Department of Distributed and Dependable Systems
Technical Report no. D3S-TR-2014-03
April 2nd, 2014

Computational Model for Gossiping Components in Cyber-Physical Systems

Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, Frantisek Plasil

{bures, iliasg, hnetynka, keznikl, kit, plasil}@d3s.mff.cuni.cz

Abstract: Developing software for dynamic cyber-physical systems (CPS) is a complex task A particular challenge in this context is the robust distributed data dissemination in dynamic networks. Gossip-based communication stands as a promising solution to this challenge. We argue, that exploitation of application-specific information, software architecture in particular, has a large potential for improving the robustness and performance of gossip-based communication. This report presents a computational model that represents a synergy between high-level architectural models and low-level communication models to effectively enable application-specific gossiping in component-based systems.

Version: April 2nd, 2014

This work was partially supported by the EU project ASCENS 257414. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7-PEOPLE-2010-ITN) under grant agreement n°264840.

1 Introduction

Cyber-physical systems (CPS) are complex networked systems where the interplay of software control with the physical environment has a prominent role. Examples range from intelligent navigation systems (cars can communicate with each other and with street infrastructure units to minimize traffic congestion, fuel consumption, etc.) to emergency coordination systems and interactive distributed games. Modern CPS are inherently distributed and large-scale, and consist of both stationary and mobile devices. They are also increasingly depending on software which has actually become their most intricate and extensive constituent [3].

Whereas the challenges and opportunities of CPS cover a range of areas, in this report we focus on the communication requirements of CPS. The main observation is that communication in CPS has to be robust in spite of frequent connection faults, transient network failures, and inherently unreliable communication mediums (e.g., wireless). At the same time, communication primitives should reflect and ideally take advantage of the CPS specifics, such as physical mobility and partitioning.

Looking at the state-of-the-art in distributed systems communication, gossip or epidemic protocols provide an efficient way to address the aforementioned specifics. Gossip protocols cope with node and network failures, are scalable due to their symmetric nature, and can exploit the physical mobility of gossiping nodes [5]. Gossiping paradigm has already been applied with success in both Internet-based systems and wireless mobile ad-hoc networks (MANETs) [7]. The gossip protocols typically combine probabilistic forwarding with counter-based, distance-based, and location-based mechanisms. These mechanisms and configuration parameters are, however, only available at the lower level of the software stack, often transparent to the application/architecture layer, which is problematic when the spread of data depends on the architectural configuration in question.

In this report, we aim at bridging the abstraction gap between gossip protocols (concerned with message sending/receiving, packet-based communication, peer selection, etc.) and application-level programming using component models and architectures tailored to the specifics of CPS. Specifically, we do it by encompassing gossiping primitives into sound software engineering abstractions, which allow for (i) systematic engineering of CPS via gossiping components and (ii) application-specific, scalable, and efficient gossip-based communication. We do so in the context of DEECo [4] – a component model that specifically supports dynamic, ever-changing architectures and reflects the specifics of CPS within its abstractions by relying on the concepts of autonomous real-time components, and dynamic ad-hoc coordination groups (ensembles). For further information on DEECo, we refer an interested reader to [4, 6].

2 Gossiping in ensembles

To integrate the communication mechanisms of gossip-based protocols into DEECo, we have adopted a fully decentralized and robust approach relying on gossip for establishing ensembles and performing knowledge exchange. In principle, we replace the network communication layer of DEECo by gossip-based communication and extend the DEECo architectural model (the definition of ensembles in particular) by the concept of a *communication boundary* so as to allow efficient functioning of the underlying gossip. In this section, we describe our approach informally and formalize it in Section 3.

To connect components at the architectural level with their physical deployment, we define node as a hardware/software platform where a number of DEECo components are deployed. Each node contains an instance of Runtime, taking the role of component container and communication middleware. A node contains both the knowledge of hosted components and copies (replicas) of the knowledge of components hosted on other nodes. Nodes communicate with each other via their network interfaces depending on the available networking infrastructure. Thus, component communication is constrained by the available networking infrastructure between the nodes the components are deployed on. Inspired by the motivating scenario, we focus on combinations of IP-based networks (wireless and wired) and MANET networks (which allow only for short range direct communication).

The main principles of our approach can be characterized by the following rules pertaining to every node:

(1) *A node has its own awareness of ensemble instances existing in the system, specifically of those that contain components deployed on the node. This is based on evaluating the membership predicates with respect to the knowledge of (a) local components and (b) replicas.*
(2) *A node performs knowledge exchange independently, based on the knowledge of both local components and replicas.*
(3) *A node proactively disseminates component knowledge, so that every other node has the knowledge relevant for realization of (1) and (2).*

In the following, the individual elements of our approach are described in more detail – points 1 and 2 are explained in Section 2.1and point 3 is elaborated in Sections 2.2 and 2.3.

2.1 Decentralized evaluation of ensemble membership and knowledge exchange

Instead of forming ensembles by looking at a snapshot of the whole system (which would imply that a global view on the system has to be available), we take a node-centric approach. Every node periodically iterates over all known ensemble definitions and checks whether a local component can act as a member or coordinator in an instance of the ensemble definition, given its replicas. For each such ensemble instance, it performs the corresponding knowledge exchange, which results in updating the local components' knowledge (but not the replicas).

As an example, consider an instance of the `TemperatureUpdate` ensemble definition show in Fig. 1, evaluated on the site of the coordinator. In this case, the knowledge exchange results into updating the coordinator's field temperatures.

Note that a consequence of this technique is that degradation of system performance when no connectivity is available is gradual: each Runtime effectively operates on the locally available replicas until they become too outdated to rely upon. Here, we count on one of the specifics of CPS, namely that values of most magnitudes (e.g., temperature in Fig. 1) evolve gradually according to physical laws [1]. Practically this means that a belief which is not too old may still be at least partly relevant.

Another consequence is that, due to belief outdatedness causing belief inaccuracy, it is possible for a component to behave as if it were in ensemble with a coordinator, which is not aware of it (and vice-versa).

2.2 Asynchronous knowledge dissemination via gossip

The decentralized solution presented in Section 2.1, requires that each node possesses all the necessary replicas from the components that can potentially participate in ensembles with its local components. We enable this by asynchronous gossip-based knowledge dissemination between all the components of a DEECo application.

The main idea is that every node periodically publishes the knowledge of its local components on the network. For MANETs, this translates to periodic broadcast within the wireless range of the node. For IP networks, it translates to periodic sending to randomly selected nodes. Upon reception of a component's knowledge, every node probabilistically decides whether to retransmit the received knowledge. The nodes that perform such re-transmission are then acting

```
role TemperatureRelay:
  position

role TemperatureSensor:
  missionID, missionAreas, temperature

ensemble TemperatureUpdate:
  coordinator: TemperatureAggregator
  member: TemperatureSensor
  membership:
    member.missionID == coordinator.missionID
  knowledge exchange:
    coordinator.temperatures ← { (m.ID, m.temperature) | m ∈ members }
  boundary:
    case relay: TemperatureRelay, replica: roleOf(member):
        ∃area ∈ replica.missionAreas: isInArea(relay.position, area)
    case relay: any, replica: roleOf(coordinator):
        false
    ip-registry: 10.10.16.35, 10.10.16.112
  …
```

Fig. 1. Example of an ensemble definition in DEECo, extended with a communication boundary.

as relays. Here, we rely on the probabilistic convergence of gossip protocols which ensures that every node will eventually receive the knowledge of every component in a bounded number of steps.

Note that this dissemination scheme dictates that retransmission is potentially done by all nodes, not only from the ones that are interested in the disseminated knowledge (i.e., nodes hosting components that could be members of the ensemble which the disseminated knowledge relates to).

2.3 Bounding the gossip

Although the aforementioned gossip-based knowledge dissemination successfully conveys the knowledge of all nodes, it raises performance issues. Specifically, a DEECo application is considered as a ubiquitous ecosystem in a real environment, the application is potentially boundless w.r.t. network reachability. In such a system, unlimited gossiping is not a viable option. Advantageously, in contrary to the assumption of traditional gossip protocols discussed above, not every node is interested in all the data being disseminated by all the components. Thus, certain application-specific bounds should be established for knowledge dissemination.

For this purpose, we define for each ensemble its communication group as the set of nodes to which the ensemble's knowledge dissemination is limited. This set consists of all the nodes where components forming the ensemble are hosted and all the relays necessary for knowledge propagation. Relying on the fact that data is disseminated via gradual flooding, we define a communication boundary as the predicate determining the limits of a particular communication group w.r.t. network topology. The relays not satisfying the communication boundary will not participate in the dissemination. In a way, a communication group forms a dynamic, architecture-specific network overlay for knowledge dissemination.

Naturally, a communication boundary includes all the nodes "potentially interested" in the disseminated replicas, while excluding as many of the other replicas as possible. Thus, a communication boundary forms a conservative approximation of the potential ensemble membership. For example, the communication boundary for the ensemble definition in Fig. 1 can be formulated as follows:

"For every mission, include all components within all the areas
in which the participants of the mission operate."

In this example, the communication boundary reflects the fact that all components satisfying the membership condition of the ensemble, i.e., those participating on the same mission, operate in one of the predefined areas. Note however, that the communication boundary predicate is generic w.r.t. a particular mission – it determines a number of different communication groups (thus approximating a number of different ensemble instances), namely a distinct group per distinct mission.

To achieve its desired functionality, a relay has to evaluate a communication boundary much more efficiently than membership condition, preferably using only locally-available information. Thus, we specify communication boundary as a predicate over the local knowledge of the relay and the particular knowledge being disseminated.

Since "communication group" is an application-specific concept relating to application architecture (namely to knowledge of a component), we extend the ensemble definition to capture communication group by definition of communication boundary. In addition, we extend the existing concept of role to apply at the level of nodes so that a node supports a role if one of the components (representative) deployed on the node has structurally-matching knowledge. The structural matching enables designing open-ended architectures.

Technically, a communication boundary is defined by a set of predicates (lines 14-17 in Fig. 1). Each of these predicates, given a relay role and a replica role, determines whether a node that has a representative matching the relay role meets the communication boundary for a replica that matches the replica role. Formally, the communication boundary is a conjunction of these predicates (having the form of implications). A relay role has to be either the coordinator or member role.

For example in Fig. 1, given a replica corresponding to the member role (TemperatureSensor), the communication boundary includes all relay nodes featuring the TemperatureRelay role, which are in one of the mission areas specified by the replica. This is captured on lines 15-16, which semantically form an implication: the line 15 forms the antecedent (i.e., "if the relay has the role TemperatureRelay and the replica corresponds to the member's role"), while line 16 forms the conclusion. Note, that we have extended the TemperatureSensor role and the knowledge of all the related components to provide the information about mission areas. Similarly, on lines 17-18 the predicate prevents any relaying of replicas matching the coordinator role (as there is no knowledge exchange towards the member).

This part of specification of communication boundary aligns well with the knowledge dissemination in MANETs, where the set of potential recipients is limited by their geographical locality. On the other hand, in large networks that enable routing based on global addressing, such as IP networks, a necessary performance optimization is to disseminate replicas only to recipients which themselves meet the communication boundary (rather than blindly pollute the entire IP network). To do this, given a replica, a sender has to be able to (at least partially) assess the validity of the communication boundary with respect to the recipient.

To address this issue, we assume that well-known registries exist providing a relay node the information which other IP-based nodes are part of a communication group (given a particular replica). To avoid unnecessary centralization, such a registry is ensemble specific. The registry either provides statically-defined recipients (well-known relay nodes) or evaluates the communication boundary with respect to a recipient. In the latter case, the potential recipient relay nodes provide the registry with the required relay knowledge. Syntactically, the communication boundary definition contains a set of IP addresses identifying the registries that are specific to the corresponding ensemble (line 19 in Fig. 1). Note that due to the nature of gossip, we do not require all the registries in a given ensemble specification to contain the same information.

3 Formalized Semantics

Having informally outlined our approach in the previous section, we provide now a precise formulation of our solution. We resort to formalization using operational semantics, which allows us to do analysis of functional and timing properties and precise simulations.

Technically, we represent the semantics via a state transition system, where the transitions are generated by the inference rules presented in the rest of this section. Specifically, each rule defines the required state in which the transition is allowed and the state change entailed by the transition.

3.1 Initial definitions

Knowledge is a partial function $knw: F \nrightarrow D$, mapping the domain of knowledge fields F (typically seen as identifiers) to the set of knowledge field values D. We denote K as the set of all possible knowledge valuations.

Knowledge update is a partial function $u: F \nrightarrow (D \cup \{undef\})$, where undef represents a special value which signifies that a knowledge field should be removed from a knowledge. We denote U as the set of all knowledge updates. We further define the knowledge update operator $\oplus$ with the following semantics. Let $knw \in K$, $u \in U$, and $knw' = knw \oplus u$, then value of $knw'(f)$ (for a knowledge field f) is:

- $knw(f)$ if $u(f)$ is not defined; or
- $u(f)$ if $u(f) \neq$ undef ; or
- $\perp$ (not defined) if $u(f) = undef$.

3.2 Internal component behavior

Component is a tuple $c = (knw, proc, rproc)$, where knw is the knowledge of the component, $proc$ is the set of processes of the component, and $rproc$ is the state of running processes (definitions follow below). Note, that a component is understood as a singleton, i.e., a component instance. We denote the set of all components as C.

- **Process** of a component c is a function $p: K \times R \rightarrow U$, where R denotes the domain of internal events which are not explicitly modeled on the level of the component model and thus from the perspective of the semantics remain non deterministic (e.g., responses from the operating system, readings from sensors, etc.).
- The **state of running processes** is a partial function $rproc: proc \nrightarrow K$, which maps each currently running process to its inputs (i.e., valuation of the knowledge at the time of the process start). If a process is not running, the value of $rproc$ is undefined.

The inference rules below describe the semantics of the concurrent, asynchronous component process execution:

$$\frac{c \in C,\ p \in c.proc,\ c.rproc(p) = \bot}{c.rproc(p) \leftarrow c.knw} \qquad \text{(p-start)}$$

$$\frac{\begin{array}{c} c \in C,\ p \in c.proc \\ c.rproc(p) \neq \bot,\ k = c.rproc(p),\ r \in R \end{array}}{c.rproc(p) \leftarrow \bot,\ c.knw \leftarrow c.knw \oplus p(k, r)} \qquad \text{(p-end)}$$

Each process executes in a cyclic manner. This is modeled in two steps: (1) inputs of the process are atomically retrieved from component's knowledge (rule p-start), and (2) outputs of the process computation are atomically written to component's knowledge (rule p-end). Essentially, this semantics of process execution is similar to our previous work [2].

Note that we use a shorthand form of the inference rules conclusions. We always assume the conclusion to be in form a state transition $S \longrightarrow S'$, where S' is the same state as S, except for assignment explicitly stated in the rule. For instance $c.knw \leftarrow c.knw \oplus u$ means S' is the same as S, except for the knowledge of component c, whose new valuation is $c.knw \oplus u$. We use the assignment operator $\leftarrow$ also to remove an assignment from a partial function – i.e., $f(x) \leftarrow \bot$ means $f \setminus \bigcup_{\forall y}(x, y)$.

3.3 Decentralized execution of ensemble membership and knowledge exchange

Ensemble definition is a tuple $e = (mem, kex, cb)$, where mem is the membership predicate, kex is the knowledge exchange function, and cb is the communication boundary predicate (definitions of membership and knowledge exchange follow below, communication boundary is elaborated in Section 3.4). We denote the set of all ensemble definitions as E.

- **Membership** is a predicate $mem \subseteq K \times K$ over knowledge of an ordered pair of components. The predicate determines whether the two components form the coordinator-member pair of an ensemble instantiated from e.
- **Knowledge exchange** is a function $kex: K \times K \to U \times U$, which for knowledge of the coordinator and a member (in this order) gives the knowledge updates corresponding to the effect of the knowledge exchange in ensemble definition.

Computational node is a tuple $n = (comp, cknw, rkex)$, where $comp \subseteq C$ denotes the set of components deployed on the node, $cknw$ the knowledge replicas available to the node, and $rkex$ is the state of running ensemble knowledge exchange (definitions follow below). We denote the set of all nodes as N.

- The **knowledge replicas of the node** n is a partial function $cknw: C \nrightarrow K$, which for a component returns the replica of the component's knowledge available to the node n. In case the component c is deployed on the node (i.e, $c \in n.comp$), then $cknw(c) = c.knw$ (i.e., the replica is equal to the actual knowledge of the component). If the node has no available replica for a component, the value is undefined.

- The **state of running ensemble knowledge exchange** is a partial function $rkex : E \times C \times C \nrightarrow K \times K$, which maps each currently running knowledge exchange to its inputs – knowledge of the coordinator and a member at the time of the knowledge exchange start. If knowledge exchange is not running, the value of $rkex$ is undefined.

The inference rules below describe the semantics of the concurrent, asynchronous, and decentralized execution of knowledge exchange in scope of a single node, as described in Section **Error! Reference source not found.**:

$$\frac{\begin{gathered} n \in N,\ c_l \in n.comp,\ e \in E, \\ c_p \in C: n.cknw(c_p) \neq \bot,\ n.rkex(e, c_l, c_p) = \bot \end{gathered}}{c_l.rkex(e, c_l, c_p) \leftarrow \big(n.cknw(c_l), n.cknw(c_p)\big)} \qquad \text{(e-start)}$$

$$\frac{\begin{gathered} n \in N,\ c_l \in n.comp,\ e \in E, \\ c_p \in C: n.rkex(e, c_l, c_p) \neq \bot = (k_l, k_p) \end{gathered}}{\begin{gathered} c_l.knw \leftarrow c_l.knw \oplus \begin{cases} [e.kex(k_l, k_p)]_1, & e.mem(k_l, k_p) \\ [e.kex(k_p, k_l)]_2, & e.mem(k_p, k_l) \\ \emptyset & , \text{otherwise} \end{cases} \\ n.rkex(e, c_l, c_p) \leftarrow \bot \end{gathered}} \qquad \text{(e-end)}$$

Specifically, the rule (e-start) describes the atomic read of knowledge exchange inputs at the start of its execution. Recall, that knowledge exchange is executed for all pairs of components in which one component (the local component, c_l) is deployed at the node itself while there is an available replica for the other (the peer component, c_p). Note that the peer component can be deployed on the node as well. The rule (e-end) describes the atomic update of the local component's knowledge with the outcome of the process w.r.t. to the stored inputs, performed at the end. Note that the local component's knowledge is actually updated only if the local and peer components meet the membership condition of the corresponding ensemble. If the local component acts in the knowledge exchange as the coordinator, the first knowledge update from the tuple returned by kex is used; if it acts as a member, the second is used.

3.4 Asynchronous knowledge dissemination

We abstract the network as a complete directed graph of nodes, where each edge corresponds to a unidirectional **communication channel** between two nodes. The communication channel is modeled as an unbounded, lossless queue. We denote the set of all channels as Q. A channel may either represent a MANET communication link or an IP communication link. For convenience, we use the function $chan: \{manet, ip\} \times N \times N \to Q$ which for each channel type and each pair of nodes returns a distinct channel connecting the first node with the second. The communication over the channels is regulated by two conditions described below.

The **gossiping condition** is a predicate $gossip \subseteq \{manet, ip\} \times N \times N \times R$, which for a channel type, a sender node, and a recipient node establishes whether the sender should disseminate replicas to the recipient via a channel of that type, based on the employed gossiping scheme and network topology. For example, in case of a MANET channel, this predicate reflects whether the recipient is within the receiving range of the sender, etc. R denotes the domain of internal events which are not explicitly modeled on the level of the component model and thus from the perspective of the semantics remain non deterministic (e.g., random algorithmic decisions, lost packets due to limited range and conflicts in wireless communication). Note, that the gossiping condition is system-specific (depending of the system's network topology etc.).

Ensemble **communication boundary** is an ensemble-specific predicate $cb \subseteq N \times K$, which for a node and a component's knowledge replica establishes whether the node meets the communication boundary w.r.t. the replica; i.e., whether the replica should be disseminated by the node for the purpose of ensemble membership and knowledge exchange evaluation. Recall, that for every ensemble e, $e.mem \subseteq e.cb$. The inference rules below describe asynchronous, gossip-based dissemination of knowledge as described in Section **Error! Reference source not found.** and Section **Error! Reference source not found.**:

$$\frac{\begin{gathered} r \in R, n_l, n_r \in N\colon\ n_l \neq n_r, c \in C,\ k_C = n_l.cknw(c) \\ (k_1, \dots, k_n) = chan(manet, n_l, n_r), \\ gossip(manet, n_l, n_r, r), \exists e \in E\colon\ e.cb(n_l, k_c) \end{gathered}}{chan(manet, n_l, n_r) \leftarrow (k_1, \dots, k_n, k_c)} \qquad \text{(manet-send)}$$

$$\frac{\begin{gathered} r \in R, n_l, n_r \in N\colon\ n_l \neq n_r, c \in C,\ k_C = n_l.cknw(c) \\ (k_1, \dots, k_n) = chan(ip, n_l, n_r),\ gossip(ip, n_l, n_r, r) \\ \exists e \in E\colon\ e.cb(n_l, k_c) \wedge e.cb(n_r, k_c) \end{gathered}}{chan(ip, n_l, n_r) \leftarrow (k_1, \dots, k_n, k_c)} \qquad \text{(ip-send)}$$

$$\frac{\begin{gathered} n_l, n_r \in N,\ n_l \neq n_r, c \in C,\ k_c = n_l.cknw(c) \\ t \in \{manet, ip\}, chan(t, n_r, n_l) = (k'_c, k_1, \dots, k_n) \neq \perp \end{gathered}}{\begin{gathered} n_l.cknw(c) \leftarrow \begin{cases} k'_c, & k'_c \text{ is newer than } k_c \\ k_c, & \text{otherwise} \end{cases} \\ chan(t, n_r, n_l) \leftarrow (k_1, \dots, k_n) \end{gathered}} \qquad \text{(receive)}$$

The rules (send) and (ip-send) describe sending a replica via a channel whose endpoints meet the gossip condition (i.e., they are within wireless communication range, etc.). Here, sending is represented by adding the replica at the end of the channel's queue. Specifically, according to (manet-send), the replica is sent via a MANET communication link only if the pair sender node-replica meets the communication boundary of an ensemble. In case of an IP communication link, as described by (ip-send), both the pair sender-replica and receiver-replica need to meet the communication boundary for the replica to be sent (as explained in Section **Error! Reference source not found.**, technically the receiver-replica pair is to be evaluated by querying a registry). The rule (receive) describes asynchronous reception of a replica from a non-empty channel, represented as removing the first element from the channel's queue. Note that the node's replica is only updated if the received knowledge is newer than the one stored in the replica.

3.5 Real-time aspects

Being completely non-deterministic, the transition system generated by the presented rules can also capture behaviors that are not realistic w.r.t. real execution. In particular, as DEECo targets in general real-time CPS we focus on real-time properties of the transition traces. In principle, we allow only those transition traces that are possible with respect to real-time periodic scheduling of the system processes, ensemble knowledge exchange, and knowledge dissemination. In a way, these restrictions impose a fairness constraint on the transition traces. The technical details on connecting the semantics with time can be found in [2].

4 Conclusions

In this report, we presented a computational model representing a synergy of software component model abstractions and gossip-based communication primitives as a promising solution for engineering scalable dynamic decentralized cyber-physical systems. Our approach relies on providing architecture-level descriptions that feature communication groups (captured by communication boundaries) and allow us "driving" the gossip efficiently. Our current and future work involves experimentation with different gossip algorithms (e.g., a promising direction is to employ location-based algorithms where GPS devices are required).

References

[1] Ali, R. Al, Bures, T., Gerostathopoulos, I., Keznikl, J. and Plasil, F. 2014. Architecture Adaptation Based on Belief Inaccuracy Estimation. *To appear in Proc. of WICSA'14* (Apr. 2014).

[2] Barnat, J., Benes, N., Bures, T., Cerna, I., Keznikl, J. and Plasil, F. 2013. Towards Verification of Ensemble-Based Component Systems. *To appear in Proc. of FACS'13* (Nanchang, China, Oct. 2013).

[3] Beetz, K. and Böhm, W. 2012. Challenges in Engineering for Software-Intensive Embedded Systems. *Model-Based Engineering of Embedded Systems*. Springer. 3–14.

[4] Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M. and Plasil, F. 2013. DEECo – an Ensemble-Based Component System. *Proc. of CBSE'13* (Vancouver, Canada, Jun. 2013), 81–90.

[5] Friedman, R. and Gavidia, D. 2007. Gossiping on MANETs: the Beauty and the Beast. *ACM SIGOPS Operating Systems Review*. 41, 5 (Oct. 2007), 67–74.

[6] Keznikl, J., Bures, T., Plasil, F. and Kit, M. 2012. Towards Dependable Emergent Ensembles of Components: The DEECo Component Model. *Proc. of WICSA'12* (Aug. 2012), 249–252.

[7] Williams, B. and Camp, T. 2002. Comparison of broadcasting techniques for mobile ad hoc networks. *Proceedings of MobiHoc'02*. (2002), 194.