Implementing Synchronous Applications on WLAN

Elsa M. Macías and Alvaro Suárez[1]

University of Las Palmas de Gran Canaria, Grupo de Arquitectura y Concurrencia
Departamento de Ingeniería Telemática, Las Palmas de Gran Canaria, Spain
e-mail: {emacias,asuarez}@dit.ulpgc.es

***Abstract*: Nowadays, Wireless Local Area Networks (WLAN) can be used to execute synchronous applications including cooperative, video conference and wireless gaming. Due to computers can bring in and out of coverage during application execution, it is important to design a simple and efficient mechanism that avoids the terminal equipment waiting a long time for data of a computer that has gone out of coverage. In this paper we present a novel and simple but efficient mechanism to prevent terminal equipment blocking indefinitely when receives (sends) data from (to) a computer outside the coverage area. We apply this mechanism to an iterative loop carried dependencies application.**

1. Introduction

Nowadays, cellular communication systems, Wireless Personal Area Networks, Wireless Wide Metropolitan Networks and WLAN have given rise to the Wireless Revolution. In the scope of WLAN, a report [13] has shown that IEEE 802.11 standard [6] is the best in practice due to its level of standardization and real market applicability.

The traditional applications of these networks can be read in [4]. Applications that work fine with disconnected devices has been widely studied [1]. Future applications of wireless networks will include extreme collaboration [12], realtime video conference and wireless gaming [9] among others. Certain set of these applications can be modelled as a sequence of strong synchronized steps (inside a step, a set of parallel and distributed actions can be done) and disconnected work is not possible for these applications. Characteristics such as channel failures, the control of energy in batteries, abrupt disconnections of terminal equipment and synchronous communication have to be efficiently controlled in order to applications end their tasks gracefully. The design of this kind of applications has not been properly studied in the literature due to they demand channel failure management mechanisms that could forewarn applications to take corrective action and in this manner prevent abrupt ending. In this paper, we explain a novel, simple but efficient mechanism that ensures the correct data communication for this kind of applications. We illustrate its use with an iterative loop carried dependencies application.

The rest of the paper is organized as follows. In section 2 we review some important characteristics about the implementation of programs on WLAN. In section 3 we present a simple but efficient mechanism to detect the wireless channel state and we analyze its theoretical efficiency and scalability. Then, we apply the mechanism to program a real engineering problem. Finally we sum up the conclusions and present directions for further research.

[1]This work is supported by the Spanish CICYT under Contract TIC2001-0956-C04-03 and University of Las Palmas de G.C. UNI2002/17.

2. Important issues to implement applications on WLAN

We focus our work studying the efficient implementation of certain kind of applications in a heterogeneous network consisting of an interconnection between a Local Area Network (LAN) and an infrastructure WLAN. In Fig. 1 we graphically show this kind of applications, where bold circles represent the actions made in the Master Node (`MN`, see Fig. 2) by the master process and the other circles represent the actions in the slave computers (slave processes running on `PC` and `FN` that stand for Portable and Fixed Computers, see Fig. 2). The arrows represent data communication and synchronization among computers. In the step `i` master process communicates data to several parallel and distributed processes on slave computers. At the end of step `i` slave processes send data to the master to start step `i+1`. This means that the application exhibits a strong synchronization that must be done in a reduced time and the communication must not fail.

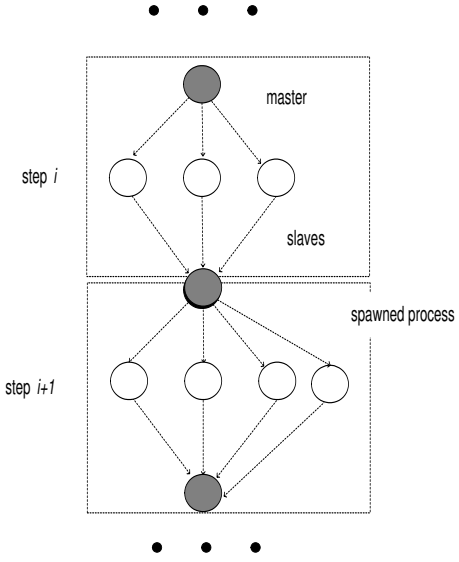


Figure 1: Programming model

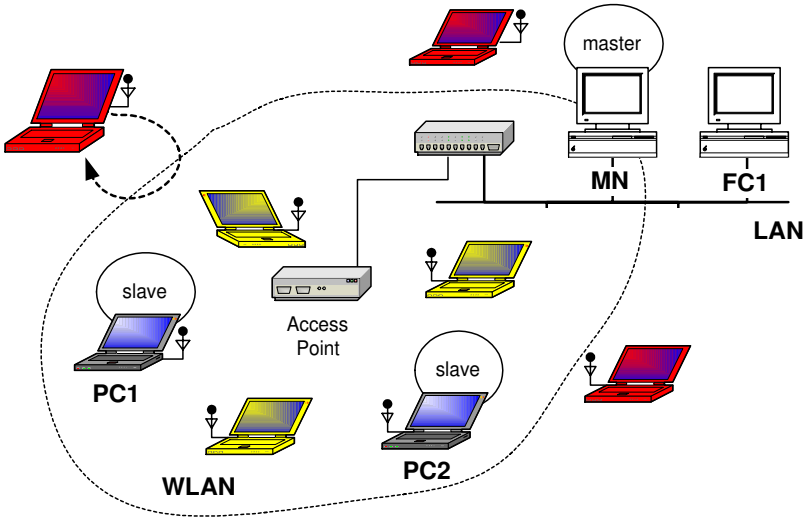


Figure 2: Portable computers inside coverage zone and processes

In Fig. 2 we show an example of the heterogenous network. In the WLAN, PC do not reassociate to a different access point to continue seamless doing computations or collaborations. Therefore, roaming between cells is not considered, that is to say, the mobility of PC is restricted to the coverage zone of its access point. Some important considerations must be taken into account: a) Processes should be spawned dynamically in PC just when they bring into the coverage (for example, in Fig. 1 one slave process is spawned at the beginning of step i+1) and should be destroyed when PC is disassociated. b) The PC could leave abruptly the computations or collaborations (leave the WLAN) and the applications would consider them as faulty. c) The PC in the coverage area could be spuriously disconnected.

Another important consideration is that although TCP could inform applications about congestion and spurious disconnections, it could be done for all PC within the coverage area. But spurious disconnections could not be handled efficiently [5] and there is no accepted way for a link level driver to inform TCP of packet loss and frequent disconnections [8]. Moreover, this technique could flood the wireless channel with a lot of irrelevant information for these applications because not all PC within the coverage area could be executing program parts as shown in Fig. 2 in which only two of the PC within the coverage area collaborate.

An interesting final remark is that an implementation of reliable sockets [14] can recover a failed connection after some specified time but it requires that the peers do not lose its IP addresses. In our environment we consider PC can change the IP address and, on the other hand, it would be an error to try to recover the connection with a PC that definitely leaves the WLAN.

For these reasons we propose a middleware that provides the applications with mechanisms to efficiently control the wireless channel failures. Next, we explain the main ideas of a particular function that applications can call to test, for example, if the receiver of a message can be reached.

3. The wireless channel fault detection mechanism

In [10] we proposed initial ideas about a simple and standard mechanism to detect the wireless channel state between two computers. The mechanism is based on injecting ICMP (Internet Control Message Protocol) echo request packets [3] from MN to PC and waiting for the echo replies. The main drawback of this implementation is that it introduces a high overhead (about ten seconds in average) to classify the channel state as a faulty one (no connectivity between both computers).

We now show an improved version that reduces the detection time by a factor of 3 (in presence of disconnections in the wireless channel) and a factor of 10 (when the wireless channel connectivity is good). Shortly, this new version is based on a client thread (running in MN) that attempts to make a connection to a TCP socket in the remote portable computer. If the connection succeeds, it means there is connection to PC. On the contrary, if the error code is such that no one is listening on the remote PC (value equal to ECONNREFUSED) the thread returns to the parent process that there is connection to PC. Otherwise, the wireless channel is broken and an error code is returned.

The overhead of this mechanism is graphically presented in Fig. 3. The tests consider different scenarios,

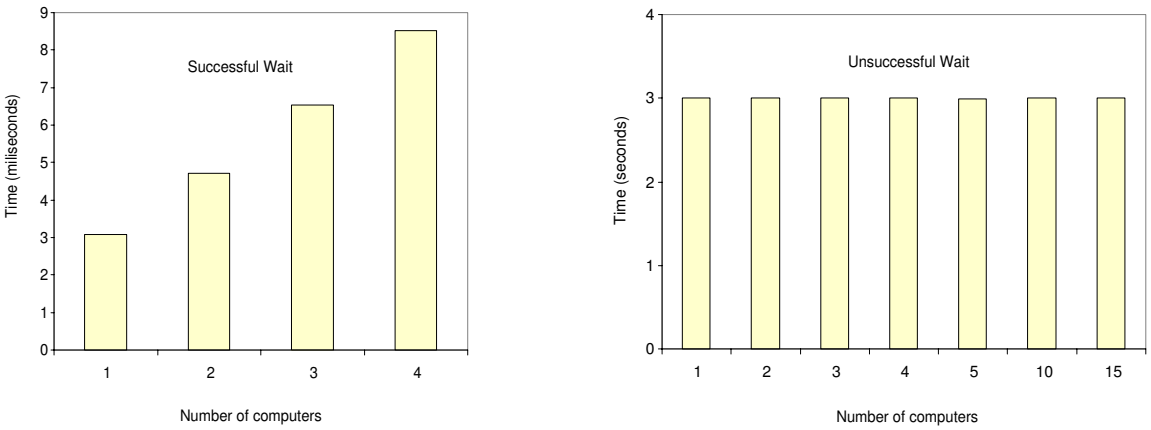


Figure 3: Average execution times (over 50 runs for Successful Wait and 20 runs for Unsuccessful Wait values) for socket based mechanism

varying the number of computers to be tested from 1 to 15 (4 with the network interface cards disabled and the remainder computers are not present in the network) to test the mechanism in presence of failures (values labelled as Unsuccessful Wait in Fig. 3), and from 1 to 4 PC with 11 Mbps wireless cards switched on and associated to an access point at 11 Mbps (values labelled as Successful Wait).

It is important to show the behavior of this mechanism in our heterogeneous target network. For this reason in the next section we show its theoretical efficiency and scalability.

3.1. The efficiency and scalability of the mechanism

Physical channel contention is of importance because it influences on the flow and congestion at TCP/IP level. The communication delay due to channel collisions is very short and it does not influence badly in our mechanism. For these reasons we now present a simple model to observe the contention on the IEEE 802.11 channel for our mechanism using the basic ideas proposed in [2]. We do not analyze the congestion control of TCP, the cost to assemble the messages into MAC (Medium Access Control) frames and the scheduling of above threads.

We consider one MN and n PC. The master transmits n messages to PC and they will reply to these messages. It is trivial that $2 * n$ messages must be interleaved in the channel. For that reason, the computers must content to gain the medium to transmit. In order to compute the delay due to this contention we take into account the following considerations according to the IEEE 802.11b standard:

- A computer monitors the channel activity for a period of time DIFS (Distributed Inter Frame Space) equal to 50μs for DSSS (Direct Sequence Spread Spectrum) physical medium and it transmits if the channel is sensed idle for this period. Otherwise, it persists to monitor the channel until it is sensed idle for a DIFS. Then, it generates a random backoff interval before transmitting (we refer to this interval as bi-backoff interval) that is multiplied by slot times of 20μs. The minimum value for bi is 32 and the maximum one is 1024 (bi is doubled up to 1024 after each unsuccessful transmission for the packet).
- The backoff time counter is decremented as long as the channel is sensed idle, frozen when a transmission is detected, and reactivated when the channel is sensed idle again for more than a DIFS. The computer transmits when bi reaches zero.
- At the end of data transmission it waits for a SIFS (Short IFS) period plus the propagation delay to receive an acknowledgement MAC frame (ACK).

Fig. 4.a shows an example of three PC contending to transmit to MN. The master transmitted before t_0. We suppose that the value of bi will not increase, and there are not other computers contending. Taking into account these considerations, the delay of transmission can be computed as Eq. 1:

$$delay = 2 * n * (DT) + n * bi^{MN} + max(bi^{PC}) \quad (1)$$

Where

- $DT = DIFS + DATA + SIFS + ACK$
- bi^{MN} is the bi for the MN
- $max(bi^{PC})$ is the maximum value for all the bi of PC

The structure and size of the MAC frame used in our mechanism is shown in Fig. 4.b. In Fig. 5 we show the delay for different number of PC communicating at 11

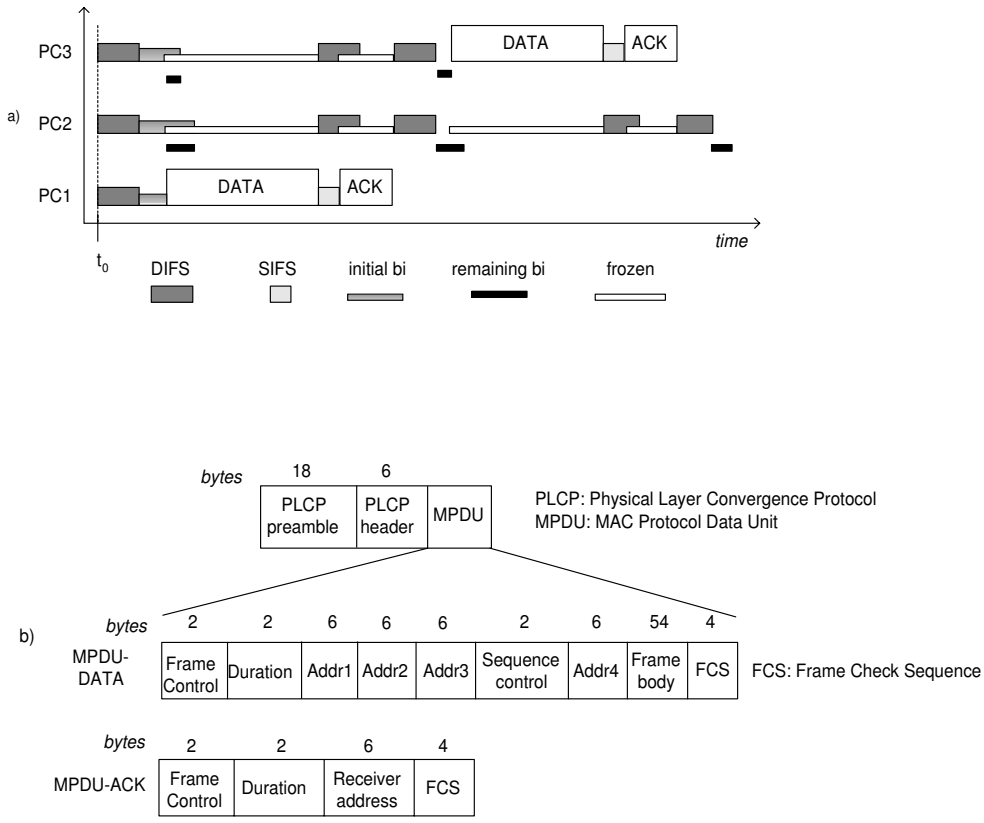


Figure 4: Scalability: a) contention for three PC b) MAC frame format

Mbps and $bi = 64$. The obtained values are slightly lower that the experimental results shown in Fig. 3 for $n = 1, 2, 3, 4$ (socket based implementation with Successful Wait). It is due to we have abstracted some characteristics such as collisions, thread scheduling, message buffering, and so on. But the results show a linear behavior of the delay with n.

It is important to observe the low overhead of our mechanism, so we can assent it is appropriated to be used in a wide range of applications. In the next section we show an efficient implementation of a numerical application that follows the scheme of Fig. 1 and that requires high performance.

4. Application example

MPI [7] is an standard middleware widely used by programmers to implement efficient and portable parallel and distributed applications. With MPI, the default behavior in case of a process, node or network failure is the immediate termination of the application (abrupt ending). We pretend to continue the computation to its ending integrating the above mechanism with the MPI parallel program. For the sake of brevity, we do not explain in this paper how this integration has been made and we only present some experimental results of a master/slave MPI parallel program that finds the global minima of a given function defined in a multidimensional space (this kind of problem belongs to one of the most interesting research areas in parallel nonlinear programming). Our LAMGAC [11] middleware provides, among other functionalities, the integration of the fault detection mechanism with MPI parallel programs.

The characteristics of computers used in the experiments are presented in Table 1. All the machines run under LINUX operating system. The wireless cards complies with IEEE 802.11b standard (11 Mbps) and LAN speed is 10 Mbps.

For all experiments shown in Fig. 6.a we assume a null user load and the network load is the application's one. The experiments were repeated 10 times obtaining a low standard deviation.

The sequential execution time is 124′18" on the slowest computer and 55′54" on the fastest computer. The stopping criteria is less strict than for the parallel program because of the convergence is too slow in the sequential program, obtaining less accurate results.

For the configurations of computers presented in Fig. 6.b, we measured the execution times for the MPI parallel program (values labelled as A in Fig. 6.a) and for the equivalent LAMGAC parallel program without the integration with the wireless channel detection mechanism (values labelled as B in Fig. 6.a). To make com-

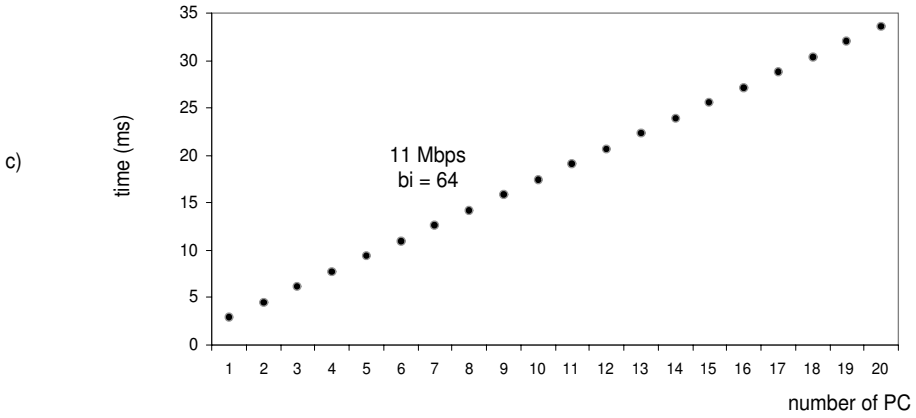


Figure 5: Elapsed time for different number of PC

parisons we do not consider neither input nor output of PC. As you can see, A and B results are similar because LAMGAC middleware does not introduce overhead. The experimental results for the parallel program with the integration of the mechanism are labelled as C and D in Fig. 6.a. The execution time is slightly higher for the C experiment compared to A and B results because of the overhead of the LAMGAC fault detection function we name LAMGAC_Fault_detection and the conditional statements added to the parallel program to consider abrupt outputs of PC. This function implements our novel mechanism and it is invoked before synchronous send and blocking receive communication primitives. In this experimental result we did not consider the abrupt outputs of PC because we just only want to test the overhead of LAMGAC_Fault_detection function and the conditional statements added to the parallel program to consider abrupt outputs of PC. Values labelled as D in Fig. 6.a represent the execution times only for the LAMGAC parallel program when the mechanism of fault detection is considered (this mechanism is called 7 times, once per iteration) and abrupt outputs of PC. We experimented with abrupt output of PC during the step 4 so the master process must restructure the computations before starting the step 5. For all the experiments, the parallel programs reduce the sequential execution times and the global minima found are better.

	Characteristics
Master Node	300 MHz dual Pentium II
LAN	FC1: SMP with four 450 MHz Pentium II FC2: 667 MHz Pentium III FC3: 667 MHz Pentium III
WLAN	PC1: 667 MHz Pentium III PC2: 550 MHz Pentium III

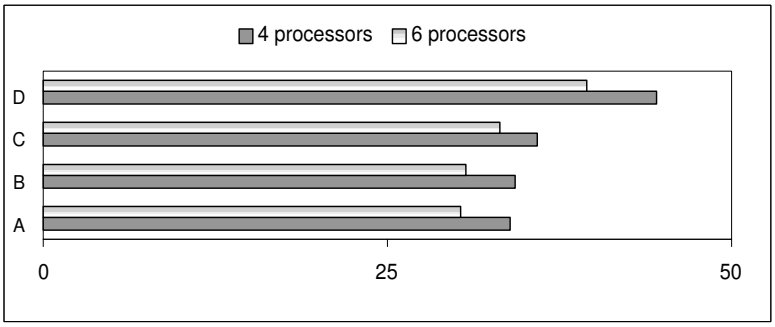
Table 1: Characteristics of the computers

5. Conclusions and future work

In this paper we presented a mechanism to detect the state of the wireless channel so synchronous applications can take corrective action in presence of communication failures between the master process and slave processes on PC. According to the experimental and analytical results, the implementation is appropriate because it introduces low overhead with the same accuracy than our first approach based on sending ICMP frames and waiting for the echo replies. We have already applied the mechanism to a real engineering problem (the unconstrained global optimization for n-dimensional functions) showing good results.

Many things remain to be done. We keep in mind to improve the mechanism to reduce the rate of false alarms, that is to say, the percentage of time the mechanism believes that there is no connection to a portable computer and in fact there is, and viceversa. To prevent these false alarms, the mechanism should, for example, combine other metrics, such as signal and noise strength, before taking a decision.

a)



b)

Parallel program	4 processors	6 processors
A: MPI program	MN,FC1(1cpu),FC2,FC3	MN,FC1(3 cpus),FC2,FC3
B: LAMGAC program	MN,FC1(1cpu),FC2,FC3	MN,FC1(2 cpus),FC3,PC1,PC2
C: LAMGAC-Fault program (with wireless failures detection)	MN,FC1(1cpu),FC2,PC1	MN,FC1(2 cpus),FC3,PC1,PC2
D: LAMGAC-Fault program (abrupt output of PC)	MN,FC1(1cpu),FC2,PC1	MN,FC1(2 cpus),FC3,PC1,PC2

Figure 6: Experimental results for the optimization problem: a) execution times (in minutes) for different configurations and parallel solutions b) details about the implemented parallel programs and the computers used to make measurements

REFERENCES

[1] F. André and E. Saint Pol. "A Middleware for Transactional Hospital Applications on Local Wireless Networks". In *The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 4, Las Vegas, Nevada, June 2000. CSREA.

[2] G. Bianchi. "Performance Analysis of the IEEE 802.11 Distributed Coordination Function". *IEEE Journal on Selected Areas in Communications*, 18:535–547, 2000.

[3] D. E. Comer. *Internetworking With TCP/IP: Principles Protocols, and Architecture*, volume 1. Prentice Hall, fourth edition, 2000. Chapter 9.

[4] A. Dornan. *"The Essential Guide to Wireless Communications Applications: From Cellular to WiFi"*. Prentice Hall PTR, 2002.

[5] H. Wu et al. "Performance of Reliable Transport Protocol over IEEE 802.11 Wireless LAN: Analysis and Enhancement". In *The 21 Annual Joint Conference IEEE Computer and Communication Societies*, pages 599–607, 2002.

[6] M. S. Gast. *"802.11 Wireless Networks: The Definitive Guide"*. O'reilly and Associates, Inc., Sebastopol, CA, 2002.

[7] W. Gropp, E. Lusk, and R. Thakur. *"Using MPI-2: Advanced Features of the Message-passing Interface"*. The MIT Press, Cambridge, MA, 1999.

[8] G. Huston. "TCP in a Wireless World". *IEEE Internet Computing*, 5(2):82–84, March/April 2001.

[9] N. Leavitt. "Will Wireless Gaming Be a Winner?". *IEEE Computer Magazine*, 36:24–27, 2003.

[10] E. M. Macías and A. Suárez. "A Mechanism to Detect Wireless Network Failures for MPI Programs". In P. Kacsuk, D. Kranzlmuller, Z. Németh, and J. Volkert, editors, *Distributed and Parallel Systems. Cluster and Grid Computing*, SECS 706, pages 211–218. Kluwer Academic Publishers, 2002.

[11] E. M. Macías, A. Suárez, C. N. Ojeda-Guerra, and E. Robayna. "Programming Parallel Applications with LAMGAC in a LAN-WLAN Environment". In Y. Cotronis and J. Dongarra, editors, 8^{th} *European PVM/MPI Users Group Meeting*, volume 2131, pages 158–165, Santorini, Greece, September 2001. LNCS, Springer Verlag.

[12] G. Mark. “Extreme Collaboration”. *Communications of the ACM*, 45:89–93, 2002.

[13] S. J. Vaughan-Nichols. “Bull Market for IEEE 802.11 WLAN Chipsets”. *IEEE Computer Magazine*, 35:17–19, 2002.

[14] V. C. Zandy and B. P. Miller. “Reliable Network Connections”. In *8^{th} Annual International Conference on Mobile Computing and Networking*, pages 95–106, Atlanta, USA, 2002.