An Efficient Scheduling Mechanism for Simulating Concurrent Events in Wireless Communications Based on an Improved Priority Queue (PQ) TDM Layered Multi-Threading Approach

P.M.PAPAZOGLOU[1,3], D.A.KARRAS[2], R.C.PAPADEMETRIOU[3]
[1] Department of Informatics & Computer Technology
Lamia Institute of Technology, Greece
papaz@teilam.gr
[2] Department of Automation Engineering
Chalkis Institute of Technology, Greece
dakarras@teihal.gr, dakarras@ieee.org, dakarras@usa.net
[3] Department of Electronic & Computer Engineering
University of Portsmouth, United Kingdom

Abstract - The physical activities of a real wireless network are represented by events which are the main components of a discrete event simulation (DES) system and are produced by its event generator during simulation time. Each network service (e.g. voice, data and video) constitutes an event for a particular mobile user. A critical component within the simulation system, called scheduler, runs by selecting the next earliest event, executing it till completion, and returning to execute the next event. Calendar queue is the state of the art implementation of the scheduler among the most popular networking simulation tools such as ns-2. However, Calendar queue time-stamping mechanism presents drawbacks in the case of complex dynamical systems, like wireless networks, where probability of events concurrency is large. In such a case sequential time-stamping of calendar queue scheduling does not reflect real network events occurrence and generation. It should be remarked that there are very few reports if any in the literature concerning research on events scheduling mechanisms in such real time systems. On the other hand, multi-threading technology offers advanced capabilities for modelling concurrent events. The main goal of this paper is to illustrate that multithreading architectures provide the means for designing efficient schedulers in the simulation of wireless networks resource allocation but, also, several critical issues such as deadlocks, synchronization and scheduling must be effectively faced. In this paper, a stable simulation model is presented based on a novel layered thread architecture and on an alternative network event scheduling mechanism, called the Priority Queue (PQ) – Time Division Multiplexing (TDM) Layered Multithreading mechanism, which supports concurrent events as compared to the state of the art approach which supports only sequential events. Moreover, specific drawbacks of the JVM multi-threading platform such as thread execution unpredictability are also faced and presented.

Key-words: - cellular network, simulation, event scheduling, concurrent events, multi-threading

1 Introduction

1.1 Discrete Event Simulation (DES) in Wireless Communication Systems (WCS)

Discrete Event Simulation [1-10] represents the most known simulation methodology, especially for communication systems. According to the DES concept, events are happening at discrete points in time within the simulation time. Simulation time is moving forward based on the event sequence. These events represent the basic physical network activities such as new call admission, etc. Each event is generated with a unique time stamp that is used for the event execution at a later time. The event processing over simulation time is defined by a scheduler that selects events with the minimum time stamp (maximum priority) [11-15]. Thus, the whole scheduling procedure is based on a priority queue. DES systems can be categorized as sequential or parallel. Sequential DES systems are the most popular among the scientific community. In such systems, the scheduling mechanism can be analyzed in a three step cycle:

- Dequeue: Removal of an event with the minimum time stamp from the queue
- Execute: processing of the dequeued event
- Enqueue: Insertion of a new generated event in the queue

NS-2 [16], which is adopted by the 44.4% of the scientific community [17], uses this scheduling

mechanism for event execution. In ns-2 [16], the next earliest event is selected by the scheduler, executing it till completion, and returning to execute the next event (highest priority or lower time stamp among the remaining events). Only one event can be executed at any given time in ns-2 [16], and so it is a single-threaded simulator. If two or more events are scheduled to take place (to be executed) at the "same time", their execution is performed on a first scheduled – first dispatched manner (based on the time stamp of each event) and so the adaptability of the simulation model to the real network behaviour (physical activities-events) is strongly based on the type of the scheduler.

1.2 Event Scheduling

1.2.1 Why Event Scheduling

The physical activities of a real wireless network are represented by events which are the main components of a DES system. Regarding the above mentioned network services, each service constitutes an event for a particular Mobile User (MU). An event generator produces events (e.g. new voice calls) during simulation time. The scheduling mechanism constitutes a model for the event service occurrence sequence within the real wireless network. If an event is not executed, it remains in the pending event set (PES). PES, is the set of all events generated during simulation time that have not been simulated (processed) yet [4,18,19]. Thus, PES corresponds to a priority queue which controls the flow of event simulation based on current minimum time stamp (highest priority) [4]. The selected scheduling method determines how realistically the occurred real network activities will be reflected in the simulation model. In other words, scheduling is a mapping method of the real network events (activities) within the simulation time of the DES system.

1.2.2 The State of the Art Event Scheduling Mechanism

The major application of priority queues is the implementation of PES inside DES systems [20-23]. The CQ concept was first introduced by [24], was analysed by [19] and until today constitutes the most popular scheduling mechanism among DES systems such as ns-2 (Berkeley, USA)[16], Ptolemy II (Berkeley, USA) [14], Jist (Cornel University, USA) [25]. Several other variations of the CQ for improving performance of the queue itself (e.g. by optimal resizing of the queue) have been proposed in the literature such as DSplay [12], MList [4], Markov hold model [13], and SNOOPY CQ [18]. The idea of the CQ is derived from the ordinary desk calendar with one page for each day. Every event is scheduled on the appropriate page. The scheduled time of each event defines its priority. When an event is enqueued on the calendar, then this event is scheduled for future execution. The earliest event on the calendar is dequeued by searching the page for today's date and removing the earliest event written on that page [24].

Inside the computer, a CQ consists of an array of lists. Each list contains future events. According to CQ principle, the large list of *N* events is partitioned to *M* shorter lists called Buckets. Each bucket is associated with a specific range of time corresponding to future events. Any event with the occurrence time *t(e)* is associated with the *m-th* bucket in year y (y =0, 1, 2, . . .) if and only if

$$t(e) \in \left[\left((yM + m)\delta \right), \left(yM + m + 1 \right)\delta \right] \qquad \text{(a)}$$

In order to find the bucket number *m(e)* where an event *e* will occur at time *t(e)* the following type is used :

$$m(e) = \left\lfloor \frac{t(e)}{\delta} \right\rfloor \bmod M \qquad \text{(b)}$$

Assume that M=8, N=10, δ=1 and t(e)=3.52 (fig. 1) for a new event *e*.

Using the equation (b), the bucket number for event *e* is m(e)=3.

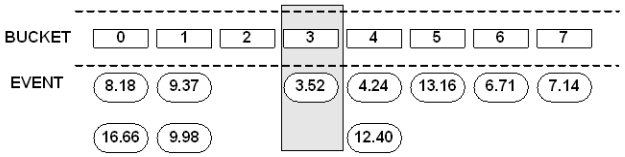

Fig. 1 Instance of a CQ with 8 buckets

The existing state of the art event scheduling strategy (CQ) suffers in supporting concurrent network events as they happen in a real wireless network and thus an improved event scheduling mechanism implemented through multi-threading technology is proposed.

An analytical study of the CQ can be found in [26].

1.3 Multi-Threading Technology

1.3.1 Multi-Threading as a Necessity

Multi Tasking is a prerequisite for today's software development. All modern operating systems (OSs) support this feature which enables different programs to run at the "same time". In an ideal environment, the best performance can be achieved by using one processor for each running program, but in real

computers there is usually only one processor available. Due to that fact, the processor is shared between the applications. Even if there are two or three processors available, the processes outnumber the processors. Pieces of the same program can run concurrently as two or more different programs do. This capability is called threading. Most of the languages that support multi-threading (MT), are OS-specific. Thus, no portability to different OS platforms is achieved.

MT technology is a well known general development technology. This technology can be applied in areas where concurrent events (tasks) exist. Existing programming languages support MT but on the other hand, significant drawbacks such as deadlocks and synchronization must be faced effectively. Java supports native MT and constitutes the most known development tool among the scientific community.

1.3.2 The JVM Platform

The Java implementation can be viewed as an example; the developer may use another technology for the implementation of the MT concept.

The MT operation in the case of Java is exclusive responsibility of the Java Virtual Machine (JVM). Thus, any Java code can be executed directly in any platform. Threading methodology can be applied when concurrency is needed. MT is a very efficient methodology for a number of scientific tasks such as simulations of real phenomena where many processes occur at the same time (simultaneously). In the context of the conceptual design of a program, this approach can be quite useful even if it runs on a single-processor machine. Thus, more reliable modelling and simulation of a real system can be achieved. ***When the code interleaving and the execution sequence of the active threads inside the processor affects the system behaviour and results, only controlled scheduling by the designer must be used.*** More information about the threading capabilities of Java can be found in [27-30].

A variety of methods are available for changing the thread status within an application. Using these methods, efficient synchronization control between threads can be achieved. The start() method activates the selected thread that is running concurrently with other threads.

A thread is terminated when the run() method returns or when the stop() method is activated. Figure 2 shows all the possible thread status and the corresponding status transitions.

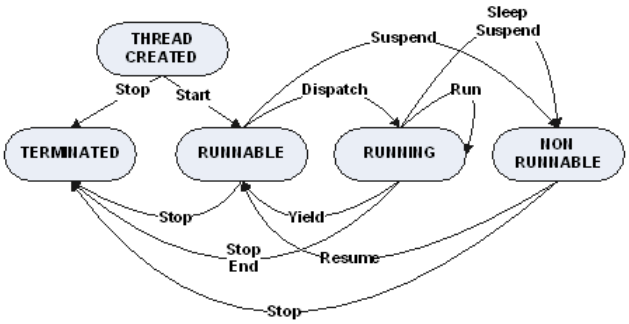

Fig. 2 Thread status transitions

An internal mechanism of JVM, called scheduler, defines the real-time order of thread execution. Scheduling can be controlled by the programmer and is categorized as follows:

- Non pre-emptive
- Pre-emptive

In non pre-emptive scheduling, the scheduler runs the current thread forever and requires from this thread to tell explicitly if it is safe to start another thread. In pre-emptive scheduling, the scheduler runs a thread for a specific time-slice (usually a tiny period within a second) and then "pre-empts" it, (calling suspend()), and resumes another thread for the next time-slice. The non-pre-emptive scheduling can be very useful especially in time critical (e.g. real time) applications when the interruption of thread execution can happen in the wrong moment. Modern schedulers are usually pre-emptive, therefore, the development of MT applications is easier. JVM uses priorities for scheduling threads. Initially, it gives equal priorities to all threads.

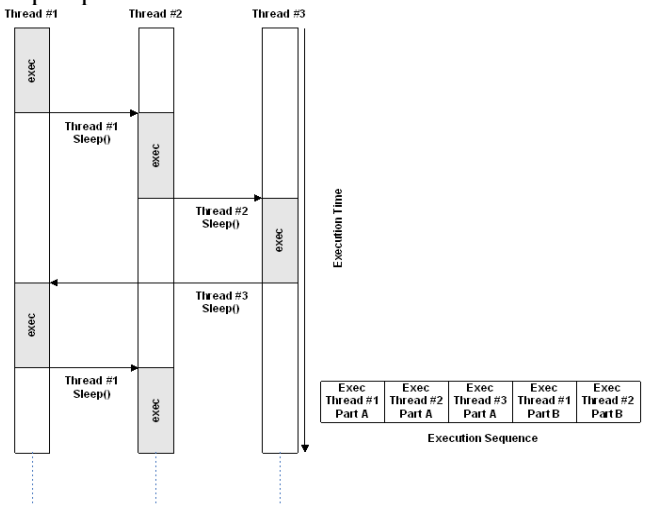

Fig. 3 Three threads share the same processor (sample scenario)

A major drawback of JVM is that the behaviour of its scheduler can not be predicted [31]. Actually, JVM specifications have not

been designed by taking into account event scheduling constraints.

Figure 3 illustrates three threads sharing the same processor. Note that there is no real parallel execution between threads but only high speed switching between them. The sleep() method deactivates temporarily the current thread in order to give time for the next thread execution. If the OS uses pre-emptive multi tasking, the sleep() method is not needed for switching between threads.

2 Proposed WCS Simulation model

2.1 Services Oriented Modelling

The effectiveness and functionality of a wireless network are based on the offered services. The services type combined with the offered communication quality constitute the most critical issues. For the above reasons, the evaluation of a wireless network is performed through the performance evaluation of the offered services using specific statistical metrics.

2.1.1 New Call Arrival (NC)

The number or MUs is large, the calls by each MU are limited and so the call arrivals can be assumed as random and independent. In the simulation program, the new calls result from a random or a Poisson distribution according to a predefined daily model. In the case of multimedia services, multiple channels are allocated.

2.1.2 Call Termination (FC)

The program uses an exponential function that generates the call duration for each new MU. The call holding time is added to current simulation time for later examination. The associated procedure examines the progressive call time for connected MUs. If this time has expired the User Registry (UR) is updated with the new connection status of the particular MU. In the case of data transfer, a call is terminated only when the transfer is completed.

2.1.3 Reallocation Check (RC)

The computations are based on the signal strength and the way it is affected by other connected MUs in neighbour cells. If a MU signal does not fulfil the Carrier to Noise plus Interference ratio (CNIR) threshold the procedure tries to find another appropriate channel. At first, the algorithm calculates the signal strength between MU and base station and later on it calculates any interference coming from other connected MUs. If an accepted channel is found, it is allocated to the new MU, otherwise the call is dropped. In the case of multimedia services, partial channel reallocation is also supported.

2.1.4 Mobile User Movement (MC)

The algorithm locates the connected MUs and changes their current positions. A MU movement is generated based on Gaussian distribution. This distribution is also used in similar simulation models found in the literature [32].

2.2 The Proposed Priority Queue (PQ) –TDM Layered Multithreading Event Scheduling Algorithm for Complex Real Time Simulation Models

The proposed PQ-TDM Layered Multithreading algorithm extends the CQ paradigm to handle concurrent events in simulation models of complex systems, where event occurrence time cannot be specified as easily as in the CQ described simulation systems. The basic entity of the algorithm, the event, is specified as a thread. Besides, the basic new event entity, instead of occurrence time t(e), is its priority p(e). The basic architecture of the algorithm consists, like CQ, of an array of lists. Each list contains future events with their own priorities. According to the PQ-TDM principle, the list of *N* concurrent/non-concurrent events taking place within the system is partitioned to shorter lists called Priority Buckets. Moreover, there exists a basic periodic event-thread, the Time Clock, which synchronizes all other events-threads. This event-thread is associated with the largest Priority P_{MAX}. Provided that the Multithreading Simulation Platform supports P priorities, let's D_P be the priority distance between P_{MAX} priority of the Time Clock event-thread and the priority associated with the priority bucket having the largest priority. If we assume that the priorities supported are 1,2,…P, then P_{MAX}= P, and P_{MAX} - D_P are the supported buckets in which the event list is partitioned. Each such bucket, on the other hand, is associated with a specific range of priorities corresponding to future events. Any event with the occurrence priority *p(e)* is associated with the *m-th* bucket in Basic Priority p (p =0,1,2, ..) if and only if

$$p(e) \in [(p(P_{MAX} - D_P) + m)\delta, (p(P_{MAX} - D_P) + m + 1)\delta] \quad (1)$$

In order to find the bucket number *m(e)* where an event *e* belongs with priority *p(e)* the following formula is introduced:

$$m(e) = \left| \frac{p(e)}{\delta} \right| \bmod (P_{MAX} - D_P) \quad (2)$$

Regarding time *t(e)* of the event-thread e, it should be remarked that now it is determined by the Multithreading Simulation Platform by a Time

Division Multiplexing (TDM) procedure. Time slice ΔT is the basic entity in this TDM procedure. That is, ΔT computational time is given to each event out of the events list to proceed its computations, within which it might finish or not. If it doesn't finish then, it waits for a future assignment of a ΔT computational time again. Events with higher priority are assigned with more such ΔT time slices. An important aspect in the proposed scheduling algorithm is that D_P should be as maximum as possible in order for the Time-Clock Thread to be a reliable controller of the multithreading architecture of events-threads and face the known difficulties of multithreading technology to reliably schedule threads, due to absence of such specifications and definitions in Multithreading Simulation Platforms like JVM etc. Therefore, the proposed PQ-TDM Layered Multithreading algorithm reliably extends CQ capabilities in order to handle events concurrency in complex simulation systems like WCS, at least theoretically. It is the purpose of this paper to show that such a proposed algorithm leads to efficient scheduling of events and thus, to efficient simulation modelling of WCS.

2.3 Implementation Issues Based on the JVM Specification

2.3.1 Synchronization and Deadlocks

The proposed simulation system uses a User Registry (UR) for keeping a detailed record of each connected MU. Due to that fact, the UR constitutes a shared resource area. When concurrent events take place, two or more threads try to access the UR at the "same time". An active thread can be pre-empted by the scheduler, when an access activity in a shared resource is not completed. While this thread is now pre-empted, another thread tries also to access the shared resource. Due to the given time slice to that thread by the scheduler, the access activity is completed. After the re-activation of the first thread, the semi-completed access activity of the first phase is now complete. If the above two threads try to access the same record in the UR, then, the resulting data are incorrect depending on the thread switching. For the above reasons, the UR of the simulation system must be accessed through the synchronized method. This synchronization will prevent the shared resource area from simultaneous access by two or more threads. When a thread has locked an object (e.g. access method for UR), and is waiting for another thread to finish, while that other thread is waiting for the first thread to release that same object before it can finish then a deadlock occurs.

In most cases, when two or more threads try to access the shared resource area, this is not for the same MU. Thus, the synchronization mechanism is necessary only in some cases. In order to remove any deadlock possibility, the following features must be designed and developed:

- Controlled thread switching (execution)
- Control points where the UR will be accessed
- Conditional synchronization (synchro-nization only if needed)

Every time a thread tries to access UR for a specific MU, flag values are written in a local table that belongs to that thread. After the update of the local table, the thread scans all the local tables and decides to access the main UR through non-synchronized methods; otherwise, the access is completed through synchronized methods. Using this mechanism, high speed and data manipulation is achieved with no deadlocks.

2.3.2 JVM Scheduler Unpredictability

As mentioned previously, the JVM scheduler behaviour can not be predicted [31] and so a more controlled scheduling mechanism must be applied. In other words, the execution sequence of the threads can not be guaranteed across all Java virtual machines [27]. The event execution sequence plays a major role for the network performance due to the fact that the events represent the modelled network services.

2.4 Proposed Event Scheduling Mechanism

The proposed scheduling mechanism supports concurrent network events instead of the state of the art approach (CQ) where only sequential events are supported based on the above defined PQ-TDM Layered Multithreading algorithm. Moreover, the proposed mechanism has been implemented through a multi-threading platform where improvements took place in order to face the default JVM unpredictability in thread execution sequence.

The basic goals of the proposed event scheduling mechanism are:

- The control of thread execution sequence.
- The control of thread core code execution time.

The thread execution sequence and activation is controlled by a clock inside a thread (super thread, the Time Clock thread of the PQ-TDM Layered Multithreading algorithm) with maximum priority instead of minimum priorities which are applied in the rest of the threads, as discussed in the PQ-TDM

Layered Multithreading algorithm. Figure 4 shows the pseudo code of clock implementation.

```
Thread CLOCK
{
  Main action
   {
     While (simulation time step)
         {
          Thread#1_active=1;
          Sleep(Thread#1_limit);
          Thread#2_active=0;
          …
          …
          Thread#n_active=1;
          Sleep(Thread#2_limit);
          Thread#n_active=0;
         }
   }
}
```

Fig. 4 Super Thread (Clock) implementation

Figure 5 shows the thread implementation.

```
Thread #1
{
  Main action
    {
      While (simulation time step)
         {
           If (Thread#1_active==1)
                 {
                   //core code
                 }
         }
    }
}
```

Fig. 5 Thread implementation

Using the Thread.start() method, all the threads become active but the core code of the threads (except clock) is disabled while Thread#n_active=0. The thread CLOCK is under execution in most of the times because its priority has the maximum value as compared to the rest of the threads. Thus, the clock (super thread) synchronizes the thread core code activation and execution. Initially, the core code of thread#1 is activated. After Thread#1_limit time the core code is deactivated. Thus, the rest of the threads (core codes) are executed in a controlled order and for a specific time period.

2.5 Experimental Models

A set of multi-threaded model variations has been implemented in order to show:

- The model instability based on default JVM when different numbers of computational threads are used.
- How thread execution sequence and core code activation time can be controlled based on a complementary to JVM scheduler, the PQ-TDM Layered Multithreading proposed above.

For achieving the above goals, two sets of implementation model variations have been tested. Tables 1 and 2 show the corresponding implementation parameters.

Model	No of threads	Thread type	Additional computational time
A1	4	basic	-
A2	8	4 basic 4 dummy	-
A3	16	4 basic 12 dummy	-
A4	4	basic	Loop in RC
A5	4	basic	Conditional loop in RC
A6	8	4 basic 4 extra	Loop in 4 extra
A7	8	4 basic 2 dummy 2 extra	Loop in 2 extra
A8	6	4 basic 2 extra	Loop in 2 extra

Table 1. Computational thread variations

Model	Thread	Pri	Sleep	core code exec	core code active time	exec seq
B1	Net	5	-	Def JVM	Sym	Unpr
B2	Net	10	Sym	Clk	Sym	Pred
	Clk	1	Sym	Clk	Sym	Pred
B3	Net	1	Sym	Clk	Sym	Pred
	Clk	10	Sym	Clk	Sym	Pred
B4	Net	1	Asym	Clk	Asym	Pred
	Clk	10	Asym	Clk	Asym	Pred

Table 2. Thread scheduling variations
(Pri=Priority, Net=Network threads, Clk=Clock thread, Sym=Symmetrical, Asym=Asymmetric, Unpr= Unpredictable, Pred=Predictable)

3 Network Model Validation

The implemented algorithms and models in this paper have been tested in a simulation environment that integrates the basic simulation and network components. The necessary validation of the simulation environment consists of three different validation levels which are:

- Calendar Queue (State of the art) scheduling mechanism implementation algorithm
- Network environment which includes signal propagation, interference and signal measurements
- Network performance compared to theoretical computations

The most known statistical metrics for the wireless network performance evaluation are blocking and dropping probabilities [33-41]. Blocking probability represents the blocked calls, while dropping represents the unsuccessful channel reallocation for an ongoing call. The dropping probability is strongly connected to R_{cni} (eq. 3), because when this ratio is not above the accepted threshold and the network can not allocate an appropriate channel, the call is dropped. On the other hand, blocking probability can be theoretically calculated. If the received power of each MU is high enough, it is assumed that the interference from other MUs can be ignored. The theoretical formula known as Engset's formula [42] is as follows:

$$P_{blocking\_theoretical} = \frac{\binom{n-1}{s}(vh)^s}{\sum_{i=0}^{s}\binom{n-1}{i}(vh)^i} \qquad (3)$$

where **n** is the number of users, **s** is the number of channels, **v** is the average call arrival rate (for no connected MU) and **h** is the average call holding time.

Equation 3 shows only the basic relation between channels and users, and does not take into consideration critical factors that affect the blocking probability such as traffic conditions, service type, channel allocation strategy, etc.

Figure 6 shows the theoretical blocking probability that is derived from eq. (3) as compared to the simulated blocking probability, derived in the proposed WCS simulation system. The simulated probability has been generated from the large scale network based on the classical DCA algorithm [41] and the above reported network services.

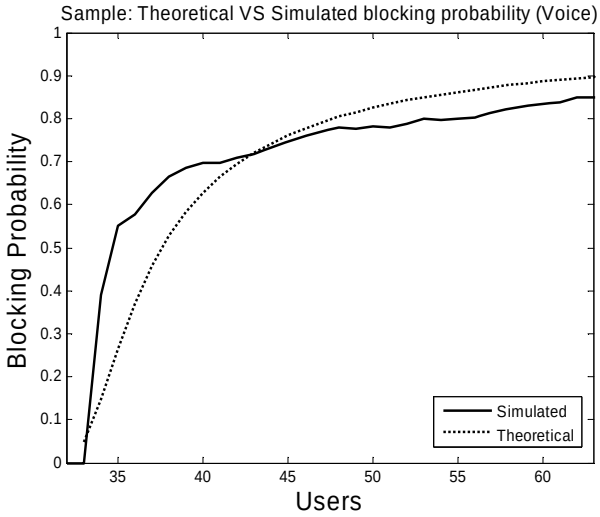

Fig. 6 Theoretical blocking probability versus simulated.

4 Experimental Results

Figures 7 through 11 contain sample executions that show the thread execution sequence and core code activation based on the tested models. The numbers 1 through 4 inside figures 7 to 11 represent the network services NC, RC, MC and FC respectively. Figures 7 and 8, show that the thread execution sequence is different in every simulation execution and also, that threads do not have constant execution periodicity.

```
33333333333333333333333331421423142314231423142314
23142314231423142314231423142314231431431231243
24324324324324324324324324324324324324324324324
31431241241241241241241241241241241241241241241
24124124324312312312312312312312312312312312312
31231231231231241243143143143143143143143143143
14314314314314314314312312432 4
```

Fig. 7 Thread execution #1,
Default JVM, equal priorities

```
32323232323232323232323412121212121212121212121
21212121243232341313131313131313131313131421212
12121243232323232323232323232323413131313131421
21212121212121212121212432323232323413131313131
31313131313131421212121212432323232323232323232
32323413131313131421212121212121212121212124323
23232323413213213213213213213 2
```

Fig. 8 Thread execution #2,
Default JVM, equal priorities

```
111111……111111{NC:1031,332}
{RC:1031,0}
{MC:1031,0}
{FC:1031,0}
---------------------------------
111111……111111{NC:1235,14575}
{RC:1031,0}
{MC:1031,0}
```

```
{FC:1031,0}
---------------------------------
111111……111111{NC:1219,21591}
222222……222222{RC:1093,17704}
333333……333333{MC:1000,21929}
444444……444444{FC:1188,15734}
```

Fig. 9 Thread execution, Symmetrical Sleep, Clock priority=1, Network thread priorities=10

Figure 9 shows that some threads are not executed due to the fact that the clock is rarely activated by the default scheduler when minimum priority is applied and therefore, the required thread control is not possible. Moreover, the thread activation time leads to different number of core code execution times for each thread (e.g. NC:1219=real thread core code active time in ms, 21591 executions, RC:1093=real thread core code active time in ms and 17704 executions)

```
111111……111111{NC:1000,70}
222222……222222{RC:1000,72}
333333……333333{MC:1000,72}
444444……444444{FC:1000,72}
---------------------------------
111111……111111{NC:1000,72}
222222……222222{RC:1000,72}
333333……333333{MC:1000,72}
444444……444444{FC:1000,72}
---------------------------------
111111……111111{NC:1000,72}
222222……222222{RC:1000,72}
333333……333333{MC:1000,72}
444444……444444{FC:1000,72}
```

Fig. 10 Thread execution, Symmetrical Sleep, Clock priority=10 (super thread), Network thread priorities=1, Dp =9

Figure 10 shows that the thread core code active time and also, the number and sequence of executions can be controlled based on the proposed PQ-TDM layered multithreading approach.

```
111111……111111{NC:500,34}
222222……22222222222{RC:1000,72}
333333……3333333{MC:500,36}
444444……44444444444{FC:1000,72}
---------------------------------
111111……11111111{NC:500,36}
222222……22222222222{RC:1000,72}
333333……33333333{MC:500,36}
444444……44444444444{FC:1000,72}
---------------------------------
111111……11111111{NC:500,36}
222222……22222222222{RC:1000,72}
333333……33333333{MC:500,36}
444444……44444444444{FC:1000,72}
```

Fig. 11 Thread execution, Asymmetric Sleep, Clock priority=10 (super thread), Network thread priorities=1, Dp =9

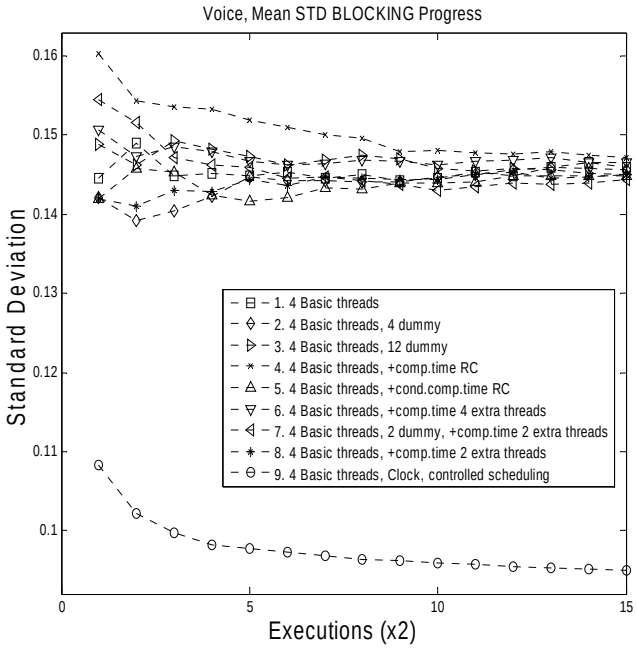

Fig. 12 Standard deviation of blocking probability for different multi-threaded implementation models

Figure 12 illustrates that the proposed PQ-TDM Layered Multithreading scheduling mechanism improves simulation model stability in the resource allocation (channel assignment) problem for voice services in a cellular network with 100 cells, 50 users maximum per cell and 32 available channels per cell. This is shown by examining standard deviation of blocking probabilities (above discussed) over simulation time in 30 Monte Carlo executions of the wireless network simulation model. The channel assignment algorithm used in these experiments is the classical DCA one [41, 42]

5 Conclusions and Future Trends

In this paper, an efficient event scheduling mechanism, which supports wireless network concurrent events occurrence is presented, based on a novel extension of the classical CQ algorithm to handle concurrent events. The proposed PQ-TDM layered multithreading scheduling algorithm based on TDM principles is suitable for Multithreading Architectures, exploiting concurrent execution capabilities of threads. Moreover, the implementation issues concerning the default features of the JVM multi-threading platform are also presented. The service sequence and interleaved execution of the network events are the most critical modelling issues that affect the network performance in a dynamically changing wireless environment. The scheduling mechanism must guarantee the

efficient processing of user requests according to current user needs and network performance. Our PQ-TDM layered multithreading proposed algorithm controls the thread execution sequence and also, the core code active time. It is shown that the proposed simulation model, by taking into account the above scheduling principles in its design, outperforms all conventional simulation models concerning its stability validated through specific statistical metrics. Such a research opens a new field in wireless networks simulation modelling since scheduling mechanisms and specifically, how they affect designed network performance evaluation, have been so far ignored by the scientific community.

References

[1] Pasquini, R. (1999). *Algorithms for improving the performance of optimistic parallel simulation*. Unpublished Doctoral dissertation, Purdue University.

[2] Brade, D. (2003). *A generalized process for the verification and validation of models and simulation results*. Unpublished Doctoral dissertation, University of Bundeswehr, Munchen.

[3] Barr, R. (2004). *An efficient, unifying approach to simulation using virtual machines*. Cornell University.

[4] Goh, R. S. M., & Thng, I. L.(2003). MLIST: An efficient pending event set structure for discrete event simulation, international journal of simulation. 4(5-6).

[5] Di Caro, G.A. (2003). *Analysis of simulation environments for mobile ad hoc networks*. Technical Report No. IDSIA-24-03. Dalle Molle Institute for Artificial Intelligence.

[6] Schriber, T.J., & Brunner, D.T. (1997). Inside discrete-event simulation software: how it works and why it matters. Proceedings of the 1997 Winter Simulation Conference.

[7] Misra, J. (1986). Distributed Discrete-event Simulation. *ACM Computing Surveys*, *18*(1).

[8] Overeinder, B.J. (2000). *Distributed Event-driven Simulation*. Unpublished Doctoral dissertation, University of Amsterdam.

[9] Preiss, B.R., Loucks, W.M. & Hamacher, V.C. (1988). A unified modeling methodology for performance evaluation of distributed discrete event simulation mechanisms. The 1988 Winter Simulation Conference.

[10] Perumalla, K.S. (2006). Parallel and distributed simulation: traditional techniques and recent advances. Proceedings of the 2006 Winter Simulation Conference.

[11] Lacage, M., & Henderson, T.R. (2006). Yet Another Network Simulator, ACM, Proceeding from the 2006 workshop on ns-2: the IP network simulator.

[12] Siow, R., Goh, M., & Thng, I.L.J. (2004). DSplay: An Efficient Dynamic Priority Queue Structure For Discrete Event Simulation. Simtect Simulation Conference.

[13] Chung, K., Sang, J., & Rego, V.A. (1993). Performance Comparison of Event Calendar Algorithms: an Empirical Approach. *Software—Practice and Experience*, *23*(10), 1107–1138.

[14] Muliadi, L. (1999). Discrete event modeling in ptolemy ii. University of California, Berkeley.

[15] Naoumov, V., & Gross, T. (2003). Simulation of Large Ad Hoc Networks, MSWiM'03, September, San Diego, California, USA.

[16] Fall, K., & Varadhan, K. (2007). *The ns Manual*. UC Berkeley, LBL, USC/ISI, and Xerox PARC, January 14.

[17] Kurkowski, S., Camp, T., & Colagrosso, M. (2005). MANET Simulation Studies: The Current State and New Simulation Tools.

[18] Tan, K.L., & Thng, L.J. (2000). Snoopy Calendar Queue. Proceedings of the 2000 Winter Simulation Conference.

[19] Siangsukone, T., Aswakul, C., & Wuttisittikulkij, L. (2003). Study of Optimised bucket widths in Calendar Queue for Discrete Event Simulator. Thailand's Electrical Engineering Conference (EECON-26).

[20] Blackstone, J.H., Hogg, C.L., & Phillips, D.T. (1981). A two-list synchronization procedure for discrete event simulation. Commun *ACM*, *24*(12), 825-629.

[21] Henriksen, J.O. (1977). An improved events list algorithm. Proceedings of the 1977 Winfer Simulation Conference (Gaithersburg. Md., Dec. 5-7). IEEE, Piscataway, N.J. pp. 547-557.

[22] Kingston, J. H. (1984). *Analysis of Tree algorithms for the simulation event list*. Unpublished Doctoral dissertation, Basser Dept. Computer Science, University of Sydney, Australia.

[23] McCormack, W.M., & Sargent, R.G. (1981). Analysis of future event-set algorithms for discrete event simulation. *Communications of the ACM, 24*(12), 801-812.

[24] Brown, R. (1988). Calendar queues: A fast O (1) priority queue implementation for the simulation event set problem. *Communications of the ACM, 31*(10), 1220–1227.

[25]http://jist.ece.cornell.edu/javadoc/jist/runtime/Scheduler.Calendar.html

[26] Erickson, K.B., Ladner, R.E., & LaMarca, A. (1994). Optimizing Static Calendar Queues. *Annual IEEE Symposium on Foundations of Computer Science*, *35*, 732-743.

[27] Oaks, S., & Wong, H. (2004). Java Threads, O'Reilly, 3rd edition.

[28] Kramer J.M. (2006). *Concurrency: State Models & Java Programs* (2nd Ed.). John Wiley & Sons.

[29] Lindsey, C.S., Tolliver, J.S., & Lindblad, T. (2005). *An Introduction to Scientific and Technical Computing with java*. Cambridge University Press.

[30] Pidd, M., & Cassel, R.A. (1998). Three phase simulation in java. Proceedings of the 1998 Winter Simulation Conference.

[31] Mengistu, D., Lundberg, L. and Davidsson, P. (2007). Performance Prediction of Multi-Agent Based Simulation Applications on the Grid. *international journal of intelligent technology volume 2 number 3 2007 issn 1305-6417*

[32] Tripathi, N.D., Jeffrey, N., Reed, H., & VanLandingham, H.F. (1998). Handoff in Cellular Systems. IEEE Personal Communications.

[33]Bigham, J., & Du, L. (2003). Cooperative Negotiation in a MultiAgent System for RealTime Load Balancing of a Mobile Cellular Network. *AAMAS'03, July*, 14–18.

[34]Cheng, M., Li, Y., & Du, D.Z. (2005). *Combinatorial Optimization in Communication Networks*. Kluwer Academic Publishers.

[35] Cherriman, P., Romiti, F., & Hanzo, L. (1998). Channel Allocation for Third-generation Mobile Radio Systems. *ACTS' 98, 1*, 255-261.

[36]Godara, L.C. (1997). Applications of Antenna Arrays to Mobile Communications, Part I: Performance Improvement, Feasibility, and System Considerations. *Proceedings of the IEEE*, *85*(7).

[37]Grace, D. (1998). *Distributed Dynamic Channel Assignment for the Wireless Environment*. Unpublished Doctoral dissertation, University of York.

[38]Haas, H. (2000). *Interference analysis of and dynamic channel assignment algorithms in TD–CDMA/TDD systems*. Unpublished Doctoral dissertation, University of Edinburg.

[39]Hollos, D., Karl, H., & Wolisz, A. (2004). Regionalizing Global Optimization Algorithms to Improve the Operation of Large Ad Hoc Networks. *Proceedings of the IEEE Wireless Communications and Networking Conference*, Atlanta, Georgia, USA.

[40]Katzela, I., & Naghshineh, M. (1996). Channel assignment schemes for cellular mobile telecommunication systems: A comprehensive survey. *IEEE Personal Communications*, 10–31.

[41]Salgado, H., Sirbu, M., & Peha, J. (1995). Spectrum Sharing Through Dynamic Channel Assignment For Open Access To Personal Communications Services. Proc. of IEEE Intl. Communications Conference (ICC), pp. 417-22.

[42]Lee, W.C.Y. (1995). Mobile Cellular Telecommunications. McGraw-Hill