [8]. Examples of on-demand routing protocols include Ad-hoc On-demand Distance Vector Routing [9], AODV, and its multipath version of Ad-hoc On-demand Multipath Distance Vector Routing [10], AOMDV.

In contrary to table-driven routing protocols in ad-hoc networks, distance vector routing over PalCom device discovery [3] does not need to exchange routing table updates. Also, in contrary to on-demand routing protocols, distance vector routing over PalCom device discovery does not need to implement a route discovery stage. The PalCom device discovery mechanism ensures that a device continually gets event-based updates about the availability of other devices via different network interfaces. In turn, each node maintains its routing table internally, without the need for exchanging routing table updates or running a route discovery mechanism.

Another challenge to distance vector routing protocols in ad-hoc networks is the need *to flood a network with control messages* [11] [12]. E.g. on-demand routing protocols may flood a network with route-request messages. To address this challenge, these protocols may employ a multi-point relaying approach that selects only a specific sub-group of a node neighbors to forward a broadcast message [11]. Alternatively, these protocols may employ a neighbor assisted approach that utilizes information about the destination recent neighbors to forward route-request messages [12].

Compared to these approaches, PalCom device discovery already limits broadcasting control messages within the boundaries of local networks while it forwards device discovery events across those boundaries [3]. In turn, a distance vector routing protocol over PalCom device discovery does not need to implement a separate mechanism to limit control messages flooding.

A third challenge that distance vector routing protocols in ad-hoc networks need to address is *the detection of, and overcoming, broken links* [4]. Usually, these protocols utilize information from the underlying Media Access Layer, MAC, protocols to detect broken links and reduce duplicate control messages that may be broadcasted to overcome them [7] [8] [9]. Oppositely, a distance routing protocol in PalCom cannot make use of MAC layer information as

networking protocols are abstracted by the media abstraction layer [2], as we explained above in section 2.

Alternatively, device discovery in PalCom [3] provides a device undiscovery mechanism that enables detecting the disappearance of a device via a specific network interface, possibly indicating a broken link. Additionally, the PalCom media abstraction layer [2] provides the ability to reach a device via different networking interfaces. Combining this with device discovery, a routing protocol in PalCom can overcome a broken link.

3 The Proposed Support For Distance Vector Routing in PalCom

In principle, in order to support distance vector routing, a source node or a router node forwards a data message to a destination device via a network interface that represents the lowest cost route to that device [13]. In our proposal for distance vector routing in PalCom, we use the number of hops between two nodes as the route cost between them. Devices need to exchange route cost updates among each other while they build and maintain their routing tables as part of the device discovery process in PalCom [3], which we summarized above in section 2. In turn, wherever necessary in the PalCom discovery process, we need to convey route cost updates in the exchanged discovery messages.

We assume the route cost between two devices in a local network to be a single hop. In turn, the exchanged messages in a PalCom device discovery sequence within a local network do not need to convey any route cost updates. However, the exchanged messages in a cross-networks PalCom device discovery sequence must convey the accumulated route cost to the discovered device. That accumulated route cost is calculated while a device discovery event is being forwarded, as a discovery message, from a node to another across interconnected networks.

In particular, the main goal of our design is to extend one of the PalCom stack components [3] that are involved in the device discovery process to:

- *Maintain route cost based natural ordering of the routes set to a known device*, reflecting route cost updates that are conveyed in discovery messages that are received via introducer routes to that known device.
- *Report route cost updates to neighbor nodes* based on that natural ordering of the introducer routes.

As summarized above in section 2, the PalCom stack components that are responsible for device discovery are the discovery state machine, the discovery forwarding manager, and the discovery forwarding flows. We need to choose the one of these components with the logic that extensible to maintain a route cost based natural ordering among the elements in the routes set of a known device.

Ideally, the logic of the chosen component needs to be already designed to maintain some relation among the routes to a known device. We refer to such logic as *routes-set oriented*. We distinguished that logic from *device-state oriented* logic that makes decisions only based on device appearance, disappearance, and status updates. Extending a component with only device-state oriented logic to support distance vector routing will increase its complexity and decrease its modularity.

In this section, we start the discussion by analyzing the discovery state machine, the discovery forwarding manager, and the discovery forwarding flows [3], summarized above in section 2, to identify which of them has the most routes-set oriented logic and the least device-state oriented logic. Then, we emphasize our design principles that focus on preserving the discovery state consistency among the PalCom stack components, the separation of concerns among them, and minimizing the routes changes that result from triggered discovery events and route cost updates. Finally, we detail our design.

3.1 Design Options

As mentioned above in this section, our criterion is to choose the component that already implements routes-set oriented logic more than device-state oriented logic to be extended for the support of distance vector routing.

1) Analyzing the discovery state machine: We start by considering the discovery state machine, as the first component of the PalCom stack that is responsible for device discovery [3]. As mentioned in section 2, it implements both the local discovery logic as well as the cross networks discovery logic. By analyzing that logic, we find that the discovery state machine makes its decisions, about updating the routing table and/or triggering a device discovery event, based on the combination of:

   a) The parameters of the concerned device that are conveyed in an input device discovery message.
   b) The current state of the source route, to the concerned device, via which the input message was received.
   c) The current values of the parameters of the concerned device.
   d) The collective state of the concerned device via its routes set.

It is clear that the discovery state machine does not maintain any relation among the elements in the routes set of a known device and it makes it decision only based on the device state. In turn, the discovery state machine is more device-state oriented than routes-set oriented. In turn, we exclude it as a candidate for the update to support distance vector routing.

2) Analyzing the discovery forwarding manager: Alternatively, considering the discovery forwarding manager [3], summarized in section 2, it implements the logic that triggers forwarding discovery events to neighbor nodes and aligns remote routes to their introducer local routes. By analyzing that logic, we find that the discovery forwarding manager makes its decisions, about triggering discovery event reports to neighbor nodes, based on the combination of:

   a) A device discovery event that was triggered by the discovery state machine. If that event originated from a local route, then the concerned device is on the local network and it may function as a router to discovered devices on remote networks. Otherwise, if that event originated from a remote route, then the concerned device is a discovered device on a remote network.

   b) The latest forwarded discovery event to neighbor nodes about the concerned device.

   c) In the case of a local concerned device, the decision considers the current state of the remote routes that are discovered via the local route via which the discovery event was received.

   d) In the case of a remote discovered concerned device, the decision considers the state of the local route that introduced the remote-route via which the discovery event was received.

The logic of the discovery forwarding manager considers the state of the event concerned device as well as the relation among the states of associated local and remote routes. Those routes are routes to different devices. In turn, the logic of the discovery forwarding manager is not oriented to the routes-set of a given device and we exclude it as a candidate for supporting distance vector routing.

3) Analyzing the discovery forwarding flows: Finally, considering the discovery forwarding flows [3], summarized in section 2, it implements the logic that arranges the network interfaces on the local node into flows via which triggered discovery events are forwarded to neighbor nodes. It makes its decision, about forwarding a triggered discovery event to neighbor nodes, based on the combination of:

   a) The temporal order of the discovery of the introducer routes to the concerned device. In particular, the interface of the first discovered route to that device becomes the introducer of the

first forwarding flow, which advertises discovery events via all the network interfaces on the local machine except the interface of that first discovered route. Then, the interface of the second discovered route, to the concerned device, functions as the introducer of the second forwarding flow, which advertises discovery events only via the interface of the first discovered route.

b) The current set of network interfaces on the local node.

To sum up, the logic of the discovery forwarding flows maintains a relation among the interfaces of the local node that are associated with routes to a known device. Such logic can be modified to be routes-set oriented. In turn, we decide to extend the discovery forwarding flows with the necessary logic to support distance vector routing in PalCom. Next we emphasize the design principles that we build for that extension.

3.2 Design Principles and Properties

In this section, we list and discuss the three design principles that we follow to specify the details of extending the discovery forwarding flows to process and advertise route cost updates about a known device.

1) Discovery state consistency: As summarized in section 2, the discovery forwarding flows advertise messages that reflect the discovery state of a known device as seen by the advertiser PalCom node, regardless of whether the node functions as router. That reflection ensures the state consistency between the discovery state machine and the discovery forwarding flows on that node. Our design must preserve such consistency.

In particular, after extension to process and advertise route cost updates, the discovery forwarding flows must maintain two properties:

a) To only function as a processor of discovery state events that are triggered from the state machine and refined by the discovery forwarding manager.

b) To never influence or create discovery state events.

2) Separation of concerns: As we explained in section 2, the discovery forwarding flows separate the discovery advertisement logic from the device discovery logic, which is implemented by the discovery state machine and the discovery forwarding manager. Thus, it provides a clear separation of concerns between advertising discovery events and triggering them.

Such separation of concerns minimizes the complexity of the discovery forwarding flows data structure. In turn, extending the discovery forwarding flows to process and report route cost updates shall produce a

PalCom stack with reasonable modularity and acceptable complexity. The extension of the discovery forwarding flows, to process and advertise route cost updates, must preserve the separation of concerns by maintaining two properties:

a) To only maintain the relations among interfaces on the local node and routes to discovered devices, which are necessary to advertise discovery events and route cost updates.
b) To never implement or influence device discovery logic, which is implemented by the discovery state machine and the discovery forwarding manager.
c) Minimizing route dynamics: In addition to considering the internal details of the local PalCom node, we need to consider how the changes to the roles of its network interfaces, as discovery flows introducers and advertisers of known devices, affect the stability of routing decisions that are made by the neighbor nodes of that local node.

As we summarized in section 2 and discussed in section 3.1, at any time a PalCom node maintains only two discovery forwarding flows for advertising discovery events about a known device [3]. Assuming a static set of network interfaces on the local PalCom node, for a single discovery event about a known device, the mechanism of maintaining these two discovery forwarding flows ensures a maximum of two interfaces to change their roles with respect to that known device, as introducers and advertiser.

The case of the maximum changes in interfaces roles happens only when a discovery event indicates the disappearance of a known device via one of its two introducer interfaces, those of its first and second discovered routes, and the interface of the third discovered route to the known device currently functions as an advertiser. Those two interfaces swap their roles. We shall consider this maximum number of changes in the interfaces roles as a best case that our design shall not go much beyond it.

Additionally, in our design, we must consider the implications of supporting a dynamic set of network interfaces on a PalCom node, i.e. interfaces that can be enabled and disabled by the user. Basically, a newly enabled interface can be considered as an advertiser for every known device. On the other hand, for a known device that is discovered via more than one interface, disabling an introducer interface may at most result in one advertiser interface changing its role to be an introducer. That is the interface of the third discovered route to that device.

To be more specific, our modifications of the discovery forwarding flows to process route cost updates shall minimize the changes in routes by maintaining two properties:

a) To keep the maximum number of changes in interfaces roles, as introducers and advertisers, per discovery event as close as possible to the above mentioned maximum of two.
b) To keep minimal changes to interfaces roles per interface enable or disable event.

To sum up, we modify the discovery forwarding flows to process and advertise discovery events and route cost updates while preserving the discovery state consistency among the PalCom stack components, maintaining the separation of concerns among the logic of these components, and minimizing the routes dynamics that result from changes in interfaces roles. Next, we details our design.

3.3 Design Details

In this section, we start to detail our design by defining the necessary modification to the format of the discovery message types that will convey route cost values. Then, we specify which PalCom stack components are responsible for updating route entries, in the routing table, with the received route cost updates. Afterwards, we define the interface between the discovery forwarding flows and the other stack components. Finally, we detail the mechanism of the discovery forwarding flows that maintains the route cost based natural ordering of the routes set to a known device, and the advertisement of route cost updates and discovery events over these forwarding flows.

1) Discovery messages formats and routing table updates: As we summarized in section 2, PalCom discovery messages are composed of nodes. Moreover, we define a message type as a dash separated sequence of nodes, where a node is symbolized as a capital letter. To convey route cost values, we define a new node, symbolized as C. If the local PalCom node receives a discovery message that triggers a device discovery event, then a C-node is inserted between the two nodes of S-H message or the S-I message that is broadcasted to advertise that event [3], which we explained their triggering in section 2. If the received discovery message triggers no device discovery event, then only an S-C message is broadcasted to advertise the route cost update.

Moreover, according to *the separation of concerns* design principle of, the discovery forwarding flows cannot modify the routing table to update routes entries with route cost updates. Instead, we consider the route cost to be a route entry parameter that is updated by the discovery state

machine and the discovery forwarding manager, similarly to other route parameters.

2) Defining the interface of the discovery forwarding flows: Before detailing the internal mechanism of the modified discovery forwarding flows, we define its interfacing to the discovery state machine and the discovery forwarding manager. In particular, we specify which of these components shall be interfaced, the necessary procedures for such interface, and the parameters that we need to pass.

As explained in section 2, and conforming to the design principle of discovery state consistency, a discovery event disseminates from the discovery state machine, to the discovery forwarding manager, and finally to the discovery forwarding flows.

Also, we specified above that only the discovery state machine and the discovery forwarding manager may update a route cost update to a route entry, in the routing table. From all of the above, we can preserves the current design properties where:

a) The discovery forwarding flows needs only to be interfaced to the discovery forwarding manager.
b) Among the parameters of that interfacing, we may only pass route entries with already updated route cost values.

Moreover, as we explained before, the modified forwarding flows aims to maintain a route cost based natural ordering of the routes to a known device and arrange the interfaces on the local PalCom node as introducers and advertisers of that device. That behavior specifies the interface procedures of the discovery forwarding flows to be only route and interface oriented. These procedures need to handle the events of adding, deleting, and updating the routes to the concerned known device and the interfaces on the local node.

We define these procedures in table I, below. At the end of these procedures, they call a procedure to update the discovery forwarding flows and advertise discovery events. That procedure is called *update_advertisements*, and it take two parameters, that we describe as follows:

a) The first parameter is the type of the message node that indicates a device appearance or update that will be advertised via interfaces that become advertisers after updating the **d**iscovery forwarding flows. If a caller procedure, from those in table I, does not specify a value for this parameter, we assume a default of an information-reply node, I. We refer to that value as default-appearance-or-update-node.

b) The second parameter is the type of the message node that

Table 1. Procedures of the discovery forwarding flows, continued on next page.

Event		**Discovery Flow Element**	
Name	***Details***	***Introducer route***	***Network interface***
Add	Parameter	A newly discovered route to a known device, with updated route cost.	An enabled network-interface on the local node.
	Processing	For the conerned known device: 1. Add the parameter route as a introducer to the *route cost based sorted set* of the device introducer routes. 2. Call update_advertisements(default-appearance-or-update-node , default-disapperance-node).	For every known device: 1. Add parameter interface as an advertiser to the device lowest cost discovery flow. 2. Boradcast an S-C-I message about that knonw device via that interface.
Remove	Parameters	Two parameters are required for this procedure: 1. An introducer route via which a known device has disappeared. 2. The node type in the advertisement discovery message that indicates the reason of the device disapperance via this route. Only two values are expected, an interface-closed node, Y, or a heart-attack node, X.	A disabled network-interface on the local node.
	Processing	For the conerned known device: 1. Remove the parameter route from the *route cost based sorted set* of the device introducer routes. 2. Call update_advertisements(default-appearance-or-update-node , X or Y node).	For every knonw device: 1. If the parameter interface is an advertiser in any of that device two discovery forwarding fllows, remove it from that flow. 2. If parameter interface is of an introducr router in any of that device two discvoery forwarding flows, remove that route from the discovery frowarding flows. 3. Call update_advertisements(default-appearance-or-update-node , default-disapperance-node).

indicates a device disappearance via non-introducer interfaces

c) that are just detache d from the discovery flow of a disappearing introducer, after updating the discovery forwarding flows. If a caller procedure, from those in table I, does not specify a value for this parameter, we assume a default of an interface-closed node, Y. we refer to this value as default-disappearance-node.

3) Maintaining the route cost based sorted routes set and the discovery forwarding flows: As we summarized in section 2, we define a discovery forwarding flow as the pair (introducer route, advertiser interfaces) [3]. Accordingly, for a known device, we define the two maintained route cost based discovery forwarding flows as follows:

a) DFF1 = (the local node interface of the lowest cost route to the known device, the rest of the local node interfaces).

b) DFF2 = (the local node interface of the 2nd lowest cost route to the known device, the local node interface of the lowest cost route to the known device).

As explained above, for a known device, the discovery forwarding flows

Cont. table 1. Procedures of the discovery forwarding flows.

Event		**Discovery Flow Element**	
Name	***Details***	***Introducer route***	***Network interface***
Update	Parameters	Three parameters are required for this procedure: 1. An intoducer route via which the parameters of a known device and/or the route cost to it has been updated. 2. The node type in the advertisement discovery message that indicates the type of the device parameters update via this route. Only three values are expected, a heart-beat-reply node, H, an information-reply node, I, or a Null to indicate only route cost update and no device parameters update. 3. A boolean to indicate if the discovery state machine and the discovery forwarding manager has updated the route cost of that introducer route.	N/A
	Processing	For the concerned known device: 1. If the third parameter indicates an updated route cost, resort the *route cost based sorted set* of the device introducer routes. 2. Call update_advertisements(H, I, or Null , default-disapperance-node).	

maintain *a route cost based sorted set of the discovered routes to that device*. On a call from one of the procedures in table I, the sorted routes set is used to maintain the discovery forwarding flows as we show in Fig. 1, below.

From Fig. 1, we can see that the design principle of *minimizing routes dynamics* is achieved as the maximum number of changes in interfaces roles remains the same as specified in section 3.2 of this section. In particular, in case both of the discovery forwarding flows remain to exist after their maintenance, a maximum of two interfaces may change their roles, as advertisers and introducers of the concerned known device.

4 Implementation And Evaluation

We implemented our solution as part of the Java implementation of PalCom. We used the test setup in Fig. 2-a to evaluate routing decisions of a message stream

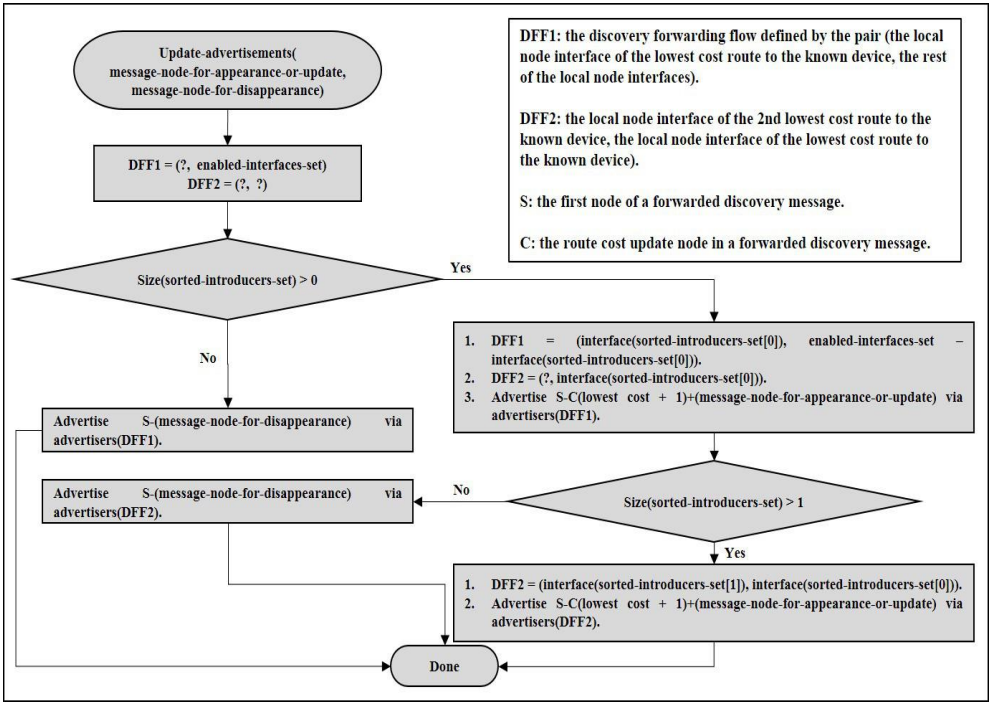

Figure 1. Updating discovery forwarding flows and advertisements.

with a periodic sending rate of 1 second from device-1 to device-2 in the condition of changing routes availability. At the start of the test, routes from device-1 to device-2 are available by enabling the connections via LAN-1, LAN-2, and LAN-4. Then, during a test round, we use an automated script to disable LAN-1, LAN-2, and LAN-4 respectively with a period of 15 seconds between each of these actions. Afterward, in the same round, we re-enable those networks in the reverse order. In Fig. 2-b, we show the sequence of received test messages on device-2 during three test rounds. The support for distance vector routing in PalCom is proven to be working properly in response to changing networking connectivity.

5 Conclusion and Future Work

Supporting distance vector routing in PalCOm over device Discovery Forwarding

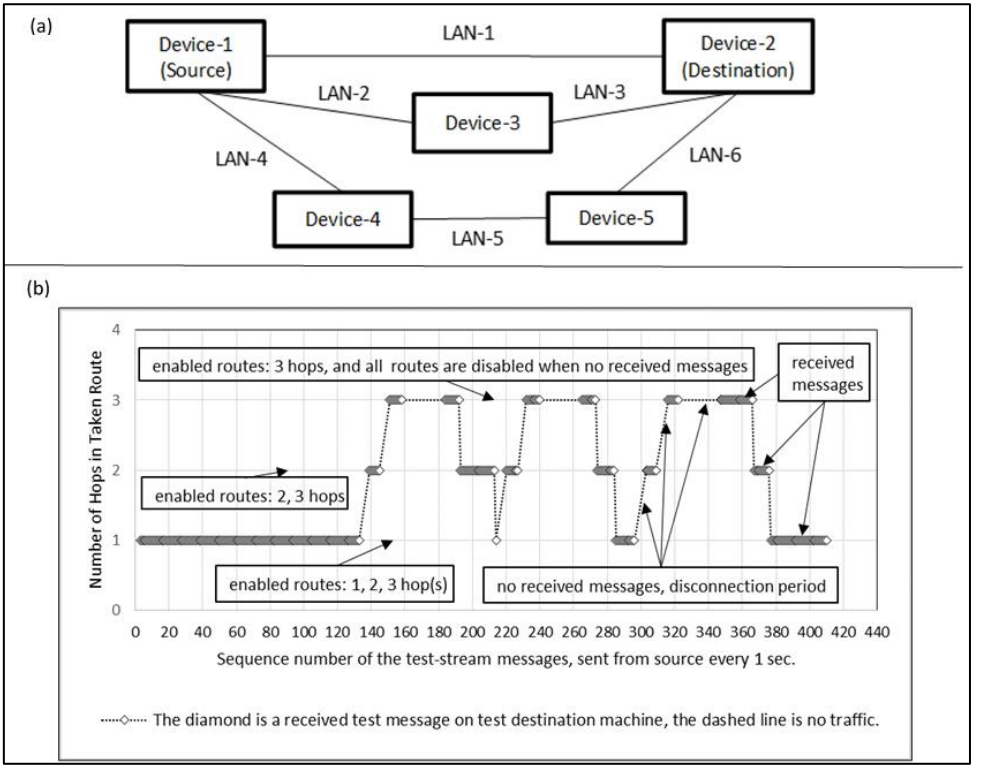

Figure 2. Test setup and results of evaluating the implemented support of distance vector routing in PalCom.

Flows, DFF, avoids the need for specialized signaling and control messages compared to table-driven and on-demand ah-hoc networking protocols [4]. In particular the PalCom device discovery mechanism already provides the necessary autonomous management of the ad-hoc network of the discovered PalCom devices. The proposed solution was integrated into the Java implementation of PalCom. The evaluation results show that correct and timely routing decisions are made by the system in response to changes of the availability of network connects.

As a future work, we plan to enhance the device discovery mechanism itself, the basis of the proposed distance vector routing, by adding a revision system of the routing table contents that enables re-synchronizing discovery information among PalCom devices after temporary disconnections that last below the configured heart-beat periods. Moreover, we plan to review the device discovery mechanism to prevent a noticed problem where a router node may forward an appearance discovery event about a routing route to a just disappeared neighbor node while it has been the only gateway to that disappeared neighbor node.

References

[1] D. Svensson Fors, B. Magnusson, S. Gestegård Robertz, G. Hedin and E. Nilsson-Nyman, "Ad-hoc Composition of Pervasive Services in the PalCom Architecture", in Proceedings of the 2009 international conference on Pervasive services, London, 2009.

[2] Amr Ergawy and Boris Magnusson, “Media Abstraction Framework for the Pervasive Middleware PalCom” in Proceedings of the 2nd International Conference on Future Internet of Things and Cloud, FiCloud-2014, Barcelona, Spain, 2014, in press.

[3] Amr Ergawy and Boris Magnusson, “Device Discovery for the PalCom Pervasive Middleware with Eliminated Cross-networks Periodic Heart-beat Messages”, Procedia Computer Science, Volume 37, 2014, Pages 64-71, ISSN 1877-0509.

[4] Ghazani, S.H.H.N.; Lotf, J.J.; Alguliev, R.M., "A new survey of routing algorithms in ad hoc networks," Computer Engineering and Technology (ICCET), 2010 2nd International Conference on , vol.3, no., pp.V3-684,V3-688, 16-18 April 2010, doi: 10.1109/ICCET.2010.5485743.

[5] Yongjun Hu; Tao Luo; Junliang Shen, "An Improvement of the Route Discovery Process in AODV for Ad Hoc Network," Communications and Mobile Computing (CMC), 2010 International Conference on , vol.1, no., pp.458,461, 12-14 April 2010, doi: 10.1109/CMC.2010.123.

[6] Charles E. Perkins and Pravin Bhagwat. 1994. “Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers”. In Proceedings of the conference on Communications architectures, protocols and applications (SIGCOMM '94). ACM, New York, NY, USA, 234-244. DOI=10.1145/190314.190336.

[7] Ting Liu; Kai Liu, "Improvements on DSDV in Mobile Ad Hoc Networks," Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on , vol., no., pp.1637,1640, 21-25 Sept. 2007, doi: 10.1109/WICOM.2007.412.

[8] Fang Xie; Lei Du; Yong Bai; Lan Chen, "A Novel Multiple Routes Discovery Scheme for Mobile Ad Hoc Networks," Communications, 2006. APCC '06. Asia-Pacific Conference on , vol., no., pp.1,5, Aug. 2006, doi: 10.1109/APCC.2006.255962.

[9] Perkins, C.E.; Royer, E.M., "Ad-hoc on-demand distance vector routing," Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on , vol., no., pp.90,100, 25-26 Feb 1999, doi: 10.1109/MCSA.1999.749281.

[10] Marina, M.K.; Das, S.R., "On-demand multipath distance vector routing in ad hoc networks," Network Protocols, 2001. Ninth International Conference on , vol., no., pp.14,23, 11-14 Nov. 2001, doi: 10.1109/ICNP.2001.992756.

[11] Po-Jen Chuang; Po-Hsun Yen; Ting-Yi Chu, "Efficient Route Discovery and Repair in Mobile Ad-hoc Networks," Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on , vol., no., pp.391,398, 26-29 March 2012, doi: 10.1109/AINA.2012.120.

[12] J. Gomez, V. Rangel, M. Lopez-Guerrero, and M. Pascoe. 2011. NARD: Neighbor-assisted route discovery in MANETs. **Wirel. Netw.** 17, 8 (November 2011), 1745-1761. DOI=10.1007/s11276-011-0375-2.

[13] Hedrick C, Routing Information Protocol, RFC 1058, June 1988, available at http://www.rfc-editor.org/rfc/rfc1058.txt**.**

Paper IV: Synchronizing device discovery on loss of update messages in the pervasive middleware PalCom

Amr Ergawy and Boris Magnusson,

Published in Proceedings of the 11th International Conference on Future Networks and Communications (FNC 2016). Montreal, Canada, 2016. Procedia Computer Science, Volume 94, 2016, Pages 347-352.

Abstract: PalCom is a pervasive middleware that enables users to combine the services on devices into useful configurations. Interconnected PalCom devices can discover, and keep track of, the existence of each other by exchanging periodic heartbeats within local networks, and once-sent device appearance/disappearance notifications across interconnected networks. This approach has the advantage of eliminating the need to forward periodic heartbeats beyond the boundaries of the local networks of their originator devices. However, when a device appearance/disappearance notification is sent only once over an unreliable channel, there is a possibility of losing that notification. As a result, the device discovery information on PalCom devices will be out-of-sync. In this paper, we present the design, model-based evaluation, and the implementation status of a solution to this synchronization problem.

Keywords*:* pervasive middleware; device discovery; synchronization; model checking

---

1 Introduction

PalCom is a pervasive middleware that enables users to combine the services of their devices into useful application scenarios[1]. In particular, PalCom can be deployed onto user devices to provide the necessary utility for advertising device existences to other devices on the same local networks, as well as across different interconnected networks[2,3,4]. Once interconnected devices discover the existence of each other, they can exchange description of their functionality, i.e. services[1].

PalCom provides the necessary abstraction for devices to communicate via different networking technologies[2]. On top of that abstraction, PalCom uses periodic heartbeats within the boundaries of local networks to enable devices to discover and keep track of each other[3]. For cross-networks device

discovery, PalCom uses once-sent device appearance/disappearance notifications[3]. This approach saves network bandwidth by eliminating the need to forward periodic heartbeats outside the local networks of their originator devices. However, if such appearance/disappearance notifications are sent over unreliable channels, then PalCom nodes that lose a notification will lose synchronization of their exchanged cross-networks device discovery information.

In this paper, we present the design, model-based evaluation, and the implementation status of a reliability feature for PalCom that enables synchronization of device discovery information on the event of losing discovery updates. In section 2, we summarize our previous work on networking and device discovery for PalCom, illustrating the need for a device discovery synchronization mechanism. Then, in section 3, we survey reliability in distributed systems to inspect different design options for the required synchronization support in PalCom. Afterwards, in section 4, we present our proposed synchronization mechanism, modelled using UPPAAL[9]. In section 5, we present a performance evaluation of the algorithm. We conclude with the implementation status of the algorithm and our future work.

2 Previous work and problem statement

PalCom is designed as a layered distributed system[2,3,4]. In this section, we review the lowest two layers of PalCom that provide the communication utility and state machine logic for device discovery among interconnected devices. We conclude this section by the problem statement of this paper, namely the need for a synchronization mechanism to overcome the loss of once-sent device discovery notification on unreliable channels.

2.1 Cross networks communication support in PalCom

PalCom enables devices to communicate via different types of networking technologies by defining a networking media abstraction layer, MAL, as its lowest layer[2]. The core of the MAL layer is to view a network protocol as a composition of a device addressing method, a connection establishment mechanism, a byte array encoding/decoding method, and a network interfacing utility. By inheriting and extending this framework, a PalCom developer can easily add a new connectivity driver to support a new

networking protocol. The plugging for such connectivity driver to PalCom is done in a way that considers different sides of system reliability and performance[2].

2.2 Device discovery in PalCom

On top of the MAL layer, PalCom defines the device discovery layer[3]. This layer defines two state machines, namely the local-network device discovery state machine and the cross-networks device discovery state machine. A PalCom device applies the logic of the local-network state machine to discover and keep track of other PalCom devices on its connected local networks[3].

In particular, PalCom device-A periodically broadcasts heartbeat messages via all its network interfaces on available local networks. Then, PalCom device-B that receives a heartbeat from device-A, on the same local network, starts to exchange device information request and response messages with device-A. On the successful completion of device information exchange, PalCom device-B declares device-A as visible. From that point in time, device-B may change the availability state of device-A between visible, out-of-reach, rebooted, and gone based on configured time-outs and received periodic heartbeats from that device.

Moreover, when device-B discovers device-A on a local network that connects them, device-B will advertise the appearance of device-A on other networks that it is connected to[3]. To prevent looping device discovery advertisements in a network of PalCom devices, device-B manages forwarding device discovery notifications using a data structure called discovery forwarding flow[3], DFFs. The idea behind this data structure is to prevent forwarding a device discovery notification via the network interface that it was received from, a reused concept from conventional internet protocols[3]. Similarly, device-B may forward a disappearance notification of device-A on all other local networks after a specific time-out of not receiving heartbeats from that device. When device-C receives from device-B appearance/disappearance notifications about device-A, it uses the cross-networks device discovery state machine to process this notification to keep track of the availability of device-A.

2.3 The problem of once-sent device discovery notifications over unreliable channels

When sending cross-networks device appearance/disappearance notifications over unreliable channels, there is a possibility of losing these messages. In the above discussed example in section 2.2, if a discovery

notification message from device-B to device-C about device-A is lost, then device-C will be out-of-sync from the world view that device-B meant to update it with. This will cause communication issues to the services on device-C and other PalCom devices that are connected to device-C, but not connected to device-B or device-A.

We view this problem as a reliability problem of the PalCom device discovery mechanism. And to handle it, we need a synchronization mechanism that recovers the world view on device-C whenever it detects the loss of a discovery update message from device-B. In the next section, we review reliability support in distributed systems while focusing on possible design options for the required discovery synchronization mechanism for PalCom.

3 Reliability in distributed systems and design options

As discussed above, we designed PalCom as a layered distributed system[2,3,4]. In this section, we review reliability requirements and approaches in distributed systems. We focus on different approaches of fault detection and tolerance for communication/networking faults. Below, we compare these approaches to the current status of PalCom, aiming to make proper design decisions for the required device discovery synchronization mechanism.

3.1 Reliability requirements in distributed systems vs. communication/networking faults

Among several requirements for distributed systems, reliability comes as an essential one[5,7]. It aims at ensuring system functionality, quality and scalability. A reliable distributed system must provide a specific probability of successful performance of its intended function for sufficient time periods to meet users' expectations[5]. Moreover, a reliable distributed system is required to properly operate under the specific conditions of its deployment environment and resources[5]. Among several challenges to meet these requirements, the reliability of the used communication/networking infrastructure comes as a very important factor to consider[5,7]. Fault tolerant and recovery techniques are required to overcome reliability issues at such low-level utilities.

3.2 Requests redirection reliability vs. architectural based reliability

The simplest way of failure recovery at the networking levels is to redirect operations to other resources[5,7]. However, such an approach is not suitable to recover lost device discovery messages in PalCom, because such messages represent only the perspective of their originator node. Alternatively, we can employ an architectural and state based approach[5] to add support for device discovery synchronization in PalCom.

The main challenge to applying an architectural based reliability is the need to plan it before application development[5], which is not the case for PalCom. A recommended approach to start such a design is to preserve the layered system structure in a way that keeps hiding communication and networking from the reliability module[5]. We apply this recommendation to design a synchronization mechanism that fits into the layered architecture of PalCom.

3.3 Time-out based failure detection vs. sequence number based failure detection

To design reliability into device discovery in PalCom, we need to consider failure detection[6,7]. A proper failure detector is expected to provide a specific time after which a failure is detected while a properly functional process is not wrongly detected as a failed one[6]. To support device discovery synchronization on lost update messages in PalCom, we need to complement its time based device discovery failure detection[3] with a mechanism that keeps track of the sequence of update messages. This is a packet loss failure detection approach[7].

To keep track of the sequence of update messages, we re-use the concept of update round numbers[8] in our proposed synchronization mechanism for PalCom. However, instead of using update round numbers to implement a lock-step synchronizer[8], we implement a synchronizer that can handle a continuously changing ad-hoc network of PalCom devices[3]. We detail our design in the next section.

4 The proposed algorithm for synchronizing device discovery on lost update messages

As shown in Fig. 1 and Fig. 2, we model our proposed synchronization algorithm in UPPAAL[9]. Our design has four components, we list them while describing the algorithm functionality:

- *Sequence number based fault detection*: Every heartbeat message, an h/H-message[3], from device-A to device-B contains an update-number field. As shown in Fig. 1, if device-B finds that the (h/H).update-number is not equal to the latest update-number that it has received from device-A via the network interface that received the h/H-message, then device-B concludes that it is out-of-sync with device-A. Every appearance or disappearance notification from device-A to device-B about device-*x*, an S-*-message[3], has two fields: previous-update-number and update-number.

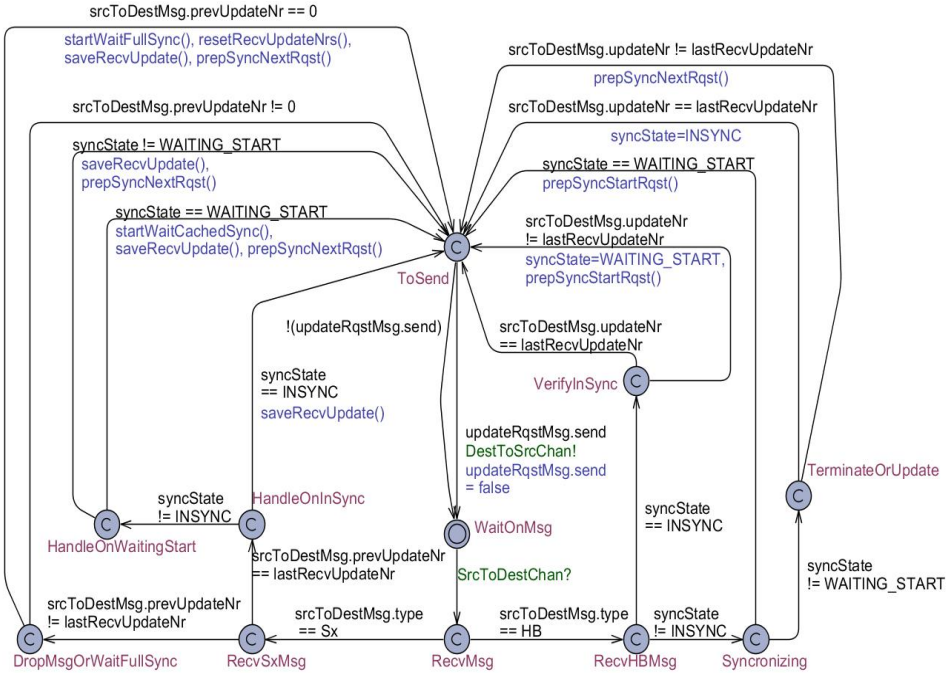

Fig. 1. The UPPAAL model of the destination node of the synchronization algorithm.

Device-B uses a received (S-*).previous-update-number in the same way as it uses a received (h/H).update-number to detect whether it is out of sync with the sender device-A.

- *Destination-device driven request/response synchronization*: When device-B detects that it is out-of-sync with device-A via one of its network interfaces, it initiates the synchronization process, as shown in Fig. 1. In particular, device-B declares all remote routes discovered via that interface as out-of-sync, and sends a sync-start request, a k-message, to device-A. Fig. 2 illustrates how device-A handles such message to respond with a relevant S-*-message to device-B.
  Fig. 1 illustrates how device-B handles a reply to its sync-start message. Device-B may ask for more updates by sending a sync-next request to device-A, a K-message. In Fig. 1, we illustrate that device-B terminates the synchronization process when a received (h/H).update-number equals its remembered latest-update-number from device-A via its network interface that received the h/H message.
- *Limited cache of continuous device appearance/disappearance history*: As illustrated in Fig. 2, the synchronization source, device-A, maintains a limited size cache of complete device-*x* appearance/disappearance history. If device-B, the synchronization destination, asks for updates within this cached history, then a smaller number of messages will be required for the synchronization process.
- *Synchronization reliability over periodic heartbeat messages*: As shown in Fig. 1, device-B, the synchronization destination, uses h/H-messages to

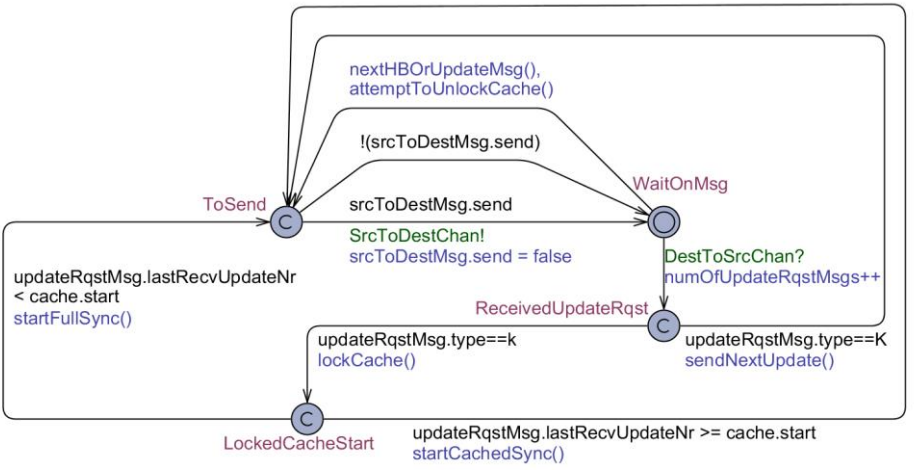

Fig. 2. The UPPAAL model of the source node of the synchronization algorithm.

make sure that sync-start and sync-next message are sent as much as required to the synchronization source device-A. This feature overcomes any lost synchronization request/response messages from device-A.

5 Model-based performance evaluation

We used the UPPAAL[9] model of our synchronization algorithm to evaluate its performance. We use two different parameters to evaluate the performance of the algorithm:

- *Cache capacity*: the number of appearance and disappearance updates that the source node remembers.
- *Out-of-sync detection sensitivity*: the number of sent discovery update messages before sending a heartbeat message. This is a definition of the *heartbeat frequency* in terms of the number of discovery update messages.

We measure *the synchronization overhead* as the average number of sync-request messages per a lost message. The input to the performance evaluation is composed of 50 message sequences with 20 messages each and randomly chosen dropped messages. These message sequences are grouped into five groups based on the ratio of dropped messages. These groups are randomly specified to contain 2/20, 5/20, 6/20, 8/20, or 10/20

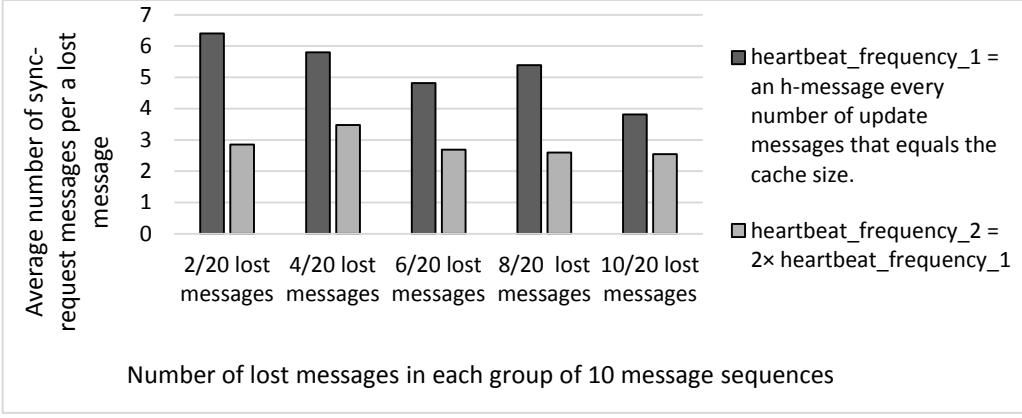

Fig. 3. The algorithm performance with two different configurations of the heartbeat frequency.

dropped messages, which are extreme message drop rates of 10% to 50%. We run the performance evaluation against two configurations of the algorithm. The first test configuration is: *heartbeat_frequency_1* = an h-message every number of update messages that equals the cache size. The second test configuration is: *heartbeat_frequency_2* = *2*× *heartbeat_frequency_1.*

As shown in Fig. 3, the test run with *heartbeat_frequency_2* produces less synchronization overhead compared to that with *heartbeat_frequency_1*. We interpret this as the more frequent the heartbeat messages compared to the update messages, the earlier the out-of-sync status is detected, and the more cached synchronization is performed. Also, for the test run with *heartbeat_frequency_2*, the majority of cases does not require more than three sync-request messages per a lost message. This is a relevant but still expensive performance. We interpret these results as at least a sync-start k-message and a sync-next K-message will be triggered per an out-of-sync detection event. Moreover, a minimal overhead of a sync-start k-message and two sync-next K-message are triggered per one out-of-sync detection event when the source cache has one more update message after the one that triggered the synchronization. From the test results in Fig. 3, the higher the heartbeat frequency, the less the synchronization overhead.

Moreover, Fig. 3 shows that, for the test run with *heartbeat_frequency_1*, the message sequences with 10/20 dropped messages have less average synchronization overhead than the message sequence with 2/20 dropped messages. We interpret this as the more dropped messages, the higher the chance that they are consequent messages, and the higher the chance that more than one message is retrieved during a single full-synchronization round. This will reduce the number of out-of-sync detection events and the average synchronization overhead.

6 Conclusion and future work

We added to PalCom a synchronization algorithm to overcome the problem of losing a once-sent device discovery update message over an unreliable channel. We evaluated the algorithm using an UPPAAL based model. The evaluation shows a stable and reasonable synchronization overhead when using high enough heartbeat frequency that provides sufficient out-of-sync detection sensitivity compared to the size of the discovery events cache. We implemented the synchronization algorithm as part of the current PalCom Java implementation. The implementation passed testing against basic scenarios of channel failure. As a future work, we need

to run more advanced test scenarios on the implementation. We may also modify the algorithm for less synchronization overhead.

References

[1] Svensson Fors D, Magnusson B, Gestegård Robertz S, Hedin G, Nilsson-Nyman E. Ad-hoc Composition of Pervasive Services in the PalCom Architecture. In: *Proceedings of the 2009 International Conference on Pervasive Services, London, UK, 2009*. ACM. p. 83-92.

[2] Ergawy A, Magnusson B. Media Abstraction Framework for the Pervasive Middleware PalCom. In: *Proceedings of the 2nd International Conference on Future Internet of Things and Cloud, FiCloud-2014, Barcelona, Spain, 2014*. IEEE.

[3] Ergawy A, Magnusson B. Device Discovery for the PalCom Pervasive Middleware with Eliminated Cross-networks Periodic Heart-beat Messages. In: *Proceedings of the 5th International Conference on Emerging Ubiquitous Systems and Pervasive Networks, EUSPN-2014, Nova Scotia, Canada*. Procedia Computer Science, volume 37, 2014. p. 64-71.

[4] Ergawy A, Magnusson B. Supporting Distance Vector Routing Over Device Discovery Flows in the Pervasive Middleware PalCom. In: *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies, ANT-2015, London, UK*. Procedia Computer Science, volume 52, 2015, p. 153-160.

[5] Waseem Ahmed and Yong Wei Wu: A survey on reliability in distributed systems. *Journal of Computer and System Sciences*. 2013, volume 97, number 8, p. 1243 – 1255.

[6] N. Xiong and Y. Yang and M. Cao and J. He and L. Shu. A Survey on Fault-Tolerance in Distributed Network Systems. In: *Computational Science and Engineering, 2009. CSE '09. International Conference on*. 2009, vol. 2, p. 1065-1070. IEEE.

[7] Lilia Paradis, Qi Han: A Survey of Fault Management in Wireless Sensor Networks. *Journal of Network and Systems Management*. 2007, volume 15, number 2, p. 171-190.

[8] Matthias Függer and Alexander Kößler and Thomas Nowak and Ulrich Schmid and Martin Zeiner: The effect of forgetting on the performance of a synchronizer. *Performance Evaluation*. 2015, volume 93, pages 1-16.

[9] UPPAAL and its tutorial material are available at http://w ww.uppaal.org/.