GCP: Gossip-based Code Propagation for Large-scale Mobile Wireless Sensor Networks

Yann Busnel*
IRISA / UR1 - ENS Cachan
Yann.Busnel@irisa.fr

Marin Bertier
IRISA / INSA Rennes
Marin.Bertier@irisa.fr

Eric Fleury
CITI / INSA Lyon
Eric.Fleury@inria.fr

Anne-Marie Kermarrec
IRISA / INRIA Rennes
akermarr@irisa.fr

ABSTRACT

Wireless sensor networks (WSN) have recently received an increasing interest. They are now expected to be deployed for long periods of time, thus requiring software updates. Updating the software code automatically on a huge number of sensors is a challenging task, especially when all participating sensors are embedded on mobile entities. In this paper, we investigate an approach to automatically update software in mobile sensor-based applications when no localization mechanism is available. We leverage the peer-to-peer cooperation paradigm to achieve a good trade-off between reliability and scalability of code propagation. More specifically, we present the design and evaluation of GCP (*Gossip-based Code Propagation*), a distributed software update algorithm for mobile wireless sensor networks. GCP relies on two different mechanisms, piggybacking and forwarding control, to balance the load among sensors without sacrificing on the propagation speed. We compare GCP against traditional dissemination approaches. Simulation results based on both synthetic and realistic workloads show that GCP achieves a good convergence speed while balancing the load evenly between sensors.

1. INTRODUCTION

Context. Sensor devices have the properties of small size, low cost, low power consumption. Combined in a network, they allow to consider many new applications.
Sensor networks are extremely constrained due to their energy limitation. This implies a special attention to reduce the number of messages exchanged and the computation time [1, 11].

*Yann Busnel is the contact author: Campus Universitaire de Beaulieu – F-35042 Rennes Cedex – France

AUTONOMICS 2007, 28-30 October 2007, Rome, Italy
Copyright © 2007 ICST 978-963-9799-09-7
DOI 10.4108/ICST.AUTONOMICS2007.2227

- Also available as Research Report INRIA-RR-6251, June 2007, Rennes, France.

Problem statement. Sensors may be deployed in either static or dynamic environments, usually for an expected long period of time. While efficient solutions to software update may be deployed in static WSNs, this is far more complex when sensors are embedded on mobile entities such as people for example.
In this paper, we consider only the code propagation service; we do not consider how to dynamically install new code on sensors. This issue has a lot in common with broadcast and data dissemination [4, 6], with an additional main constraint. In data dissemination, new nodes can ignore every message exchanged before it joins. In code dissemination however, new nodes have to be able to obtain the software's latest version as soon as possible in order to be operational.

Applying P2P paradigm to mobile WSN. WSNs can be consider as a distributed system where no entity is able to reach the entire network as in wired internet-based systems but only a small subset of neighbours. Consequently, we believe it is possible to apply in this context of the peer-to-peer (P2P) communication paradigm. In a P2P system, each node may act both as a client and a server, and knows only few other nodes. Each node is connected only to a subset of participating nodes forming a logical overlay over the physical one. Resources are aggregating and the load[1] is evenly balanced between all peers in the system. Central points of failures disappear as well as associated performance bottlenecks [3, 14].
The aim of this work is to leverage the research in P2P systems to design solution for WSNs. If there are several similarities between the two kinds of system, differences remain between them which have a strong impact on the solution design. In a sensor network, a node is able to communicate only with a subset of the network within its communication range. In addition, in a mobile WSN, the neighbourhood of a node changes according to its mobility pattern. The other difference is the multicast property of the wireless medium. When a sensor node sends a message, every node in its direct neighbourhood can receive this message while in a wired network a message is received only by the nodes, which are explicitly designated in the message.

[1]The load is composed of forwarding messages, storing data, *etc.*

Epidemic algorithm. Epidemic or gossip-based communication is well-known to provide a simple scalable efficient and reliable way to disseminate information [8]. Epidemic protocols are based on continuous information exchange between nodes. Periodically, each node in the system chooses randomly a node in its neighbourhood to exchange information about itself or its neighbourhood. Introduced in unstructured P2P overlay, gossip-based protocols can be successfully applied in WSNs. Recently, several approaches based on gossip have been proposed in this context [9, 10, 12, 13].

Contribution. The contribution of this paper is to apply the epidemic approach to achieve efficient and reliable software dissemination in mobile WSNs. We introduce a greedy protocol (*GCP: Gossip-based Code Propagation*) balancing the dissemination load without increasing diffusion time. GCP relies on *piggy-backing* to save up energy and *forwarding control* to balance the load among the nodes.

2. GCP: INTRODUCING CONTROL IN FLOODING FOR MOBILE WSNS

In mobile wireless sensor networks, routing and broadcasting is a challenging task due to the network dynamicity. To the best of our knowledge, existing approaches do not deal with persistent diffusion (*cf.* [2]).

2.1 System model

In the following, we consider a distributed system consisting of a finite set of mobile sensor nodes, which are not aware of their geographic location. The network may not be connected at any time as at a time t, a node can only communicate with nodes in its communication range. However, we consider that over application duration, there is an infinite number of paths between two nodes. In order to discover its neighbourhood, each node broadcasts locally `Hello` messages, called *beacon*.

2.2 GCP design

Flooding paradigm is a simple way to disseminate information. Used commonly in the network area, it consists in forwarding to everyone known new received information. Rather than applying classical flooding algorithms having ideal speed propagation at the price of a high-energy consumption, GCP is inspired from the flooding paradigm enhanced through*Piggy-Backing* and *Forwarding Control.*

Piggy-Backing mechanism. In order to avoid unnecessary software transmissions, nodes have to be aware of the software versions hold by their neighbours. To this end, each node simply piggybacks its own version number into beacon messages.

Forwarding Control mechanism. In order to balance the load among nodes and increase the overall lifetime of a system, each node sends its current software version a limited number of times. To this end, each node owns a given number of *tokens*, which value is a system parameter. Sending a software update is worth a token. When a node has spent all its tokens, it is no longer allowed to send this version of the software. The number of tokens is associated to each version. Upon receiving a new version of the software, the number of tokens is set to the default initial value. This mimics the behaviour of an epidemic protocol, where each node sends a predefined number of time a message (typically $\log(N)$, N being the size of the system) [8]. Likewise, the default value of the number of tokens can be set according to the order of magnitude of WSN size.

2.3 GCP algorithm

Figure $1.a$ represents the three possible different cases and the GCP behaviour.
Step 1, each node in the transmission range of a receives the beacon (In Figure $1.a$: nodes b, c and d; node e is out of range). A beacon message received by node b is processed as follows:

2a. If b owns the same version as a ($v_a = v_b$) due to the piggybacking mechanism or $token_b = 0$ due to the forwarding control mechanism, then no action is required.

2b. If b owns a more recent version than a ($v_a < v_b$) due to the piggybacking mechanism, and, if b still holds some tokens ($token_b > 0$) due to the forwarding control mechanism, it sends its version to a thus consuming a token ($token_b$--). Note that if other nodes, within the transmission range, holding an older version than b's, they leverage the software update and update their own version ("free update").

2c. If a owns a version newer than b ($v_a > v_b$) due to the piggybacking mechanism, the node b sends immediately a beacon message in order to request a software update from a while a is still in its radio range.

2.4 Alternative algorithms

In order to assess the efficiency of GCP, we compare it against three other protocols, directly derived from wired networks. We briefly present those protocols in this section.

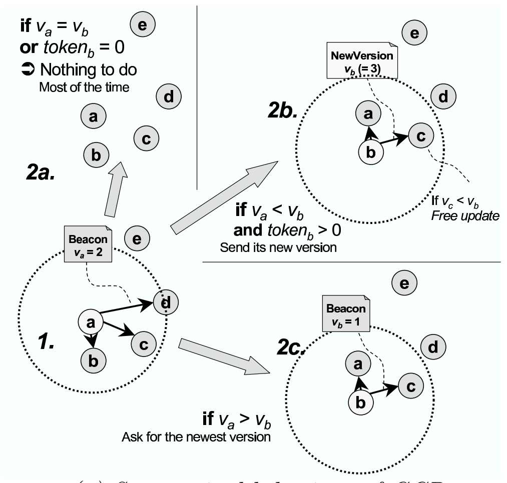
The Flooding Protocol (FP) Each time a node receives a beacon from another one, it sends its own version of the software, whether the node needs it or not. This algorithm obviously leads to load unbalance and does not take into account energy consumption. This algorithm is presented here because it provides ideal software propagation speed.

The Forwarding Control Protocol (FCP) This algorithm is an enhancement of the flooding protocol, using the forwarding control mechanism.

The Piggy-Backing Protocol (PBP) This last algorithm is an enhancement of the flooding protocol, using the piggybacking mechanism.

Thereafter, we introduce a formalized version of these protocols.

- $version$ and $version_r$ represent respectively the local and the remote version number of the software;
- $software$ and $software_r$ represent respectively the local and the remote binary of the software;



(a) Summarized behaviour of GCP

```
RECEIVEBEACON($version_r$)
1 if $version_r < version$
2     and $token > 0$
3   then
4        send ($software, version$)
5        $token \leftarrow token - 1$
6   else if $version_r > version$
7          then
8               send $Beacon(version)$

RECEIVESOFTWARE($software_r, version_r$)
1 if $version_r > version$
2   then
3        updateCode ($software_r$)
4        $version \leftarrow version_r$
5        $token \leftarrow initialNumberOfTokens$
```

(b) GCP algorithm in pseudo-code

Figure 1: Gossip-based Code propagation algorithm

- *token* represents the remaining number of tokens available on the local node;
- *initialNumberOfTokens* represents the initial number of tokens available on the local node.

The GCP algorithm is presented in Figure $1.b$. PBP has the same pseudo code without lines 2, 5 in ReceiveBeacon and without line 5 in ReceiveSoftware. FCP has the same pseudo code without lines 1, 6, 7 and 8 in ReceiveBeacon. A theoretical analysis is provided in [2].

3. SIMULATION RESULTS

3.1 System settings

We assume that nodes can communicate only by 1-hop broadcast with nodes in their transmission range, with no collision. For a node S, we distinguish two ranges of transmission, r and R, where $0 < r < R$ and $\mathcal{S}_S(r) \subset \mathcal{S}_S(R)^2$. r represents the radius where the transmission range is uniform, and thus messages sent by nodes separated by less than r are always received. The second range R represents the radius at which transmission range may be not uniform. No nodes separated by more than R can receive each other transmission. Thus, nodes separated by a distance between r and R may or not receive each other transmitted messages according to the Equation 1. In Equation 1, P_{min} is the transmission's lower bound probability parameter for two nodes separated by R. We consider that sensor nodes have equal communication ranges. Nodes have a transmission probability defined as follows:

$$\begin{cases} 1 & \text{if } d < r \\ P_{min} - \sqrt{\frac{R-d}{R-r}} \cdot \left(\frac{R-d}{R-r} - 5\right) \cdot \frac{1-P_{min}}{4} & \text{if } r < d < R \\ 0 & \text{if } d > R \end{cases} \quad (1)$$

In simulation, the transmission ranges are set as follows: $r = 3m$, $R = 5m$ and a minimum transmission probability inside R, $P_{min} = 0.3$.

We consider mobile sensors. In order to compare our results with other ones in the literature, we choose, for the synthetic workload, the widely used random way point mobility model [7].

$^2\mathcal{S}_S(x)$ is the sphere notation with center S and radius x.

3.2 Simulation setup

Simulator. In order to evaluate GCP, we developed SeNSim, a software implemented for mobile wireless sensor-based applications' simulation. SeNSim is a Java-based software, which allows the creation of mobile wireless sensor networks and analyses information dissemination under different mobility and failures scenarios. The simulator also allows the evaluation of the characteristics related to this protocol under different mobility, failures, and stimulus scenarios.

Workloads. We evaluate different scenarios with the same set of workloads for comparison purposes. By running different algorithms on a same persistent trace, we obtain a fair comparison between solutions introduced in Section 2.

We conducted experiments on various clustering scenarios (8 synthetic and one realistic). For space reason, we do not detail all of them in this paper and focus on the following. Complete results are available in [2]. In order to evaluate the performance of GCP with a realistic movement behaviour, we used the mit/reality data set [5] from CRAWDAD. This data set provides captured communication, proximity, location and activity information from 100 subjects at MIT over the course of the 2004-2005 academic year. The time is discretized in millisecond. Simulations run for 50,000 ms. A new version is sent to a sensor picked up at random after 1 s of simulation.

Each sensor node:

- is initially randomly placed inside its defined area for synthetic workloads;
- sends a beacon periodically every 100 ms;
- is mobile, following a *Random Way Point* strategy for synthetic workloads, with a maximum pause time of $100ms$, each movement duration between 100 and $500ms$, with a speed included between $0.8m.s^{-1}$ ($2.88km.h^{-1}$) and $2m.s^{-1}$ ($7.2km.h^{-1}$) (equivalent to human walking speed). The defined area bounds every

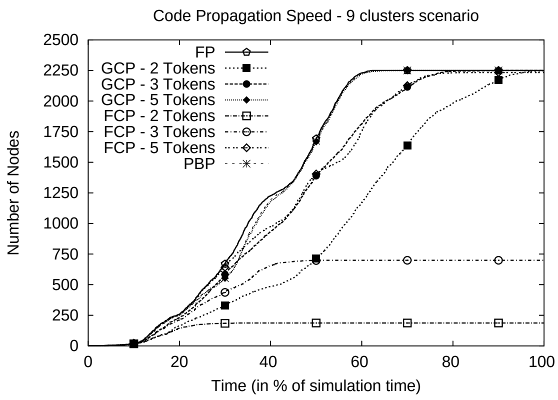


Figure 2: Convergence speed for the largest clustered scenario

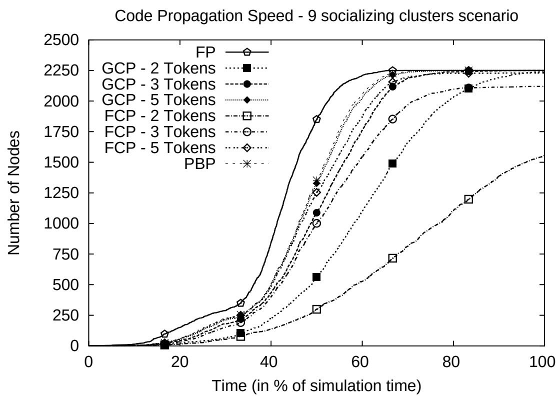


Figure 3: Convergence speed for the largest socialized scenario

movement. The border rules are defined as each node bounces back according to the bisector of incidence angle.

- has a number of tokens, for each code propagation, according to simulation configuration describes below.

In order to compare the efficiency of the four algorithms, we compared them along the following metrics:

Code propagation speed We observe the number of nodes owning the newest version of the software during all the simulations.

Load balancing At the simulation termination, we extract from each node the number of times it has sent the software.

Results are depicted in Section 3.3 for the studied scenario and algorithms.

3.3 Results

Convergence speed. Figures 2, 3 and 4 present the results according to time, for three scenarios ordered as 9 cluster scenario, 9 socializing cluster scenario (i.e. with some free moving nodes) and the MIT campus realistic workload scenario.

Each synthetic scenario has approximately the same propagation behaviour as the 9 socializing cluster scenario (cf. Figure 3). Due to the space constraints, we have represented the results for only two synthetic scenarios. The 9 cluster scenario is presented here to illustrate the fact that GCP outperforms the FCP algorithm.

For the two algorithms with token rules (Forwarding control mechanism), we have plotted the results obtained by using k tokens a node, where k is respectively equal to 2, 3 and 5. We do not represent results for more than 5 tokens as in most cases, GCP tends to approach the ideal reference by using only 5 tokens (Each synthetic simulation system counts around 2,000 sensors, so for all $k \sim \log(2000) = 3.3$ [8]).
Regardless of the number of tokens chosen, in each scenario, GCP outperforms FCP as far as propagation speed is concerned. Taking flooding algorithm as a reference (ideal case), Figure 2 shows different inflection points, due to the software transmission from a cluster to another. This is clearly denoted in this figure: when almost half of the sensor nodes have received the software's newest version, there is a period during which the newest version is moving from one cluster to another.

Figure 4 presents the propagation speed for the realistic scenario. We observe that FCP algorithm is not efficient in this scenario. In real life, some people are together most of the time. As nodes used FCP algorithm, they are not aware of the remote nodes' version, and may spend all their tokens for the same node. GCP with a small number of tokens (2 and 3 here for instance) remains better than FCP with a large number of tokens. With only 5 tokens per node, GCP is almost as efficient as the flooding algorithm with respect to propagation speed.

For each scenario, PBP is significantly slower than the flooding one and is always equivalent to GCP with 5 tokens per node. Simulation results show the propagation speed efficiency of GCP according to the FCP algorithm for the same number of tokens and to the flooding algorithm as ideal reference. We have measured as well the network load balancing, presented in the next subsection.

Load Balancing. For each simulation, Figure 5 presents the results for the 9 socializing cluster scenario. For each number of message sent (represents by the X-axes), the number of nodes, which have sent exactly this number of time its software, is represented. As the flooding consumes much more messages than the three other algorithms (PBP, FCP and GCP), the network load with flooding is represented in the upper-right corner of Figure 5.

For each scenario, the benefit of GCP or FCP over the flooding algorithm is clear. When considering the number of software binary sent, the two other algorithms save between 82 % and 93 % of messages for FCP and more than 98 % for GCP for a 50 seconds simulation only. The number of software sent message increases linearly with time in the flooding algorithm.

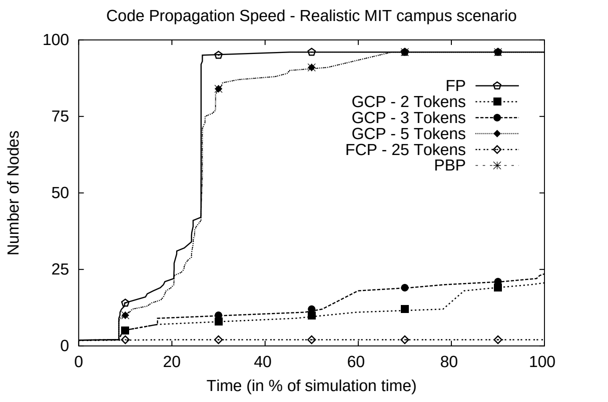


Figure 4: Convergence speed for realistic scenario

Figure 5 shows the benefits of GCP over PBP and FCP: as FCP is not aware of the remote node's version, the local node sends its own version as long as it still possesses tokens. That implies the number of software sending messages with FCP algorithm is almost constant, corresponding of two times the number of tokens owned by each node (k tokens for the first version plus k tokens for the newer version: in these simulations, the software is updated only once).
With GCP and PBP, the current local version number is sent in the beacon message (Piggy-backing mechanism). So, nodes do not send the first version. They only send the newest version, and only if the beacon sender does not own the latest version. The total number of software sending in the network is almost equivalent to the number of participating nodes. Moreover, as every node in the transmission range of a sending node receives the sending version, freely for the sender, the network load benefits using GCP is decreased all the more. Comparing GCP with 5 tokens a node and PBP (same propagation speed), we observe that almost 5 % of the nodes have sent the software more that 5 times, as opposed to GCP where nodes have consumed at most 5 tokens.

We do not represent the load extracted from simulation on the realistic trace because it shares the same aspect as the synthetic ones.

4. CONCLUSION

In this paper, we have proposed a software update algorithm for mobile wireless sensor networks. To the best of our knowledge, tackling code propagation in mobile WSN has not been done before. Leveraging epidemic protocols introduced in P2P systems, the *Gossip-based Code Propagation* algorithm tends to outperform classical dissemination algorithms, imposing a small overhead, adding little extra information on sensor nodes and in beacon messages.

We have exposed the benefit of GCP on several simulation scenarios, compared to three other dissemination algorithms: one ideal in speed convergence but with a large number of software sending and, therefore, very high power consumption, another one based on forwarding control and a last one based on piggy-backing message. GCP outperforms each of these algorithms. It can disseminate the new software with almost the same propagation speed than the ideal one while balancing the load evenly between sensors.

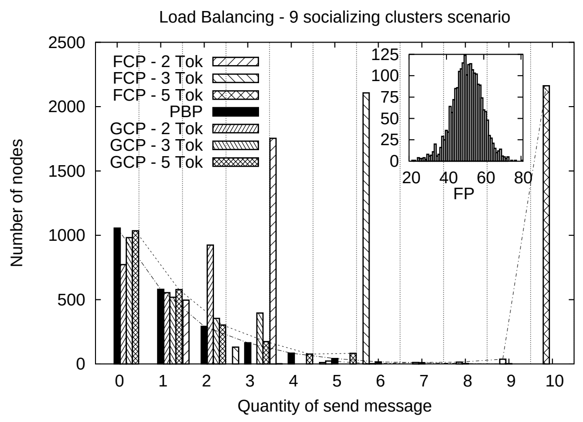


Figure 5: Load balancing for the largest socialized scenario

5. REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[2] Y. Busnel, M. Bertier, E. Fleury, and A.-M. Kermarrec. Gcp: Gossip-based code propagation for large-scale mobile wireless sensor networks. Research Report RR-6251, INRIA, Rennes, France, June 2007.

[3] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsoft Reasearch, Cambridge, UK, 2003.

[4] K. Chen and Y. Jian. Survey on peer to peer data dissemination in manet. Computer Information Science Engineering Department, November 2005.

[5] N. Eagle and A. S. Pentland. CRAWDAD data set mit/reality (v. 2005-07-01). Downloaded from http://crawdad.cs.dartmouth.edu/mit/reality, July 2005.

[6] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions Networks*, 11(1):2–16, February 2003.

[7] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, 353:153–181, 1996.

[8] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3), March 2003.

[9] P. Kyasanur, R. R. Choudhury, and I. Gupta. Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. In *The Third IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2006)*, Vancouver, Canada, October 2006.

[10] P. Levis, N. Patel, S. Shenker, and D. Culler. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *First Symposium on Network Systems Design and Implementation (NSDI)*, march 2004.

[11] P. Rentala, R. Musunuri, S. Gandham, and U. Saxena. Survey on sensor networks. In *Proceedings of International Conference on Mobile Computing and Networking*, 2001.

[12] G. Wang, D. Lu, W. Jia, and J. Cao. Reliable gossip-based broadcast protocol in mobile ad hoc networks. In *Mobile Ad-hoc and Sensor Networks: First International Conference, MSN 2005*, pages 207–218, Wuhan, China, December 2005.

[13] L. Wang and S. S. Kulkarni. Gappa: Gossip based multi-channel reprogramming for sensor networks. In *Second IEEE International Conference in Distributed Computing in Sensor Systems (DCOSS 2006)*, pages 119–134, 2006.

[14] B. Xu and O. Wolfson. Data management in mobile peer-to-peer networks. In S. V. L. N. in Computer Science, editor, *Proc. of the 2nd International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P'04)*, pages 1–15, Toronto, Canada, August 2004.