Scalability Issues of Firefly-based Self-Synchronization in Collective Adaptive Systems

Iva Bojic
University of Zagreb, FER,
Unska 3, Zagreb, Croatia
Email: iva.bojic@fer.hr

Tomislav Lipic
Rudjer Boskovic Institute
Bijenicka cesta 54, Zagreb, Croatia
Email: tomislav.lipic@irb.hr

Mario Kusek
University of Zagreb, FER,
Unska 3, Zagreb, Croatia
Email: mario.kusek@fer.hr

***Abstract*—In this paper we investigate scalability issues of self-synchronization emergent properties, described with the pulse coupled oscillator model. As in the pulse coupled oscillator model the information propagation process is a gossip-like process, huge amounts of network traffic can be generated, causing thus scalability issues of the whole collective adaptive systems. These issues are even more emphasized in collective adaptive heterogeneous systems called Machine-to-Machine (M2M) systems. Namely, these systems consist not only from one large complex network, but also from a larger number of different interconnected complex networks. The easiest way to reduce network traffic in large networks is to use different overlay network topologies. An overlay network topology can be seen as a layer of a virtual network topology on top of a physical network, enabling significantly less messages to be exchanged during a synchronization process. However, the implementation process of overlay network topologies is not very efficient in real-world environments, as will be discussed in the paper. Therefore, we propose a mechanism for selective coupling implemented on the sender side that can be used to reduce both network traffic and time to synchronization without negatively affecting the entire synchronization process. Moreover, in some cases the rate of successful synchronization outcomes can be also increased when using the proposed mechanism.**

I. Introduction

As collective adaptive systems grow in size, maintaining their scalability becomes a very important research challenge. We say that a system is scalable if its growth in size does not result in malfunctions. In this paper, the focus is on a special type of collective adaptive systems called Machine-to-Machine (M2M) systems because of an enormous number of predictions about their fast growth in the future. International Data Corporation forecasts 15 billion devices communicating over the network by the year 2015 [1]. Cisco IBSG predicts 25 billion devices connected to the Internet by 2015 and 50 billion by 2020 [2]. Ericsson claims that the vision of more than 50 billion connected devices by 2020 may seem ambitious today, but with the right approach, it is within reach [3]. Machina Research says that by 2022 18 billion M2M connections will exist globally, up from approximately 2 billion today [4].

Although scalability issues of M2M systems can be seen through different perspectives, without a doubt, network traffic can cause serious problems if its increase with the number of interconnected devices exceeds a linear growth. Namely, communication is a backbone of every distributed system and if realized in an all-to-all manner, then the increase in the number of interconnected devices results in a polynomial increase of network traffic.

In this paper we investigate a communication overhead in M2M systems during emerging firefly-based self-synchronization described with the *pulse coupled oscillator* model [5]. In this model, every firefly is modeled as an oscillator, which is in our case implemented on every device in an M2M system. During a synchronization process, every firefly/oscillator/device exchanges information with its neighbors and that causes information propagation through the whole systems. In order to make this process scalable, communication must not be realized in an all-to-all manner.

The central problem discussed in this paper is thus how to reduce network traffic during a synchronization process in an M2M system. The most commonly used approach is to use overlay network topologies. Therefore, we give an overview of related work in which scholars used different overlay network topologies to reduce network traffic. After that we propose a *mechanism for selective coupling implemented on the sender side* that can also slow down a network traffic growth. Our contribution is twofold: first we give a *theoretical discussion* on the overlay network topology implementation efficiency in real-world environments and afterwards we give a *numerical analysis* of the proposed mechanism.

The rest of this paper is organized as follows. Section II introduces the pulse coupled oscillator model for firefly-based self-synchronization emergent properties explaining the difference between *synchrony*, *phase locking* and *frequency locking*. Section III gives a short overview of related work and present our previous work, while Section IV explains the proposed mechanism for selective coupling implemented on the sender side. Finally, Section V concludes the paper and gives directions for future work.

II. Firefly-based synchronization

As collective adaptive systems become larger and more complex, more often inspiration for a different distributed algorithm design is sought in nature. Nature is one enormous system that is powered only by natural forces and has no central control. It is desirable to copy the same good characteristics to collective adaptive systems. When we talk about firefly behavior in nature, the most commonly used mathematical model was proposed by Mirollo and Strogatz [5]. In their model (later on referred to as M&S' model) every oscillator i in systems of N oscillators was described as:

$$x_i(t) = f(\varphi_i) + \sum_{j=1}^{N} \varepsilon_{i,j}\ g_{i,j}(t), \tag{1}$$

where $x_i(t)$ was the value of the state variable z_i at the moment t, φ_i denoted oscillator i internal phase, $f(\varphi_i)$ described oscillator i *excitation* evolution, $\varepsilon_{i,j}$ was a coupling constant, while $g_{i,j}(t)$ was a coupling function between oscillators i and j. They proved that if the function $f(\varphi_i) : [0, 1] \rightarrow [0, 1]$ was smooth, monotonically increasing and concave down, then a set of oscillators connected in an all-to-all manner would always achieve synchronization for any set of initial conditions.

In M&S' model when oscillator i is not coupled, its state variable z_i changes following only its own *excitation* described with the function $f(\varphi_i)$. At the moment t_i^*, when the state variable z_i reaches the *threshold*, oscillator i "fires" and then its state variable z_i jumps back to 0 (i.e., $z_i \in [0, threshold]$). A period between two "fires" is called a *synchronization cycle*. When two oscillators are coupled, oscillator i state variable z_i does not only depend on its own *excitation*, but is also adjusted upon reception of "fires" from the other oscillator. Mirollo and Strogatz proved that over time synchronization emerged from a random situation, making it seem as though all coupled oscillators were "firing" in perfect synchronization. More about their model can be found in [6].

However, as discussed by the authors, M&S' model is not directly applicable in collective adaptive systems. Two main limitations of the model are the following: it does not take into consideration that oscillators may be nonidentical nor does it account for the spatial structure of oscillators. When taking those limitations into consideration, it is mandatory to distinguish among three different levels of synchronization: *synchrony*, *phase locking*, and *frequency locking*. *Synchrony* means "firing" in unison, while *phase locking* is a weaker form of synchronization wherein oscillators do not necessarily "fire" at the same time, but differences between their state variable values are constant and nonzero. Finally, *frequency locking* means that oscillators run at approximately the same frequencies, but that differences between their state variable values are not constant because of these frequency fluctuations.

Mirollo and Strogatz used the term *synchrony* in the strongest possible sense since they were considering that all oscillators were the same. However, in real-world environments devices[1] are different,[2] and it is thus impossible to achieve the true *synchrony*, but some form of a "weaker" level of synchronization. Although in the rest of the paper we will not explicitly consider that oscillators/devices have different frequencies, this is the case when conducting experiments on "real" devices. Consequently, in real-world environments synchronization will be achieved in the form of *frequency locking*. Using more informal language, when talking about frequency locking, the *synchronization precision* is not infinitesimally fine (as in synchrony), but is limited by the *synchronization window* width. A synchronization window is calculated as the maximal difference between all oscillator state variable values at the moments when they "fired". Figure 1 shows an example of how a synchronization window width can be calculated. Circles denote the moments when oscillators "fired".

[1]We use the term *oscillator* when describing the mathematical model, and the term *device* when describing real-world environments.

[2]Devices are different because their computer clocks run on different natural frequencies. These hardware imperfections are the reason why synchronization is needed in the first place. More about the reasons that lie behind the need of synchronization and why physical clocks drift apart can be found in [7].

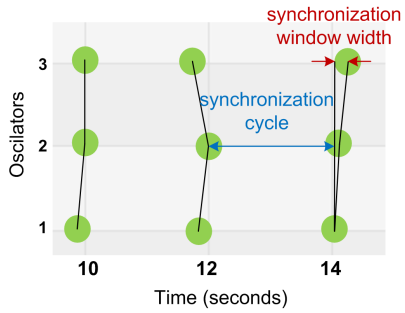

Fig. 1. An example of a synchronization window width

When a synchronization window width is zero, then the synchronization precision is not infinitesimally fine, i.e., we talk about synchrony. Otherwise, as wider this window gets, lower synchronization precision is achieved. Depending on different applications, sometimes we need higher precision and sometimes we are even satisfied with a lower one. However, it is important that the synchronization window width is always much narrower than the synchronization cycle length. If this is not the case, then we are not able to detect synchronization.

If we are not satisfied with the synchronization precision, then we can say that synchronization is not achieved. We thus define a *rate of successful synchronization outcomes* as the number of cases when we achieved desired synchronization precision compare to the all cases when we tried. Moreover, sometimes we want to achieve synchronization with desired precision within a certain period of time. We thus define *time to synchronization* as time needed to achieve synchronization of desired precision. Finally, synchronization is achieved because fireflies in nature exchange information through their pulses, oscillators in mathematical models through their "fires" and devices in real-world environments through their *synchronization messages*. We thus define *network traffic* as the number of messages exchanged during synchronization.

III. Related work

As mentioned earlier, in M&S' model it was assumed that all oscillators were connected in an all-to-all manner and that there was no spatial structure among them. This means that every oscillator is coupled with all other oscillators in the system (i.e., fully-meshed connectivity). There are at least two problems with fully-meshed connectivity. The first one is that in some cases (e.g., multi hop networks such as Internet or heterogeneous systems such as M2M systems) it is physically impossible for all oscillators to be directly connected. The second one is that in some other systems (e.g., small sensor networks) oscillators can communicate directly, but that generates huge amounts of network traffic and is energy consuming. It was thus mandatory to investigate whether it was possible to achieve synchronization even when oscillators were not connected in an all-to-all manner. Almost 15 years after Mirollo and Strogatz's seminal work, Lucarelli and Wang showed that the fully-meshed assumption could be replaced with the partly-meshed assumption and that oscillators could still achieve synchronization [8]. They proved that if oscillators were connected in such a way they formed a connected graph,[3] they would eventually become synchronized.

[3]Graph is connected if there exists a path between any two vertices in it.

Not only has Lucarelli and Wang's work paved the way for the application of the pulse coupled oscillator model in real-world environments, but also it enabled development of mechanisms for a network traffic reduction. Namely, in more recent years, as collective adaptive systems have grown in size, due to scalability issues, more research focus has been put on how to reduce network traffic. The decision to use different *overlay network topologies* to reduce traffic and consequently achieve a higher scalability came as a logical choice since an overlay network topology can be defined as a layer of a virtual network topology on top of a physical network. A lower number of messages can be thus exchanged even if there is fully-meshed connectivity on the physical level. In the rest of the section we will discuss the usage of overlay network topologies both from simulation and testbed points of view.

A. Simulation results

We can use overlay network topologies when wanting to reduce network traffic in different simulated environments. Namely, in simulated environments there is usually no limitation for devices to be connected in an all-to-all manner and when wanting to investigate a network traffic overload, different overlay network topologies can be defined. In our previous work [9], [10] we used a simulator called MASON (Multi-Agent Simulator Of Neighborhoods) [11] in order to compare the following overlay network topologies: fully-meshed, line, ring, mesh(n)[4] and star. The results showed that systems where devices were connected in an all-to-all manner, the most studied ones, were actually not the best ones when considering both time to synchronization and network traffic.

In contrast, simulation results supported our hypothesis that we might get better results when using overlay network topologies. When comparing only time to synchronization and network traffic, a star overlay network topology is the best choice. However, this topology is not completely distributed since the center of a start is a single point of failure and thus can be seen as a potential critical component that could provoke a total system failure in case of malfunction. Because of all that, a better choice would be to use mesh overlay network topologies where the parameter n have to be chosen in respect of the number of devices in a system. In our simulation we had 10 devices, and concluded that mesh(4) overlay network topology performed the best.

Heavy network traffic also leads to higher energy consumption. The amount of energy consumed during a synchronization process is important in energy constrained environments such as sensor networks. Niu et al. [12] simulated a sensor network where the main research focus was to explore a trade-off between the transmission power for each synchronization message and the number of messages exchanged during synchronization. In their simulation, devices were connected only with devices within their transmission range r. A smaller transmission radius needs less transmission power, but also may need more exchanged messages to achieve synchronization. They showed that when transmission radius r was not big enough (i.e., $r < 3$[5]) time to synchronization was too long.

[4]Mesh(n) is partly-meshed connectivity where the parameter n denotes the number of neighbors that every device has.

[5]When r was near or greater than $10\sqrt{2}$, the system was connected in an all-to-all manner.

B. Testbed results

Since simulation results were promising, scholars began implementing different overlay network topologies in real-world environments (i.e., different testbeds). When talking about the implementation of an overlay network topology, one must distinguish between two reasons behind a system with partly-meshed connectivity. Devices can be partly connected because of a lack of fully-meshed connectivity on the physical layer or because of the implementation of the overlay network topology. This means that one cannot conclude immediately that an overlay network topology is implemented in a system only from observation of its partly-meshed connectivity.

For example, Werner-Allen et al. [13] carried out experiments on an indoor wireless sensor network testbed with 24 partly connected devices. However, they did not implement an overlay network topology since there was no connectivity among all devices on the physical layer. Consequently, although they conducted experiments in the system with partly-meshed connectivity, that cannot be seen as the implementation of the overlay network topology. Nevertheless, several scholars implemented overlay network topologies in their testbeds. For instance, Leidenfrost and Elmenreich [14] made experiments on a testbed based on Atmels demonstration kit ATAVRRZ200 [15] with 5 devices ordered in a line, where a device could only communicate with its immediate neighbors. To simulate this partly-meshed connectivity, they implemented a message filter. However, they did not discuss if their solution (i.e., the message filter) is an efficient one.

The reason why the subject is brought up here is because, as far as we know, nobody made a theoretical discussion on the overlay network topology implementation efficiency. Namely, without a doubt, network traffic is reduced when using overlay network topologies. However, there is a cost to it. In the rest of the section we will discuss a calculation cost of finding neighbors, a memory cost of keeping records about neighbors, and finally a time cost of sending *multicast* messages.

1) Calculation and memory costs of finding neighbors: The first problem is how each device can find its neighbors. This heavily depends on the type of an overlay network topology that is used. Overlay network topologies can be regular ones (e.g., ring, line, star) or random ones. If a topology is regular, then every device has to either save its list of neighbors or has to calculate it every time before it sends its synchronization messages. In the former, there is a memory cost and in the latter a calculation cost. Of course, this is true only when there is no mobility in the system. On the other hand, if a topology is random, then there is usually only a calculation cost since there is no point of saving something that changes over time.

These costs can be avoided if regulating a transmission power of each device. As mentioned earlier, Niu et al. [12] implemented an overlay network topology by regulating transmission powers. However, some devices cannot regulate their transmission powers, and even if they could, there is a question how to calculate how much to reduce it to still have a connected network. Namely, in a fully decentralized system, each device has to make this decision on its own, without a central entity telling it how much to reduce its power transmission. And it is very hard to make the local decision that affects the whole system using only local information.

2) Time cost of sending multicast messages: Once when a device knows who are its neighbors, the problem is how it can send its synchronization messages only to a group of devices from all of devices that it can communicate with (i.e., how to send only *multicast* messages). When overlay network topologies are not used, then it is easy to send synchronization messages using the sending option for *broadcast*. However, when devices have to use a *multicast* option for sending, then things get complicated. The main problem is the way *multicast* functions are realized. For example, when using XBee on Waspmote sensors [16], there is virtually no function for *multicast* and it can be thus only realized as sending many *unicast* messages. Of course, that is time consuming and has the negative effect on the synchronization precision.

As discussed earlier, in real-world environments real synchrony is virtually impossible to achieve. Even if messages were delivered without delays and devices were identical (both of which is impossible), there would still be a problem of *deafness*. Namely, at one moment, a device can either send or receive messages, i.e., it is impossible to send and receive messages simultaneously. It can be thus said that a device is *deaf* while sending its synchronization messages. This means that as longer time needed for sending synchronization messages is (e.g., because of sending many *unicast* messages), lower synchronization precision can be achieved. Although this problem is not present with every communication technology, it is very likely that it will appear in M2M systems since they are heterogeneous systems composed of different networks.

IV. The mechanism for selective coupling

In previous sections we showed that maintaining scalability in M2M systems is an important research challenge and that network traffic can be reduced when using overlay network topologies. However, we also gave several reasons why it is hard to implement them in real-world environments. We thus developed a *mechanism for selective coupling implemented on the sender side*. Different mechanisms for selective coupling have been already proposed in the literature, but they were mostly implemented on the receiver side with purposes different from a network traffic reduction. When a mechanism for selective coupling is implemented on the receiver side, then each device, once it receives a message, decides whether this message will affect its synchronization process or not.

For example, Niu et al. [12] proposed a *selective coupling synchronicity algorithm* in which every device reacted to the received message only if it led to a faster convergence of the synchronization precision. However, from a network traffic reduction perspective, there is no use to drop synchronization messages on the receiver side after they have already traveled through the network. A better solution would be to decide on the sender side whether a synchronization message is going to be sent or not, because this can reduce network traffic. Scholtes et al. [17] integrated a measure for a *coupling probability* into the pulse coupled oscillator model. They showed that halving the coupling probability (i.e., probability to send synchronization messages) meant doubling time to synchronization. Moreover, Degesys et al. [18] showed that a selective reduction of transmitted information could both save energy and improve the convergence rate of desired synchronization precision.

Algorithm 1 shows the pseudo code for the proposed mechanism. The main idea behind the mechanism for selective coupling implemented on the sender side is to set *thresholdSyn* parameter value that is then used as an indicator whether to send synchronization messages or not. Namely, at the end of each synchronization cycle, every device calculates the *ratio of the number of devices it is synchronized with to the number of devices it is connected with*. If this ratio is greater than *thresholdSyn* parameter value, then the device will send synchronization messages to its neighbors[6]. Additionally, since at the beginning of a synchronization process, devices are usually not synchronized enough to send their messages, we propose to include randomness in the decision process. Randomness is introduced with the *probability* variable, while the ratio of the number of devices each device is synchronized with (*synchronizedNeighborsNumber*) to the number of devices each device is connected with (*allNeighborsNumber*) is stored in *synRatio* variable. Messages are send only if *synRatio* value is greater than *thresholdSyn* parameter value or if *probability* value is less than *thresholdProbability* parameter value.

Algorithm 1 The proposed mechanism pseudo code

1. *probability* = *random.uniform(0,1)*
2. *synRatio* = $\frac{synchronizedNeighborsNumber}{allNeighborsNumber}$
3. **if** ((*synRatio* $>$ *thresholdSyn*)
   or (*probability* $<$ *thresholdProbability*))
4. sendMessages()
5. **end if**

Since it was shown earlier that a network topology has a great influence on the synchronization behavior [17], in this paper we study the emergence of synchronization, when using the proposed mechanism, for different static network topologies. In our analysis we are considering graphs generated by *Watts/Strogatz* model [19]. Watts and Strogatz proposed a model that can produce graphs with small-world characteristics for which it was shown that the number of required edges for synchronization to occur is especially low [20]. In future work we will also include graphs generated by *Barabási/Albert* model [21], which is based on a preferential attachment[7], because it has been shown that synchronization benefits from disassortative networks in which high degree devices were connected to low degree devices [22].

For Watts/Strogatz model, there are two parameters of the model WS(p, k): k denoting the number of nearest neighbors in initial ring lattice and p denoting reconnection probability. If parameter p is set to 0, then the resulting graph is a regular ring lattice, and when p = 1, then we have a completely random graph. In our simulations, which were conducted using MASON simulator [11], we used the model with the following parameters: p = 0.5 and k = 10. We tried different parameters of the model and the results showed that for the given parameters, the hugest rate of successful synchronization outcomes is achieved, as suggested by Scholtes et al. [17].

[6]Being synchronized with more devices, the synchronization message of such a device is likely to be more important for the whole synchronization process and it is thus more probably that it is going to be sent over the network.

[7]A preferential attachment denotes the probability of incrementally added nodes to establish links to existing nodes proportional to the target node degree.

Although our main goal is to reduce network traffic, we do not want that the proposed mechanism affects negatively the rate of successful synchronization outcomes. We thus conducted experiments in which we measured the rate of successful synchronization outcomes in dependence of different *thresholdSyn* and *thresholdProbability* values on different network topologies generated using Watts/Strogatz model with WS(0.5, 10). Both parameters (i.e., *thresholdSyn* and *thresholdProbability*) were chosen from the following set {0.1, 0.3, 0.5, 0.7, 0.9}. Each simulation was repeated for 500 times with 100, 1000, 2000, ..., 10000 devices and with a different uniform distribution of device initial state variable values.

The results showed that in almost all cases (i.e., for every combination of their values) the rate of successful synchronization outcomes was 100%. Only when *thresholdSyn* value was fairly large (i.e., larger or equal to 0.9) and *thresholdProbability* value was fairly small (i.e., around 0.1), the rate of successful synchronization outcomes dropped to 98.4%. The reasons behind this are different distributions of devices initial state variables values. Namely, if *thresholdProbability* value is fairly small, then depending on devices initial state variables values, some devices will not be able to send their messages during the entire synchronization process.

Figure 2 shows synchronization results when using our mechanism with respect to the number of exchanged synchronization messages during synchronization and time to synchronization. x-axis denotes the number of nodes used in Watts/Strogatz model, while y-axis denotes the number of messages or time. Since MASON is a discrete simulator, time is expressed in time steps. Due to lack of space, we show results only when *thresholdSyn* value is within {0.1, 0.5, 0.9} and *thresholdProbability* value within {0.1, 0.3, 0.5, 0.7, 0.9}.

The black dashed line in Figure 2, which denotes that both *thresholdSyn* and *thresholdProbability* values are equal to 0.0, presents the number of exchanged synchronization messages and time to synchronization without the proposed mechanism. Namely, when *thresholdSyn* value is 0.0, then all synchronization messages are always sent regardless of *thresholdProbability* value. All other lines present relative difference between our mechanism results and synchronization process without our mechanism. This means that lines that are "above" x-axis (i.e., black dashed line) present results that are worse than when not using our mechanism, and lines that are "below" show cases when our mechanism gives better results.

For example, when *thresholdSyn* value is equal to 0.1 (first column in Figure 2), then for any *thresholdProbability* value both lines denoting the number of exchanged messages and time to synchronization are "below" the black dashed line. This means that for those combinations of *thresholdSyn* and *thresholdProbability* values, when using our mechanism, both the number of exchanged messages and time to synchronization are reduced. On the other hand, when *thresholdSyn* value is equal to 0.9 (last column in Figure 2), for any *thresholdProbability* value time to synchronization is prolonged. Moreover, when *thresholdProbability* value is equal to 0.1, not only did the rate of successful synchronization outcomes drop, but also the number of exchanged messages, in cases when synchronization was achieved, was larger for up to 30 %.

Finally, we can conclude that the optimal combination of parameters is: 0.1 for *thresholdSyn* and 0.3 for *thresholdProbability*. For example, in M2M systems with 10000 devices, when using our mechanism with aforementioned parameter values, 2.5 million (i.e., 52 %) less messages are exchanged during the synchronization process. Moreover, around 1300 (i.e., 53 %) less steps are needed to achieve synchronization.

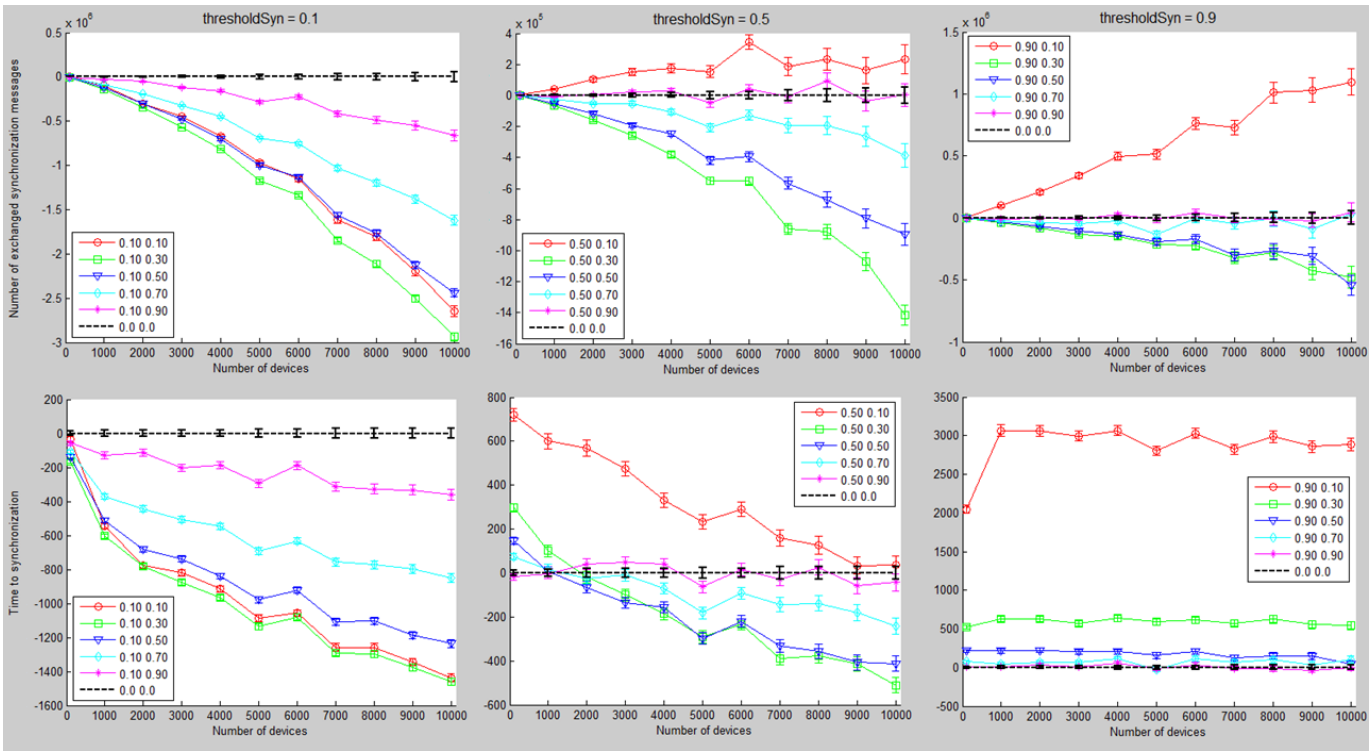

Fig. 2. Relative difference of exchanged synchronization messages and relative difference of time to synchronization

From the implementation perspective, the proposed mechanism can be easily implemented on devices in real-world environments because only one additional variable has to be used - the one in which the number of neighbors is stored (i.e., *allNeighborsNumber*). This number is calculated as the number of synchronization messages received during one synchronization cycle. Additionally, every device calculates how many messages it received exactly at the end of each cycle - this number denotes with how many neighbors this device is synchronized in this cycle (i.e., *synchronizedNeighborsNumber* variable). Compared thus to overlay network topologies, this mechanism has negligible both calculation and memory costs. Moreover, the problem of *deafness* does not affect negatively the proposed mechanism proper functioning since we calculate the *ratio* of the number of messages received exactly at the end of each synchronization cycle to the number of all messages received during the whole synchronization cycle. Therefore, if some device does not receive all synchronization messages from the neighbors it is synchronized with, because of the *deafness* problem, this will affect both *allNeighborsNumber* and *synchronizedNeighborsNumber* values. Consequently, it will not cause that the ratio value changes.

V. Conclusion

Although collective adaptive systems can be very different, they all have in common that they are very large in size. The important research challenge is thus how to develop algorithms for them that are scalable. From a firefly-based synchronization perspective, we want to find ways how to reduce network traffic without affecting the rate of successful synchronization outcomes. In order to do that, we proposed our mechanism for selective coupling implemented on the sender side for which we showed that for a certain combination of *thresholdSyn* and *thresholdProbability* values, both time to synchronization and the number of exchanged messages during synchronization can be reduced. Moreover, we showed that the rate of successful synchronization outcomes remained the same (i.e., 100%).

The main deficiency of the paper is the lack of the practical implementation in real-world environments. Unfortunately, we did not have enough devices to test scalability of our mechanism in real Machine-to-Machine (M2M) systems. Therefore, we used MASON simulator to simulate large systems. However, there are at least two problems with this approach: we used Watts/Strogatz model to generate networks representing communication channels in real-world environments, and moreover we did not simulate limitations of communication channels that are present in real-world environments. Regarding the formal, the question is how good Watts/Strogatz model represents heterogeneous M2M systems. In future work we will use models that generate multilayer networks to see if these models can more realistically represent M2M systems. Regarding the latter, in future work we will also include simulations of M2M systems under the effect of churn to access the resilience of our approach against this kind of dynamics. Another important aspects that have not yet been considered, and of which in future work we will give a more comprehensive analysis, are communication latency and different distributions of device frequencies.

Acknowledgment

Authors would like to acknowledge Machine-to-Machine Communication challenges project funded by Ericsson Nikola Tesla, Croatia.

References

[1] J. Gantz, "The Embedded Internet: Methodology and Findings," 2009.

[2] D. Evans, "The Internet of Things: How the Next Evolution of the Internet is Changing Everything," 2011.

[3] H. Vestberg, "More than 50 Billion Connected Devices," 2011.

[4] M. Hatton, "The Global M2M Market in 2013," 2013.

[5] R. E. Mirollo and S. H. Strogatz, "Synchronization of Pulse-Coupled Biological Oscillators," *SIAM Journal on Applied Mathematics*, vol. 50, no. 6, pp. 1645–1662, 1990.

[6] S. H. Strogatz, *Sync: The Emerging Science of Spontaneous Order*. Hyperion, 1990.

[7] A. S. Tanenbaum and M. van Steen, "Synchronization," in *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, 2001.

[8] D. Lucarell and I.-J. Wang, "Decentralized Synchronization Protocols with Nearest Neighbor Communication," in *Proceedings of the International Conference on Embedded Networked Sensor Systems*, 2004, pp. 62–68.

[9] I. Bojic, V. Podobnik, I. Ljubi, G. Jezic, and M. Kusek, "A Self-Optimizing Mobile Network: Auto-Tuning the Network with Firefly-Synchronized Agents," *Information Sciences*, vol. 182, no. 1, pp. 77–92, 2012.

[10] I. Bojic and M. Kusek, "Comparing Different Overlay Topologies and Metrics in Pulse-Coupled Multi-Agent Systems," in *Proceedings of the 6th KES International Conference on Agent and Multi-Agent Systems: Technologies and Applications*, 2012, pp. 464–473.

[11] "MASON web site, http://www.cs.gmu.edu/ eclab/projects/mason/," visited on 30th January 2014.

[12] Y. Niu, B. J. d'Auriol, X. Wu, J. Wang, J. Cho, and S. Lee, "Selective Pulse Coupling Synchronicity for Sensor Network," in *Proceedings of the 2nd International Conference on Sensor Technologies and Applications*, 2008, pp. 123–128.

[13] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, 2005, pp. 142–153.

[14] R. Leidenfrost and W. Elmenreich, "Firefly Clock Synchronization in an 802.15.4 Wireless Network," *EURASIP Journal Embedded Systems*, pp. 7:1–7:17, 2009.

[15] "Atmel demonstration kit, www.mscbp.hu/documents/ATAVRRZ200.pdf," visited on 30th January 2013.

[16] "Waspmote sensors, www.libelium.com/products/waspmote," visited on 30th January 2013.

[17] I. Scholtes, J. Botev, M. Esch, and P. Sturm, "Epidemic Self-Synchronization in Complex Networks of Kuramoto Oscillators," *Advances in Complex Systems*, vol. 13, no. 1, pp. 33–58, 2010.

[18] J. Degesys, P. Basu, and J. Redi, "Synchronization of Strongly Pulse-Coupled Oscillators with Refractory Periods and Random Medium Access," in *Proceedings of the ACM Symposium on Applied Computing*, 2008, pp. 1976–1980.

[19] D. J. Watts and S. H. Strogatz, "Collective Dynamics of "Small-World" Networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[20] M. Barahona and L. M. Pecora, "Synchronization in Small-World Systems," *Physical Review Letters*, vol. 89, no. 5, pp. 101–104, 2002.

[21] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[22] M. di Bernardo, F. Garofalo, and F. Sorrentino, "Effects of Degree Correlation on the Synchronization of Networks of Oscillators," *International Journal of Bifurcation and Chaos*, vol. 17, no. 10, pp. 3499–3506, 2007.