Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 94 (2016) 347 – 352

Procedia Computer Science

The 11th International Conference on Future Networks and Communications (FNC 2016)

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

Amr Ergawy*, Boris Magnusson

Department of Computer Science, Lund University, Ole Römers väg 3, Lund SE-223 63, Sweden

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].

* Corresponding author. Tel.: +46736399143; fax: +46 46 13 10 21.
E-mail address: amr.ergawy@cs.lth.se.

doi:10.1016/j.procs.2016.08.052

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].

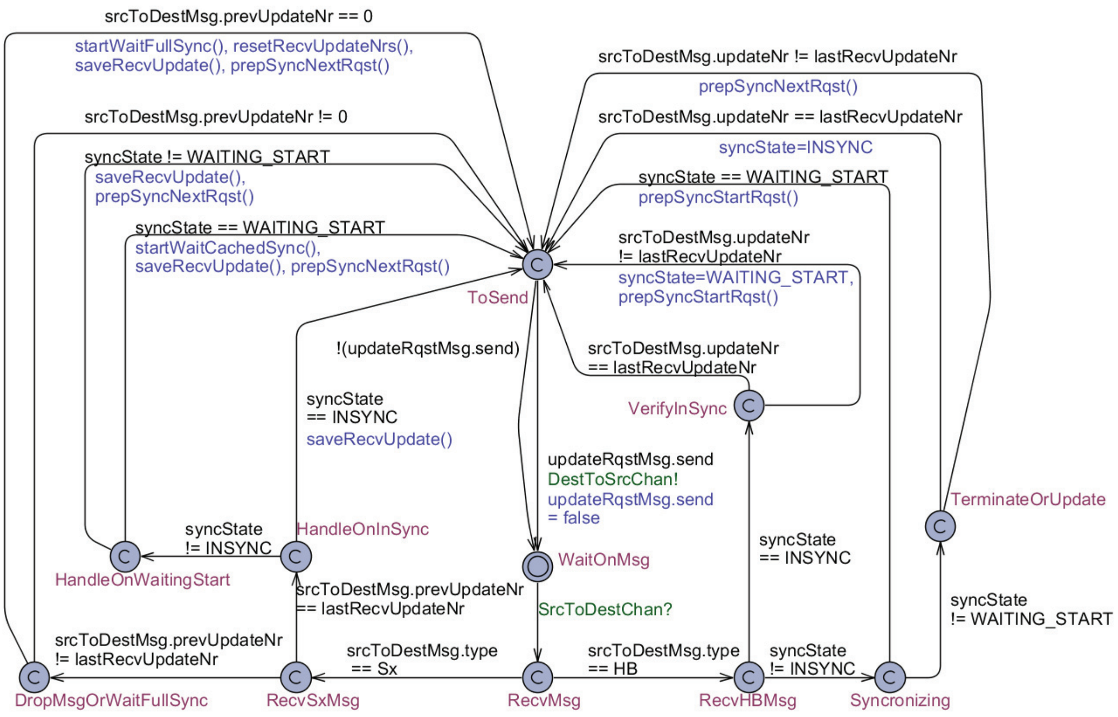
srcToDestMsg.prevUpdateNr == 0
startWaitFullSync(), resetRecvUpdateNrs(), saveRecvUpdate(), prepSyncNextRqst()
srcToDestMsg.prevUpdateNr != 0
syncState != WAITING_START
saveRecvUpdate(), prepSyncNextRqst()
syncState == WAITING_START
startWaitCachedSync(), saveRecvUpdate(), prepSyncNextRqst()
srcToDestMsg.updateNr != lastRecvUpdateNr
prepSyncNextRqst()
srcToDestMsg.updateNr == lastRecvUpdateNr
syncState=INSYNC
syncState == WAITING_START
prepSyncStartRqst()
srcToDestMsg.updateNr != lastRecvUpdateNr
syncState=WAITING_START, prepSyncStartRqst()
ToSend
!(updateRqstMsg.send)
srcToDestMsg.updateNr == lastRecvUpdateNr
VerifyInSync
syncState == INSYNC
saveRecvUpdate()
updateRqstMsg.send
DestToSrcChan!
updateRqstMsg.send = false
TerminateOrUpdate
syncState != INSYNC
HandleOnInSync
HandleOnWaitingStart
WaitOnMsg
syncState == INSYNC
srcToDestMsg.prevUpdateNr == lastRecvUpdateNr
SrcToDestChan?
syncState != WAITING_START
srcToDestMsg.prevUpdateNr != lastRecvUpdateNr
srcToDestMsg.type == Sx
srcToDestMsg.type == HB
syncState != INSYNC
DropMsgOrWaitFullSync
RecvSxMsg
RecvMsg
RecvHBMsg
Syncronizing

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

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. 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

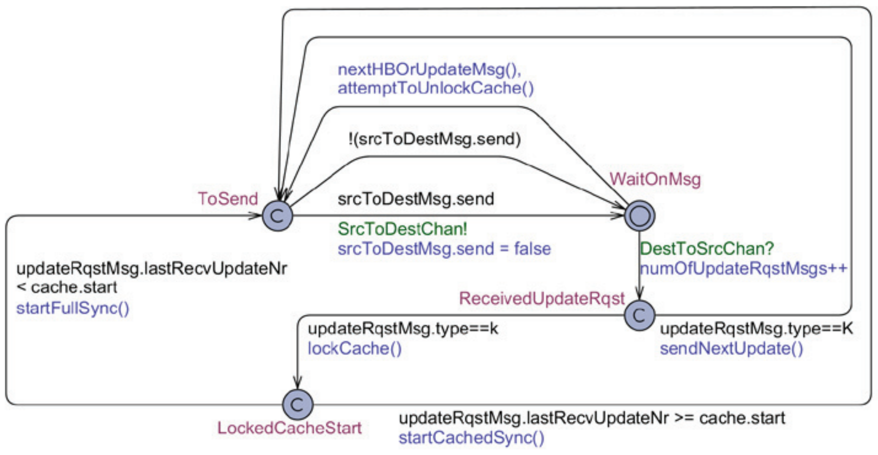

Fig. 2. The UPPAAL model of the source node of the synchronization algorithm.

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 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 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

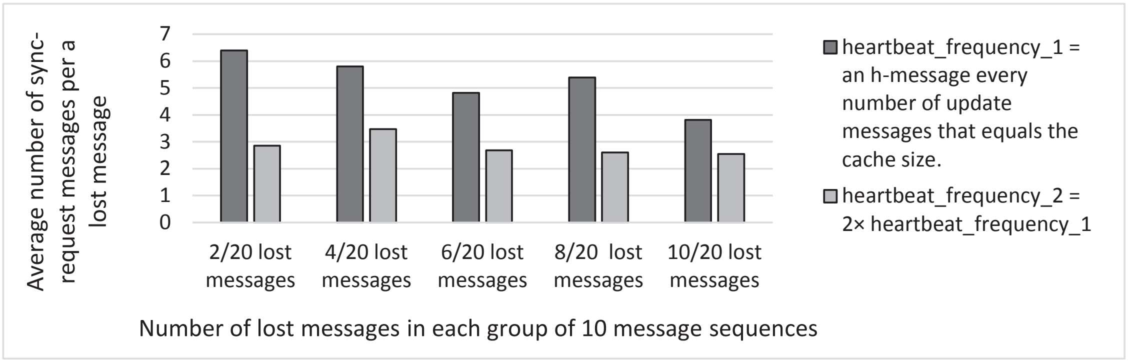

Fig. 3. The algorithm performance with two different configurations of the heartbeat frequency.

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/.