Poster: Network Bootstrapping and Leader Election Utilizing the Capture EEect in Low-power Wireless Networks

Beshr Al Nahas, Simon Duquennoy, Olaf Landsiedel

Poster: Network Bootstrapping and Leader Election Utilizing the Capture Effect in Low-power Wireless Networks

Beshr Al Nahas
beshr@chalmers.se
Chalmers University of Technology
Gothenburg, Sweden

Simon Duquennoy
simon.duquennoy@ri.se
RISE SICS, Sweden
Inria Lille - Nord Europe, France

Olaf Landsiedel
olafl@chalmers.se
Chalmers University of Technology
Gothenburg, Sweden

ABSTRACT

Many protocols in low-power wireless networks require a root node or a *leader* to bootstrap and maintain its operation. For example, Chaos and Glossy networks need an initiator to synchronize and initiate the communications rounds. Commonly, these protocols use a fixed, compile-time defined node as the leader. In this work, we tackle the challenge of dynamically bootstrapping the network and electing a leader in low-power wireless scenarios, and we focus on Chaos-style networks.

ACM Reference format:
Beshr Al Nahas, Simon Duquennoy, and Olaf Landsiedel. 2017. *Poster*: Network Bootstrapping and Leader Election Utilizing the Capture Effect in Low-power Wireless Networks. In *Proceedings of SenSys '17, Delft, Netherlands, November 6–8, 2017,* 2 pages.
DOI: 10.1145/3131672.3137002

1 INTRODUCTION

Context and Challenge. Many protocols in low-power wireless networks require an entity to bootstrap and maintain the operation, which we denote a *leader*. For example, RPL networks need a network root to build the routing tree, TSCH networks need a time-source root to synchronize the network and Glossy/Chaos/LWB networks need an initiator to synchronize and initiate the communications rounds. In applications that build their operation on consensus; *e.g.*, two-phase commit, the leader is responsible for proposing and committing transactions. In the recent work [1, 3], the common solution was to use a fixed, compile-time defined node to be the leader.

The use of a statically defined leader exhibit the following weaknesses; (a) it assumes a known network deployment; thus, it does not suit random deployments; *e.g.*, throwing nodes from the air; (b) it assumes a static network; thus, mobility is limited, and, (c) initiator failure means a network failure and might require manual intervention to restart the network operation. While the problem of clustering and leader election is not new as it was tackled by Heinzelman *et al.* in LEACH [2] and subsequent work, there is a need for an approach that both suits and benefits from the low latency of recent approaches to synchronous transmissions, such as Glossy and Chaos.

Approach. In this paper, we tackle the challenge of dynamically electing a leader in low-power wireless networks. We propose mechanisms that achieve (a) network bootstrapping; *i.e.*, network synchronization, and clustering; (b) leader election and ensuring the convergence toward one leader and (c) leader failure recovery. We build on top of our work A^2 [4] and its lower layer, Synchrotron, the synchronous transmission protocol that is inspired by Chaos.

Outline. We provide the required background on A^2 and synchronous transmission in §2. Then we explain the design in §3 and conclude with preliminary results in §4.

2 BACKGROUND: A^2 AND SYNCHROTRON

A^2 builds on top a synchronous transmissions kernel, Synchrotron, and utilizes in-network processing to provide primitives for network-wide, all-to-all dissemination, collection, aggregation, voting, consensus (two- and three-phase commit) and membership services. A^2 operates in *rounds* where nodes send packets synchronously and receive data thanks to the *capture effect*.

Synchrotron: Synchronous transmissions and capture effect. Synchrotron roots in approaches to synchronous transmissions, such as Chaos, where multiple nodes synchronously transmit the data they want to share. Nodes overhearing the concurrent transmissions receive one of them with high probability, due to the capture effect. For example, to achieve capture with IEEE 802.15.4 radios, nodes need to start transmitting within the duration of the preamble of 160μs [3].

Synchrotron operates as a time-slotted protocol. The minimum time unit is a *slot*, which fits one packet transmission/reception and processing. Slots are grouped in *rounds*, where a designated function, such as collect or disseminate is run network-wide. Within each slot, a node transmits, receives or sleeps according to the transmission policy of the application.

In-network aggregation. In A^2, each packet contains so-called *progress flags*, where one bit is assigned to each node in the network. The coordinator node starts an A^2 *round* by sending a packet with only its own flag set. Upon successful reception, a node sets its flag and merges the received packet with its own. It transmits in the next time slot when it received new information, *i.e.*, new flags, or when it sees that a neighboring node is transmitting messages with fewer flags set, *i.e.*, a neighbor knows less than the node itself. The process continues until all nodes have set their flag.

Similar to Chaos, the rules for merging are application specific. With the *Max* operation, for example, A^2 identifies the maximum value: Next to the flags, the only payload is the maximum value collected so far. Upon reception, nodes compute the max between their local value and the payload, write it to the packet payload, merge the flags, and set their flag before transmitting in the next time-slot.

SenSys '17, Delft, Netherlands

3 DESIGN

To start a proper network operation in A^2, there shall be an initiator node or a *leader* that (a) ensures network-wide *synchronization*; (b) *joins* the nodes that wish to participate in the network and assigns each of them a unique ID; and (c) initiate the communication rounds and ensures the application objectives are met.

To be able to communicate, we need to synchronize the nodes. We start by forming clusters that ensure neighborhood synchronization. Then, we merge the clusters gradually to form one network-wide cluster with a single leader.

3.1 Bootstrapping and Clustering

We start by having two assumptions (a) the nodes are homogeneous and any of them could be a leader. This assumption is by no means compulsory, but it simplifies the discussion; and (b) the maximum number of nodes is known before hand.

Every node start by listening to the radio and generates a random timeout. It keeps listening until it hears a valid A^2 message to synchronize on. If it times out without hearing, then it proposes itself a leader and starts sending *join* announcements such that other nodes hear them and join it to form a cluster. At this phase, multiple clusters could form as different nodes might not hear each other. The next step is to converge towards one leader in one cluster.

3.2 Leader Election

To ensure the convergence toward one leader, we put a quorum stability condition: A stable cluster is the one that has more than half of the nodes. Given that the nodes cannot join more than one cluster, we can ensure convergence.

Until a cluster is stable, it keeps running join rounds, and its members keep sampling the medium between the communication rounds looking for bigger clusters to join. When a node hears another cluster, it saves the synchronization information of the largest foreign cluster it heard. Each node shares this information in the next join round with its cluster, and use the *Max* primitive to find the information about the largest cluster. At the end of the join round, nodes drop their cluster, their IDs and jump to join the new cluster if it is bigger than their current cluster. With time, only the largest cluster survives, and only one leader exists.

3.3 Failure Recovery

Upon leader failure, nodes no longer hear packets, and the random timeout mechanism kicks in. This restarts the whole process and elects a new leader as illustrated in §3.1 and §3.2.

4 PRELIMINARY RESULTS AND CONCLUSION

Implementation. We implement the algorithm explained in §3 in C for the Contiki OS targeting wireless nodes equipped with a low-power radio such as TelosB and Wsn430 platforms which feature a 16bit MSP430 CPU @ 4 MHz, 10 kB of RAM, 48 kB of firmware storage and CC2420 radio compatible with 802.15.4.

Figure 1 illustrates an example run on a network of 5 nodes. Figure 2 summarizes the results of running on the testbeds FIT-IoTLAB Euratech and Flocklab. First, we vary the maximum timeout

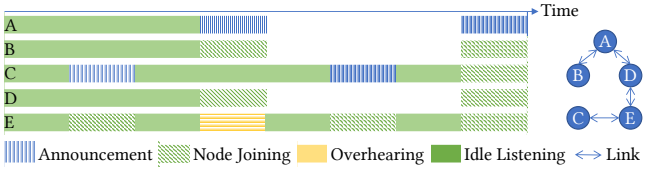
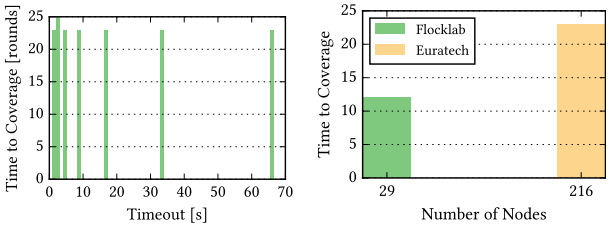


Figure 1: An example run with five nodes. *Nodes A and C propose themselves as leaders. Due to the network setup, two clusters form. E and C keep listening between rounds since their cluster has less than half of the nodes. E overhears A's cluster and notifies its cluster members. The network converges to select A as the only leader.*



(a) Election timeout does not affect convergence time. *The Euratech network always converges within 23 rounds or 41 seconds on average. Each round is 1.8 seconds long.*

(b) Leader election on a sparse and a dense testbeds; *repectively, Flocklab and Euratech.*

Figure 2: The time it takes the leader election procedure to converge to one cluster and join all the nodes in the network.

period and run on Euratech. We find that the choice of the timeout has minimal effect on the time to convergence to one cluster that inlcudes all the nodes. Second, we compare the performance when running on sparse and dense testbeds. The network converges to one leader and all the nodes join the leader within 41 seconds for 216 nodes (on Euratech) and 21 seconds for 29 nodes (on Flocklab).

ACKNOWLEDGMENTS

This work was supported by the Swedish Research Council (VR) through the project ChaosNet, the Swedish Foundation for Strategic Research (SSF) through the project LoWi, CPER Nord-Pas-de-Calais/FEDER DATA and Sweden's innovation agency (VINNOVA).

REFERENCES

[1] Federico Ferrari, Marco Zimmerling, Luca Mottola, and Lothar Thiele. 2012. Low-Power Wireless Bus. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys).*

[2] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. 2000. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the Annual Hawaii International Conference on System Sciences.*

[3] Olaf Landsiedel, Federico Ferrari, and Marco Zimmerling. 2013. Chaos: Versatile and Efficient All-to-All Data Sharing and In-Network Processing at Scale. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys).*

[4] Beshr Al Nahas, Simon Duquennoy, and Olaf Landsiedel. 2017. Network-wide Consensus Utilizing the Capture Effect in Low-power Wireless Networks. In *Proceedings of the Conference on Embedded Networked Sensor Systems (ACM SenSys).*