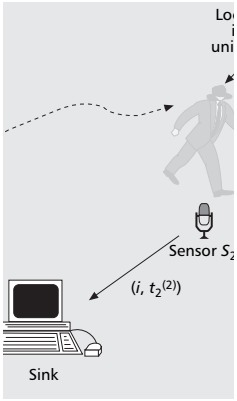
SECURE TIME SYNCHRONIZATION PROTOCOLS FOR WIRELESS SENSOR NETWORKS

AZZEDINE BOUKERCHE, UNIVERSITY OF OTTAWA
DAMLA TURGUT, UNIVERSITY OF CENTRAL FLORIDA ORLANDO



We review the most commonly used time synchronization algorithms and evaluate these algorithms based on factors such as their countermeasures against various attacks and the types of techniques used.

ABSTRACT

Time synchronization is essential in wireless sensor networks as it is needed by many applications for basic communication. The inherent characteristics of sensor networks do not permit simply applying traditional time synchronization algorithms. Therefore, many new time synchronization algorithms have been proposed, and a few of them provide security measures against various degrees of attacks. In this article we review the most commonly used time synchronization algorithms and evaluate these algorithms based on factors such as their countermeasures against various attacks and the types of techniques used.

INTRODUCTION

Distributed wireless sensor networks heavily depend on time synchronization for various reasons such as determining location and proximity of deployed sensor nodes, intranetwork coordination among different sensor nodes, temporal message ordering, security, time-division multiplexing in wireless communication, improving energy efficiency of sensor nodes by scheduling their sleep times, and so on [1].

Most computer devices contain an internal clock, usually designed to be synchronized with the exact real-world time at the specific location of the computer. Although many functionalities depend on the clock even on desktop computers, such as the scheduled Friday afternoon virus checks or the popular "make" program, which determines whether a file needs to be recompiled by comparing the timestamp of the source and object files. However, in practice, a desktop computer can function correctly even if its internal clock is minutes or even years away from the correct time.

Let us first define the ways in which the clocks of two nodes, A and B, might be out of sync. Let us note the clock of a node X with a function $C_X(t)$, which returns the reading of the clock at real time t. The first type of difference is *clock offset*: $\delta_{AB} = C_A(t) - C_B(t)$. That is, the two clocks are identical, except that the clock of node A is early (if $\delta_{AB} > 0$). Now, we might fix this by setting the clock of node A back; however, that would create a problem, because the same time slice would appear twice for node A. This creates major problems for a number of protocols. It is better to set the clock of node B forward; however, most protocols simply require the nodes to keep track of their offsets without actually changing the internal clock.

The second type of synchronization difference is *clock skew*, η_{AB}: one clock is running faster than the other. This can be expressed as a difference in the derivatives of the clock function in respect to the time:

$$\eta_{AB} = \frac{\partial C_A(t)}{\partial t} - \frac{\partial C_B(t)}{\partial t}.$$

While this appears to be a more difficult problem, if a node is aware of its clock skew, it can very easily account for it.

Neither clock offset nor clock skew requires periodic synchronizations. If we know the offset and skew of a node's clock, we can calculate the time difference at any moment in time. However, the frequency of the clocks can change randomly due to environmental conditions such as temperature differences or aging of the hardware, a condition called *drift error* and denoted λ_{AB}. Drift error appears as a nonzero second derivative in one or both clocks:

$$\lambda_{AB} = \frac{\partial^2 C_A(t)}{\partial t^2} - \frac{\partial^2 C_B(t)}{\partial t^2}.$$

The clocks of sensor nodes usually accumulate several seconds of drift error per day; as drift is not predictable, it needs to be solved using clock synchronization.

However, for sensor networks the correct synchronization of clocks is frequently a necessary component of the ability of a sensor network to function correctly. Unsynchronized clocks can yield invalid observations, create uncovered areas and timeslots, and in the worst case disable the communication architecture of the network.

1536-1284/07/$20.00 © 2007 IEEE

Let us consider several examples. The individual nodes of the sensor network send their timestamped observations to the sink.

In Fig. 1, an intruder is sensed consecutively by sensors S_1 and S_2, and their reports are sent to the sink. Based on reports (intruder, S_1, t_1) and (intruder, S_2, t_2), knowing the locations of the sensors S_1 and S_2, and noticing that $t_1 < t_2$ and $t_2 - t_1 < 1$ s, the sink can correctly infer that the observations refer to the same intruder who is moving from left to right (in certain cases this inference can be performed through in-network processing). However, this inference is valid only under the assumption that the clocks of the two sensors are synchronized at the level of tenths of seconds. Let us explain this further.

If clocks 1 and 2 are synchronized (i.e., they have the same offset $\delta^{(1)} = \delta^{(2)}$ compared to a universal time t), $t_2^{(2)} - t_1^{(1)} = t_2 + \delta^{(2)} - t_1 - \delta^{(1)} = t_2 - t_1 > 0$. Thus, $t_2 - t_1$ indicates the correct order of arrival to the sensors. However, if there is a large offset between these two, $\delta^{(1)} - \delta^{(2)} \ll 0$; that is, the clock of $\delta^{(1)}$ is early. We might have a situation that $t_2^{(2)} - t_1^{(1)} = (t_2 - t_1) + \delta^{(2)} - \delta^{(1)} < 0$; that is, the sink will infer incorrectly that the intruder is moving from right to left. This is so if the clock of sensor S1 is 2 s late, the inference would be that the intruder moves from right to left. As a drift of several seconds per day is a normal occurrence for the internal oscillators of the devices, we cannot rely on the initial setting of the clocks at deployment time. The clocks need to be synchronized periodically in the field.

Notice that the faster the intruder moves, the smaller $(t_2 - t_1)$, and the more accurate the synchronization needed in order to make the correct inferences.

Our second example concerns the wake-up time of sensors. Sensors have limited power resources. To extend the lifetime of a deployed network, sensor nodes are frequently selectively put to sleep. The idea of the method is that the set of currently active nodes at any given moment in time covers the area to be surveyed and forms a connected network. If an attacker can modify the internal clock of certain sensor nodes, such that these nodes, for instance, do not wake up in time, certain areas might not be surveilled by the sensors for a certain amount of time, allowing an intruder to operate unreported.

Finally, time-division multiple access (TDMA)-based channel sharing protocols rely on the participating nodes to transmit at well defined time slots. Relatively small time drifts in the clocks of individual nodes can make transmission intrude on an adjacent time slot, causing a collision. Repeated collisions can significantly disrupt the network. A detailed survey of clock synchronization protocols can be found in [2, 3].

The rest of the article is organized as follows. We survey some of the time synchronization protocols, including possible attacks and proposed countermeasures against these attacks. The types of attackers and attacks are presented. We give a detailed discussion of the approaches to secure time synchronization. We then conclude the article.

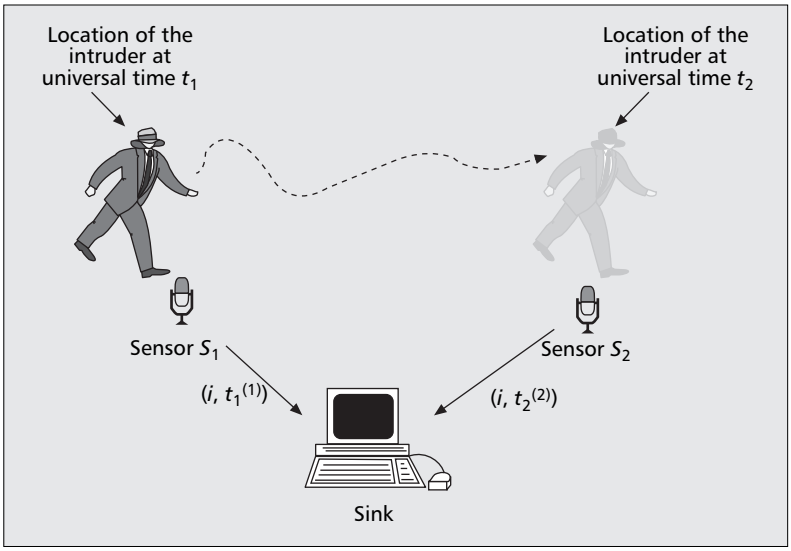


Figure 1. *Intruder movement detected by sensors 1 and 2 at times $t_1^{(1)}$ according to clock 1 and $t_2^{(2)}$ according to clock 2.*

Secure Time Synchronization

These examples show us that time synchronization is vital for the correct operation of a sensor network. As relatively small time drifts can cause significant disruption, we cannot rely on the precision of the hardware; we need to use external synchronization protocols. Furthermore, it was found that relatively small changes in the clocks can disturb the operation of the sensor network or even cause it to make erroneous inferences about the observed event. Time synchronization protocols are a convenient target for malicious attackers. Most time synchronization protocols were not designed with security in mind. Recently, however, several research groups have performed analyses of various vulnerabilities, and proposed countermeasures against them.

In the following, we survey some of the time synchronization protocols, discuss their benefits and outline possible attacks and proposed countermeasures.

Reference Broadcast Synchronization

The Reference Broadcast Synchronization (RBS) [4] method is based on a synchronization signal broadcast by an external unit. The receivers record their local time when they receive this reference message, and then exchange this information among themselves. The recording of a message is not 100 percent exact, because of hazards such as the propagation time of the message or the processing time of the packet at the lower protocol layers. To improve precision, a number of reference messages can be broadcast; the nodes exchange the arrival times for each message and then find the best approximation using a least squares fit.

Possible attacks: In RBS two nodes, upon receiving a broadcast signal, exchange their local clock times. An attack can happen if one of the receiver nodes is compromised with an incorrect time. The compromised node can then send the incorrect time information to its neighbor, causing the uncompromised node to calculate an incorrect offset.

Any node can declare itself a root, and the protocol relies on the node to step back if a lower id root appears. A compromised node can easily masquerade as a root, and by declaring a very low id it can actually dislodge the existing legitimate root.

Time Synchronization Protocol Sensor Networks

Time Synchronization Protocol for Sensor Networks (TPSN) [5] creates a spanning tree for the sensor network. The root of the tree is usually a base station. Clock synchronization is done by synchronization of the child nodes to the parent. Synchronization is initiated by a child node, which sends a synchronization packet at time t_1. This is received by the parent at t_2, and an acknowledgment packet is sent in response at time t_3. The values of t_2 and t_3 will be included in the acknowledgment packet. This packet is received by the child node at time t_4. Knowing these four time values, the child node can calculate its clock offset relative to the parent node as

$$\Delta t = \frac{(t_2 - t_1) - (t_4 - t_3)}{2}.$$

Possible attacks: Naturally, the child node relies on the parent for its clock synchronization; by providing incorrect values for t_2 and t_3, the parent can set an arbitrary offset on its child node. What is more, this incorrect offset will then be propagated down the tree. Therefore, the number of nodes whose synchronization can be affected by the compromised node depends on the location of the compromised node on the tree. One way for a malicious attacker to compromise a larger number of nodes is to reposition itself in a higher location on the tree or to answer queries instead of the proper parent. This is surprisingly easy to do in the original algorithm.

Flooding Time Synchronization Protocol

In the Flooding Time Synchronization Protocol (FTSP) [6], nodes participate in a process in which a root node is elected. The root is the origin of the time synchronization messages. If a node does not hear a time synchronization message for a while, it declares itself the new root. The protocol requires that if at a later time the node receives a time synchronization message from a node with a lower id than itself, it gives up its root status. When a node receives a time synchronization message from the root, it adjusts its clock and broadcasts its own time to its neighbors.

Possible attacks: FTSP is more robust to node failures than TPSN, as there is no need to maintain a tree structure, which is notoriously vulnerable to single point failures — the failure of a single node can disconnect a whole subtree. The weak point of FTSP is the election process. Any node can declare itself a root, and the protocol relies on the node to step back if a lower id root appears. A compromised node can easily masquerade as a root, and by declaring a very low id it can actually dislodge the existing legitimate root. Then, by sending a synchronization message with a fake timestamp, it can make the nodes synchronize to an incorrect time.

Attackers and Attacks

Possible attacks against time synchronization protocols depend on the nature and capabilities of the attackers. We first identify three different types of attackers, outline the types of attacks of which they are capable, and then discuss the various types of defenses proposed against these types of attacks.

We describe the system with the characters regularly used in the description of cryptographic protocols. We assume that Alice and Bob (and potentially additional nodes Carol, Dave, and so on) are engaging in a time synchronization process.

The **malicious outsider**, Malory, is a wireless device inserted in the range of nodes of the sensor network, which has the ability to send and receive packets. We assume that the attacker can eavesdrop on any ongoing transmission; we can also assume that the attacker can transmit messages which are physically indistinguishable from other nodes' messages. However, this type of attacker does not have access to keys or other confidential information, other than what it can infer from eavesdropping on transmissions.

An **attacker with jamming and replay ability**, Jimmy, has the ability to jam a message, record it, and possibly replay it at a later time. This type of attack is called a *pulse delay attack*. Although the existence of jamming in principle can be detected, it requires significant resources, and by default most nodes are not prepared for it.

A **compromised node** is a node taken over by an attacker. We denote one as Zach (for zombie). One example of this is the physical capture of a node by an attacker, although a node can, in principle, be compromised by purely software methods. Compromised nodes have access to all the keys and other information of the original node, and represent the most difficult type of attackers to defend against.

The challenge of secure time synchronization is to defend against all three types of attackers. An attacker is considered successful if it succeeds in making the nodes calculate an incorrect offset. By default, all three time synchronization protocols we have described are vulnerable to all three types of attackers.

Approaches to Secure Time Synchronization

Malicious outsiders can affect all types of protocols. The primary defense against a malicious outsider is cryptographic techniques for authentication of messages. If the sender and receiver share a key K_B, they can use it to sign messages. The nodes can be provided with the shared key at the time of deployment, or they can rely on secure key exchange algorithms such as the Diffie-Hellman protocol [7]. To prevent an attacker capturing a valid message and inserting a copy of it later in a different synchronization round, the sender sends a random *nonce* in the initial message, which then needs to be signed by the synchronization partner. While the attacker can still replay the same message in the same synchronization round, by simply considering

only the first arrived message, the receiver can ignore the malicious outsider.

Ganeriwal *et al.* [8, 9] proposed a series of secure time synchronization protocols based on this idea. The protocols are adapted to pairwise single-hop, multihop, and group synchronization. The protocols can also detect the existence of a pulse delay attack by calculating the end-to-end delay d of the message. If the delay is larger than a predetermined threshold d^*, the protocols assume that an attack is in progress and abort the synchronization. We should note that key exchange is a major problem for these types of algorithms, due to the ways in which sensor nodes are deployed, which does not always permit the exchange of keys in a secure environment.

Notice that cryptographic methods are not feasible against a compromised node, which has all the keys and knowledge to correctly answer all challenges, and appropriately sign its messages. If the time synchronization protocol happens only between Alice and Zach, it is impossible for Alice to detect or mitigate the attack. However, for protocols with a larger number of participants, we can use redundancy in the synchronization messages to identify malicious participants or messages. We note that delay attacks can be performed by either Zach or Jimmy, but not Mallory.

Song *et al.* [10] propose a method for making time synchronization protocols resilient to delay attacks based on techniques of *outlier detection*. The essential assumption behind this method is that the synchronization signals received from compromised nodes will be "much different from others." Thus, messages coming from compromised nodes can be identified, using statistical techniques, as outliers and eliminated from the package, and the synchronization performed with the remaining nodes.

The authors propose two alternative methods. One of them uses the generalized extreme studentized deviate (GESD), a generalization of the well-known Grubb's test from statistics. GESD can identify multiple outliers in a sample drawn from a normal distribution. GESD requires as one of its outputs the estimated number of malicious nodes.

A somewhat simpler approach is based on a delay threshold. At system setup, the nodes determine the maximum amount of time offsets they will tolerate, based on information about the typical drift rate of the nodes. A received offset that is higher than this value is considered to come from a malicious node and discarded.

As an observation, naturally Zach the compromised node would have exact knowledge about the thresholds used (but not Jimmy). Therefore, Zach has the possibility to remain undetected by setting the delay just below the threshold (or, in the GESD case, such that it will not be identified as an outlier), but still have a distorting effect on the time synchronization process. Thus, statistical techniques can only reduce but not necessarily eliminate the effect of delay attacks by nodes with insider knowledge.

Sun *et al.* [11] propose a statistical method for secure and resilient clock synchronization in the presence of compromised nodes. The techniques are applied for both *level-based clock synchronization*, where a hierarchical structure of nodes is developed that determines which node is synchronized with whom, and *diffusion-based clock synchronization*, which does not use such a structure and simply relies on the reachability information of the network. Naturally, the level-based approach allows more disciplined control of the synchronization flow, and thus higher accuracy, whereas the diffusion method has the advantage that it can be applied to dynamic sensor networks with mobile nodes.

Furthermore, the authors consider both the case with a single source of synchronization information and that with multiple sources. For instance, in the single source case, the goal is to find the clock offset δ_{iS} from the node to the source. The technique assumes that at every level a normal node collects $2t + 1$ candidate source clock differences from its $2t + 1$ neighbors and chooses the *median* of them. Thus, the node can tolerate up to t compromised nodes while retaining correct synchronization. Similar considerations apply to the diffusion-based approach. In the case of multiple sources, the node can receive synchronization information from $2s + 1$ sources, synchronized to the same external standard (e.g., a GPS signal) and tolerate up to s compromised sources by selecting the median.

Note that this approach uses the whole redundancy of the system to defend against an external attack: out of $2t + 1$ recorded offsets, the method will pick a single one, the offsets median. Approaches that assume a benign environment usually select the *mean* of these measurements, thereby improving accuracy; however, the mean is vulnerable to even a single malicious node.

In addition, this approach requires unique pairwise-key-based authentication of the nodes. Otherwise, the malicious node could impersonate multiple nodes (the so-called Sybill attack).

A significantly improved version of this technique was presented in [12]. In the approach called TinySeRSync, time synchronization is performed in two phases. While we call them phases I and II for convenience, these two processes take place asynchronously in the sensor network. In the first phase single-hop pairwise synchronization is performed. The main feature of the pairwise synchronization process is that it relies on a hardware enhanced authenticated medium access control (MAC) layer timestamping. The hardware is programmed to add a timestamp authenticated with a message integrity code (MIC) to every MAC packet transmitted. This is especially challenging for newer radios, such as the ones on the newer-generation MICAz motes, where the time required to authenticate the timestamp can interfere with the transmission rate of the radio. The authors propose a prediction-based approach where the authenticated timestamp includes a prediction of the time required to calculate the MIC.

Through these techniques (Table 1), the nodes achieve sufficiently good local level synchronization, which is exploited in the second phase. The second phase implements global synchronization using the ηTESLA broadcast

The level based approach allows for a more disciplined control of the synchronization flow, and thus a higher accuracy, whereas the diffusion method has the advantage that it can be applied to dynamic sensor networks with mobile nodes.

Approach	Protects against	Uses crytographic techniques?	Uses statistical techniques?
Ganeriwal *et al.* [8]	Jimmy, Mallory	Yes	Yes
Song *et al.* [10]	Zach, Jimmy, Mallory	No	Yes
Sun *et al.* [11]	Zach	No	Yes
TinySeRSync, Sun *et al.* [12]	Zach, Mallory	Yes	Yes
Manzo *et al.* [13]	Zach	Yes	Yes

Table 1. *A concise summary of the various techniques proposed for secure time synchronization.*

authentication protocol. This protocol relies on loose time synchronization between nodes using a unidirectional keychain. Messages received need to be stored by the receiver and are authenticated only after several time slots. This prevents an attacker from forging messages, but opens the doors to a denial of service attack. The messages received need to be buffered for future authentication, and as the memory of sensor nodes is limited, Malory can create fake messages that will not pass the authentication test but will fill the buffer, preventing the node from receiving legitimate messages. To prevent denial of service attacks, the authors propose a modified version of ηTESLA. To reduce time slots when the adversary nodes can flood the node with messages based on captured keys (which the receiver node needs to store for future authentication), TinySeRSync uses an implementation with very short delays r (made possible by the good local synchronization achieved in phase I). However, such short delays would require the generation of a large number of keys; the implementation alternates short intervals r used for message broadcasting with long intervals R used for broadcasting the disclosed keys.

The global synchronization in TinySeRSync still relies on the selection of the median from the $2t+1$ candidate offsets, thus tolerating the presence of at most t compromised nodes.

Notice that the approach presented in [11, 12] uses the median rather than the mean as a choice of estimated time offset, thus obtaining high protection against malicious nodes (provided the technique is coupled with cryptographic defenses). However, it sacrifices the ability to improve precision through multiple independent observations.

We can attack the general problem of finding the best estimation of the time offset δ_{best} from a set of candidate offsets $\{\delta_1, \ldots, \delta_n\}$ by applying the principles of *robust estimation*. We note that the individual offset measurements δ_i might have natural noise, but some of them might be a result of a malicious attack. For any estimation method, the *breakdown point* is the smallest number of contaminated values that can move the estimate arbitrarily far from the correct value. Unfortunately, the most frequently used estimators, the average and least squares estimator, have very low breakdown points; a single malicious value can modify the estimate arbitrarily far. Manzo *et al.* [13] propose the use of the least mean squares (LMS) estimator for more robust modeling. GESD used by [10] is another example of the application of the techniques of robust estimation.

Conclusions

Among the many challenges in designing and employing wireless sensor networks is the clock synchronization between the sensor nodes. Agreeing on a common time is needed and even required by many of the sensor applications to carry out the sensing, communication, and processing of sensed data. The time synchronization protocols in traditional wired networks cannot simply be reused in the wireless sensor network domain due to the inherent characteristics and limited resources of these networks. Therefore, several time synchronization protocols have been proposed recently; however, most of them do not consider security during the design stages. There are only a handful of protocols where security has been taken into the consideration.

In this article we review the three most common secure time synchronization protocols:

- Reference Broadcast Synchronization (RBS)
- Time Synchronization Protocol Sensor Networks (TPSN)
- Flooding Time Synchronization Protocol (FTSP)

We then evaluate these algorithms based on factors such as their countermeasures against various attacks and the types of techniques used (cryptographic vs. statistical).

References

[1] K. Romer, "Time Synchronization and Localization in Sensor Networks," Ph.D. diss., ETH Zurich, Switzerland, 2005.

[2] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock Synchronization for Wireless Sensor Networks: A Survey," *Ad Hoc Networks*, vol. 3, no. 3, May 2005, pp. 281–323.

[3] F. Sivrikaya and B. Yener, "Time Synchronization in Sensor Networks: A Survey," *IEEE Network Mag.*, Special Issue on *Ad Hoc Networking: Data Communications and Topology Control*, vol. 18, no. 4, July/Aug. 2004, pp. 45–50.

[4] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts," *Proc. 5th Symp. Op. Sys. Design and Implementation*, Dec. 2002, pp. 147–63.

[5] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-Sync Protocol for Sensor Networks," *ACM Proc. 1st Int'l. Conf. Embedded Networked Sensor Sys.*, Nov. 2003, pp. 138–49.

[6] M. Marhoti *et al.*, "The Flooding Time Synchronization Protocol," *ACM Proc. 2nd Int'l. Conf. Embedded Networked Sensor Sys.*, Nov. 2004, pp. 39–49.

[7] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Info. Theory*, vol. IT-22, no. 6, Nov. 1976, pp. 644–54.

[8] S. Ganeriwal *et al.*, "Secure Time Synchronization Service for Sensor Networks," *Wireless Sec. Wksp.*, Sept. 2005, pp. 97–106.

[9] S. Ganeriwal, S. Capkun, and M. B. Srivastava, "Secure Time Synchronization in Sensor Networks," *ACM Trans. Info. and Sys. Sec.*, Mar. 2006.

[10] H. Song, S. Zhu, and G. Cao, "Attack-Resilient Time Synchronization for Wireless Sensor Networks," *Ad Hoc Networks J.*, Elsevier, vol. 5, no. 1, Jan. 2007, pp. 112–25.

[11] K. Sun, P. Ning, and C. Wang, "Secure and Resilient Clock Synchronization in Wireless Sensor Networks," *IEEE JSAC*, vol. 24, no. 2, Feb. 2006, pp. 395–408.

[12] K. Sun *et al.*, "TinySeRSync: Secure and Resilient Time Synchronization in Wireless Sensor Networks," *Proc. 13th ACM Conf. Comp. and Commun. Sec.*, Nov. 2006.

[13] M. Manzo, T. Roosta, and S. Sastry, "Time Synchronization Attacks in Sensor Networks," *Proc. 3rd ACM Wksp. Sec. of Ad Hoc and Sensor Networks*, Nov. 2005, pp. 107–16.

BIOGRAPHIES

AZZEDINE BOUKERCHE (boukerch@site.uottawa.ca) is a full professor and holds a Canada Research Chair position at the University of Ottawa. He is the founding director of PARADISE Research Laboratory at the university. Prior to this he held a faculty position at the University of North Texas and worked as a senior scientist in the Simulation Sciences Division, Metron Corporation, San Diego, California. He has also been on the faculty of the School of Computer Science at McGill University and taught at the Polytechnic of Montreal. He spent a year at JPL/NASA-California Institute of Technology, where he contributed to a project centered on the specification and verification of the software used to control interplanetary spacecraft operated by JPL/NASA. His currrent research interest include wireless ad hoc and sensor networks, wireless networks, mobile and pervasive computing, wireless multimedia, QoS service provisioning, performance evaluation and modeling of large-scale distributed systems, distributed computing, large-scale distributed interactive simulation, and parallel discrete event simulation. Dr. Boukerche has published several research papers in these areas. He was the recipient of the Best Research Paper Award at IEEE/ACM PADS '97 and ACM MobiWac '06, and the Third National Award for Telecommunication Software 1999 for his work on distributed security systems for mobile phone operations, and was nominated for the Best Paper Award at IEEE/ACM PADS '99 and ACM MSWiM '01.

DAMLA TURGUT [M] (turgut@eecs.ucf.edu) is an assistant professor with the School of Electrical Engineering and Computer Science at the University of Central Florida. She is affiliated with the Networking and Mobile Computing Research Laboratory (NetMoC), Interdisciplinary Information Science Laboratory (I2Lab), and Institute for Simulation and Training (IST) at UCF. She received her B.S., M.S., and Ph.D. degrees from the Computer Science and Engineering Department of the University of Texas at Arlington in 1994, 1996, and 2002, respectively. She was included in *Who's Who among Students in American Universities and Colleges* in 2002. She has been awarded an outstanding research award and has been a recipient of the Texas Telecommunication Engineering Consortium (TxTEC) fellowship. She is a member of the ACM and the Upsilon Pi Epsilon honorary society. Her research interests include wireless networking, mobile computing, embodied agents, distributed systems, and software engineering.