PALTA: Peer-to-peer AdaptabLe Topology for Ambient intelligence

Alfredo Cádiz[1], Boris Mejías[1], Jorge Vallejos[2], Kim Mens[1], Peter Van Roy[1], Wolfgang de Meuter[2]
[1] Département d'ingénierie informatique, Université catholique de Louvain – Belgium
[2] Programming Technology Lab, Vrije Universiteit Brussel – Belgium
{alfredo.cadiz|boris.mejias|kim.mens|peter.vanroy}@uclouvain.be,
{jvallejo|wdmeuter@vub.ac.be}

Abstract

Many Ambient Intelligence (AmI) scenarios fit perfectly for auto-generated distributed networks, but they assume the existence of good enough network topology organizing the connected devices. AmI scenarios need to handle an unanticipated number of participants and inappropriate distributed network topologies can affect the network's efficiency by making it unstable and hard to manage. This paper introduces PALTA*, a self-adapting hybrid topology capable of dynamically adjusting its configuration by using a combination of existing topologies.* PALTA *allows the incremental construction of self-maintained distributed networks which take advantage of the current network state.*

Index Terms: Ambient intelligence, Decentralized systems, Network topologies, Self-management, Network variability

1. Introduction

The Ambient Intelligence (AmI) field envisions people constantly surrounded by hardware technology in the form of interconnected mobile and embedded devices [5]. AmI applications have been proposed for a diverse number of everyday situations ranging from small and close applications (e.g. home and office applications [11, 16]), to those large and open (e.g. traffic and massive-events applications [2]). Often, the limited capacities of the devices used in these scenarios cause the emergence of networks that lack any infrastructure, dynamically changing their configuration as the people move about and the devices become (un)available. Still, these networks are expected to be *self-maintainable* and *efficient*.

Most existing AmI software approaches assume the existence of an underlying network reconfiguration mechanism which copes with the connection volatility of the devices. However, thus far we observe that such mechanisms commonly imply full connectivity among the devices (i.e. the devices are directly connected to each other). As shown in other domains of distributed computing, this configuration has dramatic consequences on the scalability of the networks.

In this work, we explore advanced peer-to-peer (P2P) reconfiguration algorithms [14] to deal with the dynamicity and scalability of AmI networks. These algorithms implement efficient network configurations – known as *network topologies* – and message rerouting techniques that enable each network participant – known as *peer* (e.g. a device) – to have complete access to the network by having a small number of direct connections to other peers. Yet, these techniques mostly make sense for large-scale networks. For small networks they are considered overkill, unnecessarily using resources which, for the devices of AmI scenarios, is a critical issue.

In this paper, we propose our hybrid network reconfiguration mechanism called PALTA*: a Peer-to-peer AdaptabLe Topology for Ambient intelligence*. This algorithm takes advantage of the best features of a fully connected network when the number of peers is small enough to allow the devices manage this kind of topology. When the network becomes too large to maintain a fully connected topology, the algorithm will automatically adapt the network configuration to become a Relaxed Ring [14], which can handle a large number of peers by executing more complex algorithms for self-managing the distributed network. We consider different aspects concerning the transition between networks: adaptation of the base algorithms, maintaining the network's coherence and self-healing from inefficient configurations.

The main contribution of this work is the design of a dynamic network topology, PALTA, oriented to self-organize and self-adapt efficiently highly variable networks with an unanticipated number of participants. We present the concrete application for Ambient Intelligence, but this approach can be applied to any application which needs to deal with variable number of connected peers.

Section 2 presents our motivating example for dynami-

cally adaptable and scalable distributed networks. Section 3 details the Full Connectivity and Relaxed Ring topologies. Section 4 describes our hybrid approach for size-efficient topologies. We provide our validation results in Section 5. Section 6 compares our approach with existing technologies. We discuss future work in Section 7 and conclude in Section 8.

2. Motivation

An interesting scenario consists of the implementation of Ambient Intelligence Games [1, 18]. Real world games can augment the interaction between players by using wearable electronic devices. For instance, a Paintball game consists in players equipped with compressed air guns that shoot paint capsules for eliminating opponents. Games can be played indoors or outdoors and they can take various forms depending of the rules agreed by the players. An *electronically-enhanced paintball* game can supervise in real time the state of the whole match, offers more complex games and prevents cheating. A player can join a game at any time as soon as he turns on his device. The player's device receives the information about the already existing participants (e.g. location, health, game rules) and the device sends the local state of the player to the other devices. Since every player's device maintains a view of the game, they can collaborate in order to coordinate the matches and provide online information like team conditions, match state, scoring and messages (global or private). Players can join or leave the game at any time and the game can self-organize the active participants in order to balance the match. We can see how the game can benefit from by sharing the player's information in real time, but in order to implement such a scenario there are some design decisions to consider.

The implementation of such kind of scenario can use a traditional approach where users get connected to wireless networks installed by the event's organizers. The drawbacks of using these infrastructure-dependent networks are that they require equipment investment and installation in the game fields. Also this architecture introduces fragile points in the network: Routers and access points can fail, breaking down the connectivity to some part or even the whole local network. An alternative approach for interconnecting devices is a *distributed network* created by the mobile devices themselves. This kind of networks only need the presence of other devices in order to connect with each other, no infrastructure coordinates the communication. In this way the failure of some devices does not affect the network stability since the rest of the correct devices can conserve their communication and still exchange information. In addition to a decentralized communication, this kind of networks are maintained by the devices themselves and any existing device can be used as an entry point. Finally we need to deal with the unanticipated number of incoming nodes in the network. Since the players in a game can vary in quantity, the network generation should be able to self-adapt in order to optimize the network usage.

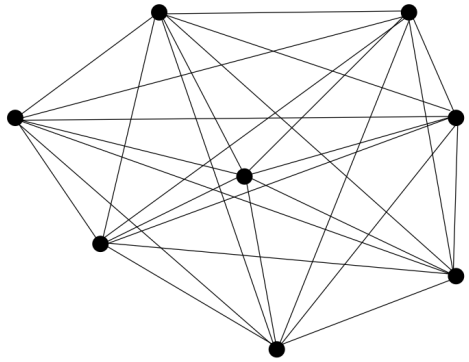

Figure 1. The fully connected topology.

There are already topologies for generating distributed networks but they target different scenarios: some of them are suitable for few interconnected devices and other provide complex algorithms to provide better scalability. We want to define an incremental network topology that can take advantage of the network size and provide both good connectivity and scalability.

So far we have described a motivating example for our approach and identified a number of requirements to provide suitable scenario for this application:

- Distributed topology without intermediate infrastructure.
- Efficiency in the network composition depending of the network state.
- Adaptability to an unanticipated number of peers.

In the next section we analyze two networks topologies, we show their advantages and weaknesses in order to present our adapted approach in section 4.

3. Network topologies

As we state in the motivation section, we look for an algorithm that can take full advantage of the current network size. First we will review two existing network topologies: fully connected and Relaxed Ring [14]. We discuss why these topologies are only partially suitable for the requirements shown in the previous section before introducing our approach called PALTA in Section 4.

3.1. Fully connected networks

A fully connected network means that every peer has an open channel (i.e. a bidirectional link) to every other peer in the network, allowing a direct communication between all the network members. Figure 1 shows the topology of a small fully connected network.

When a new node wants to join the network, it follows the protocol shown in Figure 2: It contacts an arbitrary peer sending a *join* message. The connected peer responds with a *join_ok* message including its full list of connected peers and includes the new peer in its own list. The joining node then repeats the operation sending *join* messages to all the peers still not connected to it.

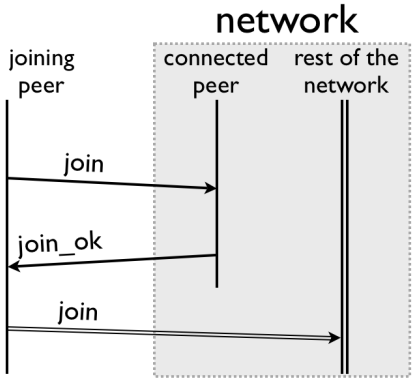

Figure 2. Joining phase in fully connected topologies.

When no network communication is presented, this algorithm ensures the direct point-to-point communication of all the peers in the network. Since every *join* message is replied with the complete and updated list of current peers, concurrent joins are supported and *the information of new peers will be propagated even when joining nodes are still contacting the rest of the network.* The algorithm 1 describes the implementation of this joining method. Note that the code belongs to the module FULL, which we will reuse for building PALTA in the next Section.

The algorithm is described using an *event-driven model*. Peers exchange messages through any communication channel that devices can use. When the message is handled by the communication layer on the device, the correspondent event is triggered at the application level. To avoid race conditions within the same device, only one event is handled at a time. As in any distributed system, devices run their own event scheduler in parallel.

In case of intentional or unintentional disconnections, the network remains stable. This is because all peers are interconnected and no special fixes are needed to repair the network.

Algorithm 1 Join for fully connected networks

```
1:  module FULL

2:  upon event ⟨ join | new ⟩ do
3:      if id ∉ peers then
4:          send ⟨ join_ok | self, peers ⟩ to new
5:          peers := peers ∪ {new}
6:      end if
7:  end event

8:  upon event ⟨ join_ok | src, src_peers ⟩ do
9:      peers := peers ∪ {src}
10:     to_contact := src_peers \ peers
11:     for all p in to_contact do
12:         send ⟨ join | self ⟩ to p
13:     end for
14: end event
```

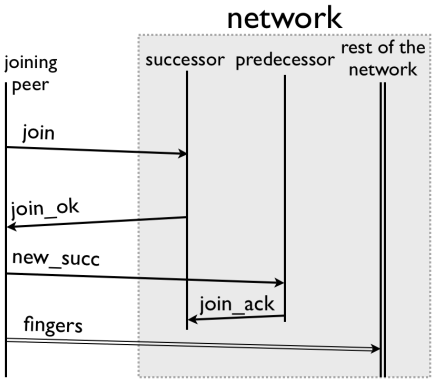

Figure 3. Joining phase in Relaxed Ring topologies.

A fully connected topology helps to maintain a schema where every peer is directly connected to the whole network. However there is an important disadvantage in this topology: it lacks scalability. When the distributed network becomes too big, every node still needs to maintain the complete list of connected peers and joining nodes will have to contact every other connected node in the network in order to be properly connected. In order to solve this issue, there are approaches providing solutions for scalable distributed networks

3.2. Relaxed Ring

The Relaxed-Ring [14] is a structured overlay network (SON) inspired on the Chord ring [17] and using a distributed hash table (DHT). It is fully decentralized and fault tolerant with self-organization of peers. It is meant for

building large-scale distributed systems where the amount of connections that peers need to maintain is efficiently chosen. It provides a routing mechanism that takes $O(log_k N)$ hops to reach any other peer in the network, where N is the size of the network, and k is the amount connections a peer maintains. These connections are known as *fingers*. One of the reasons for using the ring topology is that it is one of the most resilient to failures, and it is competitive with any other SON with respect to proximiy [9]

The ring is formed by connecting every peer with a successor ($succ$) and a predecessor ($pred$). Peers are identified with an integer taken from the range $[0, N-1]$. This identifier is used as its key for implementing the DHT. The $succ$ of a peer is chosen by taking the peer with the next identifier found clockwise. The $pred$ is chosen anti-clockwise. Having a connection with the successor is important because it means that the peer is connected to the network. Having identified the key of the predecessor gives responsibilities to the peer in the network. Every peer p is responsible for all the keys found between its predecessor and itself $(pred, p]$.

Among the systems that use the ring topology, there are different approaches for performing joins and leaves [7, 12, 17]. We have observed that all of these approaches rely on the synchronization of three nodes, which is not always guaranteed due to connection problems. That is the reason for the introduction of the relaxed ring, where the join operation is divided into two steps synchronizing only two peers at the time, allowing a relaxation of the ring between both steps. Leaves are treated as failures, and the recovery mechanism triggers the join algorithm to fix the ring.

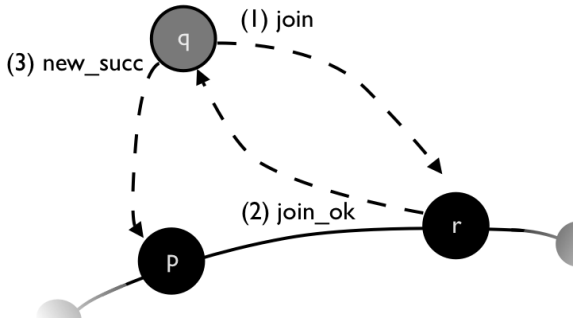

Figure 4. Joining phase in a Relaxed Ring topology.

Algorithms 2 and 3 describe how the messages depicted in Figure 3 are handled by every peer. Algorithm 2 describes the first step of a join. According to Figure 4, the most important thing is that if node q enters the network, it has to do it between p and r, where $q \in (p, r]$, to keep the share of key responsibilities consistent. The joining peer sends the $join$ message to its entry point on the network. This node checks if the new peer is a better predecessor that the current one, being the better predecessor the one who maintain the ordered structure of the network. If it is not the best predecessor, the message is forwarded to the best finger, the closest to the new peer, until the message reaches the responsible of the key of the new peer. The $pred$ pointer of the found successor successor is updated and it answers with the message $join\_ok$ to the new peer. This message contains the $pred$ and $succ$ of the joining peer, and a successor list for recovery in case of failure.

Once the joining peer receives $join\_ok$, it verifies that the information about $pred$ and $succ$ is correct. If the successor is correct, we consider that the peer is connected to the network. If the predecessor is correct, the responsibility is well defined, and the new peer contacts its predecessor with the message new_succ. The procedure $findAndConnectToFingers$ will be explain later on this section.

Algorithm 2 Relaxed-Ring join step 1 - adding a new node

```
1:  module RING

2:  upon event ⟨ join | new ⟩ do
3:      if betterPredecessor(new) then
4:          send ⟨ join_ok | pred, self, succlist ⟩ to new
5:          pred := new
6:      else
7:          send ⟨ join | new ⟩ to bestFinger(new)
8:      end if
9:  end event

10: upon event ⟨ join_ok | p, s, sl ⟩ do
11:     if betterSuccessor(s) then
12:         succ := s
13:         succlist := {s} ∪ sl
14:     end if
15:     if betterPredecessor(p) then
16:         pred := p
17:         send ⟨ new_succ | self, succ, succlist ⟩ to p
18:     end if
19:     findAndConnectToFingers()
20: end event
```

The second step of the join algorithm is dedicated to close the ring. Considering our example with nodes p, q and r, after messages $join$ and $join\_ok$, p and q have r as successor, but q is the $pred$ of r, and q is aware of p but they have not established connection yet. Peer q sends message new_succ to p to close the ring. Peer p accepts q as successor and acknowledge its old successor r.

In order to achieve efficient routing, the new peer needs to contact the corresponding fingers, which are chosen strategically. This is done in the first step of the algorithm,

Algorithm 3 Relaxed-Ring join step 2 - closing the ring

```
1: module RING

2: upon event ⟨ new_succ | s, olds, sl ⟩ do
3:     if (succ == olds) ∧ betterSuccessor(s) then
4:         send ⟨ join_ack | self ⟩ to succ
5:         succ := s
6:         succlist := {s} ∪ sl
7:     end if
8: end event
```

at the end of event $join\_ok$, using the functionality of the procedure $findAndConnectToFingers$. Using the strategy of Chord[17], if the key of a peer is p, the fingers are chosen using $f_i = (p + N/2^i) mod N$ with $i = 1..k$. When messages need to be routed to a particular key, they are sent to the closest preceding finger. There are two situations to consider here. If the distribution of the keys is not uniform, some of the peers will be responsible for a larger range of keys. This can result in having two or more fingers pointing to the same peer. The same situation can occur with very small networks, because in that case peers will be responsible for larger ranges of keys, even if the distribution is uniform. The issue is that peers will not be able to take advantage of k fingers but less, degrading unnecessarily the efficiency of the routing. This situation is solved by PALTA.

The Relaxed-Ring does not make a difference between nodes gracefully leaving the network and nodes crashing. This makes the topology more fault-tolerant. When a node is detected to have crashed, only the predecessor takes action contacting the first peer in its successor list. This peer is usually the successor of the crashed peer. The message used to trigger the failure recovery is $join$, reusing the join algorithm. More details are explained in Section 4.

4. PALTA

In this section we present our hybrid topology called PALTA, which tries to the combine the best features of the topologies shown in the previous section to provide a dynamic topology capable of self-adapting depending on the network size. Our topology will change its organization when the network size reaches a defined threshold. We will refer to this limit as ω. Even when during the discussion of this paper we consider the value of ω as uniform for all nodes, the algorithm is designed such that every node can define its own ω to adapt its behaviour according to its own capacities.

In order to successfully implement this dynamic schema, we need to analyze how the topology will evolve when peers join or leave the network. When the network is created and the number of peers is below ω, the joining peers will perform the *fully connected* algorithm shown in Section 3.1. When peers detect a network size above ω, all the incoming joining requests will be handled using the *Relaxed Ring* algorithm as shown in Section 3.2. The same methodology is followed when, after a number of disconnections, the network becomes smaller than ω. In such case, peers will change their joining algorithm from Relaxed Ring to fully connected.

To be able to make the transition from a fully connected network to a ring, peers need to precisely identify their successors and predecessors at all times. Algorithm 4 shows that the $join$ even in PALTA can be seen almost as a method dispatcher, with the subtlety that it checks its predecessor and predecessor pointers in every join before triggering the $join$ event of the FULL module. In case that ω is already reached, it is the algorithm of the relaxed ring who will take care of $pred$ and $succ$ accordingly.

The procedure that checks predecessor and successor uses functions $better\_successor$ and $better\_predecessor$, which are equivalent to the one discribed in 3.2, verifying if the key belongs to the corresponding range.

Algorithm 4 Join for PALTA: Adapted full connected algorithm with transition to relaxed-ring

```
1: module PALTA

2: upon event ⟨ join | new ⟩ do
3:     if size(peers) < ω then
4:         check_succ_pred(new)
5:         trigger ⟨ FULL.join | new ⟩
6:     else
7:         trigger ⟨ RING.join | new ⟩
8:     end if
9: end event

10: procedure check_succ_pred(id) is
11:     if better_successor(id) then
12:         succ := id
13:     end if
14:     if better_predecessor(id) then
15:         pred := id
16:     end if
17: end procedure
```

The value of ω can change right after sending message $join$ and-or $join\_ok$. Due to that, there is no way of knowing if a reply message $join\_ok$ will correspond to the full connected topology or to the relaxed-ring. Peers need to adapt dynamically to this situation whenever is needed. Algorithm 5 shows how this event is overloaded in PALTA. The first case corresponds to $join\_ok$ as in the ring, with information of the predecessor, successor and sucessor list. If the routing table is higher than ω, the event is delegated

to the relaxed-ring module. If we are in a small network, predecessor and successor are accordingly check, and the successor list is used to trigger the full connected algorithm, which will be used until reaching ω.

Algorithm 5 Join for PALTA: Overloaded event $join\_ok$

```
1:  module PALTA

2:  upon event ⟨ join_ok | p, s, sl ⟩ do
3:      if size(peers) < ω then
4:          check_succ_pred(new)
5:          trigger ⟨ FULL.join_ok | s, sl ⟩
6:      else
7:          trigger ⟨ RING.join_ok | p, s, sl ⟩
8:      end if
9:  end event

10: upon event ⟨ join_ok | src, srcPeers ⟩ do
11:     if size(peers) < ω then
12:         check_succ_pred(new)
13:         succList := succList ∪ srcPeers
14:         trigger ⟨ FULL.join_ok | src, srcPeers ⟩
15:     end if
16: end event
```

4.1. Self-healing network transitions

The consolidation of two different algorithms into an hybrid one carries a number of new issues. The transition phase between algorithms can give rise to inconsistency problems in the network's topology. For instance, the joining algorithm could not converge, constantly changing between Full Topology and Relaxed Ring or the network size could exceed the limit ω, producing topologies too big to manage properly. In a distributed network the peers cannot instantly access the complete state of the whole network. Because of this limitation the messages produced by peers in PALTA are based only in local view of the network. The design of PALTA also considers self-healing procedures to minimize unstable network states.

We need to clearly define how joining peers will connect to the network: a joining node will contact with any connected peer in the network. The connected peer reply with either the message FULL.*join_ok* or RING.*join_ok* to trigger the corresponding joining algorithm. When this choice is made without changes in the network size that pass by the limit ω, the joining methodology will follow the description given in Section 3.

We find a special case when the joining peer is in process to contact the rest of peers in the network according to the fully connected algorithm (See algorithm 1). After receiving the first response FULL.*join_ok* from the contacted node,

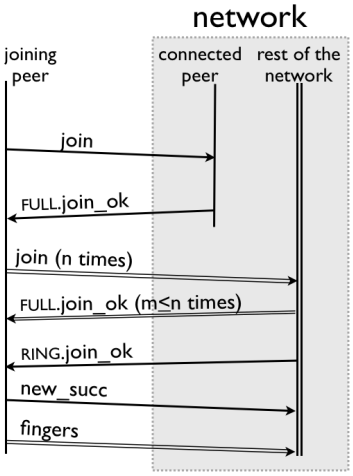

Figure 5. Algorithm change during the joining phase in PALTA.

the joining peer will be sending *join* messages and receiving responses from the network members. Since it is possible having concurrent joins in different sections of the network, in a given time a node can perceive that the topology has to change. As we can see in Figure 5, this connected node will reply RING.*join_ok* to the joining node. This action will change the joining mechanism and the joining node will be integrated as a peer of the Relaxed Ring, with its already connected nodes as fingers.

Algorithm 6 Fault Recovery in PALTA

```
1:  module PALTA

2:  upon event ⟨ crash | p ⟩ do
3:      if (p == succ) ∨ (p == succ_cand) then
4:          succ := nil
5:          succ_cand := getFirst(succList)
6:          send ⟨ join | self ⟩ to succ_cand
7:      end if
8:  end event
```

In order to solve inconsistencies because of crashes on the peers, in PALTA we use a simplified version of the *failure recovery* mechanism used in the Relaxed Ring. This is a complete and eventually accurate failure recovery system, meaning that all the crashed will be detected eventually and false suspicious will be corrected. The failure recovery mechanism is shown in Figure 6. In this case, when a peer p suspects of a possible crashed successor q, this successor is removed from p's routing table. The p will search for a new successor in its successor list, being t the best candidate, p sends a *join* message to t in order to fix the ring. In case that the suspicion over q is false, t will refuse the new connec-

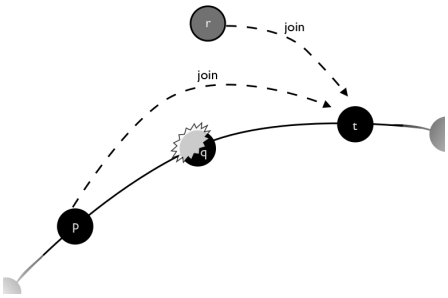

Figure 6. Triggering failure recovery.

tion request from p by rerouting the *join* message to q and the suspected peer will be considered correct again. Algorithm 6 shows how this recovery is implemented in PALTA.

5. Validation

This section shows the results obtained from simulations performed over the algorithms that we review on this paper: fully connected networks, Relaxed Ring and PALTA. All the algorithms are implemented in the Mozart-Oz programming system [15], where the Relaxed Ring is already part of P2PS [4]. The implementation of the fully connected network is a bit more complex than the algorithms presented in Section 3.1, but it is a lot more efficient in terms of message exchange. PALTA is implemented following the description given on this work. We perform the simulations using CiNiSMO: Concurrent Network Simulator in Mozart-Oz [13]. Every topology was tested building networks from 20 to 1000 nodes, increasing the size by 20 nodes at every iteration. Plotted values represent the average of running every experiment with several seeds for random generation. In the case of PALTA, we tested the algorithm using two different values for ω, being 100 and 200.

In order to measure the network efficiency according to our requirements, we gather information from

Active connections The average number of active links in the connected peers.

Messages sent to create a network The accumulated number of messages sent in order to create a network of a given size.

5.1. Results

We can observe in Figure 7 that fully connected networks increment their amount of connections linearly. Part of the curve is missing, but it clearly corresponds to $n-1$, being n the size of the network, because every node is connected to all other peers. The Relaxed Ring keeps a small amount of connections which corresponds to the successor, predecessor and fingers. The fact that this value grows very slowly with respect to the size of the network confirms the scalability of this approach. But, it also confirms that peers are not connected to many nodes in small networks, requiring more steps to reach any other peer, being less efficient.

PALTA is shown using two different values of ω. We can observe that for $\omega = 100$, the amount of connections increases as in a fully connected network until the size of the network reaches ω. At that point, the new peers joining the network will behave as peers in a ring, requiring only a few amount of fingers to reach the whole network. Since the network has already reached a reasonable size, it is good that the amount of connections does not increase. The more the network grows, the more PALTA converge to the behaviour of a ring, confirming than PALTA is a scalable topology being able to provide efficient connectivity in small network. The behaviour of $\omega = 200$ is analog.

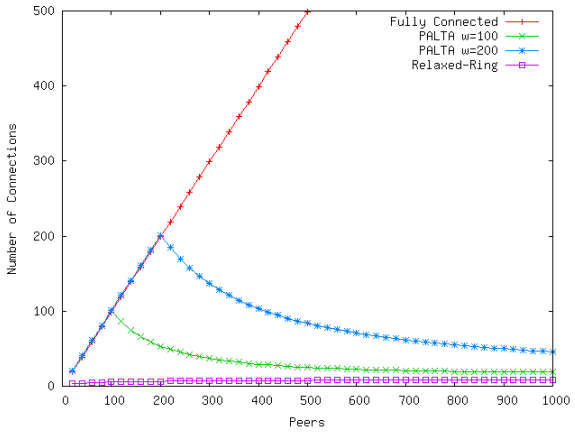

Figure 7. Active connections vs number of peers.

Another interesting result can be observed in Figure 8 where we measure the amount of messages required for building every kind of network. Once again, the fully connected topology proves to be very inefficient as soon as the network becomes a bit large. The amount of messages follows $n * (n-1)$ with n being the size of the network. This means that for a 1000 peers network, almost 1 x 10^6 messages are needed to create the network.

The comparison between PALTA and the Relaxed Ring is very interesting, because for small networks the ring performs as good or better than PALTA. On the other hand, once the network reaches the value of ω, it is interesting to see how the amount of messages needed in PALTA begins to grow slower that the ring. This is because at that point, peers in PALTA have more available connections that the fingers of the ring. Given that, PALTA takes in average

less hops to reach other peers in the network, reducing the amount messages needed by the routing algorithm. As a conclusion, we can see that PALTA is well prepared for the transition between small and large networks.

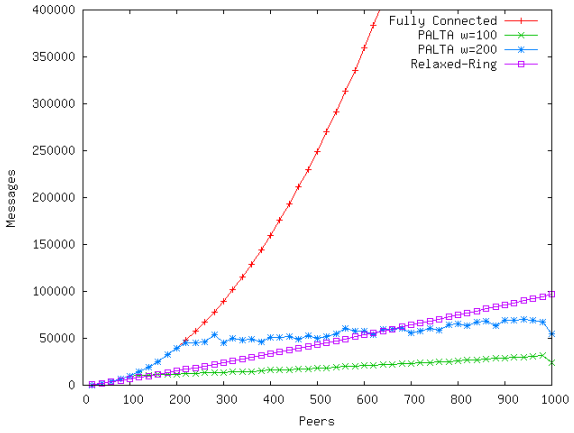

Figure 8. Messages vs number of peers.

6. Related Work

A work mentioning explicitly topologies for Ambient Intelligence is [10], but they use the concept of topology in an higher level. They provide a middleware capable of organize devices found in the environment and allow them behave coherently to the user's demands. They mention the volatility of devices in a different domain and they do not cover the logic organization of networks in AmI.

Research on peer-to-peer networks has done a lot of effort on designing many different topologies to build structured overlay network[6], but all those solutions are thought having huge networks in mind, too big for our gold. As described in section 4, our solution is inspired in on of the simples SON, but we use it only when it is needed.

7. Future Work

7.1. Variable values for ω and number of fingers

Even when we have tested different values of ω on this work, we have fixed it uniformly for all peers on the network. In future research we want to analyze the impact of adapting this values and how the topology can benefit from it to become more efficient. We still need to investigate whether we can change these values separately for every connected peer or whether we need to propagate the same value across the whole network.

7.2. Language abstractions for modularizable algorithms

In order to maintain simplicity and understandability we have presented the algorithms using if-statements in sections 3 and 4. But in order to maintain a modularizable code we need to apply other techniques. We can use traditional OO techniques like the *Strategy pattern*, but we would prefer use dynamic approaches. The Ambience programming language [8] provides facilities to enable *context-oriented programming*. By enabling different *contexts* software can adapt its behavior at runtime maintaining the software modularity. A similar approach is provided by ContextL [3] which can refine behavior using different *layers* in objects.

8. Conclusions

There is no doubt that Ambient Intelligence implementations are going to be more and more recurrent in a close future. Mobile devices are now powerful enough to allow the construction of ad-hoc networks and enable complete mobility without depending of any infrastructure maintaining the networks. But this kind of ad-hoc networks are not for free. If they are not well designed they can lead to serious scalability problems.

We have presented PALTA: **P**eer-to-peer **A**daptab**L**e **T**opology for **A**mbient intelligence. It is based on the existing full connectivity and Relaxed Ring topologies, with adaptations to make them work together. This hybrid topology features self-organizing and self-adapting mechanisms to ensure a complete connectivity among the connected peers and take advantage of the current network state to have a better use of the available resources. It benefit from fully connected small networks and it makes a smooth transition to larger ones, being able to scale as a large scale structured peer-to-peer network.

9. Acknowledgements

This work has been supported by the VariBru project of the ICT Impulse Programme of the Institute for the encouragement of Scientific Research and Innovation of Brussels (ISRIB), the SELFMAN project funded by the European Commission, and by the MoVES project of the Interuniversity Attraction Poles Programme of the Belgian State, Belgian Science Policy. The authors would like to thank S. González and P. Hass for their comments on this work.

References

[1] S. Björk, J. Holopainen, P. Ljungstrand, and K.-P. Åkesson. Designing ubiquitous computing games – a report from a

workshop exploring ubiquitous computing entertainment. *Personal Ubiquitous Comput.*, 6(5-6):443–458, 2002.

[2] G. Cabri, L. Ferrari, L. Leonardi, and F. Zambonelli. The laica project: Supporting ambient intelligence via agents and ad-hoc middleware. In *WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pages 39–46, Washington, DC, USA, 2005. IEEE Computer Society.

[3] P. Costanza and R. Hirschfeld. Language constructs for context-oriented programming: an overview of contextl. In *DLS '05: Proceedings of the 2005 symposium on Dynamic languages*, pages 1–10, New York, NY, USA, 2005. ACM.

[4] DistOz Group. P2PS: A peer-to-peer networking library for Mozart-Oz. *http://p2ps.info.ucl.ac.be*, 2008.

[5] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman. Scenarios for ambient intelligence in 2010. Technical report, EC Information Scociety Technologies Advisory Group (ISTAG), 2001.

[6] S. El-Ansary and S. Haridi. An overview of structured overlay networks. In *Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks*. 2005.

[7] A. Ghodsi. *Distributed k-ary System: Algorithms for Distributed Hash Tables*. PhD dissertation, KTH — Royal Institute of Technology, Stockholm, Sweden, Dec. 2006.

[8] S. González, K. Mens, and A. Cádiz. Context-oriented programming with the ambient object system. In *1st European Lisp Symposium*, 2008.

[9] R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity, 2003.

[10] M. Hellenschmidt and T. Kirste. A generic topology for ambient intelligence. In P. Markopoulos, B. Eggen, E. H. L. Aarts, and J. L. Crowley, editors, *EUSAI*, volume 3295 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2004.

[11] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. D. Mynatt, T. Starner, and W. Newstetter. The aware home: A living laboratory for ubiquitous computing research. In *CoBuild '99: Proceedings of the Second International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture*, pages 191–198, London, UK, 1999. Springer-Verlag.

[12] X. Li, J. Misra, and C. G. Plaxton. Concurrent maintenance of rings. *Distributed Computing*, 19(2):126–148, 2006.

[13] B. Mejías. CiNiSMO: Concurrent Network Simulator in Mozart-Oz. *http://p2ps.info.ucl.ac.be/cinismo*, 2008.

[14] B. Mejías and P. V. Roy. A relaxed-ring for self-organising and fault-tolerant peer-to-peer networks. In *SCCC '07: Proceedings of the XXVI International Conference of the Chilean Society of Computer Science*, pages 13–22, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Mozart Community. The Mozart-Oz programming system. http://www.mozart-oz.org, 2008.

[16] N. Sadeh, F. Gandon, and O. B. Kwon. Ambient intelligence: The mycampus experience. Technical Report CMU-ISRI-05-123, School of Computer Science, Carnegie Mellon University, July 2005.

[17] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[18] R. Wakkary, M. Hatala, R. Lovell, and M. Droumeva. An ambient intelligence platform for physical play. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 764–773, New York, NY, USA, 2005. ACM.