社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－DPS－125（1）
2005－ EIP － 29（1）
2005／11／14

Fault-Tolerant Communication Protocol in Wireless Sensor-Actor Networks

Keiji Ozaki†, Satoshi Itaya†, Naohiro Hayashibara†, Tomoya Enokido††, Ashraf Uddin Ahmed†, and Makoto Takizawa†
Tokyo Denki University†, Japan Risho University††, Japan
E-mail {kei, Itaya, haya, ashraf, taki}@takilab.k.dendai.ac.jp, eno@ris.ac.jp

Abstract

In a wireless sensor and actor network (WSAN), a group of sensors and actors are geographically distributed and linked by wireless networks. Sensors gather information sensed for an event occurring in the physical world and send them to actors. Actors can perform appropriate actions by making a decision on receipt of sensed information from sensors. Sensors are low cost, low powered devices with limited energy, computation, and wireless communication capabilities. Sensors may not only stop by fault but also suffer from arbitrary faults. Furthermore, wireless communication is less reliable due to noise and shortage of power of sensors. Reliable real time communication among sensors and actors is in nature required in WSAN applications. In order to realize the reliability and realtimeness, we newly propose a multi-actor/multi-sensor (MAS) model where each sensor in an event area sends sensed information to multiple actors and each actor receives sensed information from multiple sensors. Actors are required to causally and totally order sensed information and actions. In this paper, we discuss how to realize reliable, causally/totally ordered delivery of sensed information and actions with realtime constraints.

ワイヤレスセンサ-アクタネットワークにおける耐障害通信プロトコル

尾崎 敬史† 板谷 智史† 林原 尚浩† 榎戸 智也†† Ashraf Uddin Ahmed† 滝沢 誠†
東京電機大学 理工学部 情報システム工学科† 立正大学 経営学部††

ワイヤレスセンサアクタネットワーク (WSAN) **は、無線媒体で接続されたセンサとアクタから構成される。センサはイベント情報を収集し、アクタに送信する。アクタはセンサからの情報を基に必要な動作を実世界に対して行う。センサは、低電力の安価な機器である。このため、センサは停止障害だけでなく、任意の障害を起こす。信頼性とリアルタイム性を向上させるために、多アクタ/多センサ** (MAS) **モデルを提案する。アクタにはセンサの観測情報と動作の因果順序、全順序配送が求められる。本論文では、どのように信頼性を実現するか、また、どのようにセンサが観測した情報と動作の順序づけ配送を行うかを議論する。**

1. Introduction

A wireless sensor and actor network (WSAN) is a group of sensors and actors linked by wireless medium to perform distributed sensoring and acting tasks [1, 2, 13]. Sensors gather information about physical world. Actors are capable of making a decision on actions and perform appropriate actions for information gathered by sensors. WSAN is used in microclimate control, home automation, environmental monitoring, target tracking [1, 2]. There are many discussions on how to reliably and efficiently broadcast messages among sensors and actors [5, 13]. WSAN is one of the most significant technologies to realize ubiquitous societies [18].

Sensors are low-cost, low-power devices which are equipped with limited energy, computation, and wireless communication capabilities. Sensors may stop, even malfunction [11] due to the out-of-charge and fluctuation of observed phenomena in the physical world. In addition, the wireless communication between sensors and actors is less reliable, i.e. messages may be lost due to noise. We discuss how to make WSAN tolerant of faults of sensors and wireless communication links. If some event occurs in the physical world, sensors gather physical phenomena of the event occurring in an event area and send sensed information to actors. In our approach, each sensor sends sensed information to multiple actors and an actor receives sensed informations on a same event from multiple sensors in order to be tolerant of faults of sensors and wireless networks. Even if some sensors are faulty and messages are lost in the wireless link, each actor can receive proper sensed information from the other proper sensors. If sensors are arbitrarily faulty [11], an actor takes the majority-based decision on sensed information from multiple sensors. If multiple actors receive the same sensed information, the same action may be performed multiple times by the actors even if the action should be performed only once for the sensed information. On receipt of sensed information from sensors, an actor makes a decision on what action to be performed. Here, actors cooperate with each other to make a

consensus on which actor performs what actions in what order based on not only the sensed information but also a history of events, i.e. actions and sensed information. In distributed systems where multiple peer processes are cooperating, the processes are required to be causally delivered to the processes [17, 4, 12, 10]. We discuss how to order sensed information and actions in each actor. In addition, each event in the physical world is characterized by properties like temperature and location. Different sensors may collect properties of an event different from each other. Here, we define *quality of event* (QoE) to be properties of event. It depends on observed QoE of events how to order events. We discuss how to order events and actions by considering QoE of events.

In section 2, we present a system model of WSAN. In section 3, we discuss how to make sensors and sensor-actor communicates fault-tolerant. In section 4, we discuss how actors perform actions.

2. System Model

2.1. Sensors and actors

A wireless sensor and actor network (WSAN) is composed of sensors, actors and actuation device objects interconnected with wireless medium [1, 2, 13]. A sensor gathers information about the physical world and sends it to actors. An actor makes a decision on what actions to do and then performs the actions on devices. Let ***S*** be a set of sensors, ***A*** be a set of actors and **O** be a set of actuation device objects in WSAN.

Phenomena in the physical world is changed at some event. Each event is characterized in attributes like temperature and location. Let $\Omega(e)$ show a set of attributes of an event e. $\Omega(e)$ is named a *scheme* of an event e. An event e is represented as a collection of values of attributes. For example, $\langle \cdots, 15[^\circ C], N35^\circ 69'11.74" E139^\circ 22'19.33", \cdots \rangle$ is an event of a schema $\langle \cdots, temperature, location, \cdots \rangle$ where an event occurs in Hatoyama, Japan. Here, $e.a$ shows a value of an attribute a of an event e. If an event occurs in some location of the physical world, sensors in some distance from the location gather values of the event. An event area is a geographical unit of WSAN. Sensors gather physical phenomena of events occurring in an event area. Actuation devices which act to the physical world like air conditioners are modeled to be objects. Actuation of a device is modeled to be execution of methods on the device object. On receipt of a method, the method is performed on the object. For example, cooler air is ventilated by the *air conditioner* object *ac* on receipt of a method (*turn*) *down*. An *event area* is a collection of device objects, sensors which sense an event e, and actors which perform actions on objects.

A sensor s gathers information on phenomena, i.e. properties of an event *e* occurring in an event area. A sensor s gathers information on an event e occurring in an event area. A *type* of a sensor is a subset of the attributes, i.e. information on which the sensor can gather. For example, a voice sensor can gather voice information. Let $\Omega(s)$ be a type of a sensor s, i.e. a collection of attributes. For example, a temperature sensor t can gather information on temperature, i.e. $\Omega(v) = \{temperature\}$. $e[s]$ shows information on an event e which a sensor s gathers, i.e. values of attributes of the event e. Multiple sensors sense a same event e in an event area. For a pair of sensors s_1 and s_2 in an event area, $e[s_1] = e[s_2]$ may not hold due to sensor error, i.e. a sensor s_1 gathers information on an event e different from another sensor s_2 even if $\Omega(s_1) = \Omega(s_2)$. Then, the sensor s sends the sensed information $e[s]$ of the event e to actors in the event area. Figure 1 shows the relations of sensors, actors, and objects.

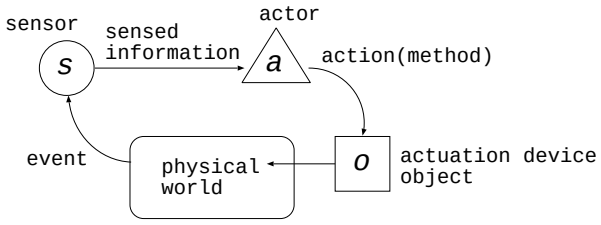

Figure 1. Sensor and actor.

2.2. Multi-actor/multi-sensor (MAS) model

In the single-actor (SA) model [1], all the sensors in an event area send sensed information to one actor [Figure 2]. Here, the actor can be a single point of failure. On the other hand, in the multi-actor (MA) model [1], each sensor sends sensed information to one actor but some pair of sensors in an event area may send sensed information e to different actors. In the SA and MA models, each sensor sends sensed information to one actor. In our *multi-actor/multi-sensor* (MAS) model, each sensor s sends sensed information to multiple actors and each actor receives from multiple sensors. Let *Actor*(s) be a set of actors to which a sensor s sends sensed information. Let *Sensor*(a) show a set of sensors which send sensed information to an actor a. For a pair of sensors s_1 and s_2, every actor a in *Actor*(s_1) $\cap$ *Actor*(s_2) receives sensed information from both the sensors s_1 and s_2. Every sensor s in *Actor*(a_1) $\cap$ *Actor*(a_2) sends sensed information to a pair of actors a_1 and a_2.

Each sensor broadcasts a message of sensed information. An area where each sensor can deliver a message with wireless medium is referred to as *broadcast cell*. Durresi *et al.* [5] discuss how to distribute sensors in a sensor/actor area so that every event to occur can be sensed by some number of sensors.

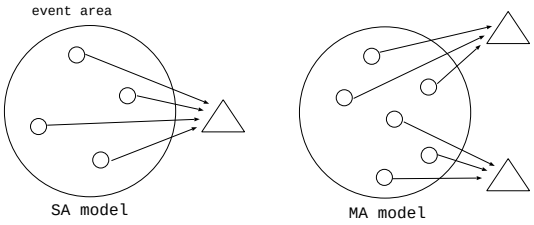

Figure 2. SA and MA models.

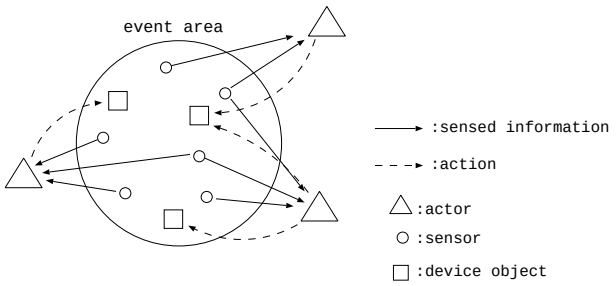

Figure 3. Multi-actor/multi-sensor (MAS) model

2.3. Object-based actions

An actor performs appropriate actions through some actuation devices in an event area on receipt of sensed information from sensors. An actuation device is modeled to be an object in this paper. An action is modeled to be the execution of a method on an object. On receipt of a method issued by an actor, the method is performed on an actuation device object. An object is composed of subobjects. A method issued to an object invokes methods on the subojects. Thus, methods are invoked in a nested manner.

Let $op(s)$ show result obtained by performing a method op on a state s of the world. Here, let op_1 and op_2 be a pair of methods of an object o. Let $op_1 \circ op_2$ show a serial execution of methods op_1 and op_2 on an object o. Let $op_1 \parallel op_2$ denote a parallel execution of methods op_1 and op_2 on an object o. A method op_1 is *equivalent* with another method op_2 ($op_1 \equiv op_2$) if the result obtained by performing the method op_1 is the same as op_2. ϕ shows a null method which does nothing on the object o. There are following relations among a pair of methods op_1 and op_2 of an object o:

1. op_1 and op_2 *conflict* with one another iff $op_1 \circ op_2(s) \neq op_2 \circ op_1(s)$ for some state s, i.e. the result obtained by performing the methods op_1 and op_2 depends on the computation order. $op_1 \circ op_2 \not\equiv op_2 \circ op_1$.
2. op_1 and op_2 are *compatible* iff $op_1 \circ op_2(s) = op_2 \circ op_1(s)$ for every state s, i.e. $op_1 \circ op_2 \equiv op_2 \circ op_1$.
3. op_2 *absorbs* op_1 iff $op_1 \circ op_2(s) = op_2(s)$ for every state s, i.e. $op_1 \circ op_2 \equiv op_2$.
4. op_1 is *compensated* by op_2 iff $op_1 \circ op_2(s) = s$ for every state s, i.e. $op_1 \circ op_2 \equiv \phi$. Here, op_2 is referred to a compensating method of op_1, denoted by $\widetilde{op_1}$.
5. op_1 is *idempotent* if $op_1(s) = op_1 \circ op_1(s)$ for every state s, i.e. $op_1 \equiv op_1 \circ op_1 \circ \cdots \circ op_1$.

Suppose an *air-conditioner* object ac supports methods, *on*, *off*, *up*, *down*, and *temp*. The air-conditioner object ac is turned on, off, up, and down by the methods, *on*, *off*, *up*, and *down*, respectively. In addition, the current temperature is obtained by a *temp* method. The method *temp* conflicts with all the other methods. A pair of the methods *on* and *off* conflict with one another while the methods *up* and *down* are compatible. The method *off* absorbs the other methods *temp*, *on*, *up*, and *down*. The methods *on* and *off* are idempotent. The method *up* can be compensated by *down* and vice versa. The effects done by methods performed are removed by compensating methods of the method. In a printer object *prt*, a method *print-out* is not compensatable. If a missile is launched, it cannot be compensated. Yasuzawa *et al.* [19] discuss compensating methods in object-based systems where objects are hierarchically structured.

An actor may issue a sequence of methods to objects. For example, a *door* object is required to be *locked* after *closed*. Thus, some multiple methods are required to be sequentially performed in a sequence specified. Suppose a *lock* method is issued before a *door* is closed. Here, suppose the *door* cannot be locked due to some trouble. In one case, it is all right because the door is closed. In another one, all the actions performed have to be undone. That is, the *door* object is opened again. In the later case, a subsequence of *close* and *lock* methods should be *atomically* performed. A sequence of methods to be atomically performed is referred to as *transaction* [7]. A transaction is required to be undone if some method in the transaction cannot be successfully performed. Suppose a transaction issues a method *up* to the *air-conditioner* object ac and is performed. Here, the transaction is required to be aborted. In order to abort the transaction, a compensating method *down* of *up* is issued to the air-conditioner object ac. A method like *print-out* cannot be compensated. Hence, transactions cannot issue uncompenasatable methods.

3. Fault-Tolerant Sensors

3.1. Types of faults

Sensors are low-cost, low-energy devices with limited energy, computation, and wireless communication capabilities. A sensor may stop by the shortage of the power supply, e.g. out of charge of the battery. A pair of sensors may collect different information for the same event due to sensor errors. That is, $e[s_i] \neq e[s_j]$ for some pair of sensors s_i and s_j in an event area. Even if $e[s_i] \neq e[s_j]$, $e[s_i]$ and

$e[s_j]$ are *equivalent* with respect to the quality of event QoE ($e[s_i] \equiv e[s_j]$) if QoE of sensed information $e[s_i]$ and $e[s_j]$ are considered to be the same from the application point of view. For example, some sensor s_1 shows $15°C$ ($e[s_1]$ = $\langle 15 \rangle$) and another sensor s_2 shows $16°C$ ($e[s_2] = \langle 16 \rangle$). If an application does not care the difference of $5°C$, $e[s_1]$ and $e[s_2]$ are considered to be the same, i.e. $e[s_1] \equiv e[s_2]$. Thus, sensors not only stop by fault but also show arbitrary fault, i.e. Byzantine fault [11].

The wireless communication link with a sensor is not reliable. That is, messages sent by sensors to actors may be lost and damaged.

We assume actors are communicating with each other by using highly reliable links like wire links. Actors are higher-cost devices with enough energy and computation capability. Actors are reasonably assumed to only stop by fault in this paper. In addition, we also assume device objects are reliable and communication links among actors and objects are reliable.

In summary, we make the following assumptions on faults to occur in WSAN:

1. A sensor may be arbitrarily faulty, i.e. suffer from Byzantine fault. That is, a sensor may not send a message, may send a different value from other sensors to actors for a some event, and may send a message even if an event does not occur.
2. An actor may only suffer from stop fault.
3. Actuation device objects are reliable. In reality, device objects may suffer from arbitrary fault. We make this assumption in this paper for simplicity.
4. Omission faults occur in the communication between sensors and actors, i.e. messages may be lost.
5. Actors reliably communicate with each other. Actors also reliably communicate with actuation device objects.

We discuss how to increase the reliability of sensors and sensor-actor communication.

3.2. Fault-tolerant sensors

For each event e occurring in an event area, multiple sensors gather the information of the event e. Then, sensors send sensed information to actors. Sensors might be arbitrarily faulty. That is, a sensor may not send sensed information and may send incorrect information to actors. We assume that the maximum number of faulty sensors is bounded to be f_1 in each event area. We assume every actor is proper as long as the actor is operational. That is, an actor may only suffer from stop fault. We assume every proper sensor sends the same value of sensed information for an event e to actors. On the other hand, a faulty sensor may not send any value, may send a value different from the proper sensor, or may send a value even if no event occurs.

Here, let $t_i[e]$ show global time when a sensor s_i senses an event e in an event area. We assume that for every pair of sensors s_i and s_j in an event area, $|t_i[e] - t_j[e]| \leq \epsilon_1$. Let $s_i[e]$ be global time when an actor a_s receives a message of an event e from a sensor s_i. Here, we assume $|r_{si}[e] - r_{sj}[e]| \leq \epsilon_2$ if an actor a_s receives messages of sensed information of an event e from a pair of proper sensors s_i and s_j. ϵ_2 is given by the maximum difference of delay and ϵ_1. Suppose an actor a_s receives sensed information $s_i[e]$ of an event e from a sensor s_i at time t_i. If $|t_i - t_j| \leq \epsilon_2$, the sensor a_s is referred to as *simultaneously* receive $s_i[e]$ and $s_j[e]$. If each actor a receives sensed information from more than $2f_1 + 1$ sensors, at most f_1 sensors are faulty and at least $f_1 + 1$ sensors are proper. Hence, the actor a takes the majority value of the sensed information from the sensors [Figure 4]. A sensor gathers information of an event for each time unit.

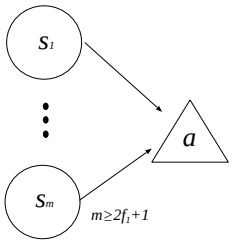

Figure 4. An actor which receives sensed information from ($2f_1 + 1$) sensors

4. Actors

4.1. Multiple instances of an action

On receipt of sensed information of an event e, an actor a makes a decision on what methods to be performed and then performs the methods on actuation device objects in an event area. In the multi-actor/multi-sensor (MAS) model, multiple actors $a_1, \cdots, a_m$ receive sensed information of an event e from multiple sensors in the event area. Suppose that a method op is to be performed on an object o in the event area for an event e. If each actor a_i performs the method op, the method op is $m(\geq 1)$ times performed on the object o. Here, the state of the object o may get inconsistent. For example, suppose each of two actors a_1 and a_2 receives sensed information of an event e on the temperature, each of the actors a_1 and a_2 makes a decision on cooling the air with $2°C$ degree. Then, each of the actors a_1 and a_2 issues a method *up* (2) to the *air-conditioner* object ac. The temperature is increased by $4°C$ because the object ac receives a pair of *up* (2) methods. Here, the method up should be performed only once on the *air-conditioner* object ac for each event e even if multiple actors receive the

sensed information of the event e from sensors.

Next, suppose each of the actors a_1 and a_2 issues a method *temp*. Here, the state of the *air-conditioner* object ac is not changed. Thus, multiple actors can multiple times issue an idempotent method like *temp* to an object.

Following the examples, we classify methods supported by objects. There are following types of methods:

1. Change method.
2. Non-change method.

By a change type of method op, state of an event area is changed, i.e. $op(s) \neq s$ for some state s. The method *up* is a change type. On the other hand, the state is not changed by a non-change method, i.e. $op(s) = s$ for every state s. The method *temp* is a non-change method. For each event e, a change type method op can be performed only once even if multiple instances of the method op are issued by multiple actors. A non-change method can be performed multiple times on an object, i.e. idempotent.

There are following ways to realize the unique execution of a change method on an object:

1. Actor-side solution.
2. Object-side solution.

In the actor-side solution, only one method is issued to an object from multiple actors. In one way, the actors cooperate with each other to make a consensus on what action to issue the method. It takes time to exchange messages among the actors, e.g. three rounds if the two-phase commitment protocol [15] is taken.

In the object-side solution, an object o takes a method op only once even if each of multiple actors sends the method to the object. Each of the actors issues a method op to an object o. Here, each instance of the method op to be taken for an event should be uniquely identified. On receipt of a method op, an object o checks the identifier id of an instance of the method op. If the method id had not been performed on the object o, the method op is performed on the object o. Then, the identifier id is recorded in the log. If the identifier id of the method op is found in the log, the method op is not performed since another instance of the method op is performed already on the object o. In WSAN, realtime communication is in nature required. We take the object-side solution by giving the unique identifier to each method. Each instance op_i of a method issued by an actor s_i is identified in a pair of method type op and time t_i when s_i receive an event. Here, a pair of identifiers $\langle op_i, t_i \rangle$ and $\langle op_j, t_j \rangle$ of instances op_i and op_j, respectively, are temporarily *equivalent* ($op_i \cong op_j$) iff $|t_i - t_j| \leq \epsilon$.

4.2. Synchronization of multiple actors

Next, suppose a pair of events e_1 and e_2 occur in an event area. An actor a_1 receives sensed information of the event e_1 and another actor a_2 receives sensed information of the event e_2. A pair of methods op_1 and op_2 are to be performed on an object o_1 for the events e_1 and e_2, respectively. A pair of methods op_1 and op_2 are to be performed on an object o_2 for the events e_1 and e_2, respectively. A pair of the methods op_{11} and op_{21} are performed on the object o_1 and a pair of the methods op_{12} and op_{22} are performed on the object o_2. Here, suppose a pair of methods op_{11} and op_{21} conflict on the object o_1 as well as a pair of the methods op_{12} and op_{22} conflict on the object o_2. The serializability [3] is required, i.e. op_{11} is performed on the object o_1 before op_{21} iff op_{12} is performed on o_2 before op_{22}.

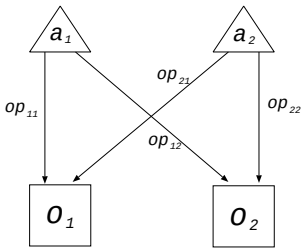

Figure 5. Serializability

In order to realize the serializability, objects are locked in one way [3, 6]. A method op waits if an object is locked by another method conflicting with the method op. In another way, method requests are timestamped in each action [3]. Here, every pair of conflicting methods are performed in the time-stamp order. Actuation device objects are classified into a pair of types: synchronous and non-synchronous object. A synchronous object supports some synchronization mechanisms. A synchronous object is a locking type if *lock* and *unlock* methods are supported. Another type (L) is TO (timestamp ordering) one. In order to prevent deadlocks, we take TO objects. Each object takes methods from multiple actors and conflicting methods are performed in the timestamp order.

A non-synchronous object does not support any synchronization mechanism. An actuation device object normally does not support any synchronization mechanism like locking and time-stamp ordering one. Methods are performed on a non-synchronous object in a receipt sequence of the methods. Hence, actors are required to issue synchronously conflicting methods with each other.

4.3. Ordered delivery of sensing and action messages

A sensor s senses an event e and sends sensed information $e[s]$ to actors in *Actor*(s). An actor a receives sensed information $e[s]$ of an event e from a sensor s in *Sensor*(e). **[Definition]** If an actor a receives sensed information $e_1[s_1]$ before $e_2[s_2]$, $e_1[s_1]$ *precedes* $e_2[s_2]$ in the actor a ($e_1[s_1] \rightarrow_a e_2[s_2]$).

[Definition] If a sensor s senses $e_1[s_1]$ before $e_2[s_1]$, $e_1[s_1]$ *precedes* $e_2[s_1]$ in the sensor s ($e_1[s_1] \rightarrow_s e_2[s_1]$).

We assume that every actor a is equiped with a GPS time server. Every event e is timestamped with real time in an actor. By the timestamp, events are ordered [9].

5. Concluding Remarks

In this paper, we discussed how to make a wireless sensor and actor network (WSAN) fault-tolerant and how to order messages. Sensors are less reliable and may be arbitrarily faulty, due to low-energy, low cost devices. We proposed multi-actor/multi-sensor (MAS) model to realize the fault-tolerant WSAN.

Acknowledgment

This research is partially supported by Research Institute for Science and Technology [Q05J-04] and Frontier Research and Department Center [16-J-6], Tokyo Denki University.

References

[1] I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks journal (Elsevier)*, 2:351–367, 2004.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks journal (Elsevier)*, 38:393–422, 2002.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[4] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, 1991.

[5] A. Durresi and V. Paruchuri. Geometric broadcast protocol for heterogeneous sensor networks. *Proc. of 19th IEEE International Conf. on Advanced Information Networking and Applications (AINA2005)*, 1:343–348, 2005.

[6] K. P. Eswaren, J. N. Gray, R. Lode, and I. L. Traiger. The Notion of Consistency and Predicate Locks in Database Systems. *Communications of the ACM*, 19(11):624–637, 1976.

[7] J. Gray and A. Reuter. *Transaction Processing : Concepts and Techniques*. Morgan Kaufmann, 1993.

[8] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The φ accural failure detector. *Proc. of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS-23)*, 1:68–78, 2004.

[9] S. Kawanami, T. Nishimura, T. Enokido, and M. Takizawa. A Scalable Group Communication Protocol with Global Clock. In *Proc. of AINA-2005 International Workshop on Ubiquitous Smart Worlds (USW 2005)*, volume 2, pages 625–630, 2005.

[10] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM*, 21(7):558–565, 1978.

[11] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[12] F. Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[13] T. Melodia, D. Pompili, V. C. Gungor, and I. F. Akyildiz. A distributed coordination framework for wireless sensor and actor networks. *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, 1:99–110, 2005.

[14] V. Paruchuri, A. Durresi, and L. Barolli. Energy aware routing protocol for heterogeneous wireless sensor networks. *Proc. of 16th International Workshop on Detabase and Expert Systems Applications (DEXA2005)*, pages 133–137, 2005.

[15] D. Skeen and M. Stonebraker. A Formal Model of Crash Recovery in Distributed Systems. *IEEE Trans. Software Engineering*, 9(3), 1983.

[16] A. K. Somani and N. H. Vaidya. Understanding fault tolerance and reliability. *IEEE Computer*, 30:45–50, 1997.

[17] T. Tachikawa, H. Higaki, and M. Takizawa. Group communication protocol for realtime applications. *Proc. of the 18th IEEE International Conf. on Distributed Computing Systems(ICDCS-18)*, pages 40–47, 1998.

[18] M. Weiser. Hot topics: Ubiquitous computing. *IEEE Computer*, pages 71–72, 1993.

[19] S. Yasuzawa, M. Takizawa, and T. Ouchi. Resolution of Parallel Deadlock by Partial Abortion. *Proc. of the 1st International Symposium on Computer Communications (ISCOM)*, pages 708–711, 1991.