

Rationale for defining NCIPs (Neighborhood and Context Interaction Primitives) - position paper - * **

J r mie Albert and Serge Chaumette

LaBRI, Universit  Bordeaux I
351 cours de la Lib ration
33405 Talence cedex
FRANCE

{jeremie.albert,serge.chaumette}@labri.fr
<http://www.labri.fr/>

Abstract. With the increasing number of mobile terminals, the development of applications that will provide new dedicated services by taking advantage of the technology is an effective challenge. The combination of such terminals communicating with each other in a peer-to-peer and dynamically self organized manner is referred to as a Mobile Ad Hoc NETWORK, MANet for short. MANets can be composed of many different kinds of devices. To help the developers to cope with their heterogeneity, we believe that it is required to precisely (re)define the basic functions that communication and context interaction primitives can effectively provide, and to give the precise associated assumptions if any. We call these primitives Neighborhood and Context Interaction Primitives, NCIPs for short. The rationale for defining NCIPs is the topic of this position paper.

Key words: neighborhood and context interaction primitives, communication, primitives, MANets, middleware, context awareness, adaptation

1 Introduction

Mobile Ad-hoc Networks can be composed of many kinds of devices like personal computers, PDAs, mobile phones or even sensors. These are very heterogeneous in terms of computing power, memory capacity, operating system (Windows or

* This work is supported by the French Agence Nationale de la Recherche under contract ANR-05-SSIA-0002-01.

** Java and all Java-based marks are trademarks or registered trademarks of Sun microsystems, Inc. in the United States and other countries. The authors are independent of Sun microsystems, Inc. All other marks are the property of their respective owners.

UNIX based systems, SymbianOS, TinyOS, etc.), supported programming languages (Java, C#, NesC, etc.), autonomy (from a few hours to several days), and radio technology (Wi-Fi, Bluetooth) which affects their potential communication range. The presence of additional features such as sensors, cameras, etc., also depends on the brand of terminal. The management of this heterogeneity imposes many constraints and makes it mandatory to decide on a number of assumptions which make application development extremely context dependent.

For instance, assume cars in a vehicular ad hoc network (also known as VANet [8]) that communicate using some radio technology. There is almost no chance, if no assumption is made regarding either their relative speeds or the radio technology that is used, that two cars moving in opposite directions can communicate because of parameters such as the latency inherent to the connection establishment process.

It thus appears that most existing middleware have been developed not to offer universal APIs that could be supported by any communicating device but for particular devices supporting particular technologies and targeted to a particular context. They make strong, even if not always clearly stated, assumptions about the environment and the mode of operation of the target platforms. An important side effect is that these middleware, making different assumptions can be neither inter-operable nor universal and the applications developed on top of them therefore work in very specific contexts.

2 NCIPS

To help the developers to cope with this heterogeneity, still being able to access the low level features of the target platforms, we believe that it is required to precisely (re)define the basic features that communication and context interaction primitives can effectively provide, and to give the precise associated assumptions if any. We call these primitives Neighborhood and Context Interaction Primitives. The topic of this paper is to explain the rationale for this approach.

The methodology that we have adopted in our preliminary work is as follows. To decide what paradigms NCIPs must support, we have studied those existing environments that we consider significant to our approach, such as TinyOS [2, 11], TOTA [9], JXTA [6], Mate [7] or Squawk [12]. We then have classified and analyzed the primitives available in these middleware or software layers. Based on that state of the art, we have defined and given a precise semantics to a number of low level primitives that support similar concepts without making any hidden assumption.

In this paper, we show by means of examples that even though these primitives are low level because they try to be universal, they can be really useful in supporting MANet dedicated applications. This is a substantial reason to pursue the development of this research.

In the future, we intend to use this approach to define a set of universal primitives (NCIPs) and to develop a middleware that will make them available on any device integrated within a Mobile Ad hoc Network.

3 NCIPS can effectively support complex operations. Example of the one way send NCIP.

The goal of this section is to show that in spite of their simplicity, NCIPs can still be used to develop complex services or high level operations.

To illustrate this point, we consider the (difficult) problem of synchronizing a number of nodes. More precisely, we have up to three nodes¹ that want to access a critical resource that must not be accessed by more than one node at a time.

This is usually achieved by some sort of rendezvous [10] and relies on a number of assumptions on the environment:

- communication is bidirectional;
- communication channels are stable during the execution of the operation;
- etc.

Our approach is different in that we do not want to impose any unnecessary or unrealistic constraint. We simply rely on basic communication primitives which are part of our NCIPs. These are:

1. a one way send method (*send*), one way meaning that it does not return any information or status and does not guarantee that the target node effectively receives the message;
2. a receive method (*receive*).

We then propose the implementation presented algorithm 1. In order to get access to the shared resource, each node goes through the following steps:

1. randomly choose a neighbor *chosenNeighbor* (line 26);
2. send a random number n (*randomNumberSent* in algorithm 1, lines 27 and 28) to *chosenNeighbor*;
3. send 0 to the other neighbors if any (line 30 to 37);
4. if the number m (*message.value* in algorithm 1) that is (possibly) received (line 8) from node *chosenNeighbor* (m can be received while any other step of the algorithm is executed) is such that $m \neq 0$ and $m < n$ then the resource is acquired for a predefined period of time t_2 (line 13) otherwise the node waits during t_2 ;
5. loop to step 1.

¹ The algorithm that we have designed to illustrate our purpose (presented later in this section) basically synchronizes pairs of nodes. Having more than three nodes could lead to several simultaneous synchronizations and the resource could be accessed by more than one node at a time.

Algorithm 1 Resource sharing

```

1  [...]
2
3  int randomNumberSent = 0;
4  Node chosenNeighbor = null;
5
6  // this method is invoked when a message is received
7
8  void receivedMessage(Node from, Message message) {
9
10     if (from == chosenNeighbor) {
11
12         if (message.value != 0 && message.value < randomNumberSent)
13             // I have the resource and I keep it for t2
14             [...]
15         else // I do not have the resource, I sleep for t2
16             sleep();
17     }
18 }
19
20 // this method is invoked when the internal timer is fired
21
22 void timerFired() {
23
24     // send a random number n to a random neighbor
25
26     Node chosenNeighbor = selectRandomNeighbor(); // from the static list of neighbors
27     randomNumberSent = randomNumber();
28     send(chosenNeighbor, randomNumberSent);
29
30     // send 0 to all other nodes
31
32     NodeList knownNodes=getNeighborhood(); // get the static list of neighbors
33     Node node= knownNodes.firstNode();
34     while (node != null) {
35         if (node != chosenNeighbor)
36             send(node, 0);
37         node = knownNodes.nextNode();
38     }
39 }
40
41 [...]
```

The temporal arrangement of the different steps of the algorithm for a sample run is shown figure 1.

Despite the fact that this algorithm relies on a very basic one way send method that does not return any information about what it has effectively done, it still ensures that the shared resource is used by at most one node at a time.

Even though this is out of the scope of this paper, we discuss the performance of this primitive to show that this is not simply a toy example and that it works in the real world. The resource sharing algorithm performance depends on several parameters:

- the number of nodes (two or three) that execute the algorithm;
- the interval where the random number is selected;
- the time t_1 required to decide which node is going to have the resource (it corresponds to steps 1, 2 and 3 of the algorithm);

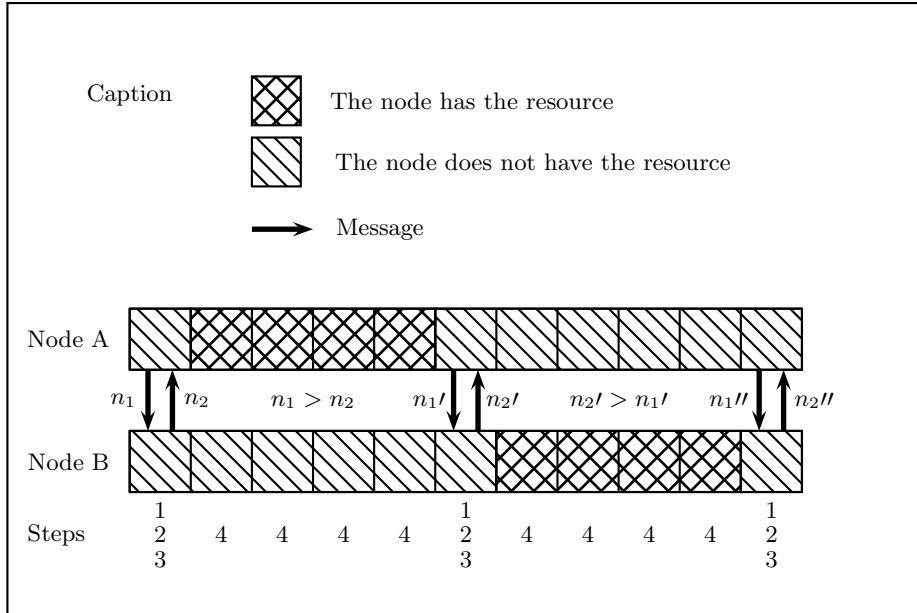


Fig. 1. Temporal behavior of the resource sharing algorithm executed by two nodes

- the time t_2 while the nodes keep the resource or wait for the following round (step 4 of the algorithm).

For example, for two nodes, if we consider that the probability for each of these nodes to choose the same random number is negligible, then the utilization ratio of the shared resource is $\frac{t_2}{t_1+t_2}$.

This algorithm has been implemented on a number of Xbow motes [1, 4] running the TinyOS [2, 11] system and the experimental results effectively confirm the above analysis.

4 NCIPS can significantly impact efficiency. Example of the neighborhood density NCIP.

The goal of this section is to show that properly defining some NCIPs to operate in a really mobile context sometimes makes it possible to improve the efficiency of algorithms. We illustrate this point with the computation of the neighborhood density NCIP and its usage in a flooding algorithm.

In a static framework it is relatively straightforward to compute the number of neighbors of a node (basically because it is stored in a table). When switching to a mobile context, either such primitives are dismissed or the problems raised by the mobility of the environment are simply ignored (see below). In our opinion these solutions are unacceptable.

Density awareness is used in several algorithms [14, 15]. Here we consider the Delayed Flooding with Cumulative Neighborhood (DFCN) algorithm presented in [5]. The basic goal of this last algorithm is to propagate information over the network by using flooding. Based on a multi-criteria optimization approach, it tries to find a compromise between the number of messages exchanged in the network and the speed at which the nodes are informed. Its behavior is directed based on several parameters, among which the number of neighbors, i.e. the neighborhood density at each node and the ratio of neighbors that already have been informed. It is shown that considering a density threshold, below which it is decided not to broadcast the information to the neighborhood, diminishes the network load without increasing too much the time necessary to inform all the nodes.

The density computation in a mobile network is usually implemented as follows. A node regularly broadcasts a beacon to signal its presence to its neighborhood. Based on the collection of all the beacons it is aware of, a node can then compute the density of its neighborhood. The thing is that because of the instability of the network, the resulting value is possibly false as soon as it has been computed. Instead of ignoring this, we propose an alternative in our NCIPs framework: the result of the density computation primitive contains a stability value that says how long the information remains true. This can be used to further improve the algorithm: if the stability period is too short, using the computed density value does not make sense; if it is long enough, the algorithm can wait for new nodes to arrive or for nodes to leave (which is useful to control more precisely the number of messages in the network).

This example shows that there can be significant advantages that come from having low level universal primitives. In this case, instead of hiding information and problems from the user, giving him access to low level context information can help optimize the algorithms.

5 NCIPs can restore hidden features.

Example of anonymity ensured by the one way broadcast NCIP.

Assume the following problem. A piece of information, say I, is stored at a given node, say N. The goal of N is to get rid of I, after making sure that it is now stored by another node of the network.

Using standard primitives this would most likely be implemented as show algorithm 2.

Let us analyze this algorithm. First, it makes a number of implicit assumptions. At line 23, it supposes that it can access the identities of its neighbor

Algorithm 2 Information storage with the standard primitives

```

1  [...]
2
3  // this method is invoked when a message is received
4
5  void receivedMessage(Node from, Message message) {
6
7      switch (message.getType()) {
8
9          case DATA_TO_STORE :      // someone passed me a piece of data (I) to store
10         I = message.getData();
11         send(from, ACK);
12         break;
13
14         case ACK :                  // some one as got my data I, and I can thus get rid of it
15         I = null;
16         break;
17     }
18 }
19
20 // this method is invoked when the internal timer is fired
21
22 void timerFired(){
23     Node neighbor = selectRandomNeighbor();
24     send(neighbor, I);
25 }
26
27 [...]
```

nodes. It furthermore assumes that the information that it gets is stable and that it can use it at line 24. Nevertheless, in a real world mobile network, it might be the case that between the execution of these two lines of code, the target node has moved or disappeared and cannot be accessed any longer. So, there is an implicit assumption about the stability of the network at that point of the algorithm. The same kind of assumption is made line 11. In the real world, there is no reason why the target of the ACK message should be in reach of the current node at that time. Second, from a functional point of view, there is an indirect side effect of using a high level communication primitive: the initial (respectively the final) owner of the information knows which node is finally (respectively was initially) owning the information.

The point is that we have considered an algorithm designed for a reliable static framework and we have ported it almost directly to a totally distributed context. This implicitly assumes a number of hypothesis that do not hold in a MANet.

Therefore we have designed algorithm 3 which is a NCIP based implementation. We just assume a communication primitive that makes it possible to broadcast a message without any knowledge of the neighborhood, without any guarantee regarding the possible reception of the message and without any knowledge about the identity of any possible receiver. We call this NCIP one way broadcast.

The behavior of the algorithm is as follows. A node that holds the information broadcasts it at regular intervals (line 27). It still holds the information till an acknowledgment gets back to it through the network. Nevertheless, this ac-

knowledge does not necessarily come from a node that directly received the message since there is no guarantee that we have bidirectional communication. Rather, when a node receives the information, it broadcasts an acknowledgment to its neighborhood (line 9). This neighborhood in turn broadcasts this ACK (line 17), until a given TTL (the TTL management is not shown on the algorithm to make it easier to read). At some point in time, the initial node might receive an acknowledgment. It then removes the information (line 14). While no acknowledgment is received, the node simply holds the information and at some point possibly restarts the whole process. The behavior of the algorithm is illustrated figure 2.

Algorithm 3 Information storage with NCIPs

```

1  [...]
2
3  void receivedMessage(Node from, Message message) {
4
5      switch (message.getType()) {
6
7          case DATA_TO_STORE :
8              I = message.getData();
9              send(*, (ACK, I.number));
10             break;
11
12          case ACK :
13              if (message.number == I.number) { // this is the ACK for our own I
14                  I = null;
15              }
16              else {
17                  send(*, (ACK, message.number));
18              }
19              break;
20          }
21     }
22
23     void timerFired(){
24
25         if (I!=null) {
26             I.number++;
27             send(*, I);
28         }
29     }
30
31     [...]

```

This algorithm has been simulated on an adaptation to a mobile context of the DA-GRS simulator [13] that is developed in our team. This simulation allowed us to validate this algorithm in an experimental way.

By adapting/developing this algorithm based on NCIPS we have gained two major benefits: no unrealistic or useless assumption on the network mobility nor communication capacities have to be made; the privacy problem described above disappears, i.e. no node knows which other nodes now store the information or have stored the information in the past.

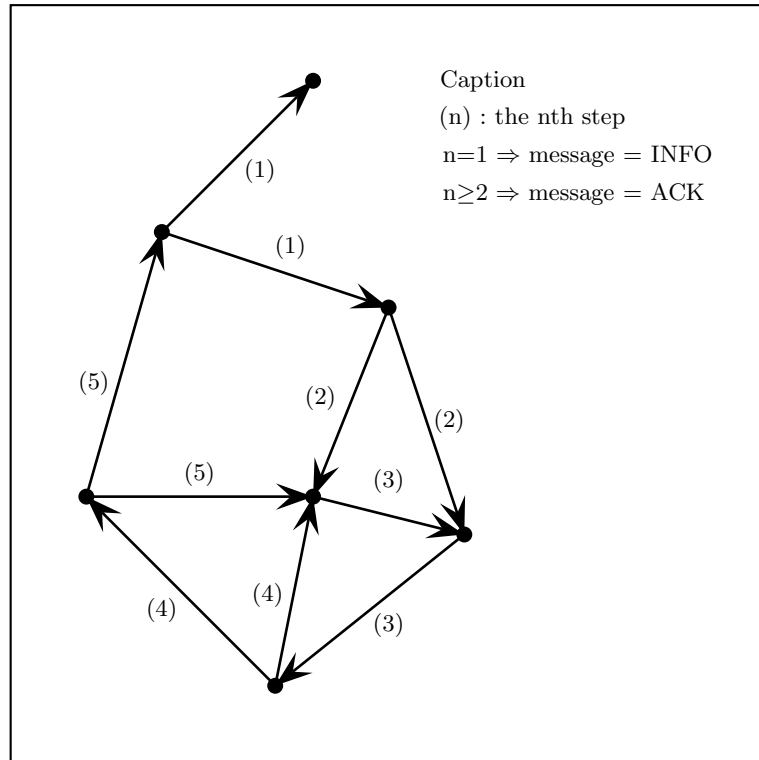


Fig. 2. The different steps of the information storage algorithm on a sample network

Once again this shows that working with too high level functions necessarily leads to implicit assumptions that are basically false in a MANet. It is furthermore especially interesting to see that we can achieve a better result (here in terms of anonymity) with less assumptions.

6 Conclusion and future work

In this paper we have introduced the notion of Neighborhood and Context Interaction Primitives (NCIPs). These are the most basic features that can be supported by a mobile network without making unrealistic assumptions. We have described a number of examples that illustrate some interesting features of NCIPs: they can be used to define complex operations (example 1); they can improve the efficiency of algorithms (example 2); they can guarantee properties that are not supported by more standard primitives (example 3).

Nevertheless, this is preliminary work and a lot remains to be done. We are currently working on the definition of a number of basic primitives. Based on these definitions we explore the higher level operations and algorithms that they make possible to implement. In the short term we will implement these primitives

on several platforms that we are working on in the team, i.e. mobile phones, PDAs, and sensors. We furthermore work on a formal model based on a graph rewriting approach[3] that we have extended to deal with dynamic networks.

References

1. Crossbow Technology. <http://www.xbow.com/>.
2. TinyOS: An open-source OS for the networked sensor regime, 2007. Available at <http://www.tinyos.net/>.
3. A. Casteigts and S. Chaumette. Dynamicity-Aware Graph Relabeling Systems (DA-GRS), a local computation-based model to describe MANet algorithms. In *International Conference on Parallel and Distributed Computing and Systems (PDCS'05)*, Dallas, USA, 2005. IASTED Press.
4. Jason L. Hill and David E. Culler. Mica: A Wireless Platform for Deeply Embedded Networks. *IEEE Micro*, 22(6):12–24, 2002.
5. Luc Hogue. *Mobile Ad Hoc Networks: Modelling, Simulation and Broadcast-based Application*. PhD thesis, University of Le Havre, University of Luxembourg, 2007.
6. Sun Microsystems Incorporation. Jxta v2.0 protocols specification, January 2007.
7. P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, Oct. 2002.
8. Jun Luo and Jean-Pierre Hubaux. A survey of inter-vehicle communication. Available at citeseer.ist.psu.edu/luo04survey.html.
9. M. Mamei, F. Zambonelli, and L. Leonardi. Tuples On The Air: a middleware for context-aware computing in dynamic networks. In IEEE, editor, *Proceedings of the 2nd International Workshop on Mobile Computing Middleware at the 23rd International Conference on Distributed Computing Systems (ICDCS)*, pages 342–347, Providence, RI, USA, May 2003.
10. Y. Métivier, Nasser Saheb, and Akka Zemmari. Randomized rendezvous. In *Colloquium on mathematics and computer science: algorithms, trees, combinatorics and probabilities*, Trends in mathematics, pages 183–194. Birkhäuser, 2000.
11. S. Madden P. Levis, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *Ambient Intelligence*, chapter TinyOS: An Operating System for Sensor Networks, pages 115–148. Springer, 2005.
12. Doug Simon, Cristina Cifuentes, Dave Cleal, John Daniels, and Derek White. JavaTM on the bare metal of wireless sensor devices: the squawk Java virtual machine. In *VEE '06: Proceedings of the 2nd international conference on Virtual execution environments*, pages 78–88, New York, NY, USA, 2006. ACM Press.
13. The DA-GRS simulator. <http://www.labri.fr/perso/casteigt/simulator.html>.
14. Ting Wang, Shuang Hao, Ping Wang, and Gang Peng. Efficient and density-aware routing in wireless sensor networks. In *The 15th IEEE International Conference on Communication and Networks (ICCCN'06)*, 2006.
15. Sau Yee Wong, Joo Ghee Lim, S.V. Rao, and W.K.G Seah. Density-aware hop-count localization (DHL) in wireless sensor networks with variable density. In *Wireless Communications and Networking Conference, IEEE*, pages 1848 – 1853 Vol. 3, 2005.