PARALLEL SIMULATOR OF WIRELESS NETWORKS

BY JIGNESH PANCHAL

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Professor Roy D. Yates

and approved by

____

____

____

____

New Brunswick, New Jersey

October, 1998

ABSTRACT OF THE THESIS

Parallel Simulator of Wireless Networks

by Jignesh Panchal

Thesis Director: Professor Roy D. Yates

This thesis describes the Wireless Propagation and Protocols Evaluation Testbed (WiPPET), a virtual testbed for parallel simulation of wireless networks. Written in the Telecommunications Description language (TeD) developed at Georgia Tech, WiPPET provides a platform to model wireless networks at multiple time scales. The current implementation supports call level and packet level time scale granularity.

Session level WiPPET has radio propagation models for distance loss and shadow fading and radio resource management modules for call admission, uplink power control, and handoff. Six handoff algorithms, including three standards-based algorithms (AMPS, IS-136, GSM) plus three others from the literature, are supported. Session level WiPPET evaluates radio resource management protocols by measuring call blocking, call dropping, average number of handoffs per call, and other statistics at the time scale of a call. The simulator is found to be consistent with results from the earlier WINLAB Mobility And Dynamic Resource Allocation Simulator (MADRAS).

For packet level WiPPET, the propagation model includes distance loss, short scale (Rayleigh) and long scale (shadow) fading. The simulator provides multiple samples of the instantaneous signal to interference ratio (SIR) during each packet. The packet error rate (PER) can be determined from SIR samples and is used to evaluate the performance of data-link layer protocols in interference and fading environments. A current study uses packet level WiPPET to assess PER in the Cellular Digital Packet Data (CDPD) standard.

This thesis discusses two methods to parallelize the interference calculation: geographic parallelization, and radio channel parallelization. These two methods are experimentally evaluated in terms of simulation scalability, efficiency and execution speedup.

Acknowledgements

I would like to thank my advisors Dr. Andrew T. Ogielski and Dr. Roy D. Yates for their patience and encouragement during the work on this thesis. I would also like to extend my gratitude to Dr. Owen E. Kelly for being so supportive and enthusiastic about this project.

I would like to thank Dr. David J. Goodman and Dr. Narayan B. Mandayam for being on my thesis committee. Finally, I would like to thank all members of WiPPET group and my friends at WINLAB for their constant love and support.

Dedication

To my father and mother,

Shanti and Indumati

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Simulations of wireless networks differ considerably from simulations of wired networks in that they have to include modeling of the geographic environment and physical channel properties (fading and interference) both of which have a profound effect on network performance. Network performance estimation typically requires very frequent calculation of received signal quality at each network node. These calculations dominate the total computational effort. Therefore, the goal of efficiency requires that careful consideration be given to the manner in which signal quality is calculated. Beyond efficiency, the other important goal of a simulator must be the ease with which researchers can write simulations. In turn, this translates into clear code organization and code reuse.

A modern radio network seeks to provide a specified quality of service to the maximum number of users by proper radio resource allocation. WINLAB's previous sequential simulation tool, the Mobility And Dynamic Resource Allocation Simulator (MADRAS [2]), has figured importantly in research [3,4,23,24] and has demonstrated the necessity of an integrated approach to resource allocation. It also showed the desirability of the reusable testbed approach.

In order to be useful in multiple research projects, a wireless system simulator should be versatile enough to support modeling of radio phenomena and receiver characteristics at different temporal and spatial scales, as well as accommodate multiple protocols layers, as sketched in Figure 1.1. The levels of granularity are dictated both by the hierarchy of transmission units in the system (e.g. frames, packets, bits or chips), and by characteristic time scales for the physical processes with which transmissions interact (e.g. short-scale Rayleigh fading, long-scale shadow fading). A versatile wireless simulator should support research studies at various protocol levels and time-scale, and more importantly *across* levels and time-scales.

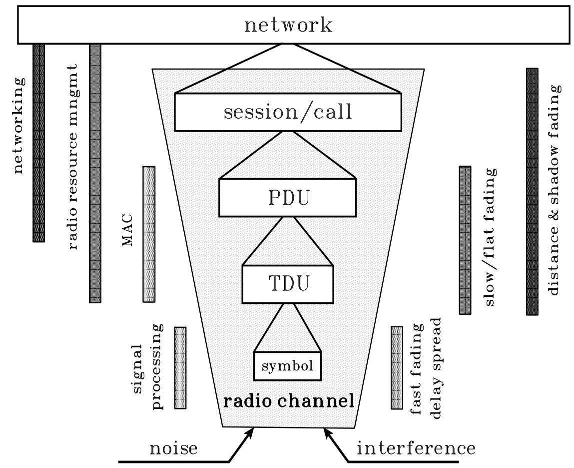


Figure 1.1: Model development at multiple time scales and protocols layers. PDU = Protocols Data Unit, TDU = Transmission Data Unit.

With such degree of complexity, simulations are often the primary tool for analysis of wireless network behavior. Yet, simulations are difficult, large systems are hard to describe in software, and simulations have high demand for computing resources. Separation of modeling from implementation helps to solve both problems. TED was designed with the goal of reducing the time required to construct new simulation models by shifting the work to design models rather than low-level programming of customized simulator software and enabling reuse of a growing database of objects for creation of new simulation models [16, 17].

This thesis gives a new parallel simulator to WINLAB. It represents one of several TED implementation efforts that will determine the success of TED in achieving those goals and thereby influence the evaluation of TED. Part of this thesis discusses results from papers which have appeared as references [14] and [15]. The plan of the thesis is as follows:

- Motivation and simulation objectives (chapter 1 section 1.2).

- Fundamentals of TeD, discrete event simulator and wireless testbed (chapter 2).
- Session level simulation: simulator design and implementation of protocols (chapter 3).
- Verification of session level simulation (chapter 3 section 3.8)
- Packet level simulation (chapter 4).
- Scalability and efficiency of parallel simulator (chapter 5).
- Summary and future work (chapter 6).

1.2 Motivations and Objectives

The earlier WINLAB simulation tool, MADRAS [2], is a customized, sequential discrete event simulator developed in C, a non-object oriented language. It supports only session level traffic (generates calls). It doesn't have modular or object structure so that each modification of MADRAS results in a separate simulator and there is no natural method to merge code variants in order to produce a revision of the program.

We want a general purpose simulator providing an application interface which can be used to design models rather than low-level programming of customized simulator software and enabling reuse of a growing database of objects for creation of new simulation models.

This thesis describes the Wireless Propagation and Protocols Evaluation Testbed (WiPPET), a virtual testbed for parallel simulation of wireless networks, written in the Telecommunications Description language (TeD) developed at Georgia Tech. TeD is a software framework for implementing models in a form suitable for parallel simulation based on the C++ object-oriented programming language [16,17]. TeD significantly increases the simulator usability by separating the domain specific modeling from internal complexity of parallel simulator engines.

In addition, within a research laboratory such as WINLAB where several investigations are in progress, it is highly desirable to have a simulator, which can be used for different protocol layers (data-link, MAC, resource management/network) and at different time-scales (session or call, packet, symbol). These wireless simulations demand significant computational power. WiPPET offers the advantages of parallel processing to help mitigate this grand demand for computation.

WiPPET, a virtual wireless simulation testbed developed, is a collection of base TeD objects modeling, i.e. *radio propagation*, *mobility*, *traffic*, and *protocols* that together with an appropriate model configuration file can be instantiated as a virtual testbed. This testbed can be used by wireless networks researchers and engineers either without modification, or extended with problem specific code that addresses a particular design question.

The objectives of this work are :

- Design and implement a wireless network modeling classes in TeD/C++ for the purpose of developing a general, versatile simulator, that can be quickly modified or extended to support modeling of radio phenomena and receiver characteristic at different temporal and spatial scales.
- Propose different methods of parallelizing the simulation to facilitate load balancing in a multi-processors environment.
- Demonstrate the simulator in selected applications – a) Verification of integrated resource management protocols; b) estimate Packet Error Rate in short-scale fading and interference environment (data-link layer analysis).

Chapter 2

Fundamentals of TeD, Discrete Event Simulator and Wireless Testbed

This chapter describes the Telecommunications Description framework (TeD) developed at Georgia Tech [6] and used to develop the parallel discrete event simulator (WiPPET) described in this thesis. This chapter gives basic introduction to Parallel Discrete Event Simulator (PDES) and explains the fundamental concepts and the working of PDES. At the end, the chapter gives brief introduction to a virtual wireless testbed.

2.1 Introduction to TeD

This section follows substantially the discussion of TeD found in [16, 17]. Software description of large systems are difficult and time-consuming to create, and therefore should be designed to have a long life time. Reusability requires clear separation between the description, implementation, and management parts of a large simulation.

TeD is a metalanguage designed for modeling telecommunication networks. TeD is more appropriately called a *framework*, because it provides a structure for model building. TeD is based on C++, a general-purpose object-oriented programming language, hence, it is called the TeD/C++ framework. The design of TeD included the following requirements:

1. A single language should provide for structural and functional description as well as dynamic simulation of designs.

2. The description language must be parsimonious, extensible, simulator-independent, and support structural and behavioral hierarchies (i.e. object-oriented).

3. The language must allow high-level formulation of concurrency control, dynamic model substitution (aggregation, deaggregation), distributed objects and runtime object creation.

TeD/C++ separates the metalanguage (framework schematic) from native language (implementation) by a domain-specific *little language* (TeD) that extends the base language in which it is implemented, i.e. C++. In the implementation of TeD used in this work, models are processed by the TeD compiler to generate the C++ code which is then compiled with a standard C++ compiler and linked with the GTW (Georgia Time Warp) optimistic parallel simulator for parallel execution [6].

TeD Abstraction

This subsection gives basic introduction to the various structures and terminology used in the TeD. The following are the fundamental structures supported in the TeD.

1. Entity
   - Architecture
     - State
     - Behavior
       - Process
       - Component
     - Result
2. Event
3. Channel

Figure 2.1 illustrates the basic framework underlying the constructs in TeD, and their relation to each other.

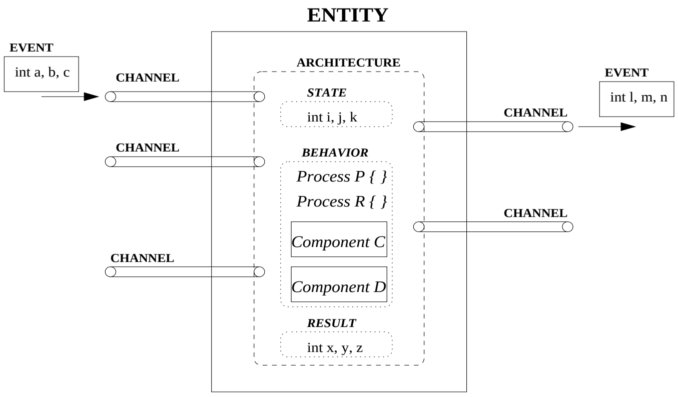


Figure 2.1: Basic framework of a TeD entity. An entity has an architecture which contains states, processes, IN (incoming) and OUT (outgoing) TeD channels, etc. A process of the architecture sends or receives events on these TeD channel.

Entity

In TeD, physical and conceptual objects in the telecommunication domain are modeled in terms of entity descriptions. Entities are connected to and interact with each other via channels, called TeD *channels* to avoid confusion with radio communication channel. TeD channels are the only means of dynamic interaction between entities. Information units, called *events*, flow through TeD channels. An output TeD channel of one entity may connect to the input TeD channel of another entity. The TeD channels of an entity also can connect to itself, these are called *internal* TeD channels. This allows for modeling of an arbitrary structure of connected entities. A channel type is defined as a set of event types. Appropriate inheritance semantics are defined for channels.

The *entity* declaration is used to define a black-box view of an entity. The entity could be a direct model of a real-life object, such as a multiplexer, a Base Station, or of an abstract object, such as a protocol. The behavior of an entity is defined solely in terms of its dynamic reactions to events on its input-mode channels, and the production of events on its output-mode channels. Special semantics are supported for interface inheritance and enhancement.

Architecture

The dynamic behavior of each entity is described by its *architecture*. The architecture of an entity is expressed using concurrent *process* semantics. The process is described in terms of the actions of the entity upon event arrivals on its input TED channel, and the production of events on its output TED channels. Internal TED channels are used to pass events from one process to another process within an entity. One or more processes can act upon arriving events or wait for events on the input channels of the entity. The processes may generate events on output channels as part of their action. An event generated on an output TED channel arrives as input on the input TED channel to which the output TED channel is *mapped*.

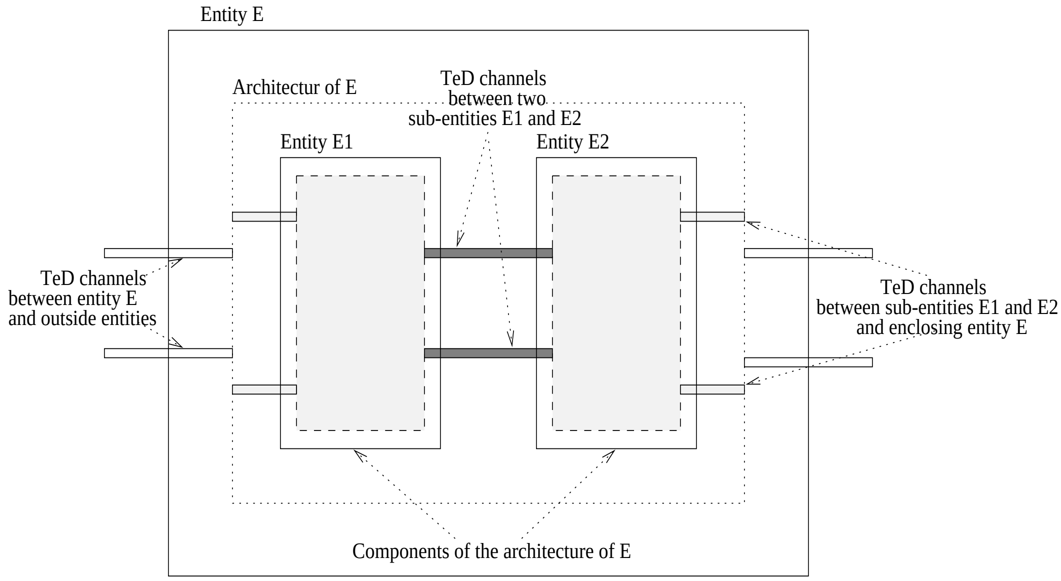


Figure 2.2: Components of an architecture. An architecture of entity E has two sub-entities E_1 and E_2 as its components.There are TED channels between sub-entities E_1 and E_2 as well between sub-entities and enclosing entity E.

The architecture of an entity can also be expressed as a composition of interacting *components*. Components are nothing but entities themselves (see Figure 2.2) logically enclosed inside the parent or bigger entity. In Figure 2.2, the enclosed entities E1 and E2 are called sub-entities of the bigger entity E. The TED channel of the components can be arbitrarily mapped amongst each other or to the TED channels of the enclosing entity. Thus, the behavior of an entity can be described in terms of its structure (components) or dynamics (processes), or a combination of both.

The architecture of an entity is composed of a small number of code blocks. Their simplified description is as follows:

- *Deferred Constants*: Variables whose values are held constant during runtime, but can be initialized to different values for different architecture instances.
- *State*: A set of variables that describes the abstract state of the modeled entity.
- *Processes*: Threads of computation that act on the events arriving on the interface and internal TeD channels, or that act upon time advances, or wait for events on TeD channels.
- *Components*: A set of entities that logically form sub-entities of an entity's behavior.

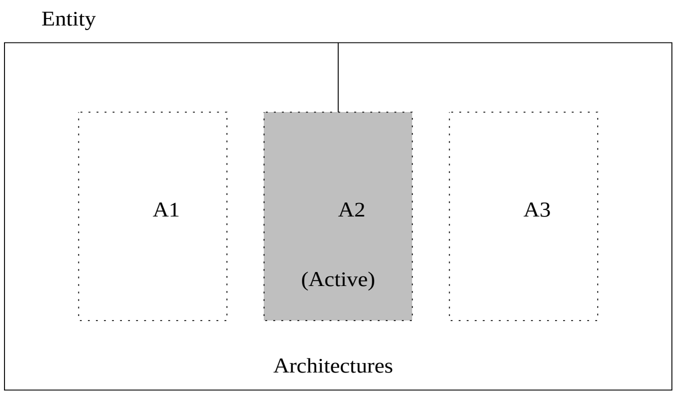


Figure 2.3: An entity with multiple architectures. A_1, A_2, and, A_3 are architectures of the entity. A_2 is currently bound to the entity. One cannot change the binding of an architecture to entity dynamically during a simulation. It is done before starting the simulation.

Entities are highly modular. Except for the interaction through their TeD channels, entities' state and behavior are completely encapsulated and invisible to other entities. The entity construct describes the entity's external input/output view, while the architecture construct describes the internal behavior of the object being modeled. More than one architecture can be defined for an entity. However, exactly one must be chosen and bound to the entity for simulation (in Figure 2.3, architecture $A2$ is bound with the entity). A configuration file is used to specify the bound architecture for each entity of the simulation.

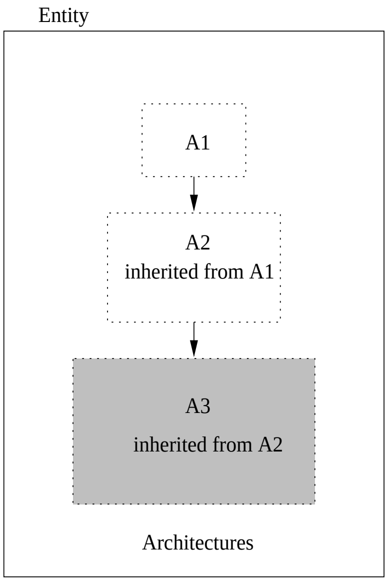


Figure 2.4: An entity with inherited architectures. A_1 is the primary or base architecture which is inherited by the architecture A_2, and then A_2 is inherited by the architecture A_3. Hence, A_3, the last architecture in hierarchy, has all the functionality of A_2 and A_1. That means, A_3 shares all the processes, states and components of A_2 and A_1.

Structural and behavior descriptions of an entity can be inherited by other similar entities, thus allowing object-oriented hierarchy-based design and development. Inherited properties can be specialized, redefined or overridden by the inheriting entity (see Figure 2.4, architecture $A3$ possesses structural and behavior descriptions of architectures $A2$ and $A1$).

Process

Processes are dynamic threads of computation acting on behalf of the entities that *own* them. Only the *owner* entity's TED channels (internal and external) and state variables are accessible to the entity's processes. The functionality of processes consists of combinations of two types of actions: *computation* (using the state variables), and *synchronization* (using TED channels or time). Computation is a sequence of operations performed on the state variables. Synchronization is a sequence of actions involving events on the channels over time.

Computation actions are specified in the executable language. Synchronization actions are specified using variants of the **wait** statement in the process body. According to the TeD semantics, the processes never stop executing, but repeatedly suspend and resume their activity depending on arrival of events or the timeout of a **wait** statement. TeD processes are conventional thread-like computations with the capabilities to make nested procedure calls and invoke that **wait** statements at arbitrary points in the process and procedure body. There are two types of processes described in TeD.

- *Self-driven process* : This is a self activated process. The process runs periodically after specific time duration.
- *Event-driven process* : This is a process activated by an event. The process listens on input TeD channels, specified in its argument list, and becomes active when an event arrives on the input TeD channels.

Configuration

A configuration file (*model.cfg*) is used to specify the binding of a particular architecture to an entity type; the configuration permits specification of different architecture bindings for different instances of the same entity type. It is also used to pass simulation parameters to different entities. The reconfiguration of simulation parameters doesn't require recompilation. It also provides a mechanism to map entity to processor in a parallel simulation. TeD includes a model management interface that allows reconfiguration of simulated models without recompilation, and user-defined entity-to-processor mapping. The latter is easily grasped by users with no prior experience in parallel computing. See [16,17] for more detailed explanation of concepts, terminology and semantics of the TeD framework.

2.2 Introduction to Parallel Discrete Event Simulator

A Discrete Event Simulation (DES) model expresses changes of state at discrete points in simulated time. The simulation model jumps from one state to another upon the occurrence of an *event*. This subsection first explains the working of DES and then discusses Parallel DES (PDES).

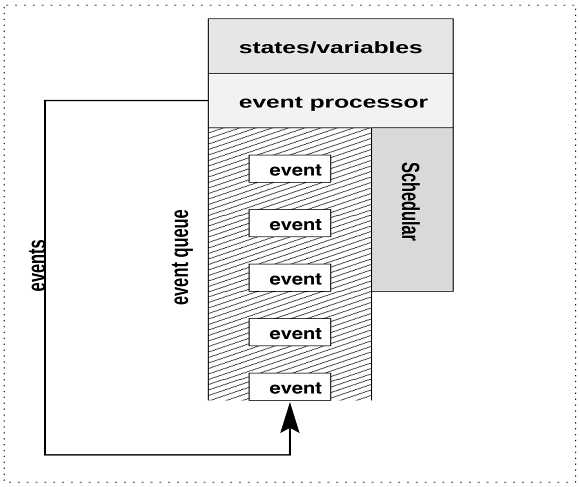


Figure 2.5: Main components of Discrete Events Simulator (DES).

Figure 2.5 shows the main components of a discrete event simulator. The simulator utilizes three principle data structures: 1) the *state variables* that describe the state of the system, 2) an *event queue* containing all pending events that have been scheduled, but have not yet taken effect, and 3) a global *time* variable to denote how far the simulation has progressed. Each event contains a timestamp and some payload data that indicate which change in the state of the simulated system should be executed. The timestamp indicates when this change occurs in the actual system, that is, the time at which the event will be executed. The scheduler attached to the queue arranges the queued events in descending order of their time stamps, so, the earliest time stamped event is at head of the queue and is processed first. The event processing may cause change in the states/variables of the simulation and may generate another events which will be queued. After the first(at the head of the queue) event is processed, it is removed from the queue, so the next event in the event queue becomes the head of the queue and will be processed next. The simulation time of the simulator is the timestamp of the last processed event. The simulator advances the simulation time by processing timestamped events.

In this execution paradigm, it is crucial that one always select the earliest timestamped event ($EventT_{min}$) from the event list in the queue as the one to be processed next. This is because if one were to select some other event containing a larger timestamp, say $EventT_x$, it would be possible for $EventT_x$ to modify state variables used by $EventT_{min}$. This would amount to simulating a system in which the future could effect the past! This is clearly unacceptable; we call errors of this type *causality* errors.

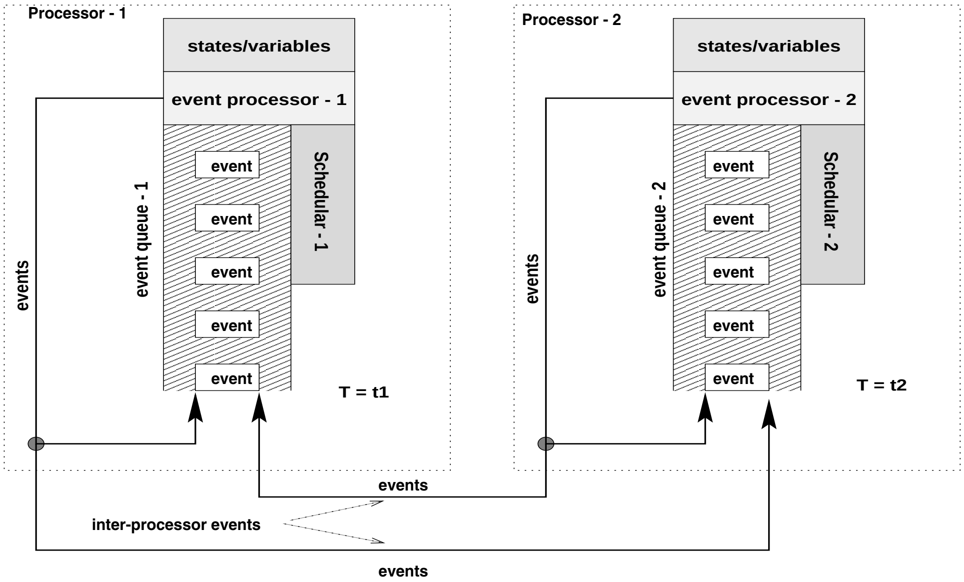


Figure 2.6: Two processor Parallel Discrete Events Simulator (PDES).

Let us now consider parallelization of a simulation program that is based on the above paradigm. The greatest opportunity for parallelism arises from processing events concurrently on different processors. Consider a two-processor PDES shown in Figure 2.6. and assume that the simulation work is distributed between two processors. If there are no inter-processor events (events passing between two processors), the processors run independently of each other. The work should be completed in half time that one processor would require. Hence, if work is divided evenly and there are no inter processor events, then the N processor simulation execution time can be approximated by

$$N \text{ processor execution time} \approx \frac{1}{N} \times (\text{uniprocessor execution time}). \tag{2.1}$$

The interesting case is in which there are inter-processor events in the two processor simulation. Consider the concurrent execution of two events E_1 sent by processor-1 to processor-2 and E_2 sent by processor-2 to itself, with timestamps T_1 and T_2 respectively. E_1 is called *inter-processor event.* Assume $T_2 > T_1$. Moreover assume that the processor-2 first processed the event E_2 with timestamp T_2. Thus, now, the simulation time of processor-2 is T_2. Then, the E_1 with timestamp T_1 sent by processor-1 is queued in the event queue of processor-2. Since $T_2 > T_1$, E_2 must be executed after E_1 to be sure a causality does not error. However, if processors 1 and 2 are not properly synchronized, one can not guarantee causality. Enforcement of causality by synchronization is the basic problem of parallel simulation. Based on the mechanism to avoid or resolve causality errors, there are two basic types of PDES: *conservative* and *optimistic.*

1. *Conservative*: In this case, the simulator avoids the causality error situation. A processor does not process an event unless it is certain that no earlier time event can ever arrive from other processors. There are several algorithms to achieve this; see [7, 11] for a detailed explanation.

2. *Optimistic*: In this case, the simulator allows causality error situations, but it has a mechanism to correct causality errors. Each processor processes events without worry. If an earlier time event arrives, the processor must undo the careless work. In other words, it has to undo the changes made in the simulation states/variables and the effect of processed events. This is called *rollback* of the second-processor.

   In above example, to resolve causality error situation the second-processor must go backward in time and undo everything done in the simulation up to the present moment T_2. The issue, here, is the possibility of an avalanche of rollbacks. It is measured in terms of *length of rollback* or *rollback distance*, i.e the *number of events* to be rollbacked. See [7, 11] for a detailed explanation.

The tradeoff between conservative and optimistic simulator is that in conservative simulation one has to estimate the *lookahead time*, which is not easy. On other hand, in case of optimistic simulation, one has to provide mechanism to rollback. A more detailed taxonomy of simulation mechanisms is described in [7,11,13].

TeD/C++ framework is based on optimistic PDES, GTW (Georgia Time Warp). Since WiPPET is based on TeD/C++, it is an optimistic PDES. Rollback is an important phenomenon to understand in optimistic parallel simulator because it is expensive in terms of simulation execution time. It causes unnecessary usage of computational resources. The execution time in an N processor environment may no longer be described by the relation 2.1. Thus, rollbacks degrade the parallel simulation efficiency. Sometimes rollbacks may even dominate so that parallel simulation performs worse than sequential uniprocessor simulation.

In the context of the earlier example, the reason for rollback is that processor 2 is executing faster, and thus running ahead of processor 1 in simulation time (the two processors are not *synchronized*). To mitigate rollbacks, one wants simulation time synchronization achieved by *load balancing*. Load balancing is the art of partitioning and distributing simulation work load uniformly among processors. If the simulation load is not distributed uniformly among processors, a processor which has the least work load executes faster and gets ahead in simulation time. In general, it is difficult to balance perfectly the simulation work load among processors during the simulation. Rollback is unavoidable in optimistic parallel simulators, but the number of rollbacks can be reduced by proper load balancing techniques and most importantly by reducing the number of inter-processor events. The effects of load balancing and rollbacks on parallel simulation performance are discussed in chapter 5 in detail.

2.3 Introduction to a Virtual Wireless Testbed

A modern radio network seeks to provide a specified quality of service to the maximum number of users by proper *radio resource management*. WINLAB's previous non-parallel simulator, the Mobility And Dynamic Resource Allocation Simulator (MADRAS [2]) addresses the issues related to radio resource management.

This thesis present the initial version of the virtual wireless testbed Wireless Propagation and Protocols Evaluation Testbed (WiPPET). The testbed provides a platform to model wireless networks two time scales : 1) session or call level ($\text{WiPPET}_{\text{session}}$), and; 2) packet level ($\text{WiPPET}_{\text{packet}}$).

$\text{WiPPET}_{\text{session}}$ implements MADRAS in the TED framework with some extensions. It models session or call level traffic, distance loss and shadow fading propagation, mobility and resource management algorithms – call admission, uplink power control, and six handoff algorithms; three system handoff algorithms (AMPS, IS-136, GSM); and other three handoff algorithms from the literature. $\text{WiPPET}_{\text{session}}$ provides system performance statistics in terms of call blocking, call dropping, average number of handoffs per call, which are used to analyze the performance of various radio resource management protocols

$\text{WiPPET}_{\text{packet}}$, a version of WiPPET for packet level traffic, models packet level traffic, propagation model having distance loss, short scale (Rayleigh) and long scale (shadow) fading, mobility, and power control. The simulator provides various statistic (e.g. PER–Packet Error Rate) to analyze the performance of data-link layer protocols in interference and fading environment. Currently, $\text{WiPPET}_{\text{packet}}$ is used to analyze the performance of data-link layer in terms of PER for different mobile station speed wireless data standards like Cellular Digital Packet Data (CDPD), and Generalized Packet Radio Service (GPRS).

This section introduces components of a virtual wireless testbed:

1. Geography and Mobility
2. Radio Propagation and Interference
3. Traffic
4. Protocols

If the testbed efficiently implements these fundamental tasks, then researchers can study and move forward quickly in understanding the nature and behavior of the system in various environments by varying these phenomena. In the real world, the first three phenomena are present naturally and have a crucial influence on the behavior of the wireless network. *Radio Resource Management* is the key by which we can limit the inverse or bad effects of these natural phenomena on the network performance.

2.3.1 Geography and Mobility

Mobility is a process describing how a mobile station (mobile) moves from one point to another within a given geography model. Mobility is of prime importance in a wireless network. To emulate mobility, a wireless simulator must include a description of the terrain in which the mobile can travel. The set of possible mobile locations in the terrain is the geography on which the simulation will be performed. Moreover, the geometry of the terrain determines gross radio propagation characteristics. This way, mobility is connected to geography as well as to radio propagation characteristics of the geography.

WiPPET allows the modeler to import an approximation to real map, or a synthetic geography, and to model mobile trajectories that either evolve at runtime, or are precomputed to simulate a particular traffic scenario. More detailed explanation of geography model is given in section 3.2.

2.3.2 Radio Propagation and Interference

It is our goal to model the performance of a system in different environments. Performance of a radio receiver depends on receiving sufficient signal energy from the selected transmitter while not receiving too much energy from other sources. In addition, the temporal resolution of the information carried by the signal that may be smeared by delay spread in multipath propagation. We describe propagation and interference phenomena that are necessary to simulate the performance of a wireless system.

Radio reception quality can exhibit very large fluctuations in space and time. Sources of these fluctuations include changes in distance between receiver and transmitter, and changes in environment or terrain and activity of other transmitters. Together these physical conditions produce five main effects :

1. *Distance loss* : signal attenuation that increases with distance

2. *Shadow fading* : signal variations due to environmental features (hills, buildings).

3. *Fast fading* : signal fluctuation due to the relative motion between transmitter and receiver.

4. *Multipath* : signal distortion due to components of the signal traveling along different paths from a transmitter to a receiver.

5. *Interference* : activity of other transmitters sharing the same radio channel, causing distortion in received signal.

It is necessary to model these spatial and temporal variations, i.e. distance fading, shadow fading, fast fading, and interference, in radio channel propagation characteristics for the wireless network simulation in order to estimate accurately the performance of a wireless system.

There is a vast body of literature on the measurements and modeling of propagation in various environments at various frequencies; for a review, see [10, 18]. The classical models are derived as follows: The i^{th} terminal, located at position $\mathbf{x}_i$, transmits signal $s_i(t)$. The received signal at location $\mathbf{b}$ over a general time varying radio channel is

$$R_i(t, \mathbf{b}) = \int_{-\infty}^{t} h(t, \tau, \mathbf{x}_i, \mathbf{b}) s_i(\tau)\, d\tau + \mathcal{N}(t) \tag{2.2}$$

where h is the channel impulse response and $\mathcal{N}(t)$ is additive white Gaussian noise. The corresponding received power at b is $P_{RX}^{(i)}(t, \mathbf{b}) = E[R_i^2(t, \mathbf{b})]$. A measure of the received signal quality at position $\mathbf{b}$ for terminal i is *signal to interference ratio* (SIR), which is defined by ratio

$$\Gamma_i(t, \mathbf{b}) = \frac{P_{RX}^{(i)}(t, \mathbf{b})}{\sum_{j \neq i} P_{RX}^{(j)}(t, \mathbf{b}) + \eta^2}$$

where η^2 is the additive noise power.

The modeling of the channel response $h(t, \tau, \mathbf{x}, \mathbf{b})$ depends on performance assessment needs: it may be site-specific [19], that is based on actual measurements or it may employ stochastic processes with values and correlations mimicking those measured in a class of similar environments (e.g. typical American suburban macro cell, urban micro cell, or highway). While the design of WiPPET allows the use of site-specific propagation models, in our research we focus on statistical characterization of the wireless network, and thus we limit the discussion to stochastic propagation models.

Signal attenuation in a radio channel is commonly represented as the product of long-scale and short-scale fading: $h(t, \tau, \mathbf{x}, \mathbf{b}) = g(\mathbf{x}, \mathbf{b}) \cdot h(t, \tau)$ [14]. The long-scale or shadow fading, g, represents the attenuation of the signal averaged due to distance and shadowing by large objects. The short-scale factor $h(t, \tau)$ represents time-varying effects of interference and delay dispersion of carrier waves arriving over distinct paths.

It is important to note that in a simulation the details of the fading model should be matched to the signal processing capabilities of the modeled receiver to obtain meaningful statistics for data loss. WiPPET$_{\text{session}}$ is implemented with distance loss and shadow fading for session level simulation.

Distance loss and Shadow fading

Empirical measurements show that average received signal strength decreases with distance from the transmitter according to

$$P_{RX} = P_{TX} \cdot \frac{K}{X^{\alpha}} \tag{2.3}$$

where K is a constant, P_{RX} and P_{TX} are the received signal power and transmitted signal power, and X is the distance between transmitter and receiver. The propagation exponent $\alpha = 2$ models a *free space* propagation environment. In wireless networks, $\alpha \geq 2$ with the precise value dependent on terrain and other environment factors such as buildings. In urban areas, $\alpha = 4$ is often used [8, 18]. However, it is also possible to find $\alpha < 2$ as a result of wave-guide effects that appear in indoor corridors and in cities when antenna heights are substantially lower than surrounding buildings.

Equation 2.3 represents the average signal strength, but the power measured at a specific location can be is generally above or below this average. As the receiver moves, it observes that the local average signal slowly fluctuates around value given by equation 2.3 occurring over distance of several dozens of the signal wavelengths. This phenomenon is referred to as *shadow fading* or *slow fading* or *long-scale fading*.

Long-scale fading is treated as constant in time and can be precomputed. For cellular networks, WIPPET$_{\text{session}}$ employs the distance and slow fading represented by path gain matrix $g_{ds}(i,j) = \log g(\mathbf{x}_i, \mathbf{b}_j)$ for all $\mathbf{x}_i \in \mathcal{M}$ and $\mathbf{b}_j \in \mathcal{B}$, where $\mathcal{M}$ is the set of mobile locations, and $\mathcal{B}$ is the set of base station locations. In WIPPET the $|\mathcal{M}| \times |\mathcal{B}|$ gain matrix g_{ds} is a large state constant that may be read from a file or precomputed in a site-specific or stochastic propagation model.

The classic stochastic model presents the geographical path losses as

$$g(\mathbf{x}, \mathbf{b}) = g_d(|\mathbf{x} - \mathbf{b}|) \cdot g_s(\mathbf{x}, \mathbf{b})$$

where g_d is deterministic function of distance and g_s is a log-normally distributed spatial random process. Measured $g_d(|\mathbf{x} - \mathbf{b}|)$ decays as an inverse power of distance, i.e. $g_d \sim |\mathbf{x} - \mathbf{b}|^{-\alpha}$, where $1 < \alpha \leq 4$ [8, 18]. A standard stochastic description of the shadow fading fluctuations due to large obstacles is modeled by a single realization of a two dimensional log-normal random process, $\log g_s \sim \mathcal{N}(0, \sigma^2)$, with exponentially decaying spatial correlations, $E[\log g_s(\mathbf{x}, \mathbf{b}) \cdot \log g_s(\mathbf{x}', \mathbf{b})] = \sigma^2 e^{(\frac{-|\mathbf{x}-\mathbf{x}'|}{d_0})}$ [9].

Fast fading

The collection of radio waves that arrives at the receiver from different directions, having traversed distinct reflecting paths from transmitter, is called *multipath*. The signal arrives at a receiver as a multipath signal or a collection of signal components coming from the different directions with different delays, with different attenuation. As the relative speed between transmitter and receiver changes, each ray undergoes a *Doppler shift* causing the apparent wavelength of the signal to either increase or decrease. Radio waves arriving at right angles to the direction of the motion experience no shift in wavelength. The different Doppler shifts and delays in the many radio waves arriving at the receiver causes the radio waves to arrive with different relative phase. At some locations the radio waves add or cancel each other and thus the received signal becomes relatively stronger or weaker respectively. These fluctuation occurs at a one wavelength scale which is much smaller spatial scale than fluctuation due to *shadow fading*. For example, in 850 MHz cellular radio band significant changes occur over a distance of the order $30cm$, which is the wavelength of the carrier frequency. In a cellular system with a relative speed at $50km/hr$, the signal exhibits approximately 40 peaks and nulls per second [8]. This effect is called *fast fading* or *short-scale fading*.

The term short-scale fading encompasses the effects of the interference of the components of multipath propagation of radio waves, and their varying travel time from the source to the receiver. Sklar [22] provides a useful taxonomy: First, the relation between the maximum excess delay T_m in impulse response (due to multipath propagation) and the symbol duration time T_s determines two signal degradation categories: i) **Frequency–selective fading** for $T_m > T_s$, producing channel-induced inter-symbol interference (ISI); ii)**Flat fading** for $T_m < T_s$, when multipath components are not resolvable. Second, the relation between the channel coherence time T_0 (characterizing temporal variability due to changing multipath components) and the symbol duration time T_s determines two other signal degradation categories: i) **Fast fading** for $T_0 < T_s$, producing symbol distortion, e.g. due to high Doppler; ii) **Slow fading** for $T_0 > T_s$, SIR fluctuations.

Modeling of frequency-selective fading, e.g., for a RAKE receiver, employs temporal fading (like Rayleigh) separately on each resolved path, together with a stochastic process of arrivals and departures of multipath components, with appropriate delay and power distribution. Such models have been derived in literature both for urban and indoor environments [9,18]. There are also various models of temporal fading: Rayleigh, Rice (LOS) are typical. With these, temporal variation appears through relative motion of the transmitter (Tx) and receiver (Rx). However, omnipresent environmental motion introduces temporal fading even if both Tx and Rx are stationary, but that case is difficult to characterize because it is highly environment dependent. The case in which relative motion is the larger effect is characterized by Rayleigh and Ricean fading.

In contrast to long–scale fading, modeling of short–scale fading need not be geographically indexed due to short–range correlations. In WiPPET the stochastically generated short–scale temporal fading is modeled independently for each Tx–Rx pair in the shared radio channel and combined at the receiver. We have implemented the models of short–scale Rayleigh and Ricean fading [5] with i) wave-superposition method, ii) spectrum-shaping method; and also iii) a quantized channel state Markovian model [1].

2.3.3 Traffic

The simulator must provide the generation of offered *traffic* (load) to the network. Traffic plays major role in understanding the system capacity for given environment and protocol instances. For session level simulation, $\text{WiPPET}_{\text{session}}$ models session arrivals and for packet level simulation, $\text{WiPPET}_{\text{packet}}$ models packet traffic. The parameters of interest for traffic modeling includes the rate of session or packet arrival and spatial and temporal variations in that rate, and the parameters of individual session or packet (e.g. duration of session or packet, temporal characterization of source).

2.3.4 Radio Resource Management

Radio resource management includes any actions that are necessary in a wireless system to provide the required quality of service (QoS). In wireless networks, the term has generally included signaling protocols that control call admission, channel allocation, power control, and handoff.

The objective of a radio resource management protocols are to assign a base station, an uplink and a downlink channels and a transmitter powers to as many communication links as possible at any instant, subject to the constraint that the quality of all links should be above a certain threshold. The radio resource management protocols consists of several algorithms [2], namely,

1. *Base Station Assignment* : The assignment of a base station with which the mobile station or mobile communicates.

2. *Channel Allocation* : The allocation of a channel which provides sufficient link quality at the assigned base station.

3. *Call Admission* : When an mobile initiates a new call or when a mobile handoff to a new base station the network must decide if it will attempt to provide the requested service to the mobile. The call admission is made up of *base station assignment* and *channel allocation* algorithms.

4. *Power Control* : This process periodically or on request adjusts the power of the transmitter to reduce the overall system interference level and maintain the required QoS.

5. *Handoff* : The switching procedure used by an mobile to change its communication from one base station to another one when the mobile is moving out of the coverage area of the prior base station.

The radio resource management for an mobile starts when the mobile is trying to make a phone call and ends when the call is blocked, finished or dropped. The radio resource management protocols are shown in Figure 2.7.

When a new call arrives, the call admission algorithm is initiated. A base station is assigned to mobile and a traffic channel is allocated to mobile by the assigned base station. If the channel allocation is successful, then the mobile starts conversation. Otherwise the call is *blocked*. During the call, the radio link quality is checked periodically by the power control algorithm and if certain handoff conditions are met, as specified by the handoff algorithm, a handoff is attempted for the mobile. The handoff algorithm invokes a call admission algorithm at the new base station (if handoff is intra-cell, then the new base station will be same as the old base station). The new base station will assign a new traffic channel to the mobile. At new base station, hand-in conditions are checked and made sure that QoS of the new link is sufficient. If the link quality remain below specified threshold (*drop threshold*) for sufficiently long time, the call is dropped.

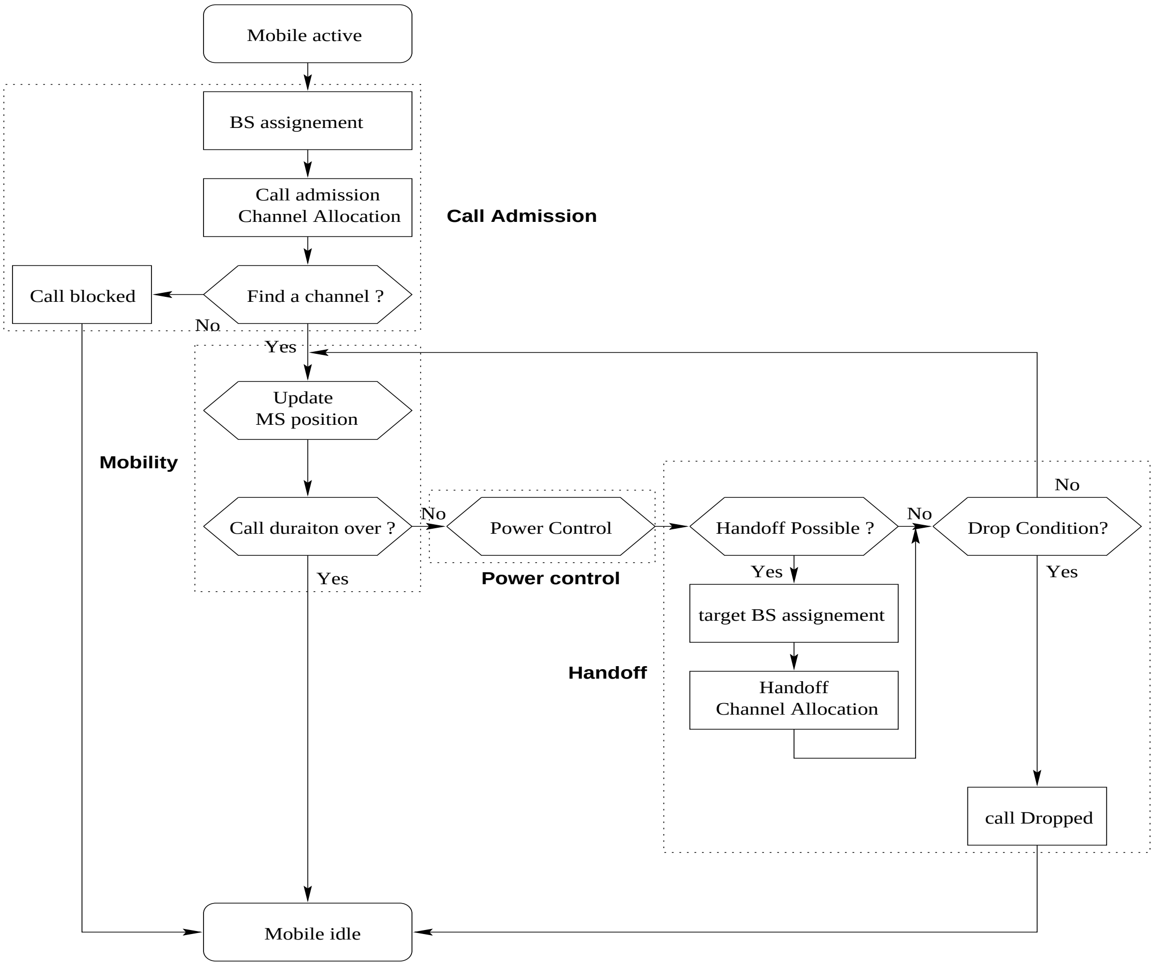


Figure 2.7: Flowchart of radio resource management protocols and algorithms

The power control algorithm changes the transmitter power according to a given QoS requirement.

In the current version of WiPPET$_{\text{session}}$, the following protocols are implemented. For call admission there are two variants of base station and channel assignment: use a channel from the base with the strongest pilot tone, or search a dynamic list of bases in order of decreasing pilot strength to find an acceptable channel. From MADRAS [2], a family of dynamic channel allocation methods have been incorporated that employ interference measurements. Six handoff algorithms have been implemented. Three are based on standards: AMPS, North American TDMA (IS-136), and GSM. The other three algorithms, Signal to Interference Ratio (SIR) based handoff, received-signal-strength (RSSI) based handoff, and a combined SIR and RSSI based handoff, are derived from MADRAS [3, 4]. The power control algorithm is an asynchronous distributed uplink power control [25] in which each user attempts to minimize its transmitter power while maintaining its required SIR.

Chapter 3

WiPPET: Session level simulation

A version of the wireless testbed has been constructed to simulate sessions in a wireless voice network. A session begins when a mobile station subscriber requests a wireless channel. The session may terminate in three ways: if no radio channel is granted or assigned, the session or call is said to be *blocked*; if the channel is granted but later fails to provide sufficient quality of service (QoS), the call is *dropped*; otherwise the session completes successfully.

This version of WiPPET generates session traffic and it provides session level system performance statistics like probability of call blocking and call dropping, etc. Hence, it is called a session level simulation. This chapter gives a detailed explanation of $\text{WiPPET}_{\text{session}}$ entities and their architectures. It also discusses the design issues of entity architecture and modularity of the architecture. It also explains the implementation of various $\text{WiPPET}_{\text{session}}$ protocols using interaction diagrams.

3.1 Overview

A wireless network consists of mobile stations, base stations, and mobiles switching centers (MSC). A mobile station is connected to a base station through the air interface (a radio link). A base station is connected to a MSC through a microwave link or an optical fiber link. A MSC has an interface between the wireless network and other public networks like Public Switching Telephone Network (PSTN). The mobile stations, base stations and switches take part in various protocols and run the system by interacting with each other. See Figure 3.1.

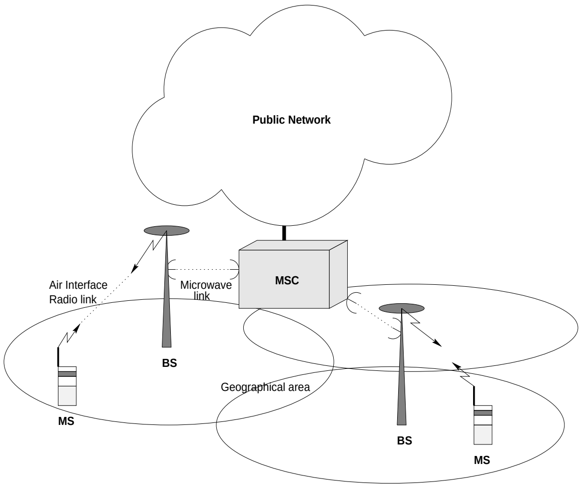


Figure 3.1: Elements of a wireless network.

Analogously, WiPPET has TeD entities: Mobile Station (`MS`)[1], Base Station (`BS`), and Mobile Switching Center (`switch`) to represent their physical counterparts. To represent the air interface, propagation environment, and geography, WiPPET has the entity `zone`. The fifth and last of the base entities is `callgen` which drives the simulation with session traffic and records the QoS given each session. The term *base* entity emphasizes the intention that the testbed serves as a starting point to which other researchers may add. Figure 3.2 shows the entities of WiPPET$_{\text{session}}$ and their connectivity through TeD channels.

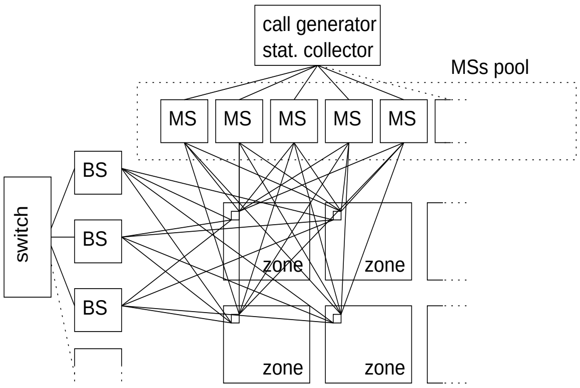


Figure 3.2: WiPPET$_{\text{session}}$ entity overview. `Callgen`, `MS`, `zone`, `BS`, and `switch` are base entities. These base entities are connected through TeD channels as shown by straight lines.

[1]The names of WiPPET entities that appear in `typewriter` font match exactly the entity name as they appear in the source code of WiPPET.

Initialization of the simulation shown in Figure 3.2 is governed by the configuration file (model.cfg). The number of `BS`s and `zone`s depends on the geographical area to be simulated and remains fixed throughout the simulation. The number of active mobiles and hence the number of `MS` entities required, is determined by offered load or traffic. However, because the current TED/GTW version 2.0/2.3 doesn't support dynamic entity creation, it is necessary to preallocate a sufficiently large fixed pool of mobiles. While configuring a simulation, the mobile pool is set to be $x\%$ larger than the average number of active mobiles.

$$\text{MS pool size} = (1 + \frac{x}{100}) \times \text{average call arrival rate} \times \text{average call duration} \qquad (3.1)$$

Experimentally, it was found that the range of x from 10 to 20 is sufficient. During runtime, if the pool becomes empty, the simulator generates an error and stops.

According to Figure 3.2, the `callgen` (call generator and statistic collector) entity is connected to all the `MS` entities of the pool. Each `MS` and `BS` is connected to all `zone`s. There is no direct TED channel from any `MS` to any `BS` entity. All `BS`s are connected to the `switch` entity. The simulation system may have one or more than one `zone` entity. The number of `zone` entities strongly influences parallel simulation performance which is addressed in detail in chapter 5.

We begin with a basic introduction of WIPPET entities.

1. Mobile Station (`MS`) : It participates in mobility, call admission, power control and handoff protocols.

2. Base Station (`BS`) : It takes part in mobility, call admission, power control and handoff protocols. It allocates a channel to a new `MS` or a hand-in `MS`. It initiates power control and handoff check periodically.

3. Mobile Switching Center (`switch`) : It takes part in call admission and handoff algorithms. It finds the appropriate target `BS` from the candidate `BS`s list for the MS handoff.

4. Call generator and statistic collector (`callgen`): It generates session or call traffic.

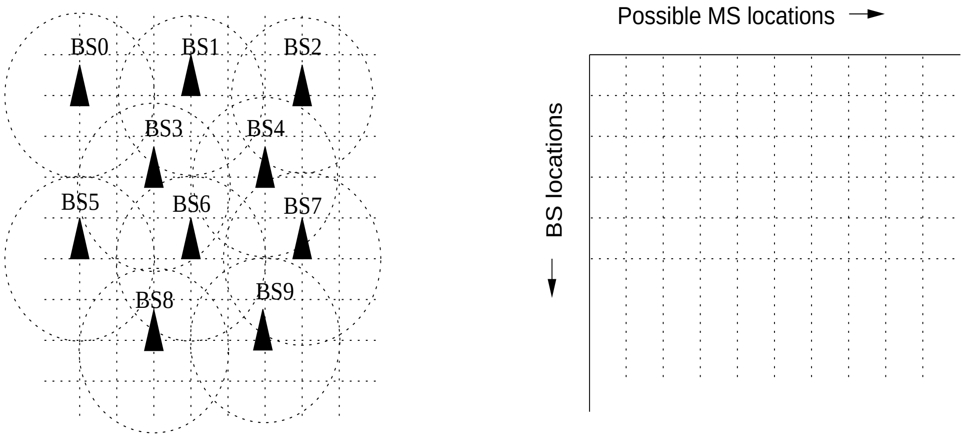
It also collects vital traffic statistics like call blocking, call dropping, average inter-cell handoff per call, and average intra-cell handoff per call, etc.

5. Geography zone or Radio channel zone (`zone`) : It is the most complex entity in WiPPET. It represents geography and radio channel and takes part in mobility, call admission, power control and handoff protocols. The `zone` is described in more detail in section 3.2.

3.2 Zone Entity

The `zone` is made up of one or more cells. It owns geographical graph of points, i.e. the possible `MS` locations. WiPPET is implemented with a Manhattan geographical model. It also contains geographically referenced path loss (distance loss and shadow fading) function $g_{ds}(\mathrm{MS}_{\mathrm{location}}, \mathrm{BS})$ via a path loss matrix. This is a two dimensional matrix with one axis representing possible `MS` locations and other axis representing `BS` locations in the system, as shown in Figure 3.3. A separate MATLAB program was written to generate the Manhattan geographical model and associated path loss matrix (file `initz_mn.m`). The program takes a Manhattan geography resolution, correlation distance, variance of shadow fading, base station locations (base station locations are stored in `BS.TXT`), etc., as inputs and generates a path loss matrix and stores it in the file `ZONE.TXT`. During the simulation parameter initialization, the `zone` reads the file and stores path loss matrix as a deferred constant. $\mathrm{WiPPET}_{\mathrm{session}}$ supports arbitrary geographies, because the geography and path loss matrix are imported during the initialization of the simulation. Manhattan is a special case used in the present work.

A `zone` combines the role of a geography container with the role of a radio channel simulator, collecting contributions to total signal power in a radio channel from all active transmitters and attenuating them by an appropriate fading factor. To enable received power and interference power calculation, the `zone` contains : 1) a large constant structure containing the geography points together with precomputed path loss matrix $g_{ds}(MS, BS)$; 2) for each `MS`, a dynamic list of data structure contains, `MS` radio



(a) Manhattan geography model

(b) Path loss matrix

Figure 3.3: (a) Manhattan geography contains streets and avenues. These streets and avenues are modeled as one dimensional lines constructed by discrete points on which mobile station moves step by step. (b) Path loss matrix is a huge deferred state constant in `zone` entity. It contains path loss (distance loss and shadow fading value) from every possible mobile station location to any base station.

channel ID, instantaneous values of `MS` position, transmitter power, parent `BS` ID (i.e., assigned `BS` for the `MS`); 3) for each radio channel, a dynamic list of `MS`s, which are active on that channel. Using these data structures, the `zone` calculates uplink Signal to Interference Ratio (SIR) on the given `MS` radio channel.

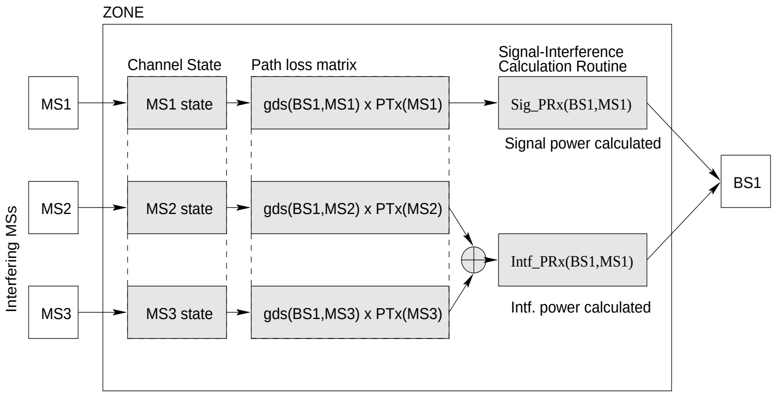


Figure 3.4: Graphical representation of uplink SIR calculation. MS_2 and MS_3 are interfering mobile stations at base station BS_1 for the mobile station MS_1. Shaded areas show used states and constants, arrows show the flow of action in calculation of uplink SIR at BS_1 for MS_1 in `zone`.

Now, we discuss how the `zone` calculates uplink SIR using the above mentioned data structures. Assume that there are three mobile stations MS_1, MS_2, and MS_3. MS_1 is connected to BS_1. Consider that MS_2 and MS_3 are the interfering mobile stations for the MS_1 at BS_1. The `zone` finds the received signal power of MS_1 at BS_1, the interference power at BS_1 for the MS_1, and the uplink signal to interference ratio $SIR(\mathrm{MS}_1, \mathrm{BS}_1)$ of MS_1 at BS_1 using equations 3.2, 3.3, and 3.4 respectively.

$$P_{RX_S}(\mathrm{MS}_1, \mathrm{BS}_1, ch) = g_{ds}(\mathrm{MS}_1, \mathrm{BS}_1) P_{TX_S}(\mathrm{MS}_1, \mathrm{BS}_1, ch) \tag{3.2}$$

$$P_{RX_I}(\mathrm{BS}_1, ch) = \sum_{n \neq 1} g_{ds}(\mathrm{MS}_n, \mathrm{BS}_1) P_{TX_S}(\mathrm{MS}_n, \mathrm{BS}_1, ch) \tag{3.3}$$

$$SIR(\mathrm{MS}_1, \mathrm{BS}_1, ch) = \frac{P_{RX_S}(\mathrm{MS}_1, \mathrm{BS}_1, ch)}{P_{RX_I}(\mathrm{BS}_1, ch)} \tag{3.4}$$

where subscripts S and I represent signal and interference quantity respectively. $P_{RX_S}(\mathrm{MS}_1, \mathrm{BS}_1, ch)$ and $P_{RX_I}(\mathrm{BS}_1, ch)$ are signal and interference power on channel ch between mobile station MS_1 and base station BS_1 respectively. MS_n are the interfering `MS`s on channel ch. $SIR(\mathrm{MS}_1, \mathrm{BS}_1, ch)$ is the uplink SIR on ch for mobile station MS_1 at base station BS_1. A graphical representation of the uplink SIR calculation is shown in Figure 3.4. The `zone` uses formulas 3.2, 3.3 and 3.4 to calculate interference in channel allocation, power control and handoff algorithms.

The `zone` has path loss matrix which contains an array of pilot signal strength from each `BS` to a given `MS` location. For the given `MS` location, the `zone` can find the strongest pilot BS_1, such that, which satisfies the condition 3.5.

$$g_{ds}(\mathrm{MS}_{\mathrm{location}}, \mathrm{BS}_1) = \max_{\mathrm{BS}_n}\{g_{ds}(\mathrm{MS}_{\mathrm{location}}, \mathrm{BS}_n)\}; \quad \mathrm{n} = 1,...,\mathrm{NumBS} \tag{3.5}$$

To save computation during a simulation, a list of the first four strongest pilot `BS`s is precomputed for every possible `MS` location and stored as a `zone` deferred constant in the *point* structure. For complete information about the point structure, a deferred constant of the `zone`, see appendix B.1. The `zone` uses this list of strongest `BS` pilots to find a parent `BS` in call admission and a target or new `BS` in handoff. For complete information about the deferred constants and the `MS` mobility state of the `zone`, see appendix B.

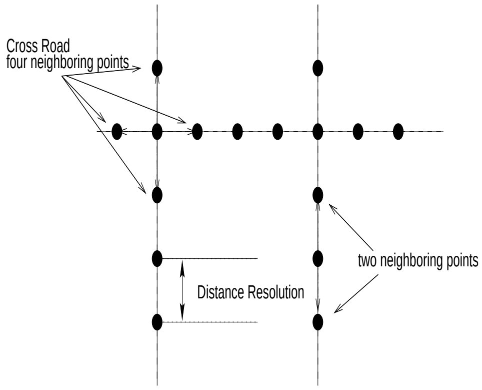


Figure 3.5: Manhattan graph with neighboring points. At cross sections of streets and avenues, there are 4 neighboring points otherwise 2. Distance between two successive points on a line is called distance resolution of the geography model.

For the mobility or position update procedure, the `zone` contains a large constant matrix having all possible graph points on which the `MS`s moves. Each graph point has N neighboring points. The value of N depends on the geography graph. In the Manhattan model, each point has maximum four neighboring points ($N = 4$), shown in Figure 3.5. The `zone` selects the next `MS` position from among its current neighbors according to a probability mass function that is specified in the configuration file. See appendix B for more detailed explanation of `zone` deferred constants and states used by mobility and interference calculation procedures in the `zone`.

3.3 Implementation

WiPPET$_{\text{session}}$ implements simulations of a wireless cellular voice network. For call admission WiPPET$_{\text{session}}$ uses the strongest pilot base station assignment algorithm and two variants of channel assignment algorithms: use a channel from the base with the strongest pilot tone, or search a dynamic list of bases in order of decreasing pilot strength to find an acceptable channel. From MADRAS [2], WiPPET has incorporated a family of dynamic channel allocation methods that employ interference measurements. Six handoff algorithms have been implemented. Three are based on standards: AMPS, North American TDMA (IS-136), and GSM. The other three algorithms, SIR based handoff, RSSI based handoff, and a combined SIR and RSSI based handoff, are derived from [3, 4]. The power control algorithm is an asynchronous distributed uplink power control in which each user attempts to minimize its transmit power while maintaining a required SIR [25]. Long–scale fading is precomputed for the specified geography (Manhattan model) by overlaying correlated log–normal shadow fading [9] on power–law distance loss [18]. Following is the list of algorithms and procedures available in WiPPET$_{\text{session}}$ model data base.

- Geography - Manhattan model.
- Mobility - Biased random walk.
- Propagation Model: Geographically referenced (distance and shadow fading).
- Base Station Assignment: Strongest pilot signal base station assignment.
- Call Admission Protocols: Least Interference based Dynamic Channel Allocation (LI-DCA), Active Channel Preservation based Dynamic Channel Allocation (ACP-DCA).
- Power Control: Asynchronous-distributed uplink power control
- Handoff Protocols: AMPS, NA-TDMA, GSM (standard based handoffs), Received Signal Strength Indication based HandOff (RSSIBHO), Signal to Interference based HandOff (SIRBHO), combined RSSI and SIR based HandOff (RSSI.SIRBHO).

WiPPET also has separate models of short–scale Rayleigh and Ricean fading [5] with (i) wave-superposition method, (ii) spectrum-shaping method; and also (iii) a quantized channel state Markovian model [1]. These models will be modified and used for WiPPET$_{\text{packet}}$, the packet level simulation (discussed in chapter 4).

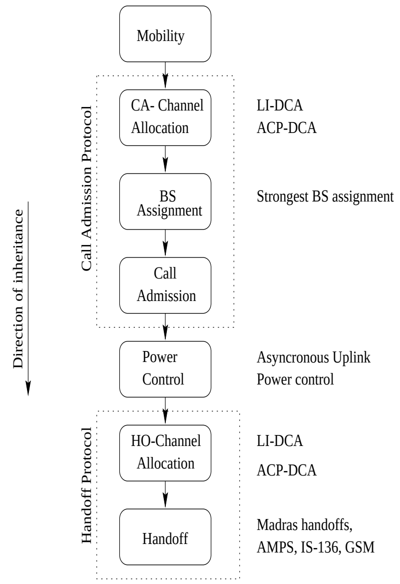


Figure 3.6: General hierarchy of inherited architectures in an entity. Down arrows show the direction of inheritance in hierarchy of architectures. Mobility is the primary architecture and handoff inherits all states and processes defined in architectures above it. Call admission protocol is divided into three architectures within an entity; 1) call admission (CA) channel allocation, 2) strongest BS assignment, and 3) call admission (decision). Similarly, handoff protocol is divided into two architecture; 1) handoff and 2) handoff (HO) channel allocation. See text for more detail.

3.4 Architectures of Entities

WiPPET$_{\text{session}}$ models mobility, call admission, power control and handoff protocols. Each entity of WiPPET$_{\text{session}}$ supports one or more protocols, i.e. a protocol functionality is distributed among entities. Protocols are represented by one or more architectures in an entity.

As we discussed in section 2.1, in TeD framework, an architecture can be inherited from another architecture with added functionality. WiPPET$_{\text{session}}$ employs inherited hierarchical architecture design method. General hierarchy of WiPPET$_{\text{session}}$ entity architectures is shown in Figure 3.6. There are seven distinct layers in entity's architecture hierarchy. Mobility is the *primary architecture.* The primary architecture is the first architecture of an entity in the hierarchy on which all other architectures are built. Handoff is the last architecture in the hierarchy.

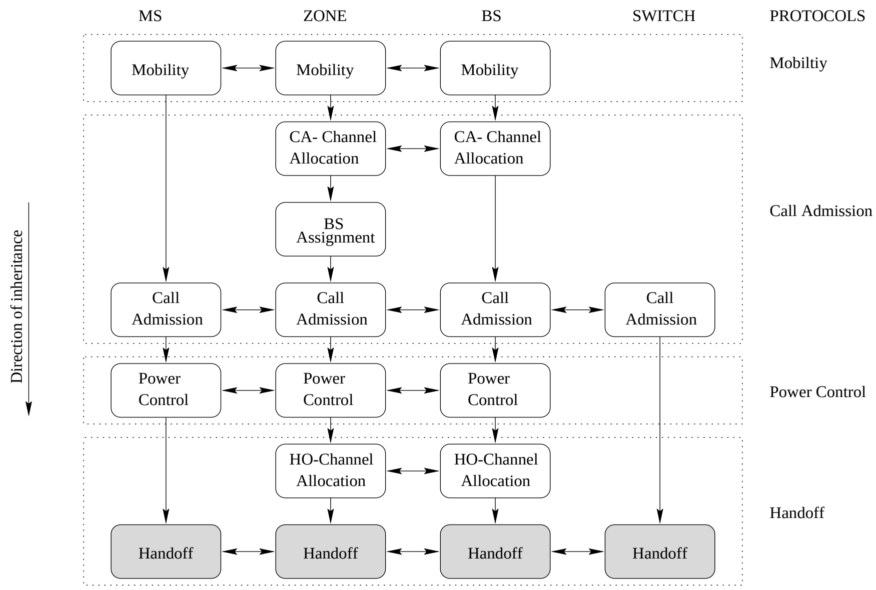


Figure 3.7: Hierarchy of inherited architectures in WiPPET$_{\text{session}}$ entities. Vertical down arrows show the direction of inheritance. Each entity supports one or more protocols, i.e. protocol functionality distributed among entities. For example, `zone` has all seven layer of architecture inheritance, so, it support all protocols of WiPPET$_{\text{session}}$. On other hand, `switch` doesn't support power control and mobility protocols. Protocols are presented by one or more architectures in an entity. For example, call admission protocol has three architectures in `zone`. Horizontal arrows show the communication between one architecture to its peer architecture in other entities. Shaded architectures are the last architectures in hierarchy and bound to their respective entities.

Figure 3.7 shows the base entities, i.e. `MS`, `BS`, `zone`, and `switch` with their inherited hierarchical architectures. Not all entities of WiPPET$_{\text{session}}$ have the entire seven layer architecture shown in Figure 3.6 and it depends on the involvement of the entities in the particular protocol. In Figure 3.7, the horizontal arrows show the interaction (event/information exchange through TeD channels) between two architectures in different entities. The vertical arrows show the direction of inheritance in the hierarchy of architectures. Mobility is the primary architecture in `MS`, `BS` and `zone`. The call admission protocol is composed of three algorithms: 1) base station assignment 2) call admission channel allocation (CA-channel allocation), and 3) call admission. Each algorithm is implemented as an architecture in related entities. Only `zone` has the base station assignment architecture. `BS` and `zone` entities have the channel allocation architecture. All base entities have the call admission architecture, and in fact, call admission is the primary architecture in `switch`. `BS`, `MS` and `zone` have the power control architecture. Each handoff protocol is composed of two algorithms: 1) handoff channel allocation (HO-channel allocation), and 2) handoff. These two algorithms are implemented as two separate architectures. `BS` and `zone` have handoff channel allocation architecture. All base entities have handoff architecture. All these protocols and algorithms are explained in detail later in section 3.7.

3.5 Design Issues

WiPPET$_{\text{session}}$ entities employs the *inherited hierarchical architecture* design method. This is not the only method to design an entity architecture in TeD. Another design method to implement an entity architecture is called *component architecture.* In this section we discuss the design issues of an entity architecture by comparing these two design methods: inherited hierarchical architecture and component architecture.

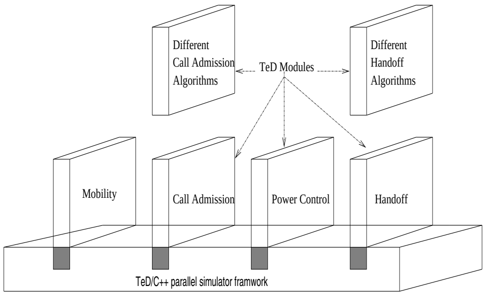


Figure 3.8: Example of modular (plug and play) architecture of simulator.

Initially, the design goal envisioned the simulator having *plug and play* architecture as shown in Figure 3.8, which can be achieved by the component architecture design. In this design, an architecture can be easily replaced with a new existing architecture without modifying the simulation code. According to the Figure 3.8, TeD/C++ parallel simulator engine acts as a backplate on which different TeD architectures reside as modules. During the development process, we learned that this design has certain disadvantages, which prevent to achieve this goal.

In component architecture design method, one can build one or more entities inside a parent entity. These entities are called sub-entities of the parent entity [section 2.1]

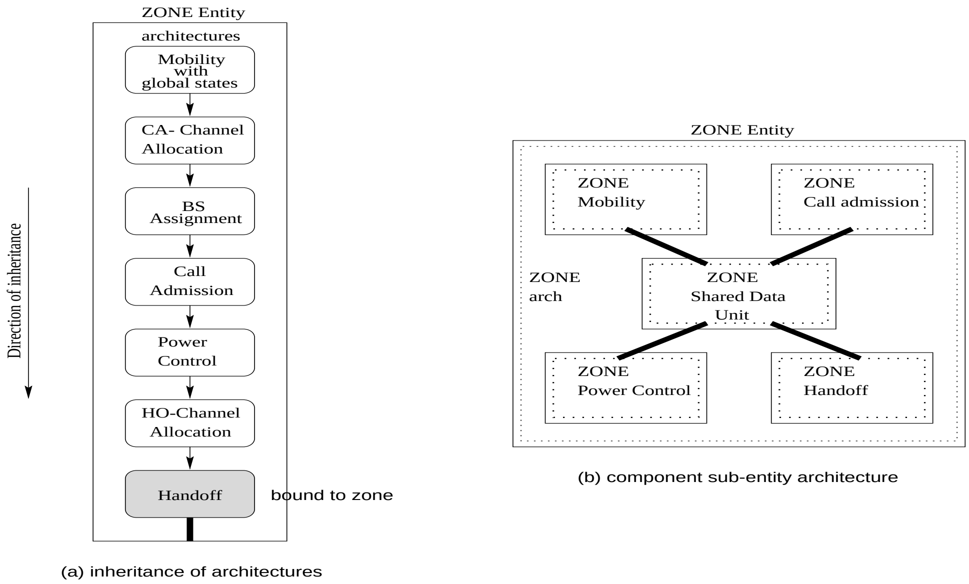


Figure 3.9: (a) Inheritance of architectures in `zone`. Mobility is the primary architecture and contains all common states and constants. The common states and constants are accessible to all hierarchical architectures in an entity. All other architectures inherit mobility and so does its states and constants. (b) component architecture in `zone`. Protocols are represented by sub-entities inside the `zone` (parent entity). Each protocol entity is connected to common data entity through a TED channel. During a simulation, these sub-entities access or update common states or constants by sending query events to the common data entity. The common data entity replies with the requested information using result-events.

and treated as components of the parent entity architecture. Considering above design method, a sub-entity can used to represent a protocol instead of an architecture and called *protocol entity*. The protocol entity resides inside the parent entity. Figures 3.9(a) and 3.9(b) show the inherited architectures and component sub-entity architectures of the `zone` respectively. Unlike in inherited architecture design, in component architecture design, mobility, call admission, power control and handoff protocols are represented by the sub-entities as components of the `zone` architecture as shown in Figure 3.9.

In WIPPET, each protocol uses common data states and constants during a simulation. A *common state* or a *constant* is defined as a state or a constant which is visible or accessible to all protocols of an entity. Hence, each entity architecture should have common states and constants accessible to all protocols or algorithms. In contrast, the current version of the TED framework does not support common states or constants, because all states and constants of an entity are encapsulated (i.e. defined *private*), and are not visible to the other entities. In component architecture design, each protocol is represented by a sub-entity. These protocol sub-entities encapsulate their states and constants. Hence, to satisfy the need of common data states and constants, there should be a common data sub-entity within the parent entity, which contains all common data states and constants. Whenever a protocol sub-entity wants to know or to update common data states or constants, it has to request the common data sub-entity using an event, which is called *query event*. Each sub-entity has TED channels to its parent entity and the common data sub-entity to communicate with the parent entity and with the common data sub-entity respectively. This greatly increases the number of TED channels. In addition, there will be lots of unnecessary exchanges of query events. Since event processing in TED is expensive, the component architecture design is not efficient.

In the inherited hierarchical architecture design, a new architecture with added functionality can be built by inheriting an old architecture and adding more methods or processes in it. The new (child) architecture has all states, constants and processes of the old (parent) architecture [section 2.1]. In WIPPET, this property of inherited architecture design method is exploited to resolve the problem of common states and constants. If the common states and constants are defined in a parent architecture (the first architecture of the hierarchy of inheritance), these common states and constants can be accessed or updated by its child or inherited architecture. $\text{WIPPET}_{\text{session}}$ employs the inherited hierarchical architecture design method, in which each protocol architecture other than the primary is built by inheriting the primary architecture. The common constants and states are defined in the primary architecture of all base entities of $\text{WIPPET}_{\text{session}}$, hence they are visible to all architectures of the base entities. Examples of the common state and constant structures stored in the primary mobility architecture of a `zone` are *channel state*, *path loss matrix*, and *MS state* (see appendix B), which are visible to all other architectures of the `zone`. This is one of the advantages of inherited hierarchical architecture design.

The disadvantage of this approach is that to replace a new architecture with an old one, one has to modify the simulation code to change the inheritance from the old architecture to the new architecture and then, one has to recompile the code. Hence, the inherited architecture design does not provide the perfect modularity or plug and play. On other hand, in the component architecture design, each protocol is represented by an entity and the protocol functionality is encapsulated in that entity architecture. One can replace an old architecture with a new one by changing the binding of protocol entity from its old architecture to a new architecture. This can be done in the model configuration file (model.cfg) without modifying and recompiling the simulation code. Hence, the component architecture design supports plug and play modularity.

3.6 Compatibility Constraints of an Architecture

In WiPPET, each architecture receives and/or sends two types of events, 1) *Inter-Entity Interface* event and; (2) *Intra-Entity Interface* event.

- Inter-Entity Interface (Inter-EI) event is an event passing between the same protocol architecture of two different entities through a TeD channel.
- Intra-Entity Interface (Intra-EI) event is an event passing between two architectures of the same entity through an internal TeD channel.

In the example shown in Figure 3.10, there are two channel allocation architectures (CA-channel allocation). One is in `BS` and the other is in `zone` and one is the peer of the other. These two channel allocation architectures communicate using Inter-EI events during a simulation as shown by horizontal bidirectional arrow. Figure 3.10 also shows two events EVDCA_REQ[2] and EVDCA_RES passing between the channel allocation and the call admission architecture of a `BS`. These two events are called Intra-EI events.

[2]The names of the events that appear in san serif font match exactly the events name as they appear in the source code of WiPPET.

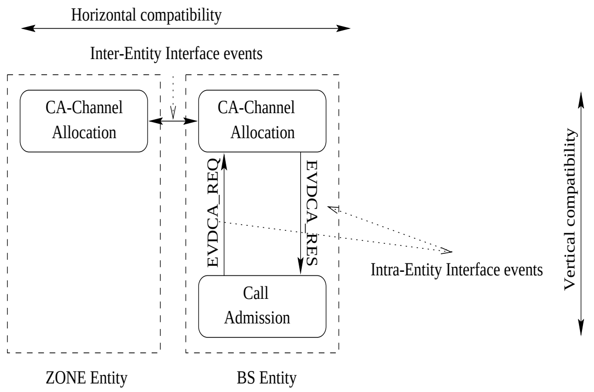


Figure 3.10: An example of inter-entity and intra-entity interface events. Vertical arrows show the direction of flow of intra-entity interface events between two architectures of an entity. Similarly, horizontal arrows show the direction of flow of inter-entity interface events between two architectures of different entities.

When a mobile station requests for a traffic channel, the call admission architecture at the BS sends EVDCA_REQ to the channel allocation architecture. The channel allocation architecture receives the request and runs a channel allocation algorithm to come up with a traffic channel. Then, the channel allocation architecture sends EVDCA_RES with the traffic channel number, which will be assigned to the mobile station by the base station.

Here, the job of the channel allocation architecture is to find a traffic channel for a mobile station on a request from the call admission architecture. In the $\mathrm{WiPPET}_{\mathrm{session}}$ model data base, there are two channel allocation architectures available based on two algorithms; 1) LI-DCA and , 2) ACP-DCA. Regardless of any channel allocation algorithm, a channel allocation architecture must send a traffic channel number for a mobile station on a request from the call admission architecture. Thus, to run the call admission protocol, any channel allocation architecture should understand or interpret the request from the call admission architecture sent by EVDCA_REQ and it should send EVDCA_RES with a traffic channel number to the call admission architecture. The call admission protocol will fail, if someone replaces a new channel allocation architecture, which does not understand EVDCA_REQ and hence, does not send EVDCA_RES with a traffic channel number. Here, the issue is the *compatibility* of two architectures. Two architectures we said to be compatible if each architecture can understand or interpret an event sent by the other architecture.

In the above example, we discussed compatibility of two architectures within an entity. The compatibility of two architectures within the entity along vertical direction is called *vertical compatibility*. Vertical compatibility means the architecture understands the Intra-EI events from other architectures within an entity. Horizontal compatibility means an architecture understands Inter-EI events. Figure 3.10 shows an example of the inter-entity (Horizontal compatibility) and intra-entity (Vertical compatibility) interface events. The next subsection discusses the Intra-EI events involved in all protocol architectures of WIPPET$_{\text{session}}$entities, which helps to build new compatible architectures. All Inter-EI events are discussed in section 3.7 separately. Before we start the subsection, let us define some terminology used in the discussion.

- *Parent Base Station* (parent `BS`) : A base station with which a `MS` is currently connected.
- *Parent Zone* (parent `zone`) : A `zone` in which a `MS` is currently active.
- *New Base Station* (new `BS`) : A candidate or new base station assigned to a `MS` after a successful handoff.
- *Timer event*: To initiate or activate a procedure or a process within an entity, an event is sent on internal TED channel with delay of required time duration after which that procedure should be activated.

3.6.1 Handoff

Let us start with the handoff protocol. Figure 3.11 shows the intra-entity interface events involved in handoff protocol. Each base entity (`MS`, `zone`, `BS`, or `switch`) has a part or an architecture of it. Only `BS` handoff architecture has Intra-EI events as shown in Figure 3.11.

An event EVHANDOFF_CHECK triggers handoff check procedure in a `BS` for a `MS`. The `MS` is recognized by its `MS` ID carried by the event as a payload. This is a timer event.

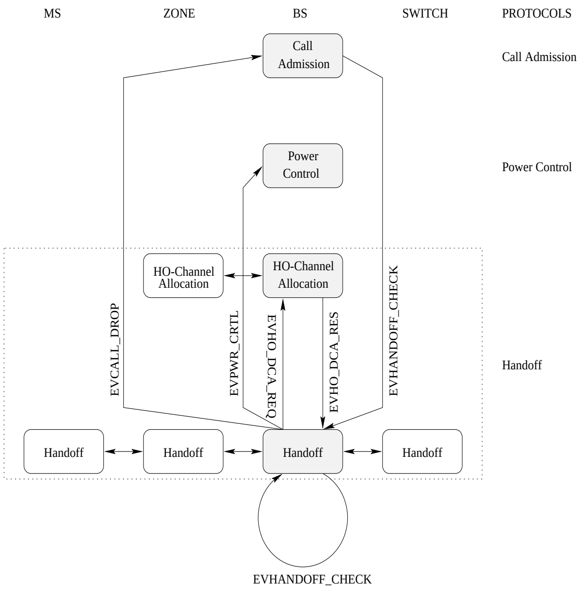


Figure 3.11: Intra-entity interface events in handoff protocol. Shaded areas represent architectures of a BS, which receive or send intra-entity interface (Intra-EI) events in handoff protocol.

At the end of the handoff check procedure, at the end, the architecture sends this timer event to activate itself after a certain delay, which will trigger the next handoff check procedure. The delay is called the handoff check interval and it is a simulation parameter. Hence, a timer event satisfies the need of a self-timer to activate a process periodically. When a MS is admitted, the call admission architecture at parent BS sends this event to its handoff architecture to initiate handoff check procedure. Similarly, when a MS hands in to a new BS, the handoff architecture at the new BS sends this event to itself to initiate handoff check procedure. While the call is in progress, the handoff architecture at parent BS sends this event to itself periodically to decide handoff condition for the given MS.

Events EVHO_DCA_REQ and EVHO_DCA_RES are a pair of Intra-EI events between the handoff and the handoff call admission architectures in the new BS. The event EVHO_DCA_REQ has the hand-in MS ID as a payload. During handoff, the handoff architecture at the new BS sends EVHO_DCA_REQ to the channel allocation architecture requesting a radio channel for a hand-in MS. The handoff channel allocation architecture sends EVHO_DCA_RES with the requested information as a payload. This information includes a radio channel, required MS transmitter power to satisfy the target SIR on the uplink, interference power and gain from the MS to the new BS.

An event EVPWR_CRTL triggers an uplink power control iteration in the parent BS for the MS. When a MS hands-in to a new BS, the handoff architecture at the new BS sends this event to its power control architecture to initiate a power control iteration.

An event EVCALL_DROP is an Intra-EI event sent by the handoff architecture to the call admission architecture in a parent BS. This event contains the dropped MS ID as a payload. The call admission architecture at parent BS receives this event and removes the dropped MS from the active list.

As we see the handoff architecture interacts with other architectures using internal TED channels. We get a better understanding of the interactions among architectures when we examine specific protocols in section 3.7.

3.6.2 Power control

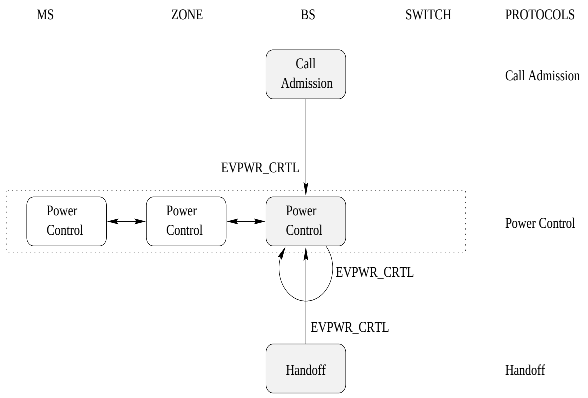


Figure 3.12: Intra-entity interface events in power control protocol. Shaded areas represent architectures of a BS, which receive or send intra-entity interface (Intra-EI) events in power control protocol.

Figure 3.12 shows Intra-EI events involved in the power control protocol. The base entities MS, zone, and BS have the power control architectures. Only the BS power control architecture has Intra-EI events as shown in Figure 3.12.

The event EVPWR_CRTL has a payload of MS ID. As we discussed in the previous subsection, the event EVPWR_CRTL triggers an uplink power control iteration in the parent BS for the MS. It is a timer event. When a MS is admitted, the call admission architecture at parent BS sends this event to its power control architecture to initiate power control iterations. Similarly, when a MS hands in to a new BS, the handoff architecture at the new BS sends this event to its power control architecture to start power control iterations for the hand-in MS. While the call is in progress, the power control architecture at the parent BS sends this event to itself periodically to control uplink power of the MS. The power control algorithm is explained in subsection 3.7.6.

3.6.3 Call admission

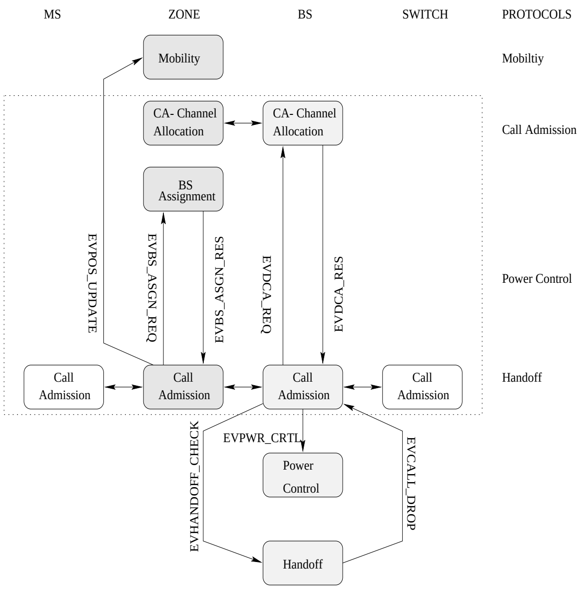


Figure 3.13: Intra-entity interface events in call admission protocol. Shaded areas represent architectures of a BS and a zone, which receive or send intra-entity interface (Intra-EI) events in call admission protocol.

Figure 3.13 shows Intra-EI events involved in call admission protocol. Each base entity (MS, zone, BS, or switch) has the call admission architecture. Only the BS and the zone call admission architectures have Intra-EI events as shown in Figure 3.13.

As we discussed earlier, EVPWR_CRTL is generated by the call admission architecture of the parent BS after successful admission of a MS. Similarly, EVHANDOFF_CHECK is also generated by the call admission architecture of the parent BS after successful admission of the given MS.

There are two separate channel allocation architectures; 1) call admission channel allocation (CA-channel allocation) and, 2)handoff channel allocation (HO-channel allocation). Events EVDCA_REQ and EVDCA_RES are a pair of Intra-EI events between the call admission architecture and the CA-channel allocation architecture of a BS entity. EVDCA_REQ has a payload of hand-in MS ID. The call admission architecture at the parent BS sends EVDCA_REQ to the CA-channel allocation architecture requesting a radio channel for a new MS. The CA-channel allocation architecture sends EVDCA_RES including requested information as a payload. This information includes the assigned radio channel ID, the required MS transmitter power to satisfy the target SIR on the uplink, the interference power, and the gain from the MS to its parent BS.

Events EVBS_ASGN_REQ and EVBS_ASGN_RES are a pair of Intra-EI events between the call admission and the base station assignment architectures of a zone. The call admission architecture at the parent zone sends EVBS_ASGN_REQ to the base station assignment architecture to find a BS for the new MS. EVBS_ASGN_REQ has a payload of the MS ID for which zone has to find a BS. As we discussed earlier in section 3.2, the zone uses the path loss matrix and the location of the given MS to find the strongest pilot BS for the MS. Hence, the base station assignment architecture finds the strongest pilot BS and sends EVBS_ASGN_RES including the strongest pilot BS ID to the call admission architecture.

An event EVPOS_UPDATE is an Intra-EI event between the call admission architecture and the mobility architecture of a zone. EVPOS_UPDATE has MS ID as a payload. When a MS is admitted, the call admission architecture at the parent zone sends this event to the mobility architecture to initiate the mobility or position update procedure for the MS.

As we discussed earlier, EVCALL_DROP is an Intra-EI event sent by the handoff architecture to the call admission architecture of a parent BS. The call admission architecture receives this event and removes the dropped MS from the active list.

3.6.4 Mobility

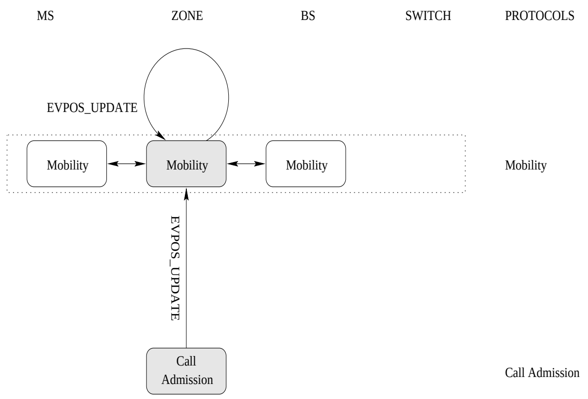


Figure 3.14: Intra-entity interface events in mobility protocol. Shaded areas represent architectures of a `zone`, which receive or send intra-entity interface (Intra-EI) events in mobility protocol.

Figure 3.14 shows Intra-EI events involved in mobility protocol. As we discussed earlier, EVPOS_UPDATE initiates position update procedure in a `zone` for a `MS`. The `MS` is recognized by its `MS` ID carried by the event as a payload. When a `MS` is admitted, the call admission architecture at the parent `zone` sends this event to the mobility architecture to initiate position update for the new `MS`. While the call is in progress, the mobility architecture at the parent `zone` sends this event to itself periodically to update the `MS` position on the geography. This is a timer event. The mobility architecture schedules the next position update for the `MS` by sending this timer event to itself and becomes active after a certain delay. The period of position updates depends on the `MS` speed and the distance resolution of the geography as shown in equation 3.6.

$$\text{Period of position updates (sec)} = \frac{\text{Distance resolution (meters)}}{\text{MS speed (meters/sec)}} \tag{3.6}$$

3.7 Protocols and Algorithms

WiPPET$_{\text{session}}$ models session or call level traffic with mobility, call admission, power control and handoff protocols. In this section, all the above mentioned protocols are discussed with implementation details and event flow interaction diagrams. We start with the call admission protocol.

3.7.1 Call Admission

A new arriving mobile station requires a new communication link, i.e. a base station and a radio channel. To guarantee a certain degree of link quality for the active calls, as well as low outage and blocking probabilities, admission control must be carefully exercised. Call admission control is closely related to channel allocation. Different channel allocation schemes imply different admission control strategies.

The call admission protocol is composed of three architectures; 1) channel allocation, 2) base station assignment and, 3) basic call admission. There are two channel allocation architectures in WiPPET$_{\text{session}}$ model library based on two distinct channel allocation algorithms: Least-Interference Dynamic Channel Allocation (LI-DCA) and Active Channel Preservation Dynamic Channel Allocation (ACP-DCA). Any one of these architectures can be represented as the channel allocation architecture in call admission protocol. WiPPET$_{\text{session}}$ also has a base station assignment architecture based on the strongest pilot base station assignment algorithm. Above two channel allocation algorithms and the base station assignment algorithm are explained later in this section. Figure 3.15 shows event flow and interaction diagram in call admission protocol. The call admission protocol is distributed among entities `callgen`, `MS`, `BS`, `zone` and `switch`.

The life cycle of a call or a session begins at the `callgen`, where EVCALL_INIT is sent to a `MS`. The `MS` receives EVCALL_INIT and becomes active. To initiate a call or a session, the `MS` has to acquire a base station and a traffic channel. Hence, it sends EVCALL_REG_REQ to a `zone`. This is an action comparable to the action in a real network when a mobile station sends a call request on control channel.

The call admission architecture of the `zone` receives EVCALL_REG_REQ from the `MS`. Then, the `zone` sends EVBS_ASGN_REQ to its base station assignment architecture. The base station assignment architecture finds the strongest pilot `BS` ID for the `MS` and sends EVBS_ASGN_RES to the call admission architecture with the strongest pilot `BS` ID as the event payload. Both these events EVBS_ASGN_REQ and EVBS_ASGN_RES are Intra-EI events discussed in previous section. This completes the base station

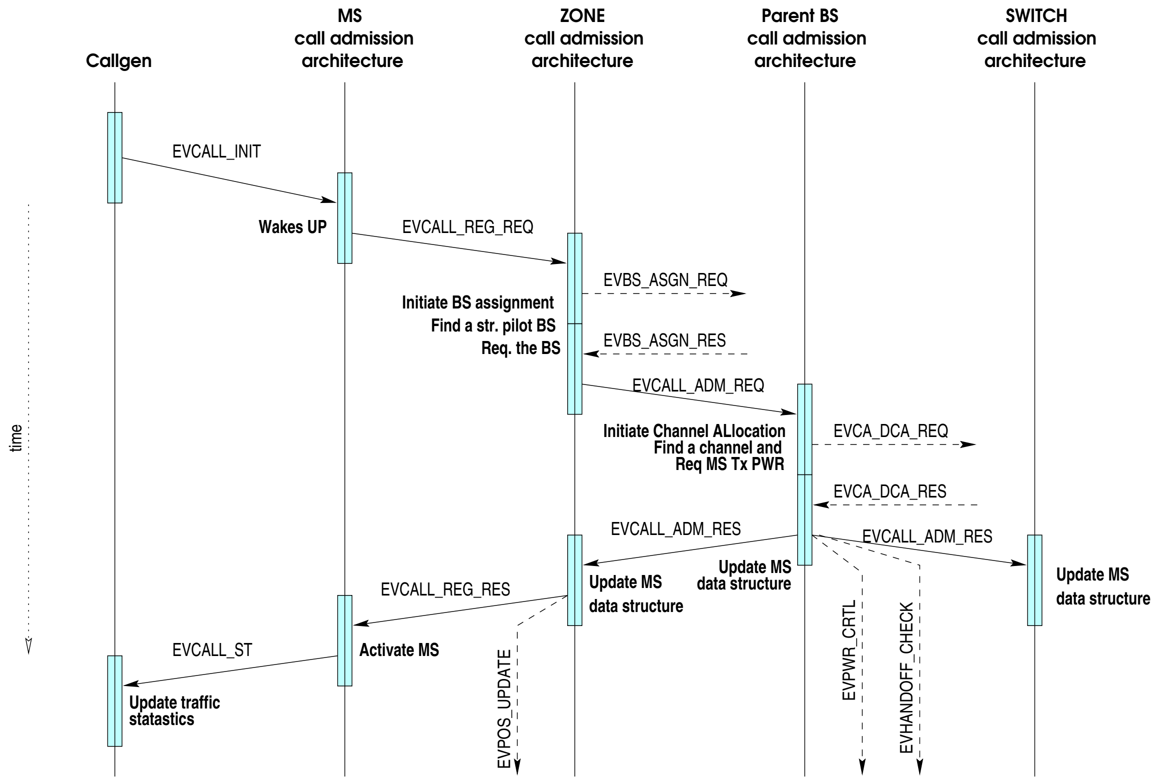


Figure 3.15: Event flow and entity interaction diagram of call admission protocol. A vertical shaded strip represents a process of a call admission architecture in an entity, which receives and/or sends Inter-EI events to the processes of its peer architectures or Intra-EI events to the processes of architecture within its entity. Solid-line arrows show the direction of flow of Inter-EI events from a call admission architecture to its peer in other entity. Broken-line arrows show the direction of flow of Intra-EI events from a call admission architecture to an architecture within an entity.

assignment phase. The next phase is channel allocation.

The call admission architecture in the zone receives the strongest BS ID for the MS. Then, the zone sends an admission request to the strongest pilot BS using EVCALL_ADM_REQ with the given MS ID. The call admission architecture in the BS receives EVCALL_ADM_REQ and invokes the channel allocation algorithm by sending EVCA_DCA_REQ to its channel allocation architecture. The channel allocation architecture in the strongest BS sends EVCA_DCA_RES to its call admission architecture with the required MS transmitter power and, an assigned channel ID as event payload. If the MS is not admitted or blocked, the channel allocation architecture sends NO_CHANNEL as the channel ID. Both these events EVCA_DCA_REQ and EVCA_DCA_RES are Intra-EI events as discussed in the previous section. The call admission architecture receives EVCA_DCA_RES and if the MS is admitted, the call admission architecture at the parent BS sends two timer events EVPWR_CRTL and EVHANDOFF_CHECK to the power control and the handoff architectures respectively.

The BS acknowledges the zone about the admission result by sending EVCALL_ADM_RES with the channel ID and required transmitter power of the MS. The call admission architecture in the zone receives EVCALL_ADM_RES, and if the MS is admitted, it sends the timer event EVPOS_UPDATE to the mobility architecture to initiate the MS motion. The zone also updates the MS state with the assigned parent BS ID, the assigned channel ID and the required transmitter power. The zone also acknowledges the MS about the call admission result by sending EVCALL_REG_RES with the parent BS ID, the channel ID and the required transmitter power.

The MS receives the EVCALL_REG_RES and sends call status report to callgen by sending EVCALL_ST. If the MS is admitted, it sends ADMISSION as call status report, otherwise it sends BLOCK as call status report to callgen. Depending on call status report from the MS, callgen will update traffic statistics such as probability of call blocking and probability of call dropping.

3.7.2 Least Interference based Dynamic Channel Allocation (LI-DCA)

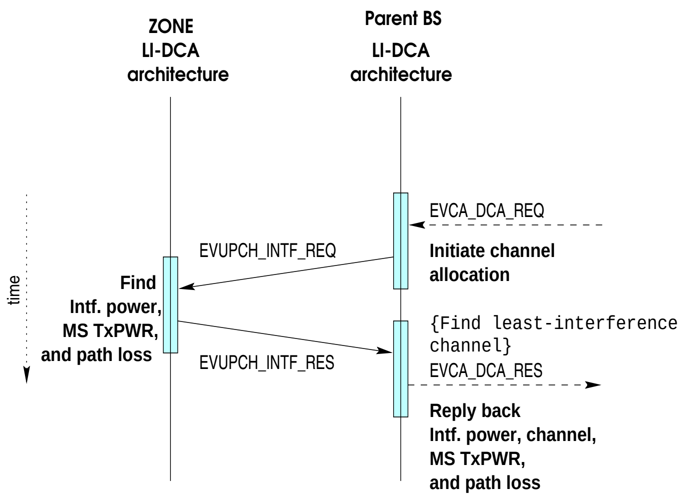


Figure 3.16: Event flow in least interference based dynamic channel allocation (LI-DCA) algorithm. Vertical shaded strips represent processes of channel allocation architectures in a `zone` and a `BS`, which receive and/or send Inter-EI and/or Intra-EI events. Solid-line arrows show the direction of flow of Inter-EI events from a channel allocation architecture to its peer in other entity. Broken-line arrows show the direction of flow of Intra-EI events from the channel allocation architecture in `BS` to an architecture within an entity.

From Figure 3.7, there are two layers of the channel allocation architectures, one is associated with the call admission protocol and the other with the handoff protocol. The reason is that one can simulate networks with a different type of channel allocation in the call admission than in the handoff protocol. The channel allocation architectures associated with the call admission protocol are called the CA-channel allocation and similarly, the channel allocation architectures associated with the handoff protocol are called HO-channel allocation. Here, we only discuss the CA-call admission and related events. The HO-channel allocation is the same as the CA-call admission (i.e. same event flow and procedure) but it has different event name prefixes to distinguish events of the two architectures. For example, the call admission architecture in a parent `BS` invokes CA-channel allocation algorithm by sending EVCA_DCA_REQ, and similarly, the handoff architecture invokes HO-channel allocation by sending EVHO_DCA_REQ. Figure 3.16 shows the event flow and interaction diagram of the LI-DCA channel allocation algorithm.

The CA-channel allocation architecture in the parent BS sends a interference measurement request EVUPCH_INTF_REQ to its peer CA-channel allocation architecture in the zone. The zone replies back using EVUPCH_INTF_RES including a list of interference values on every possible radio channel and the path loss value $g_{ds}(\mathrm{MS}_{\mathrm{location}}, \mathrm{BS}_{\mathrm{parent}})$ from the MS to its parent BS. The CA-channel allocation architecture in the parent BS receives EVUPCH_INTF_RES and initiates least interference based dynamic channel allocation (LI-DCA) algorithm to find the least interference value channel ($\mathrm{ch}_{minIntf}$) for the MS. The algorithm also finds the required MS transmitter power ($\mathrm{msTxPWR}_{req}$) to meet the uplink quality by using equation 3.7.

$$\mathrm{msTxPWR}_{req}(\mathrm{ch}_{minIntf}) = 10^{\frac{\Gamma_{new}}{10}} \cdot \frac{g_{ds}(\mathrm{MS}_{\mathrm{location}}, \mathrm{BS}_{\mathrm{parent}})}{P_{minIntf}(\mathrm{BS}_{\mathrm{parent}}, \mathrm{ch}_{minIntf})} \tag{3.7}$$

where $P_{minIntf}(\mathrm{BS}_{\mathrm{parent}}, \mathrm{ch}_{minIntf})$ is the least interference value on channel $\mathrm{ch}_{minIntf}$ for the MS at the parent BS. Note that the simulation parameter Γ_{new} is the required uplink signal to interference ratio.

Next, the CA-channel allocation architecture in the parent BS sends EVCA_DCA_RES with the $\mathrm{msTxPWR}_{req}$ and the least interference channel ID $\mathrm{ch}_{minIntf}$ to the call admission architecture. If the channel allocation algorithm fails, i.e. the call is blocked, the BS sends NO_CHANNEL as a channel ID. There are two reasons for call blocking: the required MS transmitter power is greater than the maximum allowable transmitter power (MAX_MS_PWR), or the parent BS has no channel to allocate.

3.7.3 Active Channel Preservation based Dynamic Channel Allocation (ACP-DCA)

In this channel allocation algorithm, whenever a new MS arrives, a BS attempts to predict what will be the quality of the other active links after the new MS has been admitted. Figure 3.17 shows the event flow and interaction diagram of ACP-DCA channel allocation algorithm.

The CA-channel allocation architecture at the parent BS initiates the ACP-DCA algorithm by sending a interference measurement request EVACP_UPCH_INTF_REQ to

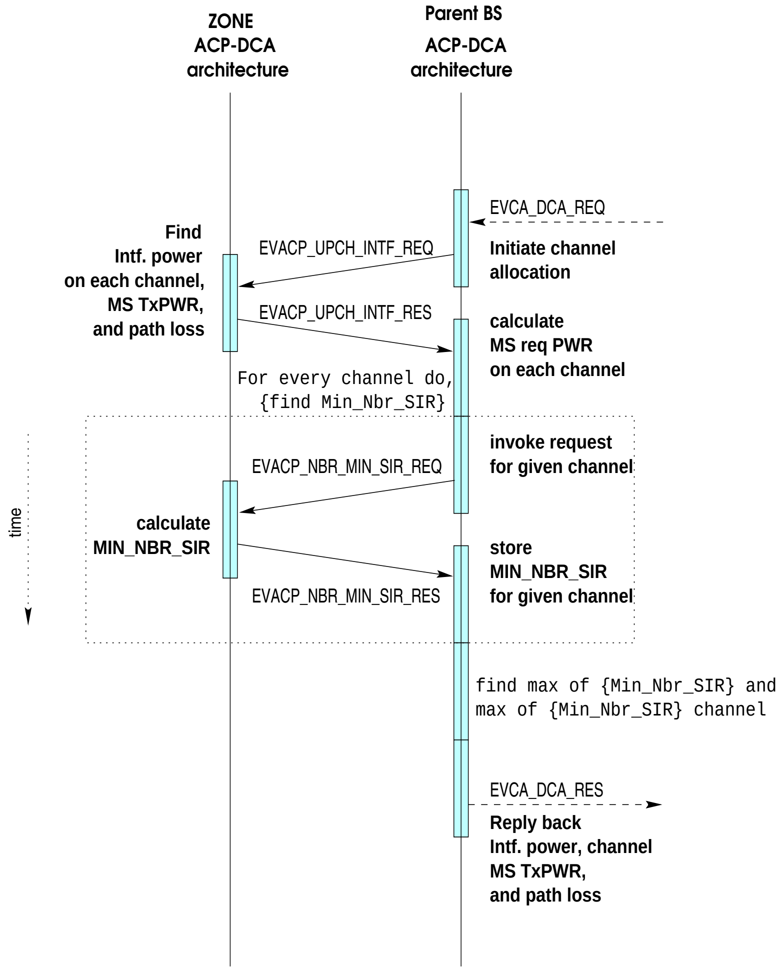


Figure 3.17: Event flow in Active Channel Preservation based Dynamic Channel Allocation (ACP-DCA) algorithm. Vertical shaded strips represent processes of channel allocation architectures in a `zone` and a `BS`, which receive and/or send Inter-EI and/or Intra-EI events. Solid-line arrows show the direction of flow of Inter-EI events from a channel allocation architecture to its peer in other entity. Broken-line arrows show the direction of flow of Intra-EI events from the channel allocation architecture in the `BS` to an architecture within an entity.

the zone. The zone sends EVACP_UPCH_INTF_RES with a list of interference values $P_{Intf}(\mathrm{BS}_{\mathrm{parent}}, \mathrm{ch})$ at the parent BS on every channel for the new MS and the path loss value $g_{ds}(\mathrm{MS}_{new}, \mathrm{BS}_{\mathrm{parent}})$ from the new MS to the parent BS. The parent BS receives EVACP_UPCH_INTF_RES and calculates the required MS transmitter power ($\mathrm{msTxPWR}_{req}$) for every channel:

$$\mathrm{msTxPWR}_{req}(\mathrm{ch}) = 10^{\frac{\Gamma_{new}}{10}} \cdot \frac{g_{ds}(\mathrm{MS}_{new}, \mathrm{BS}_{\mathrm{parent}})}{P_{Intf}(\mathrm{BS}_{\mathrm{parent}}, \mathrm{ch})}$$

where $P_{Intf}(\mathrm{BS}_{\mathrm{parent}}, \mathrm{ch})$ is the interference value on channel ch for the MS at the parent BS. The parent BS stores these required transmitter power values in its states.

At this point, the parent BS tries to find a channel for the new MS such that the quality of the other active links after the new MS has been admitted will not be below certain threshold. For that the parent BS has to estimate the uplink quality of all active MSs on every channel by taking into account the interference due to the new MS on that channel.

The parent BS sends EVACP_NBR_MIN_SIR_REQ to the zone with a channel ID ch and the required MS transmitter power $\mathrm{msTxPWR}_{req}(\mathrm{ch})$ on that channel.

The zone receives the event EVACP_NBR_MIN_SIR_REQ and calculates the uplink SIR values of all active MSs on the channel ch assuming that the new MS is connected to a BS and transmitting msTxPWR(ch) on the same channel ch. For example, suppose there are N mobile stations MS_n connected to BS_n respectively through the channel ch, where $n = 1, 2, ..., N$. Suppose a new mobile station MS_{new} is admitted by a base station BS_{new} and it is active on the same channel ch and transmitting $\mathrm{msTxPWR}_{req}(\mathrm{ch})$. First the zone calculates the received signal power of each active mobile station:

$$P_{RX_S}(\mathrm{MS}_i, \mathrm{BS}_i, \mathrm{ch}) = g_{ds}(\mathrm{MS}_i, \mathrm{BS}_i) P_{TX_S}(\mathrm{MS}_i, \mathrm{ch})$$

where $P_{TX_S}(\mathrm{MS}_i, \mathrm{ch})$ is the transmitter power of the active mobile station MS_i on channel ch, and $g_{ds}(\mathrm{MS}_i, \mathrm{BS}_i)$ is the path loss value from the MS_i to its parent BS_i. Next, the zone calculates the sum of the interference powers for the active mobile station MS_i at its parent BS_i:

$$P_{RX_I}(\mathrm{BS}_i, ch) = \sum_{n \neq i} g_{ds}(\mathrm{MS}_n, \mathrm{BS}_i) P_{TX_S}(\mathrm{MS}_n, \mathrm{ch}) + g_{ds}(\mathrm{MS}_{new}, \mathrm{BS}_i)\mathrm{msTxPWR}_{req}(\mathrm{ch})$$

Using the received signal powers and the received interference powers, the `zone` calculates the uplink SIR for the mobile station MS_i using equation 3.8.

$$SIR(\mathrm{MS}_i, \mathrm{ch}) = \frac{P_{RX_S}(\mathrm{MS}_i, \mathrm{BS}_i, \mathrm{ch})}{P_{RX_I}(\mathrm{BS}_i, \mathrm{ch}) + \eta^2} \tag{3.8}$$

Similarly, the `zone` calculates the uplink SIR values for all active mobile stations MS_n for $n = 1, 2, ..., N$, on the channel ch and stores their N uplink SIR values $SIR(\mathrm{MS}_n, \mathrm{ch})$, for $n = 1, 2, ..., N$. Then, it finds the minimum uplink SIR on the channel ch from those N uplink SIR values.

$$SIR_{min}(\mathrm{ch}) = \min_{\mathrm{MS}_i}\{SIR(\mathrm{MS}_i, \mathrm{ch})\}; \quad \mathrm{i{=}1,...,}N$$

The `zone` sends EVACP_NBR_MIN_SIR_RES to the parent `BS` with the minimum SIR value on the channel ch ($SIR_{min}(\mathrm{ch})$). The channel allocation architecture of the parent `BS` receives EVACP_NBR_MIN_SIR_RES and stores the minimum SIR values ($SIR_{min}(\mathrm{ch})$).

Similarly, the parent `BS` collects the minimum uplink SIR ($SIR_{min}(\mathrm{ch}_i)$) on every channel for i = 1 to MaxCh. Then, the parent `BS` finds a channel having the maximum of the minimum uplink SIR values,

$$SIR_{maxofmin}(\mathrm{ch}_{max}) = \max_i\{SIR_{min}(\mathrm{ch}_i)\}$$

If $SIR_{maxofmin}(\mathrm{ch}_{max}) \geq \Gamma_{new}$ (the required SIR threshold), the channel ch_{max} having $SIR_{maxofmin}$ is assigned to the MS_{new}. Otherwise the ACP-DCA algorithm fails. The channel allocation architecture at the parent `BS` assigns NO_CHANNEL as a channel ID and sends EVCA_DCA_RES with the required `MS` transmitter power on the channel ch_{max} ($\mathrm{msTxPWR}(\mathrm{ch}_{max})$) and the channel ID ch_{max} to the call admission architecture.

3.7.4 Strongest Pilot Base Station Assignment

The base station assignment is a part of the call admission protocol. Only the `zone` contains the base station assignment architecture as shown in Figure 3.7. As we discussed earlier in subsection 3.7.1, the call admission architecture in a `zone` requests the

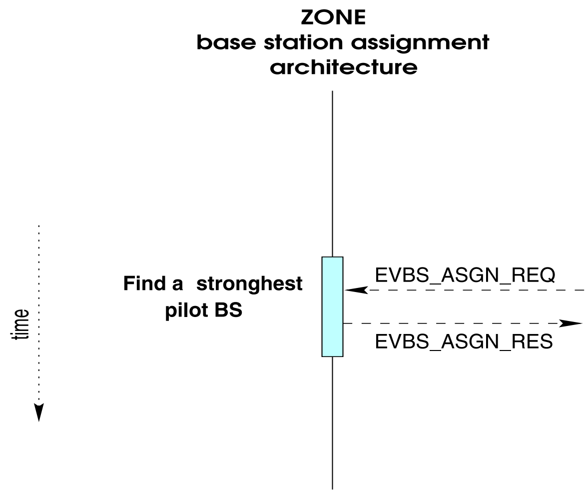


Figure 3.18: Event flow in strongest pilot base station assignment algorithm . A vertical shaded strip represents a process of the base station assignment architecture in a `zone`, which receives and sends Intra-EI events. Broken-line arrows show the direction of flow of Intra-EI events from the base station assignment architecture to the call admission architecture in `zone`.

base station assignment architecture to find the strongest pilot `BS` for a given `MS`. Figure 3.18 shows the event flow and interaction diagram of the `BS` assignment algorithm.

The base station assignment architecture in a parent `zone` receives EVBS_ASGN_REQ on internal TED channel from the call admission architecture. As we discussed earlier in section 3.2, the `zone` uses the `MS` location stored in the `MS` mobility state and the point structure stored as its deferred constant to find the strongest pilot `BS`. Then, the base station assignment architecture replies back to the call admission architecture using EVBS_ASGN_RES including the strongest pilot `BS` ID for the given `MS` as a payload. For complete information about the deferred constants and the `MS` mobility state in the `zone`, see appendix B.

3.7.5 Mobility

Mobility is the process describing how mobile station moves from one point to another within the given modeled geography. Figure 3.19 shows the event flow and interaction diagram of mobility protocols.

During the call admission protocol, after a `MS` is admitted, the parent `zone` randomly

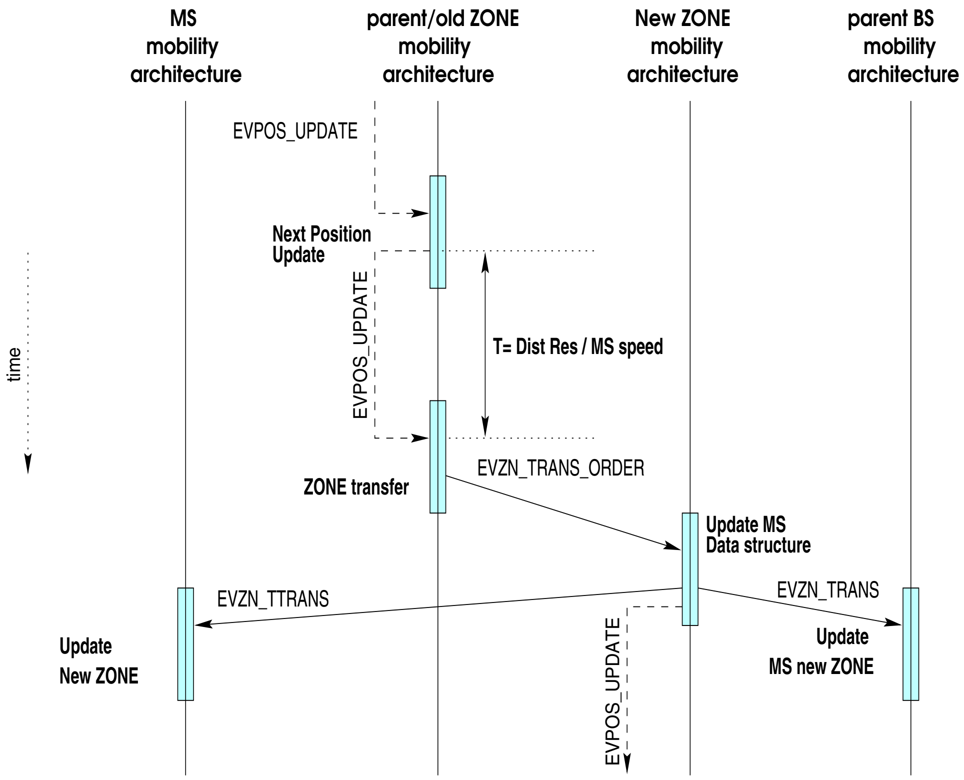


Figure 3.19: Event flow and entity interaction diagram of mobility protocol. A vertical shaded strip represents a process of a mobility architecture in an entity, which receives and/or sends Inter-EI events to the processes of its peer architectures or Intra-EI events to the processes of architecture within its entity. Solid-line arrows show the direction of flow of Inter-EI events from a mobility architecture to its peer in other entity. Broken-line arrows show the direction of flow of Intra-EI events from a mobility architecture to an architecture within an entity

selects an initial position of the MS. Then, as we discussed in section 3.7.1, the call admission architecture in the zone sends EVPOS_UPDATE to the mobility architecture to initiate mobile position update procedure. The mobility architecture in the zone receives EVPOS_UPDATE including the given MS ID as event payload. Then, the mobility architecture uses the MS mobility state and the point structure to update the position of the given MS on the geography as discussed earlier in section 3.2. The mobility architecture also schedules the next position updates by sending a timer event to itself EVPOS_UPDATE with delay equal to $\frac{\text{Distance resolution}}{\text{MS speed}}$. See B.1 and B.2 for complete information about the point structure and the MS mobility state in the zone.

In a multiple zone simulation, when a MS travels from one zone to another zone, a zone transfer occurs. Before sending the timer event EVPOS_UPDATE to schedule next position update, the mobility architecture checks for a possible MS zone transfer. If a zone transfer occurs, the old zone sends EVZN_TRANS_ORDER to the new zone and transfers all the MS data structures (i.e. the MS state, the MS mobility state) to the new zone. The new zone informs the MS and the parent BS about the zone transfer using EVZN_TRANS. Thereafter, the new zone becomes the parent zone of the MS and starts the MS mobility by sending a timer event EVPOS_UPDATE to its mobility architecture. See appendix B.1 for more details on the MS zone transfer and the related deferred constants in the zone.

3.7.6 Asynchronous Distributed Uplink Power Control

In uplink power control protocol, a MS uplink transmitter power is periodically adjusted to reduce the overall system interference level and maintain the required uplink quality. Figure 3.20 shows the event flow and interaction diagram of the asynchronous distributed uplink power control algorithm.

As we discussed in section 3.7.1, during the call admission protocol, after a MS is admitted, the call admission architecture in a parent BS sends EVPWR_CRTL to its power control architecture to initiate mobile power update procedure. The power control architecture in parent BS receives EVPWR_CRTL and initiates the power control iteration. It sends EVPWR_CRTL_REQ to the zone (if more than one zone, then it sends to all zones. Here, we assume a one zone simulation). The zone sends EVPWR_CRTL_RES including the path loss value ($g_{ds}(\mathrm{MS}_{locaton}, \mathrm{BS}_{\mathrm{parent}})$) from the MS to its parent BS and the interference value ($P_{Intf}(\mathrm{BS}_{\mathrm{parent}}, \mathrm{ch})$) on the MS uplink ch. The BS receives EVPWR_CRTL_RES and calculates the new required MS transmitter power ($\mathrm{msTxPWR}_{new}(\mathrm{ch})$) to meet uplink quality. First it calculates the required MS transmitter power:

$$\mathrm{msTxPWR}_{req}(\mathrm{ch}) = 10^{\frac{\Gamma_t}{10}} \cdot \frac{P_{Intf}(\mathrm{BS}_{\mathrm{parent}}, \mathrm{ch})}{g_{ds}(\mathrm{MS}_{\mathrm{location}}, \mathrm{BS}_{\mathrm{parent}})}$$

Then, the parent BS compares the ($\mathrm{msTxPWR}_{req}(\mathrm{ch})$) with the maximum and the

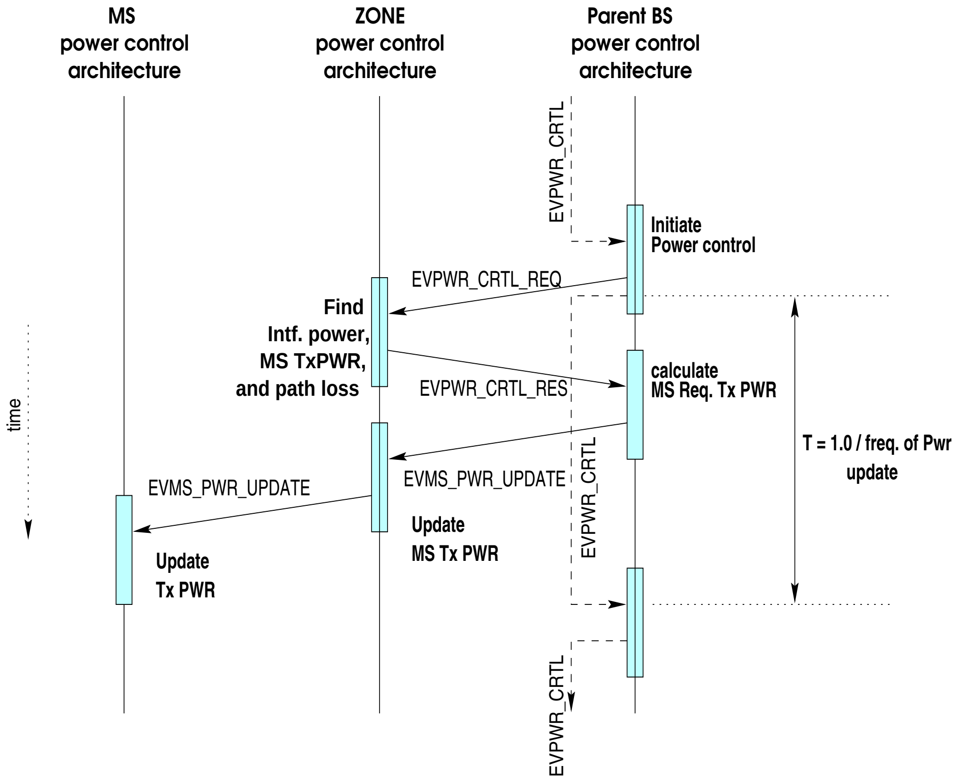


Figure 3.20: Event flow and entity interaction diagram of Asynchronous distributed uplink power control protocol. A vertical shaded strip represents a process of a power control architecture in an entity, which receives and/or sends Inter-EI events to the processes of its peer architectures or Intra-EI events to the processes of architecture within its entity. Solid-line arrows show the direction of flow of Inter-EI events from a power control architecture to its peer in other entity. Broken-line arrows show the direction of flow of Intra-EI events from a power control architecture to an architecture within an entity

minimum MS transmitter power limits and finds the new required MS transmitter power:

$$\mathrm{msTxPWR}_{new}(\mathrm{ch}) = \max\left(\mathrm{MIN\_MS\_PWR}, \min\left(\mathrm{MAX\_MS\_PWR}, \mathrm{msTxPWR}_{req}(\mathrm{ch})\right)\right)$$

Note that MIN_MS_PWR (minimum MS transmitter power) is 0 Watt, MAX_MS_PWR (maximum MS transmitter power) is 1 Watt, and $\mathrm{msTxPWR}_{new}$ is the new required MS transmitter power on uplink ch.

In a multiple zone simulation, there may be one or more MSs present in every zone using the same uplink channel. Here we consider the fact that a MS transmitting to its parent BS on a radio channel from one zone produces uplink interference on the same radio channel at BSs in other zones. Hence, to take into account the interference due to the interfering MSs, the parent BS requests all zones for interference values on a given MS uplink channel. The parent BS sums up the interference values from all zones and finds the total interference value on a MS uplink channel ($P_{Intf}(\mathrm{BS}_{\mathrm{parent}}, \mathrm{ch})$). Then, it computes the new required MS transmitter power to meet the uplink quality as discussed. The BS sends EVMS_PWR_UPDATE with the new required MS transmitter power to the parent zone. The zone updates the MS state and forwards the updated power to the MS using EVMS_PWR_UPDATE.

The BS power control algorithm also schedules next power control update by sending a timer event EVPWR_CRTL to itself with a certain delay. The delay of this timer event is equal to the reciprocal of the frequency of power update, which is a simulation parameter.

3.7.7 Handoff

Handoff is one of the radio resource management protocols. WiPPET$_{\mathrm{session}}$ implements six different handoff algorithms. Three are based on standards: AMPS , North American TDMA (IS-136) and GSM. The other three algorithms include Received Signal Strength Indicator Based Handoff (RSSIBHO), Signal to Interference Ratio Based Handoff (SIRBHO) and combined RSSI and SIR based handoff (RSSI.SIRBHO), which are derived from MADRAS (Mobile and Dynamic Resource Allocation Simulation) [3,4].

General description

There are four entities MS, zone, BS, and switch, which participate in a handoff protocol. In general, a handoff protocol is divided into four procedures and six different messages as shown in Figure 3.21.

1. **Handoff Check**: The algorithm starts at the parent BS. The parent BS takes measurements to check the quality of a MS uplink. The BS decides the possibility of a MS handoff based on the measurements taken on the MS uplink. In mobile assisted handoff, it also requests the MS to take downlink measurements of its

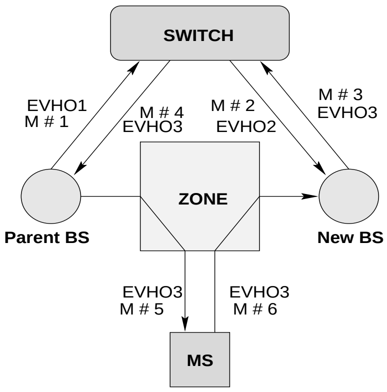


Figure 3.21: General message flow of handoff algorithms.

neighboring BSs and based on those measurements, the parent BS decides on possible candidate handoff BSs. If handoff is possible, the parent BS sends a handoff request (message 1) to a `switch`.

2. **Handoff Request**: The `switch` receives the handoff request (message 1) from a parent BS. It requests the neighboring BSs (message 2) to measure uplink quality for the MS. The neighboring or candidate BSs send the measurements back to the `switch` (message 3). Then, the `switch` finds a target BS for the MS by comparing the measurements of the parent BS and the candidate BSs. The `switch` signals the parent BS to order the MS to *handoff* to the target BS (message 4).

3. **Handoff Order**: The parent BS receives the order (message 4) from the `switch`. The parent BS directs the MS to handoff to the new BS by sending the handoff order (message 5). Since WiPPET does not have direct TeD channels from BS to MS, the above message is routed through the parent `zone`.

4. **Hand Over**: The MS receives the handoff order (message 5) from the parent BS through the parent `zone`. Then, the MS breaks the connection to parent BS. It acknowledges the new BS by sending message 6 and makes a new connection with the new BS.

From the Figure 3.21 and discussion in the previous subsection, there are six messages passing between the entities. We mapped these six messages into four different event structures with six different payloads. The last (fourth) event listed below is used only in MAHO [NA-TDMA,GSM] and it is used to send downlink measurements of neighboring `BS`s from the `MS` to the parent `BS`. The format of a payload element in the events is shown below.

- abbreviated name of the event *(full name)*
  - abbreviated name of the payload *[full name]* description

1. **EVHO1***(EVent HandOff payload 1)* Event structure
   - msgID *[message ID]* the payload type
   - msID *[mobile station ID (given MS)]*
   - zoneID *[zone ID (parent)]* of the given MS
   - pbsID *[parent bs ID]* of the given MS
   - ch *[channel (uplink)]* of the given MS
   - msTxPWR*[mobile station Tx PWR]* of the given MS
   - gain *[$g_{ds}(MS_{\text{location}}, BS_{\text{parent}})$]* from the given MS location to its parent BS
   - iPWR *[interference PWR (uplink)]* for the given MS measured at its parent BS
   - ListbsID[0 ... NO_nghBS-1] *[List of neighboring bsID]*s in descending order of their RSSI/beacon at the MS location
2. **EVHO2***(EVent HandOff payload 2)* Event structure
   - msgID *[message ID]* the payload type
   - msID *[mobile station ID (given MS)]*
   - zoneID *[zone ID (parent)]* of the given MS
   - pbsID *[parent bs ID]* of the given MS
3. **EVHO3***(EVent HandOff payload 3)* Event structure
   - msgID *[message ID]* the payload type

- msID *[mobile station ID (given MS)]*
- zoneID *[zone ID (parent)]* of the given MS
- bsID *[bsID (new/candidate/neighbor)]* of the given MS (handoff base station)
- ch *[channel (new, uplink)]* for the given MS assigned by the new BS
- TOutageTime *[Duration of Outage in seconds]* This parameter is used in dropping decisions
- msTxPWR *[mobile station Tx PWR(required)]* for the given MS to satisfy the target SIR on the uplink at the new BS
- gain *[*$g_{ds}(MS_{\text{location}}, BS_{new})$*]* from the given MS location to the new BS
- iPWR *[interference PWR (uplink)]* for the given MS measured at the new BS on the new assigned channel
- bsTxPWR *[bs station Tx PWR(required)]* for the new BS

4. **Measurement event payload** Event structure

- msgID *[message ID]* the payload type
- msID *[mobile station ID (given MS)]*
- zoneID *[zone ID (parent)]* of the given MS
- pbsID *[parent bs ID]* of the given MS
- gain[0 ... NO_nghBS-1] *downlink gain for [List of neighboring bsID]*s at the given MS location.
- ListbsID[0 ... NO_nghBS-1] *[List of neighboring bsID]*s in descending order of their RSSI/beacon at the given MS location

This subsection discusses MADRAS handoff and one of the standard based MAHO (NA-TDMA).

MADRAS handoff

Let us start with three handoff algorithms, derived from MADRAS [2]: (1) RSSIBHO (2) SIRBHO and (3) combined RSSI and SIRBHO. These three algorithms are implemented in TeD all together in one package called MADRAS_HANDOFF. One can select any one of the above three handoff algorithms by changing a handoff parameter in the configuration file. Figure 3.22 shows event flow and interaction diagram of MADRAS handoff.

Handoff check: The first event in the MADRAS handoff is EVHANDOFF_CHECK. This is an Intra-EI event sent on internal TeD channel within the parent BS. When a MS is admitted, initially, this event is generated by the call admission architecture of the parent BS. This is a timer event. The purpose of this timer event is to trigger the handoff check procedure periodically. Interval between two successive EVHANDOFF_CHECK is called handoff check interval and it is a simulation parameter. In the handoff check procedure, the parent BS checks the possibility of the given MS handoff and it is asynchronous for each active MS in the parent BS.

The parent BS initiates the handoff check procedure by sending EVPBS_MES_REQ to the handoff architecture of the parent zone requesting the uplink interference value on the MS channel. In case of more than one zone simulation, the parent BS sends the request to all zones. For simplicity, here, we considered a one zone simulation. The parent zone receives the request and calculates the uplink interference by using the equation 3.4, shown and discussed in section 3.2. It also selects a candidate BS (strongest pilot BS) for the MS corresponding to the MS location as discussed in section 3.2.

In RSSIBHO algorithm, for the given MS, if there is no stronger BS other than parent BS, the parent zone assigns NO_BS (no candidate base station) as the candidate BS. Hence, RSSIBHO does not allow the *intra-cell* handoff in which a MS is assigned a new traffic channel within the same BS. Now let us assume that the parent zone found the strongest BS other than the parent BS. Then, the parent zone compares the path loss (gain_{new}) from the strongest BS to the MS and the path loss (gain_{old}) from the parent

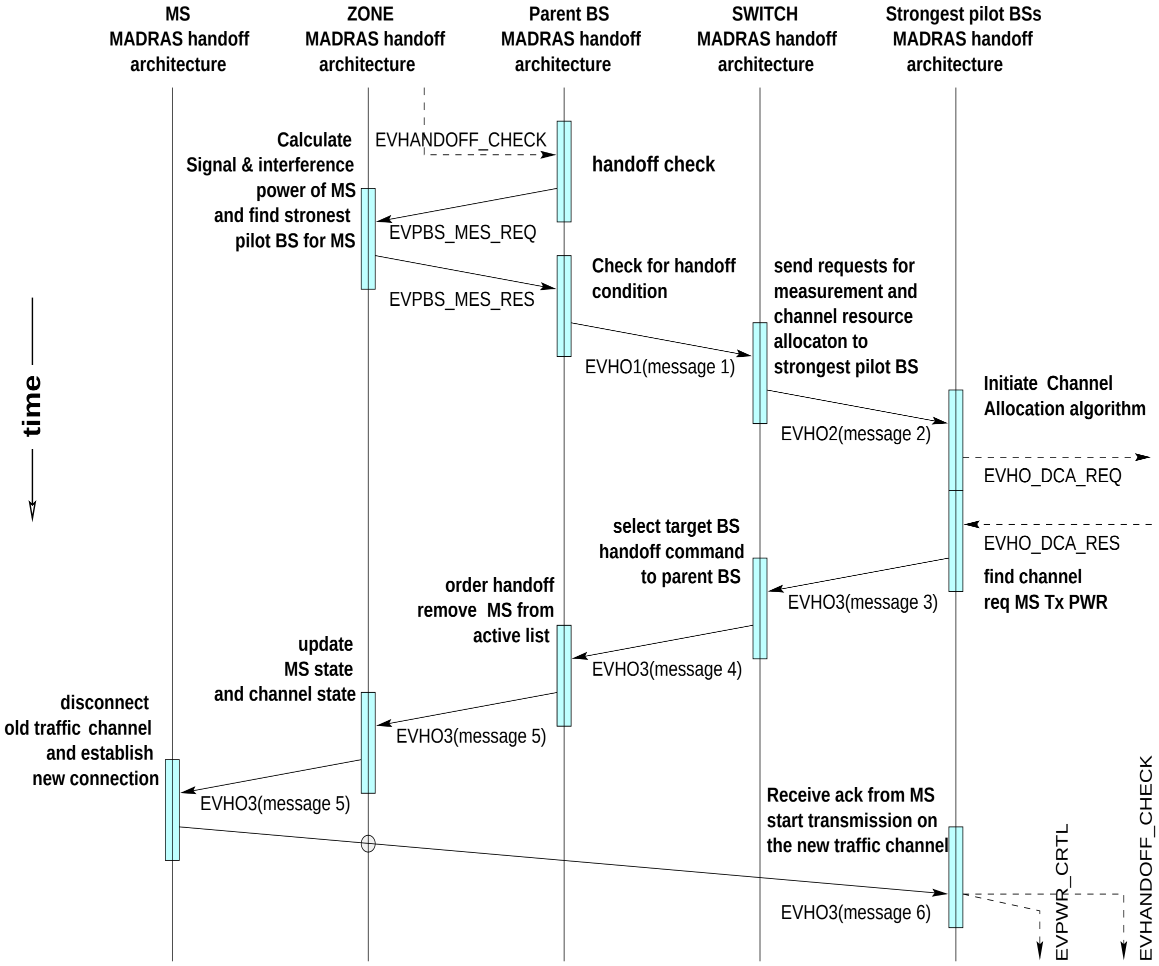


Figure 3.22: Event flow and entity interaction diagram of MADRAS handoff protocol. A vertical shaded strip represents a process of a handoff architecture in an entity, which receives and/or sends Inter-EI events to the processes of its peer architectures or Intra-EI events to the processes of architecture within its entity. Solid-line arrows show the direction of flow of Inter-EI events from a handoff architecture to its peer in other entity. Broken-line arrows show the direction of flow of Intra-EI events from a handoff architecture to an architecture within an entity. A shaded circle represents a process which acts as an event repeater in which the process receives the event and sends it to its destination without doing any event processing. Here, `zone` has a event repeater process, because, there are no direct TeD channels from any `MS`s to any `BS`s.

BS to the MS. If

$$\text{RSSI\_HY (dB)} \leq 10\log_{10}\left(\frac{\text{gain}_{new}}{\text{gain}_{old}}\right)$$

the parent zone assigns the strongest BS as the candidate BS for the MS. Otherwise, the parent zone assigns NO_BS as a candidate BS. RSSI_HY is the hysteresis threshold and it is a simulation parameter.

In SIRBHO algorithm, the parent zone selects the strongest pilot BS corresponding to the MS location. The selected BS may be the parent BS itself. Hence, SIRBHO allows *inter-cell* handoff in which a MS breaks an old connection with its parent BS and makes a new connection with the candidate BS.

The parent zone sends EVPBS_MES_RES to the handoff architecture of the parent BS. This event includes the uplink interference value (iPWR), a list of candidate base stations (in MADRAS handoff algorithms, there is only one candidate BS and that is the strongest pilot BS), the current MS transmitter power (msTxPWR) and the path loss (gain) from the MS to its parent BS. The parent BS receives EVPBS_MES_RES. In a multiple zone simulation, the parent BS collects the iPWR values from all the zones and adds them up to get total iPWR value.

In the SIRBHO algorithm, the parent BS calculates the current MS uplink SIR

$$SIR_{current} = 10\log_{10}\left(\frac{\text{gain} \cdot \text{msTxPWR}}{\text{iPWR}}\right) \tag{3.9}$$

If $\Gamma_{ho} > SIR_{current}$, the parent BS assigns NO_BS as a candidate BS. Note that the simulation parameter Γ_{ho} is the hand-in threshold. In combined RSSI and SIRBHO, the parent BS first checks the availability of the candidate BS based on RSSIBHO algorithm. If there is no candidate BS for RSSIBHO, the combined RSSI and SIRBHO becomes SIRBHO.

The parent BS checks the availability of the strongest BS by comparing the candidate BS (ListbsID) with NO_BS. If the parent BS found a candidate BS, it sends EVHO1 with payload message type 1 to handoff architecture of the switch requesting the MS handoff. Otherwise the handoff algorithm fails.

Handoff request: The `switch` receives the handoff request EVHO1 from the parent `BS`. EVHO1 has payload message type **1** including gain, msTxPWR, total iPWR on the `MS` channel, and the candidate `BS` ID (ListbsID). Then, the `switch` sends a request to the handoff architecture of the candidate `BS` as EVHO2 with payload message type **2** requesting a traffic channel for the `MS`.

The candidate `BS` receives EVHO2 and initiates channel allocation algorithm by sending EVHO_DCA_REQ to its channel allocation architecture. The channel allocation architecture sends EVHO_DCA_RES including a new traffic channel number (ch_{new}), the uplink interference power (iPWR_{new}) on the new traffic channel at the candidate `BS` and the path loss value (gain_{new}) from the `MS` to the candidate `BS`. The handoff architecture receives EVHO_DCA_RES from the channel allocation architecture. These are Intra-EI events between the handoff architecture and the channel allocation architecture in the candidate `BS` as discussed in subsection 3.6.1. Then, the handoff architecture of the candidate `BS` calculates the new required the `MS` transmitter power:

$$\mathrm{msTxPWR}_{new} = 10^{\frac{\Gamma_t}{10}} \cdot \frac{\mathrm{iPWR}_{new}}{\mathrm{gain}_{new}}$$

Note that the simulation parameter Γ_t is the required uplink SIR to satisfy the link QoS requirement. If MAX_MS_PWR $\leq \mathrm{msTxPWR}_{new}$, the channel allocation fails and the candidate `BS` assigns NO_CHANNEL (no channel available) as a new traffic channel ID. The candidate `BS` sends EVHO3 with payload message type **3** to the handoff architecture of the `switch`. The `switch` receives EVHO3 with payload message type **3** including ch_{new}, the iPWR_{new}, the gain_{new}, and the $\mathrm{msTxPWR}_{new}$.

In SIRBHO, the `switch` recalculates the new required `MS` uplink transmitter power:

$$\mathrm{msTxPWR}_{new} = 10^{(\frac{SIR_{current}+SIR\_HY}{10})} \cdot \frac{\mathrm{iPWR}_{new}}{\mathrm{gain}_{new}}$$

where $SIR_{current}$ is the SIR on the current uplink from the `MS` to its parent `BS` given by equation 3.9. If MAX_MS_PWR $\leq \mathrm{msTxPWR}_{new}$, the `switch` assigns NO_CHANNEL as the new traffic channel ID. Note that the simulation parameter SIR_HY is the hysteresis threshold.

Then, the `switch` sends EVHO3 to the handoff architecture of the parent `BS` with payload message type **4** including the candidate BS ID, msTxPWR_{new}, ch_{new}, and gain_{new}.

Handoff order: The parent BS receives EVHO3 with payload message type **4** and checks the possibility of handoff by comparing the new traffic channel ID (ch_{new}) with NO_CHANNEL. If there is no traffic channel available at the candidate BS, the handoff algorithm fails.

At this point, the parent BS checks the MS dropping condition, whether the MS handoff failed or not. If the handoff fails, the parent BS calculates the current uplink SIR ($SIR_{current}$) using equation 3.9. If $SIR_{current} < \Gamma_{drop}$ (drop threshold), the MS *outage time* is incremented. In case there is a new traffic channel available at the candidate BS, the handoff architecture at parent BS calculates the new uplink SIR for the MS at the new BS as

$$SIR_{new} = 10\log_{10}\left(\frac{\text{gain}_{new}\cdot \text{msTxPWR}_{new}}{\text{iPWR}_{new}}\right)$$

If $SIR_{new} < \Gamma_{drop}$, the *outage time* is updated. The parent BS has an outage timer for each active MS. The parent BS uses this timer to keep track of the MS outage time. Whenever a MS uplink SIR goes above the drop threshold (Γ_{drop}), the parent BS resets the outage timer to zero and starts counting again. If outage time $\geq$ MAX_OUTAGE_TIME (maximum outage time allowable), the MS is considered to be dropped. Hence, the handoff architecture at the parent BS sends EVCALL_DROP to the call admission architecture. The drop threshold (Γ_{drop}) and maximum outage time (MAX_OUTAGE_TIME) are simulation parameters. EVCALL_DROP is an Intra-EI events sent by the handoff architecture of a parent BS to its call admission architecture as discussed in subsections 3.6.1 and 3.6.3. By receiving this event, the call admission architecture removes the MS from the active list. The call admission architecture of the parent BS sends EVCALL_DROP and informs the call admission architecture of the parent zone and the switch.

If the MS is not dropped, the parent BS orders the MS to disconnect the present connection and establish a new connection with the candidate BS on the new traffic channel by sending EVHO3 with payload message type **5**. Now, the candidate BS becomes the *target* BS. Since there is no direct TED channel from a BS to any MS, EVHO3 is routed through the parent zone.

Handoff over: The handoff architecture of the parent zone receives EVHO3 with message **5** and updates the MS states: the new channel and the new parent BS ID and the new uplink transmitter power ($\mathrm{msTxPWR}_{new}$). Then, the zone resends the same event to the MS with same message ID. The handoff architecture of the MS receives EVHO3 with payload message type **5** and update its states: the new traffic channel and the new parent BS ID and the new uplink transmitter power ($\mathrm{msTxPWR}_{new}$). Then, the MS breaks the present connection to its parent BS and makes a new connection to the target BS. It acknowledges the target BS about its presence on the new traffic channel using EVHO3 with message **6**. Since there is no direct TED channel from the MS to the new BS, EVHO3 is routed through the parent zone to the target BS. The target BS receives EVHO3 with message **6** from the MS. It updates its channel state and puts the MS in the active list. It also initiates the power control and handoff check procedures for the new arrived MS by sending two timer events EVHANDOFF_CHECK and EVPWR_CRTL on the internal TED channel as discussed in subsections 3.6.2 and 3.6.1. The MADRAS handoff algorithm ends here.

The MADRAS RSSIBHO algorithm is closely matched with AMPS handoff algorithm. The difference is that the MADRAS RSSIBHO algorithm has only one candidate BS (the strongest pilot BS), while the AMPS handoff has a list of strong pilot BSs. Hence, in AMPS handoff, the switch requests all the candidate BSs of the list simultaneously for channel resources for the MS. The switch collects the necessary information from the candidate BSs as we discussed earlier, and selects one of the candidate BSs as the target BS. The target BS is one of the candidate BSs for which the new required $\mathrm{msTxPWR}_{new}$ is the least.

NA-TDMA (IS-136) handoff

WiPPET also has the NA-TDMA (IS-136) standard based handoff protocol. Figure 3.23 shows event flow and interaction diagram of NA-TDMA handoff protocols. NA-TDMA handoff is a Mobile Assisted Handoff (MAHO) algorithm. Unlike in MADRAS and AMPS handoff algorithms, the `MS` takes part in selection of candidate `BS`s, and also measures the RSSI of the candidate `BS`s.

Handoff check: The NA-TDMA handoff starts with the first event EVHANDOFF_CHECK. As we discussed earlier in MADRAS handoff [subsection 3.7.7], this is a timer event, which periodically initiates handoff check procedure in the parent `BS`. The handoff check procedure is asynchronous for each active `MS` in the parent `BS`. The handoff architecture of the parent `BS` receives EVHANDOFF_CHECK and then sends EVPBS_MES_REQ to the handoff architecture of the `zone` to find the current path loss value from the `MS` to its parent `BS` (gain) and the interference power on the `MS` channel at the parent `BS` (iPWR). The `zone` replies back with the above information using EVPBS_MES_RES. Then, the parent `BS` calculates the current uplink SIR:

$$SIR_{current} = 10 \log_{10} \left(\frac{\text{gain} \cdot \text{msTxPWR}}{\text{iPWR}}\right)$$

If $SIR_{current} \geq \Gamma_{ho}$ (the threshold to decide the handoff at the parent `BS`), the parent `BS` sends a measurement request to the `MS` using EVNGH_BS_MES_REQ. Otherwise handoff algorithm fails.

The handoff architecture of the `MS` receives the EVNGH_BS_MES_REQ requesting the measurement of the RSSI (pilot strength or gain) of the neighboring `BS`s. To fulfill this request, the `MS` sends EVMS_MES_REQ to the handoff architecture of its parent `zone` requesting a list of neighboring `BS`s ID and their relative pilot strength at the `MS`. The parent `zone` finds the first N strong pilot strength `BS`s at the `MS` and replies back these neighboring `BS`s ID with their relative pilot strength using EVMS_MES_RES. The total number of neighboring `BS`s (N) is a simulation parameter. In WiPPET$_{\text{session}}$, the `zone` deferred constant contains the first four strong neighboring `BS`s ID from every possible `MS` position (see appendix B.1). The `MS` receives the EVMS_MES_RES from the

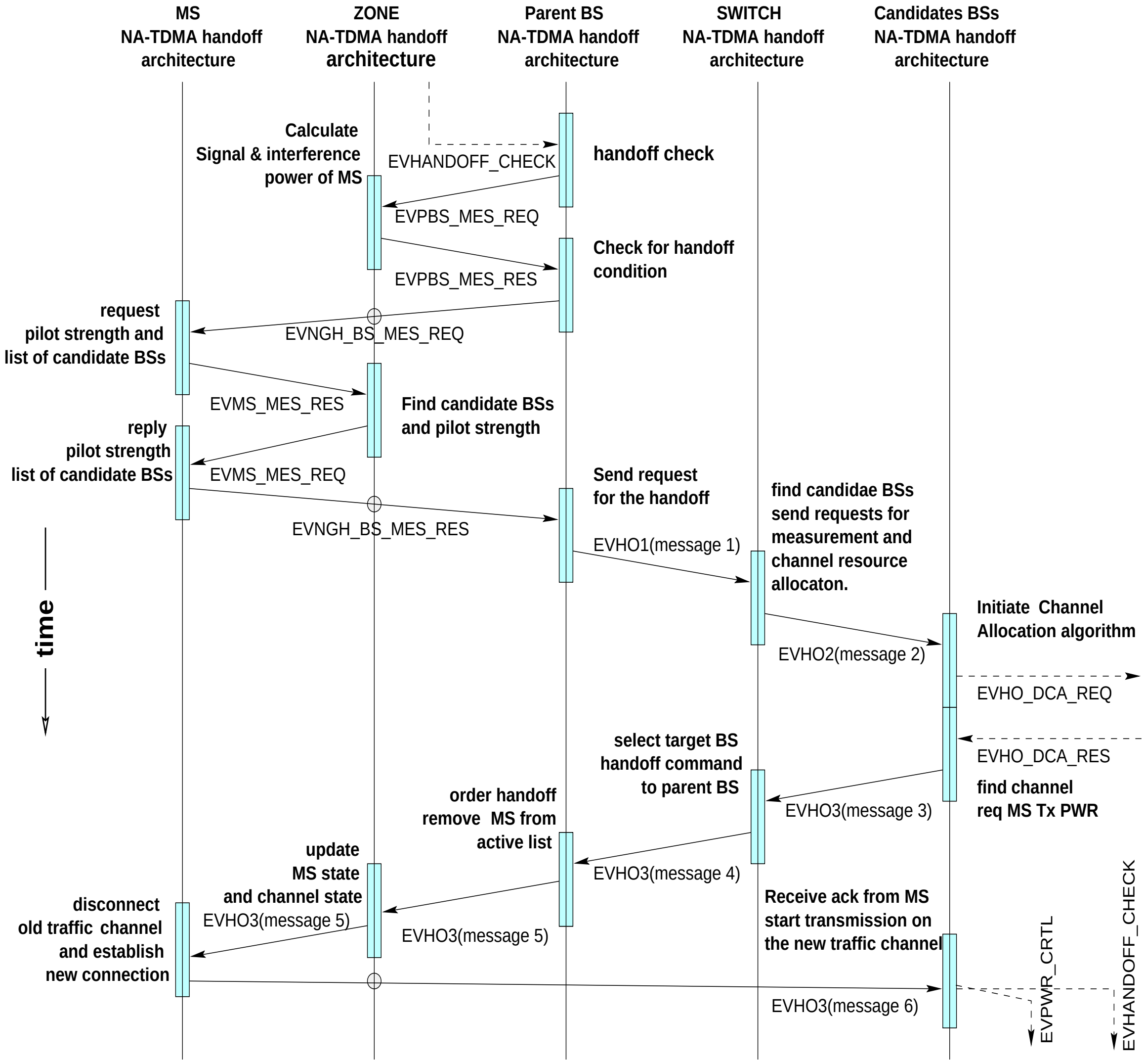


Figure 3.23: Event flow and entity interaction diagram of NA-TDMA (IS-136) handoff protocol. A vertical shaded strip represents a process of a handoff architecture in an entity, which receives and/or sends Inter-EI events to the processes of its peer architectures or Intra-EI events to the processes of architecture within its entity. Solid-line arrows show the direction of flow of Inter-EI events from a handoff architecture to its peer in other entity. Broken-line arrows show the direction of flow of Intra-EI events from a handoff architecture to an architecture within an entity. A shaded circle represents a process which acts as an event repeater in which the process receives the event and sends it to its destination without doing any event processing. Here, `zone` has event repeater processes, because, there are no direct TeD channels from any `MS`s to any `BS`s.

parent zone and transfers the measurement of the neighboring BSs to its parent BS using EVNGH_BS_MES_RES.

The handoff architecture of the parent BS receives a list of neighboring BSs (ListbsID) and their relative pilot signal strength at the MS. Then, the BS sorts the neighboring BSs list according to their pilot strength and creates a new list of candidate BSs. Then, it sends a handoff request EVHO1 with payload message type **1** to the switch including the list of candidate BSs, the current MS transmitter power (msTxPWR), the path loss value (gain) from the MS to its parent BS, and the interference power (iPWR) on the MS channel at the parent BS.

Handoff request: The handoff architecture of the switch receives EVHO1 stores the ListbsID, the msTxPWR, the gain, and the iPWR. The switch requests all candidate BSs of the list simultaneously for channel resources by sending EVHO2 payload message type **2**. All candidate BSs receive EVHO2 and initiate handoff channel allocation algorithm by sending Intra-EI events EVHO_DCA_REQ to their handoff channel allocation architectures.

As we discussed earlier in subsection 3.7.2, the channel allocation architecture at a candidate BS sends EVHO_DCA_RES including a new traffic channel number (ch_{new}), the uplink interference power (iPWR_{new}) on the new traffic channel at the candidate BS and the path loss value (gain_{new}) from the MS to the candidate BS. The handoff architecture receives EVHO_DCA_RES from the handoff channel allocation architecture and calculates the required uplink transmitted power for the MS:

$$\mathrm{msTxPWR}_{new} = 10^{\frac{\Gamma_t}{10}} \cdot \frac{\mathrm{iPWR}_{new}}{\mathrm{gain}_{new}}$$

Note that the simulation parameter Γ_t is the required uplink SIR to satisfy the QoS requirement. If $\mathrm{MAX\_MS\_PWR} \leq \mathrm{msTxPWR}_{new}$, the channel allocation fails and the candidate BS assigns NO_CHANNEL (no channel available) as a new traffic channel ID. Similarly, all candidate BSs send ch_{new}, gain_{new}, iPWR_{new}, and $\mathrm{msTxPWR}_{new}$ to the handoff architecture of the switch by using EVHO3 with payload message type **3**.

The switch receives EVHO3 with payload message type **3** from all the candidate BSs.

The next job of the switch is to decide the target BS from the list of candidate BSs based on the required MS transmitter power (msTxPWR$_{new}$) at the candidate BS and the uplink SIR of the MS at the candidate BS (SIR_{new}). First the switch decides a candidate BS as the target BS, for which the required MS transmitter power is least. If msTxPWR$_{new}$ for two or more than two candidate BSs are the same, than the switch will decide a candidate BS as the target BS, at which the MS uplink SIR is highest. Then, the switch sends EVHO3 with payload message type **4** to parent BS including the target BS ID, the new channel number (ch) at the target BS, the interference (iPWR) on the new channel at the target BS, the required MS transmitter power (msTxPWR), and the new path loss value (gain) from the MS to the target BS. Then, the algorithm enters the handoff order stage. From this point on, the flow, interaction and processing of the events are the same as the MADRAS handoff protocol.

GSM handoff is also a MAHO algorithm closely matched with the NA-TDMA handoff algorithm. The difference between these two algorithms is that unlike in NA-TDMA handoff, in GSM handoff the MS sends pilot strength measurements of neighboring BSs to the parent BS periodically not on request. The parent BS regularly updates the measurement and the list of neighboring BSs. When the parent BS finds that the current SIR on the MS uplink channel is less than the handoff SIR (Γ_{ho}), it sends the stored list of candidate BSs (ListbsID) and their relative pilot signal strength at the MS along with the current MS transmitter power (msTxPWR), the path loss (gain) from the MS to the parent BS, and the interference power (iPWR) on the MS channel at the parent BS in the event EVHO1 payload message type **1** to the switch. The rest of the protocol works the same way as NA-TDMA handoff.

3.8 Verification of Testbed

parameter [units]	2D Macrocell
Number of BSs	48
Number of Channels	40
Geography Style	Manhattan
Geographical Area	$12km \times 12km$
Cell Radius [m]	1000
Antenna Height [m]	50
Distance Resolution [m]	5
Shadow Fading Standard Deviation [dB]	6
Shadow Fading Correlation Distance [m]	50
Noise Power [dBW]	-150
Maximum Power [dBW]	0
Mobile Speed [m/sec]	25
Probability of Going Straight	0.6
Time Between Handoff Checks [sec]	1.0
Time Between Power Updates [sec]	1.0
BS Assignment Algorithm	Strongest Pilot BS
Call Admission Channel-Allocation Algorithm	LI-DCA
Handoff Algorithm	RSSIBHO
Handoff Channel-Allocation Algorithm	LI-DCA
RSSI hysteresis (RSSI_HY) [dB]	9
SIR hysteresis (SIR_HY) [dB]	3
Admission SIR (Γ_{new}) [dB]	20-26
Target SIR (Γ_t) [dB]	20
Handoff SIR (Γ_{ho}) [dB]	17
Drop SIR (Γ_{drop}) [dB]	14

Table 3.1: WiPPET$_{\text{session}}$ simulation parameters

The simulation testbed (WiPPET-session level) was verified by implementing MADRAS [2] in the TeD/C++ framework, and comparing the simulation results with previously published MADRAS simulation results [2]. The results of simulations in terms of blocking and dropping probability for given performance parameters and algorithms are shown below as an example. Tabel 3.1 shows the WiPPET$_{\text{session}}$ configuration parameters and algorithms used for comparison of the simulation results. All the algorithms used for the simulation were explained previously in section 3.7 of this chapter.

Figures 3.24 and 3.25 show the agreement in call blocking and call dropping probability versus the offered traffic respectively with reference curves from [2]. The reader may refer to [2] for more details on the algorithms and the interpretation of the results.

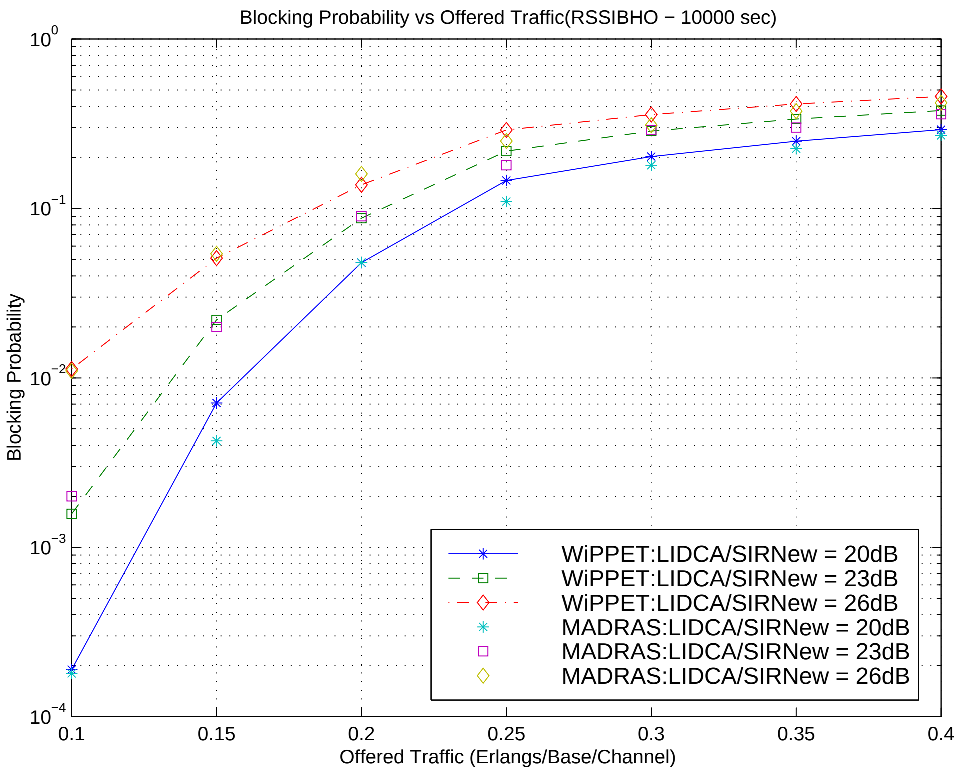


Figure 3.24: Blocking Rate of SIR admission threshold/LI-DCA versus the offered traffic. Call blocking rate as a function of offered traffic at different SIR admission thresholds using Least Interference based Dynamic Channel Allocation algorithm (LI-DCA).

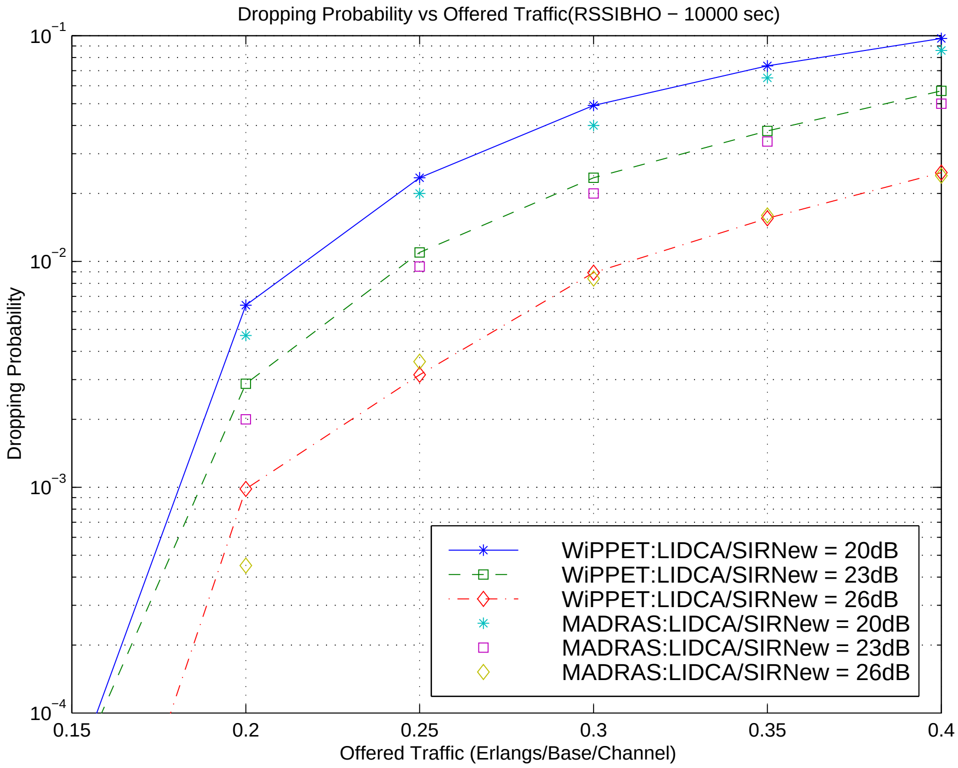


Figure 3.25: Dropping Rate of SIR admission threshold/LI-DCA versus the offered traffic. Call dropping rate as a function of offered traffic at different SIR admission thresholds using Least Interference based Dynamic Channel Allocation algorithm (LI-DCA).

Chapter 4

WiPPET: Packet level simulation

This chapter introduces $\text{WiPPET}_{\text{packet}}$, which models packet level traffic, radio propagation model including distance loss, short scale (Rayleigh) and long scale (shadow) fading, mobility, and power control. The simulator provides statistics such as packet error rate (PER) to analyze the performance of data-link layer protocols in an interference and fading environment.

This chapter gives detailed explanation of $\text{WiPPET}_{\text{packet}}$ radio propagation model, protocols and algorithms. It also compares the packet level statistics, e.g. PER versus fade margin for different mobile station speeds, obtained by $\text{WiPPET}_{\text{packet}}$ with the published reference statistics for a wireless data standard Cellular Digital Packet Data (CDPD).

4.1 Overview

$\text{WiPPET}_{\text{packet}}$ has a `callgen`, `MS`, `BS`, and `zone` entities. These entities are connected to each other according to Figure 3.2. Each `MS` and `BS` are connected to the `zone`. There is no direct TED channel from any `MS` to any `BS`. Following are the entities of $\text{WiPPET}_{\text{packet}}$ and a brief introduction to their functionalities.

1. Call Generator (`callgen`) : It generates session or call by activating `MS`s.

2. Mobile Station (`MS`) : When `MS` becomes active, it generates packet traffic and sends packets to its parent `BS`.

3. Radio channel zone (`zone`) : It is the most complex entity in the simulator. It represents a geography and radio channels. It takes part in mobility, call admission, and power control protocols. It also contains a path loss matrix and short-scale (Rayleigh) fading architecture. More detailed explanation of the `zone` is given in section 4.2.

4. Base Station (`BS`) : It receives the SIR profile of the packet, which is transmitted by a `MS`. It has a packet discriminator which decides whether the packet is in error or not. It gives performance statistics of data-link layer protocols in terms of packet error rate (PER). The `BS` also adjusts the `MS` uplink power periodically.

4.2 Zone Entity

The `zone` represents geography and radio channels. The `zone` contains the Manhattan geography model discussed in section 3.2. It also contains geographically referenced path loss values $g_{ds}(\mathrm{MS}_{\mathrm{location}}, \mathrm{BS})$ in the path loss matrix. It is a two dimensional matrix with one axis representing possible `MS` locations and other axis representing `BS`s in the system as shown in Figure 3.3. See appendix B.1 for more details.

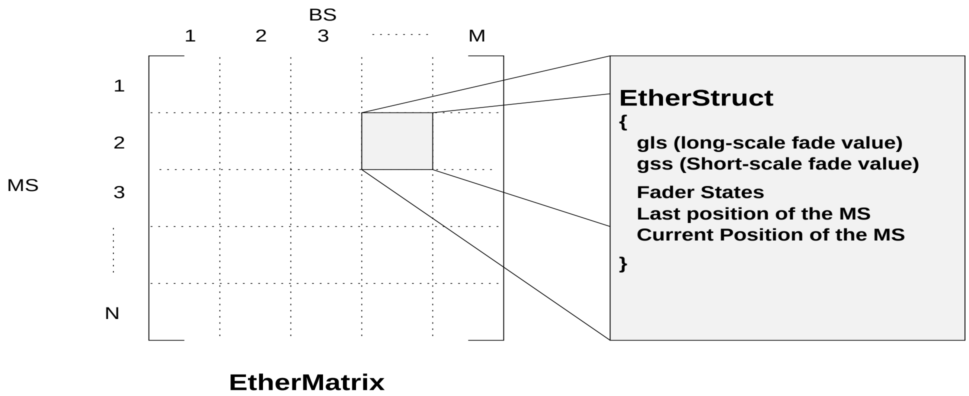


Figure 4.1: EtherMatrix containing EtherStruct in each element. Each matrix element contains a structure EtherStruct, which represents a transmitter-receiver (MS-BS) link. EtherStruct contains the current path loss value, the current short-scale fading value, fader states and, etc, associated with the link. When a `MS` changes its location, the path loss value in the EtherStruct is updated. Short-scale fading value in EtherStruct of a given MS-BS link is updates periodically along with the fader states. See text for more detail.

The `zone` combines the role of a geography container with the role of the radio channels simulator, collecting contributions to total signal power in a radio channel from all active transmitters and attenuating them by appropriate fading factor (distance, shadow fading and short-scale fading). The `zone` has a matrix associated with each radio channel. This matrix is called EtherMatrix (Here, the `zone` has only one EtherMatrix because there is only one radio channel). An abstract model of EtherMatrix is shown in Figure 4.1. Each matrix element contains a structure EtherStruct, which represents a transmitter-receiver (MS-BS) link. EtherStruct contains the current path loss $g_{ls}(\mathrm{MS}, \mathrm{BS})$ and the current short-scale (Rayleigh) fading $g_{ss}(\mathrm{MS}, \mathrm{BS})$ values for the MS-BS link, the states of Rayleigh fader for the MS-BS link, the current and the last position of the `MS`.

The elements of a row of an EtherMatrix represents the uplinks from a `MS` to all `BS`s. When a `MS` moves to the next position in the geography, the path loss values of uplinks from the `MS` to all `BS`s are changed. Hence, in the mobility protocol, along with the `MS` position updates, the `zone` also updates the path loss value of uplinks from the `MS` to all `BS`s in the EtherStructs of the entire `MS` row in the EtherMatrix. Similarly, the fader architecture in the `zone` updates the short-scale fading value and the fader states for uplinks from a `MS` to all `BS`s periodically. The fader states of an EtherStruct represents a short-scale fader. Short-scale fading on each MS-BS link is independent of other links, hence, there are separate faders for each MS-BS link. These fading updates are asynchronous for every `MS`. The frequency of fade value updates is a simulation parameter and it represents the time-scale granularity of the simulation. The resultant fading value for a MS-BS link including the path loss value and the short-scale fading value is given by

$$g_{ch}(\mathrm{MS}, \mathrm{BS}) = g_{ls}(\mathrm{MS}, \mathrm{BS}) \cdot {g_{ss}}^2(\mathrm{MS}, \mathrm{BS})$$

We call $g_{ch}(\mathrm{MS}, \mathrm{BS})$ the *radio channel gain*. The reason $g_{ss}(\mathrm{MS}, \mathrm{BS})$ is squared in the above equation is that the short-scale fader generates a signal envelope value at a given instant of time, not a power value.

To find the instantaneous received signal power and interference power at the parent

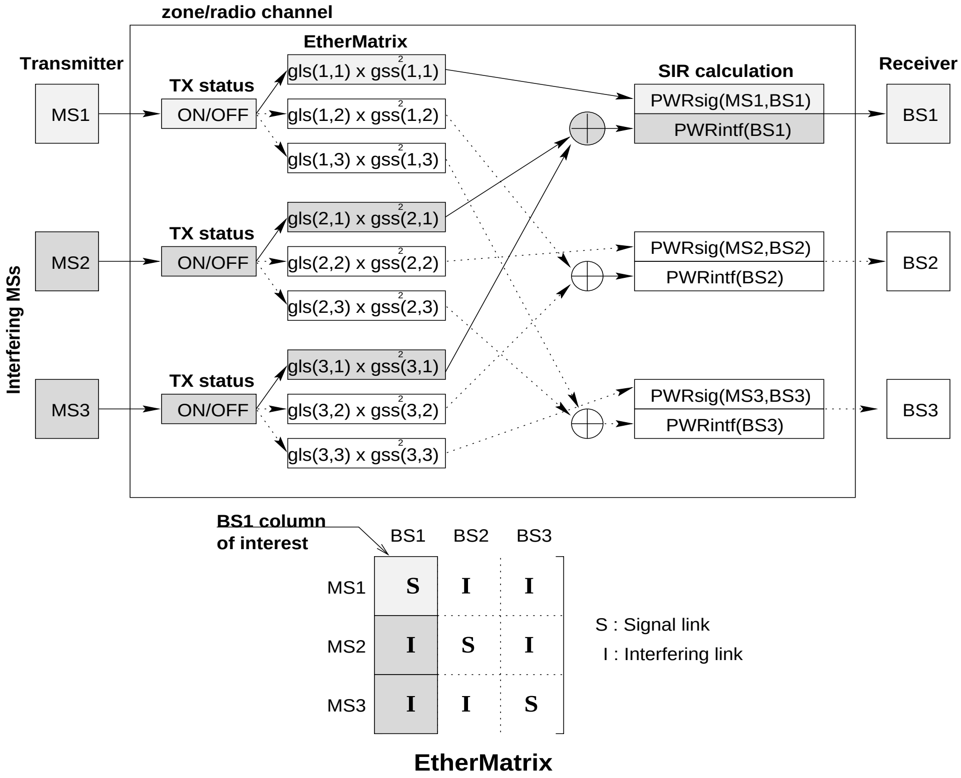


Figure 4.2: Graphical representation of instantaneous uplink SIR calculation. MS_2 and MS_3 are interfering mobile stations at base station BS_1 for the mobile station MS_1. In EtherMatrix, subscripts **S** and **I** represent the signal uplink from MS_1 to BS_1 and the interference uplinks from MS_2 and MS_3 to BS_1 respectively. To calculate the instantaneous interference power at BS_1, the instantaneous power of **I** uplinks in the EtherStruct is summed up. Shaded areas shows used states and constants, arrows shows the flow of action in calculation of instantaneous uplink SIR at BS_1 for MS_1 in `zone`. See text for more detail.

BS for the specified MS, the zone contains:

1. An EtherMatrix for each radio channel.

2. For each MS, a dynamic list containing instantaneous values of MS position, instantaneous transmitter power, MS transmitter status (whether MS is transmitting packet or not), parent BS ID.

As discussed earlier, a MS transmits packets and the zone receives those packets. The zone samples each packet and calculates instantaneous uplink signal to interference ratio (SIR) at each sample instant using the instantaneous received signal power and the interference power. Then, it sends the sampled SIR values of the packet to the parent BS one by one.

Now, we discuss how the zone calculates the instantaneous SIR values for each sample. Assume that there are three mobile stations MS_1, MS_2, and MS_3 connected to three base stations BS_1, BS_2, and BS_3 respectively. Consider mobile stations MS_2 and MS_3 are the interfering mobile station for MS_1 at BS_1. Figure 4.2 shows the associated EtherMatrix in the zone. The fist column of the EtherMatrix is the BS_1 column. The interfering uplinks from mobile stations MS_2 and MS_3 are represented by **I** elements of the EtherMatrix and **S** element of the EtherMatrix represents the signal uplink from MS_1 to BS_1. Figure 4.2 also shows the role of the EtherMatrix in calculation of signal to interference ratio in the zone. The zone finds the instantaneous received signal power of MS_1 at BS_1 using equation 4.1.

$$P_{RX\,S}(\mathrm{MS}_1, \mathrm{BS}_1) = \mathrm{TXstatus}(\mathrm{MS}_1) \cdot g_{ch}(\mathrm{MS}_1, \mathrm{BS}_1) \cdot P_{TX\,S}(\mathrm{MS}_1) \tag{4.1}$$

where $P_{RX\,S}(\mathrm{MS}_1, \mathrm{BS}_1)$ is the instantaneous signal power of MS_1 at BS_1. $\mathrm{TXstatus}(\mathrm{MS}_1)$ is the transmitter status of MS_1. If the MS_1 transmitter is on, $\mathrm{TXstatus}(\mathrm{MS}_1)$ will be 1, otherwise 0. To find instantaneous interference power at BS_1 for the MS_1, the zone sums up the instantaneous received power of MS_2 and MS_3 at BS_1 as shown in equation 4.2.

$$P_{RX\,I}(\mathrm{BS}_1) = \sum_{n \neq 1} \mathrm{TXstatus}(\mathrm{MS}_n) \cdot g_{ch}(\mathrm{MS}_n, \mathrm{BS}_1) \cdot P_{TX\,S}(\mathrm{MS}_n) \tag{4.2}$$

where $P_{RX\,I}(\mathrm{BS}_1)$ is the instantaneous interference power at BS_1 for MS_1. If any interfering mobile station is not transmitting a packet at that instant, then TXstatus will be zero and hence, the instantaneous interference power due to that mobile will be zero. The `zone` calculates the uplink instantaneous signal to interference ratio $SIR(\mathrm{MS}_1, \mathrm{BS}_1)$ of MS_1 at BS_1 using equation 4.3.

$$SIR(\mathrm{MS}_1, \mathrm{BS}_1) = \frac{P_{RX\,S}(\mathrm{MS}_1, \mathrm{BS}_1)}{P_{RX\,I}(\mathrm{BS}_1) + \eta^2} \tag{4.3}$$

4.3 Architectures of Entities

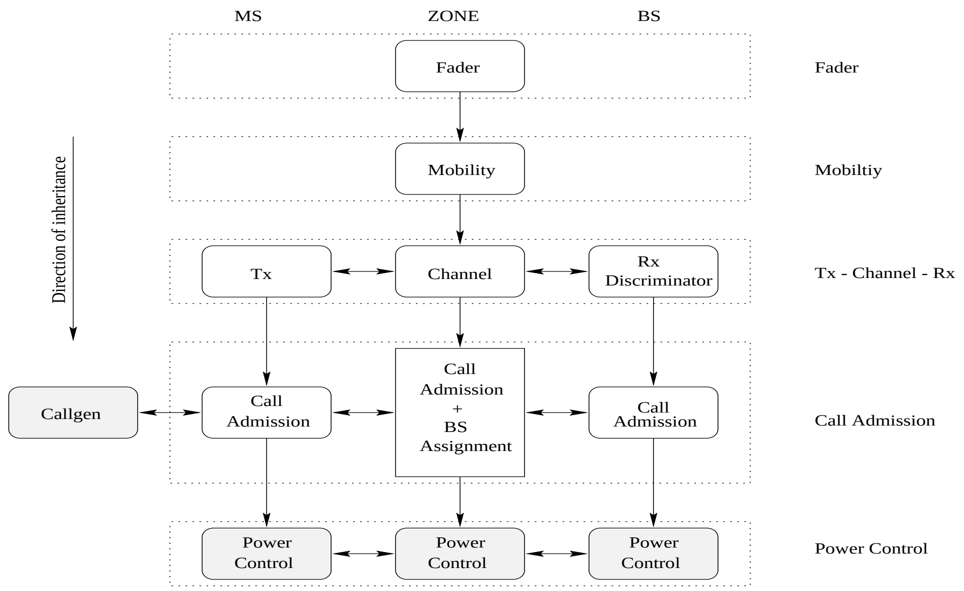


Figure 4.3: Hierarchy of inherited architectures in WIPPET$_{\text{packet}}$ entities. Vertical down arrows show the direction of inheritance. Each entity supports one or more protocols, i.e. protocol functionality distributed among entities. Horizontal arrows show the communication between one architecture to its peer architecture in other entities. Shaded architectures are the last architectures in hierarchy and bound to their respective entities.

WIPPET$_{\text{packet}}$ models packet level traffic, radio propagation model including distance loss, short scale (Rayleigh) and long scale (shadow) fading, mobility, and power control. Figure 4.3 shows WIPPET$_{\text{packet}}$ entities with their inherited hierarchy of architectures.

The `callgen` activates a `MS`. The call admission architecture of the `zone` finds the strongest pilot `BS` for the `MS` and the call admission architecture of the strongest pilot

BS assigns a radio channel to the MS. The *Fader* is the primary architecture in the zone. The fader architecture generates short-scale fading values and updates EtherMatrix for each MS-BS link periodically. The *Mobility* is the second architecture in the zone hierarchical architecture. The mobility architecture updates the position of the MS and also updates shadow (long-scale) fading value. The *Tx* is the primary architecture in the MS. The Tx architecture generates the packet traffic and sends packets to the parent BS. The *Channel* architecture in the zone samples the packet transmitted by the MS and calculates the instantaneous uplink SIR values for each sample instant of the packet. Then, it sends those calculated uplink SIR samples to the *Rx* architecture of the parent BS. The *Rx* architecture of the parent BS has a *discriminator*, which decides whether the packet is in error or not using the uplink SIR profile of the packet.

4.4 Protocols and Algorithms

This section gives detailed explanation of WiPPET$_{\text{packet}}$ radio propagation model, protocols and algorithms. WiPPET$_{\text{packet}}$ models packet level traffic, radio propagation model including distance loss, short scale (Rayleigh) and long scale (shadow) fading, mobility, and power control. The packet level simulation starts at callgen. The callgen participates only in the call admission protocol.

4.4.1 Call admission

This is a simple call admission protocol, which activates a MS to transmit packets and initiates mobility, power control and short-scale fading processes for that MS. Figure 4.4 shows an interaction diagram of the call admission protocol.

The callgen activates a MS by sending EVCALL_INIT. The activated MS sends a request for admission by sending EVCALL_REG_REQ to the call admission architecture of the zone. The zone receives the request and assigns the strongest pilot BS to be the parent of the MS. The zone requests the parent BS to admit the MS by sending EVCALL_ADM_REQ and also replies the MS with admission result by sending EVCALL_REG_RES, which

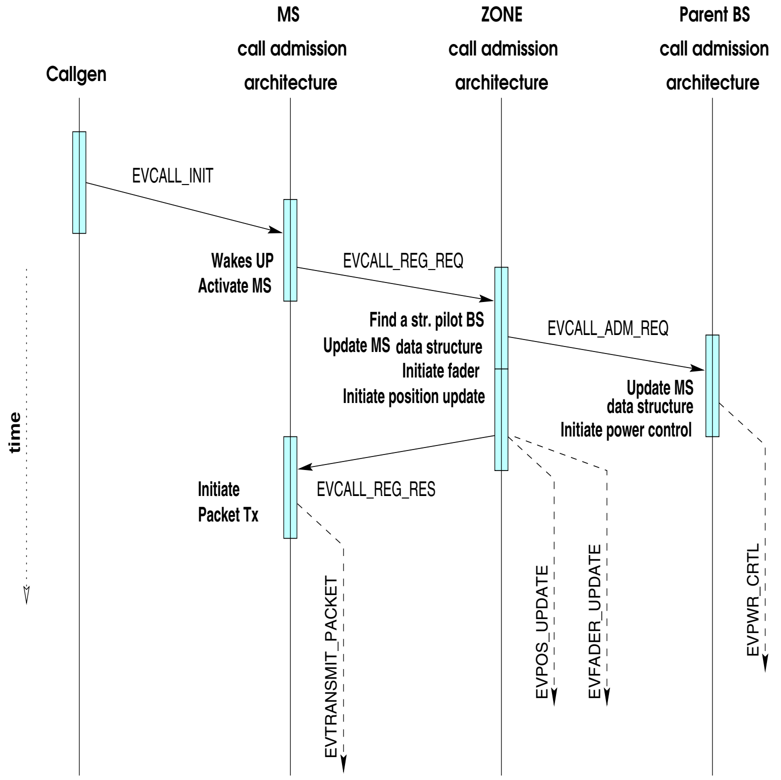


Figure 4.4: Event flow and entity interaction diagram of call admission protocol. A vertical shaded strip represents a process of a call admission architecture in an entity, which receives and/or sends Inter-EI events to the processes of its peer architectures or Intra-EI to the processes of architecture within its entity. Solid-line arrows show the direction of flow of Inter-EI events from a call admission architecture to its peer in other entity. Broken-line arrows show the direction of flow of Intra-EI events from a call admission architecture to an architecture within an entity.

includes the parent BS ID.

The zone initiates the MS position update procedure by sending EVPOS_UPDATE to its mobility architecture on an internal TED channel. It also initiates the MS short-scale fading process by sending EVFADER_UPDATE to its fader architecture. The parent BS receives EVCALL_ADM_REQ and initiates power control algorithm by sending EVPWR_CRTL to its power control architecture. The MS receives EVCALL_REG_RES from the zone with the parent BS ID. Then, the MS sends EVTRANSMIT_PACKET to its Rx architecture requesting to start transmitting packets to its parent BS.

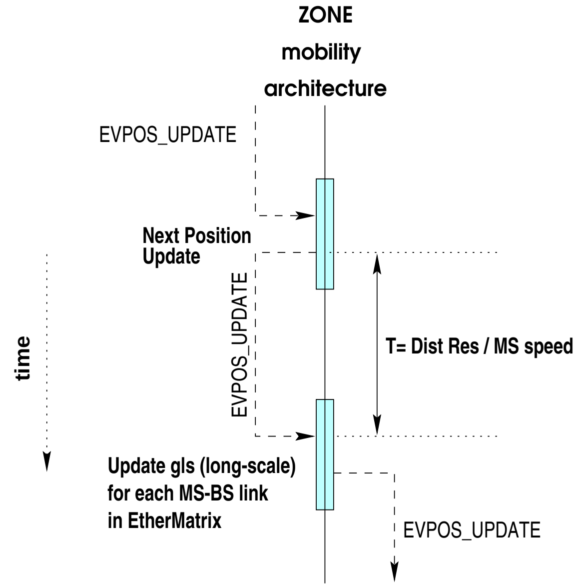


Figure 4.5: Event flow and entity interaction diagram of mobility protocol. A vertical shaded strip represents a process of the mobility architecture in an entity, which receives and/or sends Intra-EI events to itself. Broken-line arrows show the direction of flow of Intra-EI events from the mobility architecture.

4.4.2 Mobility

Only the `zone` has a mobility architecture. The mobility architecture in the `zone` updates the `MS` position and accordingly updates the path loss values in the EtherMatrix of uplinks from the `MS` to all `BS`s. Here, the mobility architecture of the `zone` is almost identical to that of the session level simulation except that there are no `MS` zone transfers because of one zone simulation. Figure 4.5 shows an interaction diagram of mobility protocol.

The mobility architecture at the `zone` receives EVPOS_UPDATE on an internal TED channel. Then, the architecture updates the `MS` position on the geography. The EtherStructs of the `MS` row in the EtherMatrix represents the uplinks from the `MS` to all `BS`s. When a `MS` moves to next position in the geography, the path loss values of the uplinks from the `MS` to all the `BS`s are changed. Hence, along with the `MS` position updates, the mobility architecture also updates the path loss values of the uplinks from the `MS` to all `BS`s in the EtherStructs of the `MS` row. At the end, the mobility architecture schedules the next position update by sending a timer event EVPOS_UPDATE with a delay equal to $(\frac{\text{Distance resolution}}{\text{MS speed}})$.

4.4.3 Fader

We have implemented the models of short–scale Rayleigh and Ricean fading [5] with i) wave-superposition method [21], ii) spectrum-shaping method [20] ; and also iii) a quantized channel state Markovian model [1]. WiPPET$_{\text{packet}}$ employs the spectrum shaping method based on Clarke's model. This subsection gives a brief introduction to this model and discusses the implementation of the model as the *fader* architecture of the `zone`.

Clarke [5] developed a random process model for the received power of random interfering waves. Each incident wave at receiver antenna undergoes a Doppler shift due to the motion of the receiver. In Clarke's model, each incident wave is characterized by amplitude, phase shift, and direction, i.e., the multipath delay is not accounted for. Hence, the model simulates flat fading. Due to the Doppler effect, the received frequency increases if the receiver is advancing into the approaching wavefront and decreases if the receiver is moving away from the wavefront. The maximum Doppler shift is given by

$$\begin{aligned} f_d &= \frac{v \cdot f_c}{c} \\ \text{where } f_d &= \text{maximum Doppler shift.} \\ v &= \texttt{MS}\text{ speed.} \\ c &= \text{speed of light in free space.} \end{aligned}$$

Each arriving wave at the receiver has in-phase and quadrature components T_c and T_s, that are independent stationary Gaussian processes. Assuming angles of arrival to be uniformly distributed from 0 to 2π and assuming an omni-directional receiver, Clarke determined that the power spectral density $S(f)$ of the in-phase and quadrature components are:

$$S_{T_c}(f) = S_{T_s}(f) = \begin{cases} \frac{1}{\pi f_d \sqrt{1-(\frac{f-f_c}{f_d})^2}} & |f - f_c| \le f_d \\ 0 & |f - f_c| > f_d \end{cases}$$

Each component has autocorrelation function ${J_0}^2(2\pi f_d T)$, where J_0 is the zero-order Bessel function of the first kind.

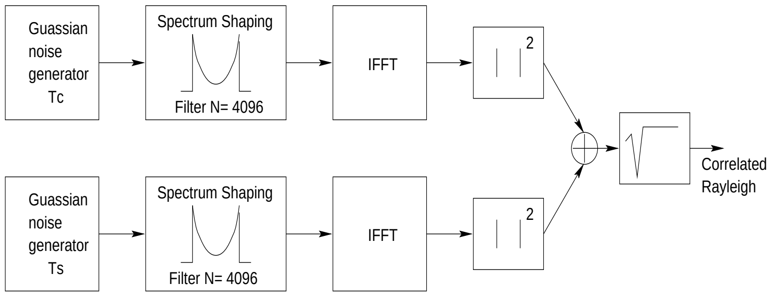


Figure 4.6: Implementation of Rayleigh fading model using FIR filters

Figure 4.6 shows implementation of Rayleigh fading generator using FIR filters. The simulator generates a one dimensional random process that obeys Clarke's model using correlation specified by the mobile's distance from the last location at which a fade value was calculated. Before the simulation, the following parameters are specified.

- Two independent seeds to initialize the Gaussian random number generators.
- Carrier frequency f_c.
- Speed of MS v.
- δt (sample time resolution) $\cdot f_d$ (maximum Doppler shift). Thus, $\delta x = \frac{c f_d \delta t}{f_c}$ (resolution in space). We choose $\delta t \cdot f_d$ to get appropriate space resolution.
- *Filter order* directly affects the correlation between two fade values. High filter order guarantees proper correlation characteristics. However, too high a filter uses a lot of memory and necessitates a lot of computation time. Thus, the filter order chosen is a tradeoff between correlation accuracy and computational efficiency. For a more detailed explanation, see reference [20].

Figure 4.7 shows the event flow and interaction diagram of short-scale fading process. The fader architecture at the `zone` initiates the fading processes, when it receives EV-FADER_UPDATE on the internal TED channel from the call admission architecture. It calculates the current fade values of the uplinks from the `MS` to all possible `BS`s and

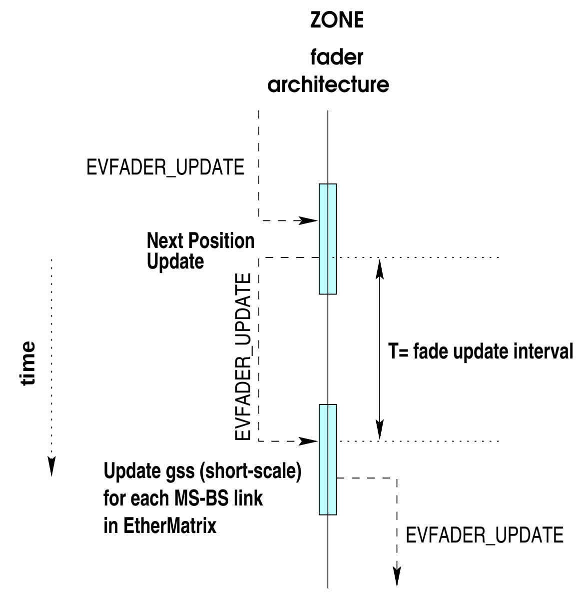


Figure 4.7: Event flow and entity interaction diagram of Rayleigh fader. A vertical shaded strip represents a process of the fader architecture in an entity, which receives and/or sends Intra-EI events to itself. Broken-line arrows show the direction of flow of Intra-EI events from the fader architecture.

updates the EtherMatrix. The fader architecture schedules the next update by sending a timer event EVFADER_UPDATE with delay equal to the short-scale fading update interval.

4.4.4 Tx-channel-Rx

The combination of the Tx architecture of the `MS`, the channel architecture of the `zone`, and the Rx architecture of the `BS` (Tx-channel-Rx) represents the data-link layer of a wireless network. The Tx architecture of the `MS` transmit packets to the parent `BS` (receiver). The *channel* architecture of the `zone` entity represents the fading and interference radio channel, which corrupts the packet by introducing errors. The channel architecture receives the packet from the `MS`. Then, the channel architecture samples and calculates uplink SIR values at each sample instant of the packet, and sends these sampled SIR values of the packet to the parent `BS` one by one. The Rx architecture of the parent `BS` receives these SIR samples of the packet and passes them to the *discriminator*. The discriminator decides if the packet is in error based on sampled SIR profile of the packet.

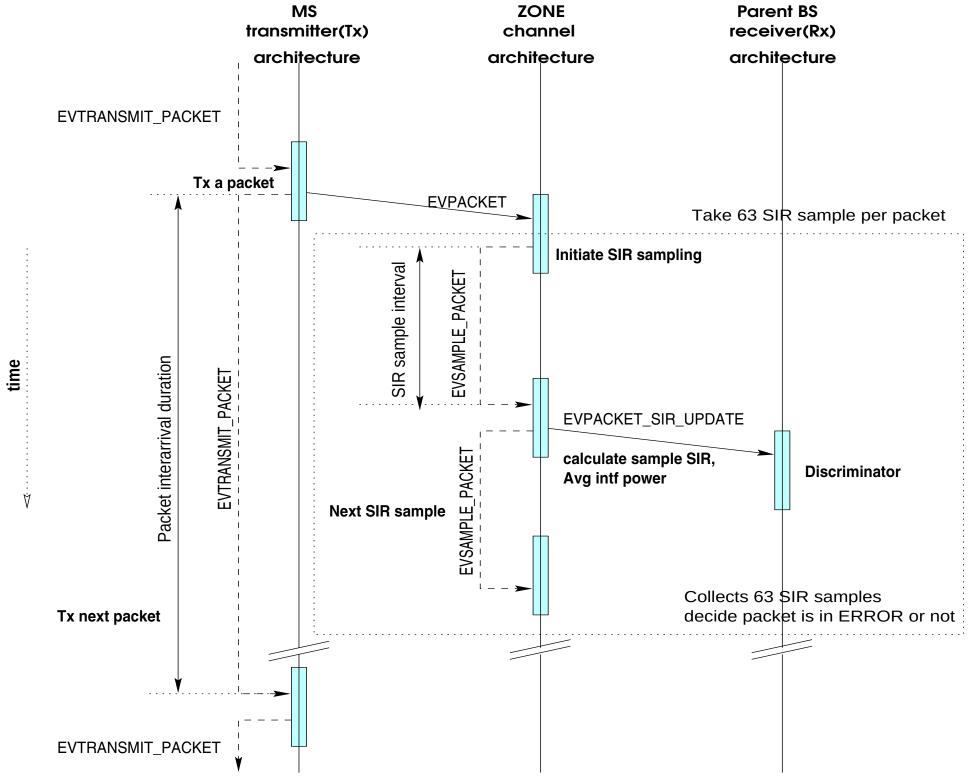


Figure 4.8: Event flow and entity interaction diagram of Tx-Channel-Rx. A vertical shaded strip represents a process, which receives Inter-EI or Intra-EI events or sends timer events to itself. Solid-line arrows show the direction of flow of Inter-EI events. Broken-line arrows show the direction of flow of Intra-EI or timer events.

Figure 4.8 shows an interaction diagram describing the working of the data-link layer. Initially, the Tx architecture of a `MS` receives EVTRANSMIT_PACKET from the call admission architecture. This event triggers the generation of packet traffic. The `MS` generates packets with a fixed time duration. The inter-arrival time between two successive packets is an exponential random variable. The packet duration and average inter-arrival time are simulation parameters.

The Tx architecture schedules the next packet transmission by sending a timer event EVTRANSMIT_PACKET with a delay equal to the exponential random inter-arrival time. The `MS` sends EVPACKET to the `zone` with the start and the end times of the packet, and the parent `BS` ID (receiver ID).

The channel architecture of the `zone` receives EVPACKET from the `MS` and stores the start and the end times of the packet, and parent BS ID. Then, the channel architecture triggers the sampling of the packet by sending EVSAMPLE_PACKET on an internal TED channel to itself. The channel architecture receives EVSAMPLE_PACKET and sets the MS transmitter status to on to indicate that the MS transmitter is emitting power.

In the SIR sampling process, the channel architecture in the zone calculates the uplink SIR at sampling instant using equation 4.3 and also calculates the average interference power at the parent BS using equation 4.4. Then, the channel architecture sends EVPACKET_SIR_UPDATE to the Rx architecture of the parent BS including the sampled uplink SIR value for the packet, the average interference power $P_{AvgIntf}(\mathrm{BS}_{\mathrm{parent}})$ at that instant on the MS-parent BS link, and the current path loss value $g_{ls}(\mathrm{MS}, \mathrm{BS}_{\mathrm{parent}})$ as event payload.

$$P_{AvgIntf}(\mathrm{BS}_{\mathrm{parent}}) = \sum_{n \neq 1} g_{ls}(\mathrm{MS}_n, \mathrm{BS}_{\mathrm{parent}}) \cdot P_{TX\,S}(\mathrm{MS}_n) \tag{4.4}$$

where $P_{AvgIntf}(\mathrm{BS}_{\mathrm{parent}})$ is the average interference power on MS - parent BS uplink. MS_n are the interfering mobile stations. $P_{TX\,S}(\mathrm{MS}_n)$ is the transmitter power of the interfering mobile stations MS_n, and $g_{ls}(\mathrm{MS}_n, \mathrm{BS}_{\mathrm{parent}})$ is the path loss value of the uplink from the interfering MS_n to the $\mathrm{BS}_{\mathrm{parent}}$ of the MS_1.

Before the end of the sampling process, the channel architecture schedules the next sampling by sending a timer event EVSAMPLE_PACKET on an internal TED channel. The delay of the event is called the sampling interval and it is a simulation parameter to control the number of samples per packet. At the end of the packet, the zone resets the MS Tx status to off indicating the MS is emitting zero power.

The Rx architecture of the parent BS receives EVPACKET_SIR_UPDATE from the zone and stores the sampled SIR value for the packet, the average interference power, and the current path loss value as in the MS state. These average interference power $P_{AvgIntf}(\mathrm{BS}_{\mathrm{parent}})$ and the path loss value $g_{ls}(\mathrm{MS}, \mathrm{BS}_{\mathrm{parent}})$ will be used later in the power control algorithm to calculate the new required MS transmitter power. The Rx architecture passes every uplink SIR sample for the packet to its discriminator, which will decide the packet is in error or not.

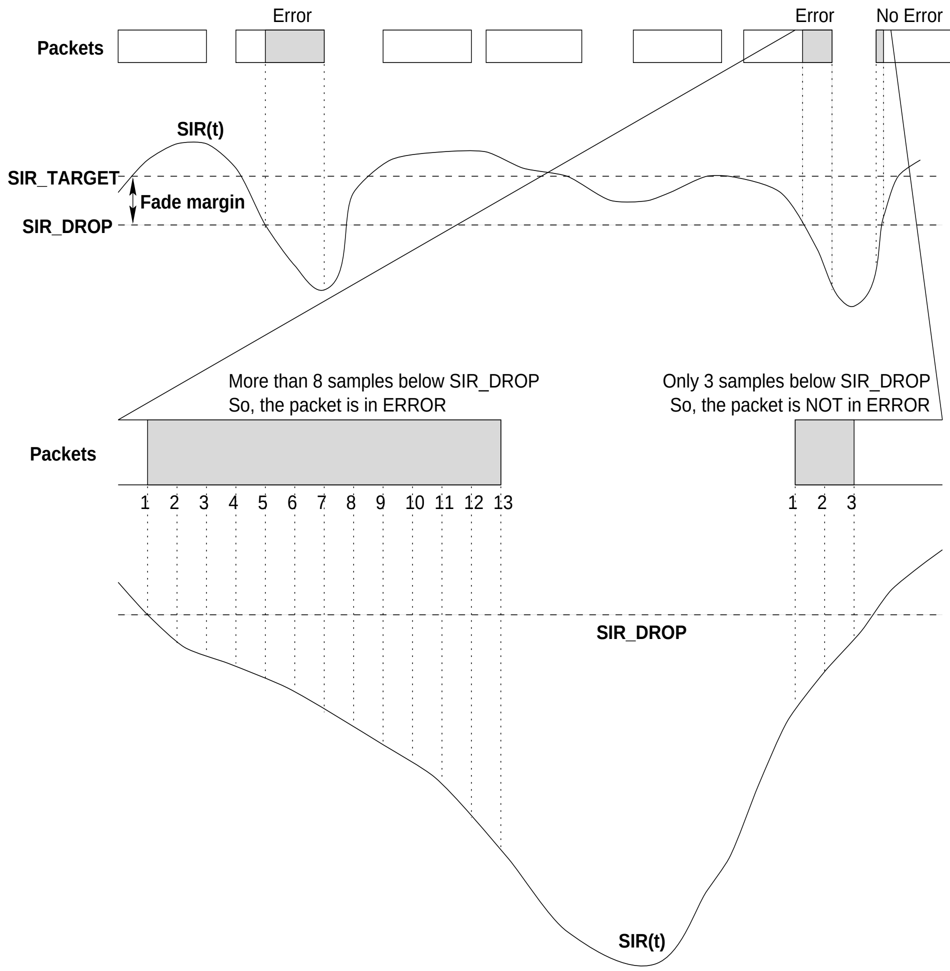


Figure 4.9: Functionality of discriminator in the receiver. In the CDPD uplink, each transmitted packet consists of 63 code symbols and each code symbol consists of 6 bits. The `zone` takes 63 samples per packet and finds a SIR value for each sample of the packet. The SIR value is assumed to be constant during a code symbol of 6 bits. At receiver, the discriminator uses 63 samples of SIR corresponding to each code symbol to determine the packet loss. If more than 8 samples are below the threshold SIR (Γ_{drop}) then the packet is considered to be in error. Shaded area of a packet shows the errors in code symbol. Fade margin (db) is equal to target SIR (Γ_t) (db) - threshold SIR (Γ_{drop})(db). See text for more detail.

WiPPET$_{packet}$ employs a simple discriminator in the Rx architecture at the parent BS receiver. The discriminator collects and analyzes each uplink SIR sample. Consider Figure 4.9, which shows the working principle of the discriminator. The discriminator compares each uplink sampled SIR with the threshold SIR (Γ_{drop}). If a sampled SIR < Γ_{drop}, a counter in the discriminator is incremented by one indicating a bad SIR sample. At the end of packet, if number of counts in the counter is greater than the predefined count (DROP_COUNT), the packet is considered to be in error and the discriminator increments the number of packets in error by one and updates PER statistics.

The purpose of building this discriminator at the receiver is to compare simulation results of WiPPET$_{packet}$ with the results shown in [12] in terms of the PER versus the fade margin for Cellular Digital Packet Data (CDPD) uplink channel. CDPD is the data standard based on AMPS. In CDPD, 20 msec long packets are transmitted from the MS to the parent BS. The packet is channel coded with the Reed-Solomon (63,47,8) block code. There are 63 symbols in the packet and the code can correct 8 symbols per packet. In reference [12], the discriminator analyzes 63 samples of each 20 msec long packet. If more than 8 samples are below the threshold SIR then the packet is considered to be in error. For a more detailed explanation, see reference [12].

4.4.5 Power Control

The main goal of the power control architecture is to adjust the required MS transmitter power to compensate the signal power loss due to distance loss and shadow fading and average interference from the surrounding cells at the parent BS. The average received uplink SIR (Γ_t) at the parent BS remains constant throughout the call duration. Figure 4.10 shows the complete interaction diagram of uplink power control algorithm.

Initially, the power control architecture of the parent BS receives EVPWR_CRTL on an internal TeD channel from the call admission architecture. The power control architecture calculates the new required MS transmitter power (msTxPWR$_{req}$) using equation 4.5 to make the average uplink SIR (Γ_t) at the parent BS remain constant

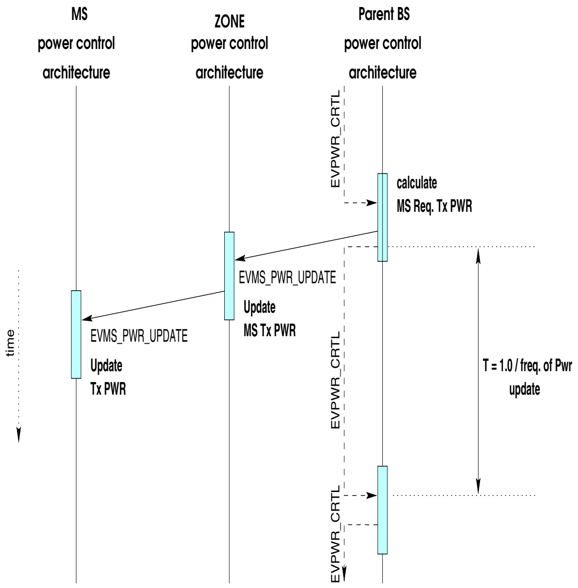


Figure 4.10: Event flow and entity interaction diagram of power control protocol. A vertical shaded strip represents a process of power control architecture in an entity, which receives Inter-EI/Intra-EI events or sends timer events to itself. Solid-line arrows show the direction of flow of Inter-EI events. Broken-line arrows show the direction of flow of Intra-EI or timer events.

throughout the call duration.

$$\mathrm{msTxPWR}_{req} = 10^{\frac{\Gamma_t}{10}} \cdot \frac{P_{AvgIntf}(\mathrm{BS}_{\mathrm{parent}})}{g_{ls}(\mathrm{MS}, \mathrm{BS}_{\mathrm{parent}})} \tag{4.5}$$

The BS sends the new required MS transmitter power using EVMS_PWR_UPDATE to the zone. The zone updates the MS state and forwards the new required power value to the MS.

4.5 Fading: performance evaluation

In this section, we verify the performance of short-scale fading with different MS speeds and compare the results. Figures 4.11 and 4.12 show the short-scale, long-scale and combined fading plots on uplink for different MS speeds.

From the Figures, it is observed that the high speed MS passes the deep fade quickly compared to slow speed MS. So, the high speed MS observes fast fades and the slow speed MS observes slow fades.

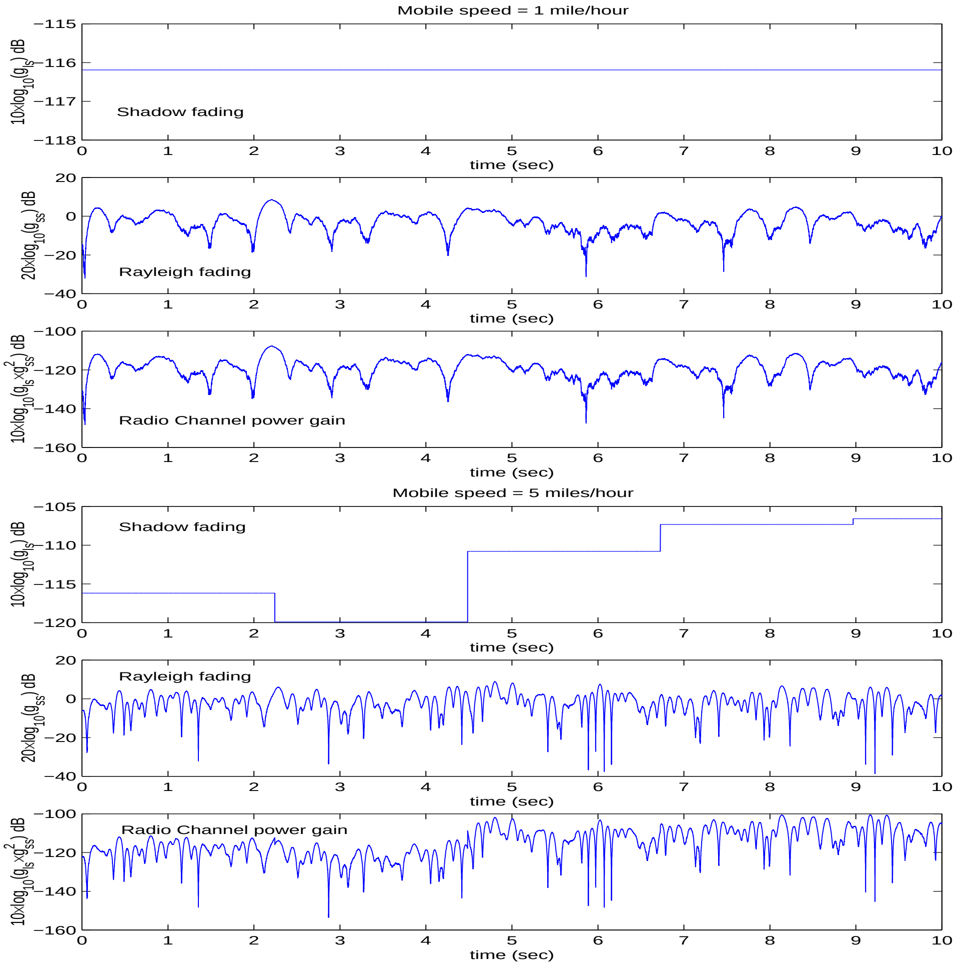


Figure 4.11: Slow speed MS observes slow fading

4.6 Packet-level Simulation: Performance Evaluation

This section discusses the results of WiPPET$_{\text{packet}}$ simulation and evaluates the performance of the simulator in terms of PER versus fade margin.

In the simulation, a `MS` transmits packets to its parent `BS` on a zero interference radio channel. The discriminator at the parent `BS` evaluates the 63 SIR samples per packet and decides if the packet is in error. The integrity of the simulator testbed (WiPPET-packet level) was verified by comparing the results of the simulations with previously published analytical results in [12].

Tabel 4.1 shows the WiPPET packet level configuration parameters and algorithms used for comparison of the simulation results. Figure 4.13 shows PER versus fade

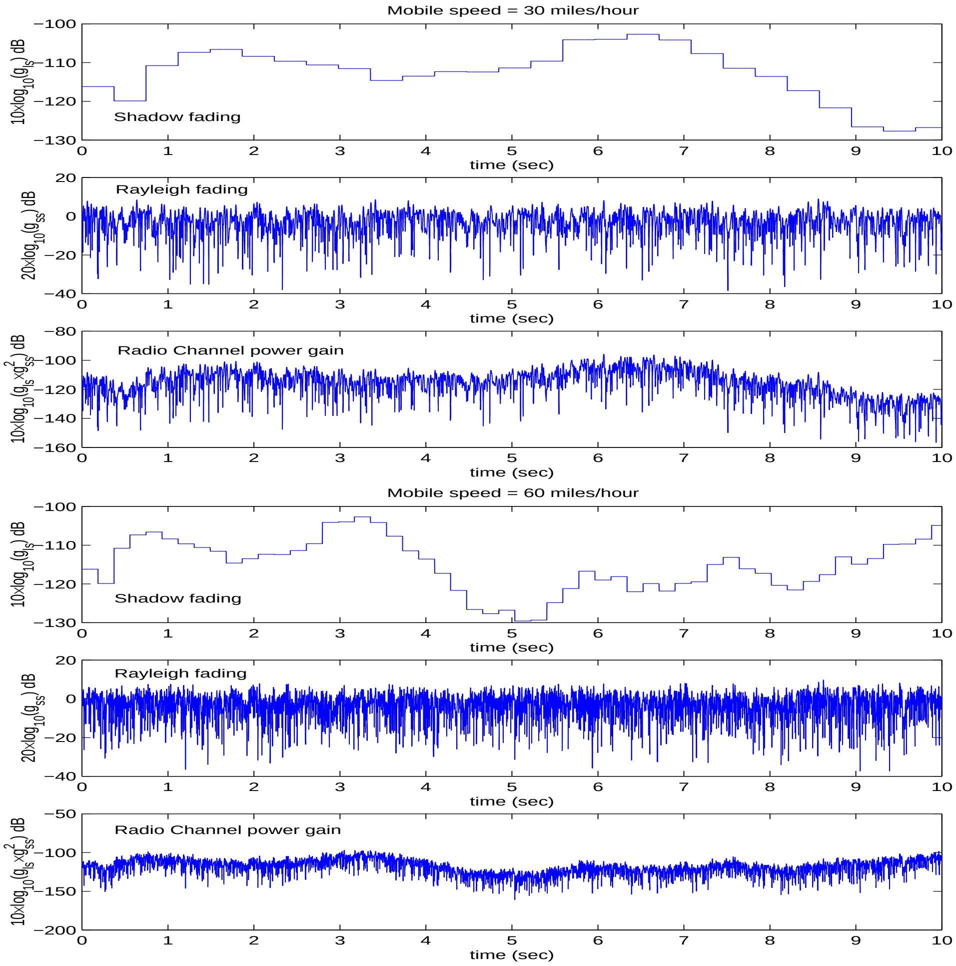


Figure 4.12: High speed MS observes fast fading

margin, which is comparable to the results shown in [12]. The reader may also refer to [12] for more details on the algorithms and the interpretation of the results.

parameter [units]	2D Macrocell
Number of BSs	1
Number of Channels	1
Geography Style	Manhattan
Geographical Area	$2km \times 2km$
Cell Radius [m]	1000
Antenna Height [m]	50
Distance Resolution [m]	5
Shadow Fading Standard Deviation [dB]	6
Shadow Fading Correlation Distance [m]	50
Noise Power [dBW]	-150
Mobile Speed [miles/hour]	1 - 60
Probability of Going Straight	0.6
Time Between fade update [msec]	0.32
Time Between Power Updates [msec]	40
BS Assignment Algorithm	Strongest Pilot BS
Packet duration [msec]	20
Packet average inter-arrival time [msec]	5
No. of SIR samples per packet	63
Max SIR drop count (DROP_COUNT)	8
Target SIR (Γ_t) [dB]	7-33
Drop SIR (Γ_{drop}) [dB]	5

Table 4.1: WIPPET$_{\text{packet}}$ simulation parameters

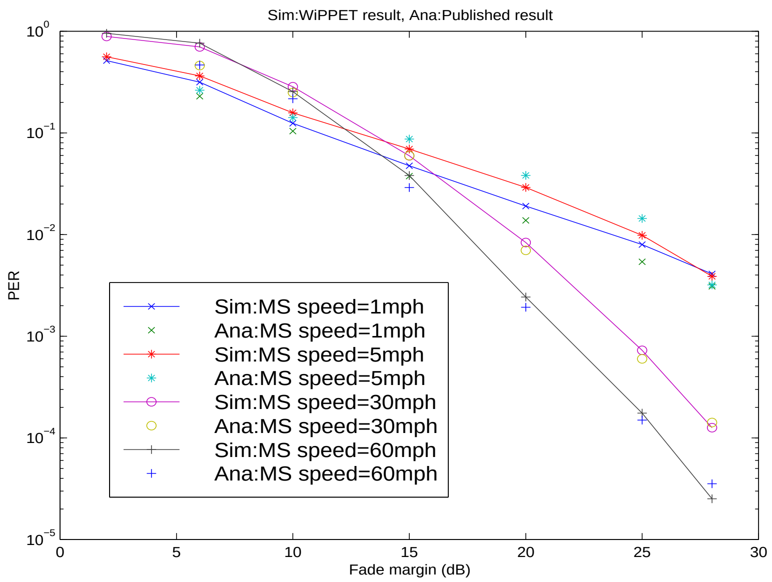


Figure 4.13: Packet error rate (PER) versus fade margin in zero interference and short-scale fading environment as evaluated by WIPPET$_{\text{packet}}$ and by analysis.

Chapter 5

Performance of Parallel Simulation

Parallel simulation performance is measured in terms of execution time speedup and efficiency based on experiments performed in a multi-processor environment. This chapter introduces and compares two methods of parallelizing WiPPET$_{\text{session}}$: geography parallelization & radio channel parallelization.

5.1 Rollbacks and Load Balancing

Rollback is an important phenomenon to understand in an optimistic parallel simulator. Rollbacks slow down the simulation, therefore, it is desirable to find a design in which rollbacks are rare. As we discussed earlier in section 2.2, rollback is initiated by an inter-processor event whose timestamp is earlier than the current simulation time of the processor to which it arrives. In the optimistic parallel discrete event simulator (PDES), rollbacks are executed by undoing the simulation until the simulation time at that processor is earlier than the timestamp of the arriving event. The effort spent on performing rollbacks depends on *length of the rollbacks* and *number of rollbacks*.

Two heuristics for minimizing rollback cost are a) to minimize the number of inter-processor events, and b) to ensure that the simulation time progresses at the same rate on every processor. The latter is called synchronization of simulation time among processors, which is directly related to load balancing. It is possible before starting a simulation to distribute simulation load approximately equally among processors by mapping equal number and same type of TeD entities to each processor. However, the method is only approximate because the dynamic nature of the simulation allows that the same entity can transit between idle or compute-intensive behavior.

Rollback is unavoidable in the optimistic PDES because it is necessary to exchange the information among processors during the simulation. However the effect of rollbacks can be reduced by proper load balancing and by reducing the number of inter-processor events. The experiments described herein evaluate the effectiveness of competing designs for mitigating rollback effects.

5.2 Methods of parallelization

A successful parallelization of a simulation should have the property of scalability, i.e., the execution time on N processors should be approximately N times smaller than the execution time on one processor. In practice, however, the reduction of simulation time is smaller due to the occurrence of rollbacks.

5.2.1 Simulation Configuration and Parameters

The following are WiPPET$_{\text{session}}$ configuration and parameters used to evaluate performance of parallel simulation. This is called the *normal* simulation configuration.

1. 10 meter geographic resolution of a synthetic Manhattan–style urban environment with 24 by 24 city blocks (geographical area 12km×12km)

2. Mobile station trajectories modeled by a biased random walk with probability of going straight 0.6.

3. `MS` pool is of 500 mobile stations.

4. Mobile station speed is 25 meters/second constant.

5. Mobile station arrival rate (λ_{MS}) is 2.00 mobiles per second.

6. Average call duration (μ_{MS}) is 120 seconds.

7. 48 base stations serving 240 mobiles (average) over 40 frequency channels (load 0.125 Erlang/channel/base).
8. Channel Allocation: Least Interference Dynamic Channel Allocation (LI-DCA) algorithm (for call admission and handoff).
9. Asynchronous distributed uplink power control.
10. Handoff: RSSI based MADRAS handoff (RSSIBHO).
11. Handoff check and power control update every 0.5 second.
12. Test simulations run for 10,000 seconds.

TeD/C++ version 2.0, GTW version 2.3 and GNU project C++ Compiler were used. The experiments were conducted at WINLAB, Rutgers University on a 14–processor Sun Ultra enterprise server 4000 machine, having 3 GB shared memory and SunOS 5.6.

5.2.2 Parallelization of the Geography

The `zone` is the most complex entity in WiPPET. As we discussed earlier in section 3.2, the `zone` manages mobility, carries a Manhattan geography and the associated path loss matrix, and calculates interference on a `MS` uplink channel during call admission, power control and handoff protocols.

This method of parallelization is motivated by an idea that it is advantageous to divide and conquer the `zone`'s heavy computation demand by partitioning the model, when possible, into multiple `zone`s for parallel simulation. This approach of dividing the geography into multiple `zone`s is called *geography parallelization*. In geography parallelization, each `zone` contains a fraction of the geography points together with the associated path loss sub-matrix. Figure 5.1 shows WiPPET overview for geography parallelization and partition of the Manhattan geography into four `zone`s and their associated path loss sub-matrices.

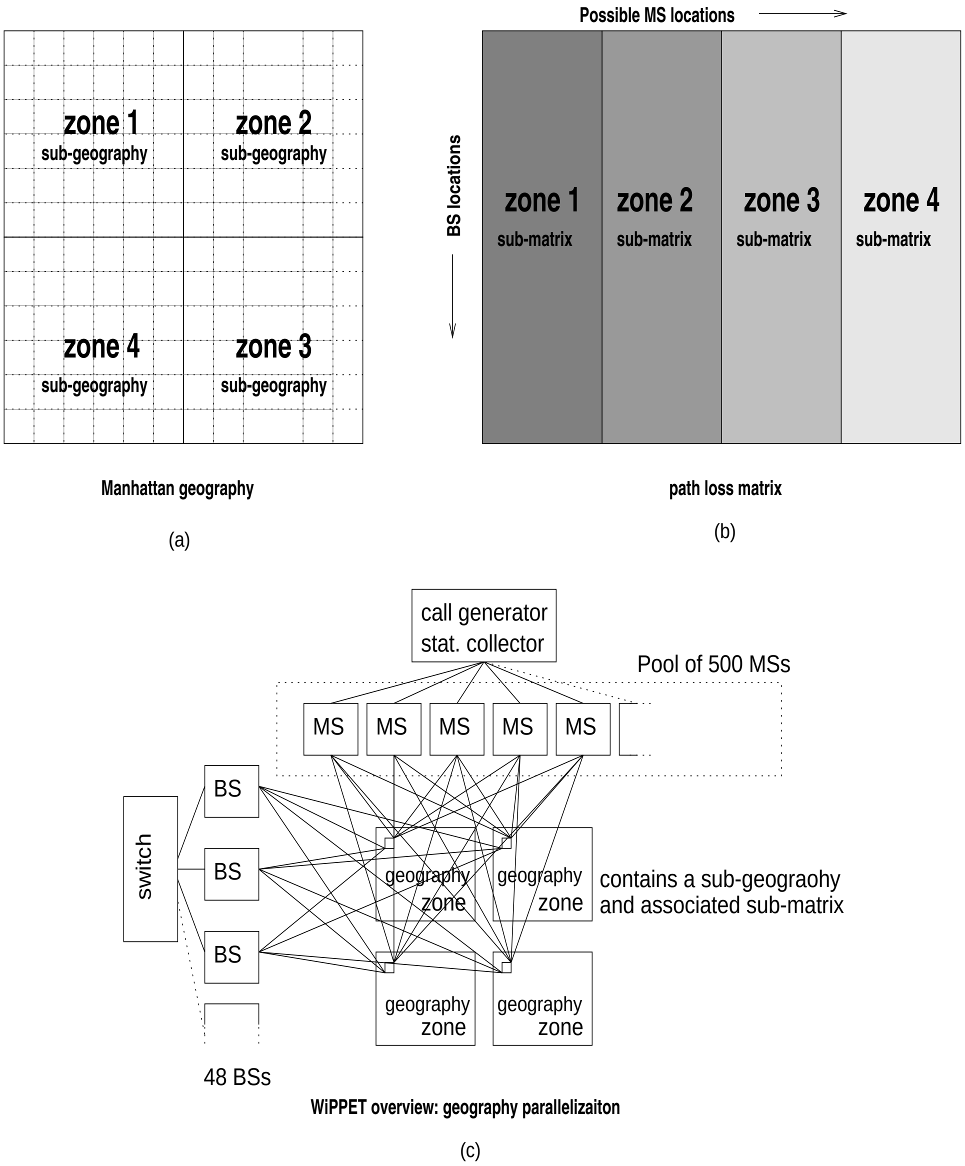


Figure 5.1: Partition of geography into four `zone`s (a) Manhattan geography is divided evenly into four sub-geographies. Each `zone` contains one sub-geography stored in its deferred constant. (b) Path loss matrix is divided into four sub-matrices. Each sub-matrix associated with the sub-geography is stored in each `zone` as its deferred constant (c) WiPPET overview with a `callgen`, 500 `MS`s in the pool, 4 `zone`s each contains a sub-geography and associated path loss sub-matrix, 48 `BS`s and, a `switch`. This configuration is used for four processor simulation. Similarly, one can partition the whole geography into N `zone`s and run N processor simulation in which each `zone` is mapped on one processor.

Load Balancing

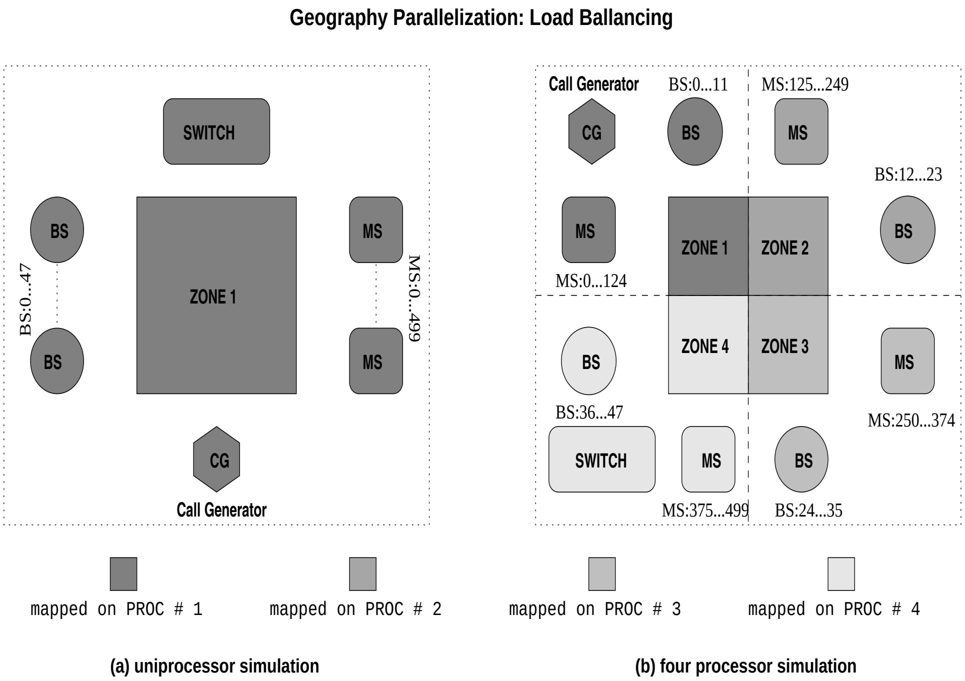


Figure 5.2: Load balancing in geography parallelization (a) uniprocessor simulation. a `callgen`, 500 `MS`s in the pool, a `zone` contains the whole geography and associated path loss matrix, 48 `BS`s and, a `switch` are mapped on one processor (b) four processor simulation. The `callgen`and the `switch` are mapped on processor 1 and processor 2 respectively. One fourth of `MS` pool, i.e. $500/4 = 125$ `MS`s, are mapped on each processor. Similarly, one fourth of `BS`s, i.e. $48/4 = 12$ `BS`s, are mapped on each processor. A `zone` including a sub-geography and associated path loss sub-matrix is mapped on each processor. This uniform load balancing among processors are done before starting of a simulation. one can extend philosophy of partitioning the whole geography into N `zone`s, divide `MS`s and `BS`s equally among N processors and run N processor simulation in which each `zone` is mapped on one processor.

In geography parallelization, the geography is divided into number of zone entities. The number of `zone`s equals to the number of processors used for a simulation. Each `zone` including a sub-geography and the associated path loss sub-matrix is mapped on one processor. The sharing of the simulation load equally among processors for a uniprocessor and four processor simulations are shown in Figure 5.2. One can extend the same mapping philosophy for N processor simulation.

Experiments and Analysis

The entire WIPPET$_{\text{session}}$ protocol stack, including mobility (Mob), call admission (CA), power control (PC), and handoff (HO) are simulated in the normal configuration mode. First we define some terminology used in discussion of parallel simulation performance.

- The *speedup* of N processor simulation is defined as the ratio of computing time or execution time on a uniprocessor to the computing time on N processors.
- The *net* events of N processor simulation is defined as total events required, excluding rollback events, to complete the simulation.
- The *excess* events of N processor simulation is defined as difference between net events of the N processor simulation and net events of the uniprocessor simulation. That is, the total number of *excess events* of N processor simulation = number of *net events* of N processor simulation – number of *net events* of uniprocessor simulation. Thus,

$$\%\text{ excess events }(N\text{ processors}) = \frac{\text{excess events }(N\text{ processors})}{\text{net events (uniprocessor)}} \times 100 \qquad (5.1)$$

The simulation experiments are conducted on 1,2,4,6, and 8 processors. Figure 5.3(a) shows the experiment results in terms of speedup versus number of processors. The results show that the speedup in execution time is not impressive as the number of processors increases. In fact, for 6 processor and 8 processor simulations, the speedup is almost 1. That is the execution or computing time for 6 processor and 8 processor simulations is almost equal to the execution time taken by the uniprocessor simulation. We now examine the source of this insignificant speedup.

Figure 5.3(b) shows percentage of excess events for 2,4,6, and 8 processors simulations with reference to 0 percentage of excess events in the uniprocessor simulation. According to Figure 5.3(b), the excess events are increasing with number of processors in the simulation. That means the number of net events in multi-processor simulations are a

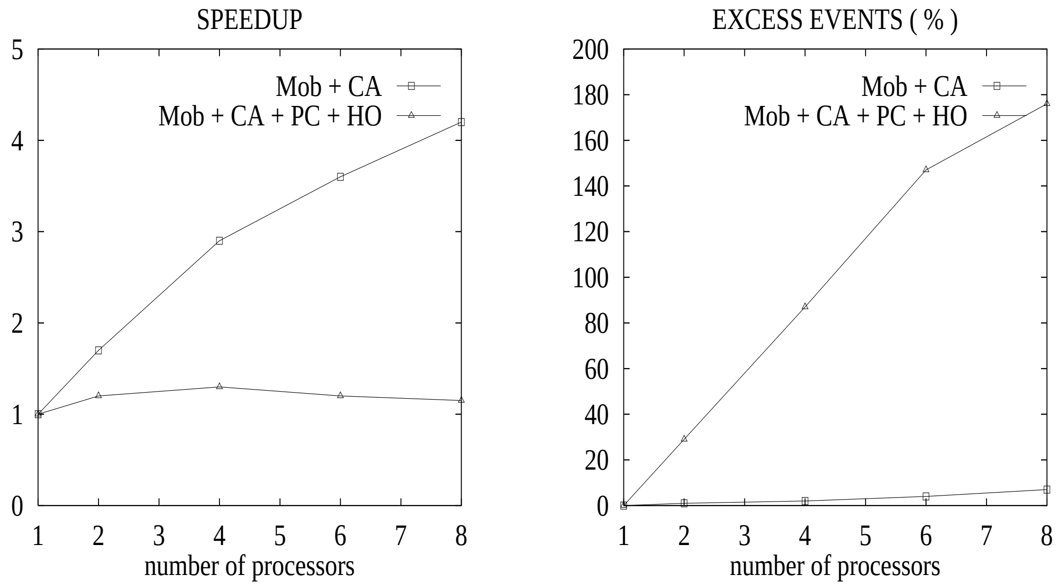


Figure 5.3: Experiment results of geography parallelization (a)[left] speedup (b)[right] percentage of excess events. See text for more analysis.

lot more than that of the uniprocessor simulation. Hence, we conclude that by porting the simulation to a multi-processor platform, the simulation problem is no longer the same. Here, these excess events may contain many inter-processor events, which cause undesirable rollbacks in the multi-processor simulation. The processes which introduce these excess events, are as follows:

1. **zone transfer – mobility** : In the uniprocessor environment, there is only one `zone`, hence, there will be no `MS` zone transfer. In the multi-processor environment, there are more than one `zone` due to division of geography into multiple `zone`s, hence, there will be `MS` zone transfers from one `zone` mapped on one processor to another `zone` mapped on a different processor. As we discussed in the mobility protocol in subsection 3.7.5, when a `MS` zone transfer occurs, the old `zone` transfers the `MS` data structures to the new `zone` using an event. This event is an excess event and it is also an inter-processor event.

2. **Interference calculation on every channel – call admission channel allocation**: In call admission, the channel allocation architecture of LI-DCA algorithm at the parent `BS` queries a list of interference values on every channel. The

parent BS collects a list of interference values on every channel and the LI-DCA algorithm finds the least interference channel, which is assigned to the new MS. In a multiple zone scenario, there may be one or more MSs present in every zone, which are using the same uplink channel. We cannot neglect the fact that a MS transmitting to its parent BS on a radio channel from one zone produces uplink interference on the same radio channel at BSs in other zones.

Hence, in the multi-processor simulation with multiple zones, to take into account the interfering MSs of all zones, the parent BS requests a list of interference values on every channel to all zones, which are mapped on different processors. On other hand, in a uniprocessor simulation, the parent BS requests only one zone.

In a multi-processor simulation, each zone responds to the request by sending a list of locally calculated interference values on every channel to the parent BS. The parent BS sums up the lists of the interference values and constructs a list of total interference values. The algorithm finds the least interference channel from the list of total interference values on every channel.

Hence combining all the request and result events for a N processor simulation, there will be (1 request + 1 result)$\times(N-1)$ excess events per call admitted.

3. **Interference calculation on a channel – power updates and handoff check**: The power control algorithm and handoff check procedure at the parent BS require an interference value on the MSuplink channel. In power control, the parent BS updates the required MS transmitter power (msTxPWR$_{req}$) based on the current interference on the MS uplink. Similarly, in the handoff check, the parent BS decides the possibility of handoff of the given MS based on the current interference on the MS uplink. Refer sections 3.7.6 and 3.7.7 for more detail.

   In a multi-processor simulation, the parent BS requests all zones for interference on the MS channel and all zones reply back. Then, the parent BS sums up the interference values from all zones. Hence, the number of excess events per handoff check and power control update is (1 request + 1 result)$\times$(1 power control update + 1 handoff check)$\times(N-1)$. Now, let us assume that frequency of handoff check and power control update is $1/\Delta T$. That means after every ΔT time, there will be one power control update and one handoff check events. According to the normal configuration mode, an average life time of a call is μ_{MS} seconds, hence, there will be $\frac{\mu_{MS}}{\Delta T}$ power control updates and handoff checks per call on average. Hence combining all the request and result events for the N processor simulation, the average number of excess events in the power control update and the handoff check per call will be (1 request + 1 result)×(1 HO + 1 PC)×$(N-1) \times \frac{\mu_{MS}}{\Delta T}$.

4. **Interference calculation on every channel – handoff channel allocation**: In handoff, the channel allocation architecture of the LI-DCA algorithm at the parent `BS` requires interference on every channel to find a channel for a hand-in `MS`. Based on our discussion of excess events due to channel allocation in the call admission protocol, for N processor simulation, the excess events per channel allocation in the handoff will be (1 request + 1 result)×$(N-1)$. Let us assume that an average handoffs per call is $A_{handoff}$. Hence, for N processor simulation, there will be (1 request + 1 result)×$(N-1) \times A_{handoff}$ excess events in the handoff channel allocation per call on average.

Let us assume that an arrival rate of mobile stations is λ_{MS} and total simulation time is T_{total} seconds, then, there will be $T_{total} \times \lambda_{MS}$ `MS`s admitted during the entire simulation. From above discussion, the average total excess events (N_{EE}) in interference calculation for N processor simulation for T_{total} seconds with the average arrival rate of λ_{MS} `MS`s per second, the average call duration of μ_{MS} seconds, and the average $A_{handoff}$ handoffs per call will be

$$N_{\mathrm{EE}} = 2(N-1)T_{total}\lambda_{MS}[1 + 2\frac{\mu_{MS}}{\Delta T} + A_{handoff}] \tag{5.2}$$

Total excess events will be the excess events due to **interference calculation** from equation 5.2 plus the excess events due to **zone transfer**. From equation 5.2, the number of excess events depends on number of processors and also linearly increases with number of processors.

Now, the next thing is to find which excess events are dominating the multi-processor simulation. For that, we simulated only mobility and call admission (Mob+CA) protocols in a multi-processor environment. The simulation configuration mode is normal and load balancing pattern is the same as it was used for the previous experiment as shown in Figure 5.2. Figure 5.3(a) and 5.3(b) show speedup and percentage of excess events for 1,2,4,6, and 8 processor simulation respectively. The results show impressive speedup and no significant increase in excess events as number of processors increases in the simulations. As we discussed earlier, in mobility and call admission protocols the excess events are due to zone transfer and channel allocation respectively. From the experimental results, we can see that only Mob and CA protocol simulations give the desired speedup. Hence, the excess events due to call admission and mobility protocols do not affect the speedup. We conclude that the culprit is the excess events in interference calculation in the power control updates and the handoff check procedure, that dominate the multi-processor simulation.

5.2.3 One Zone Parallelization

Our first approach toward reducing excess events is a one zone multi-processor simulation with entire WiPPET$_{\text{session}}$ protocol stack. Since there is only one `zone`, there will be no excess events.

The results of the 1,2, and 4 processor simulations are shown in Figure 5.4. The results show the noticeable speedup, but, in this scenario, load balancing of a `zone`, a `callgen`, a `switch`, 48 `BS`s and 500 `MS`s entities among the processors is difficult. From Figure 5.4, we can see that speedup curve shows the downward trend after 3 processor simulation. The reason for the downward trend is that it is difficult to achieve proper load balancing with four or more processors. Moreover, the one zone multi-processor simulation beyond four processors is not scalable in the normal configuration.

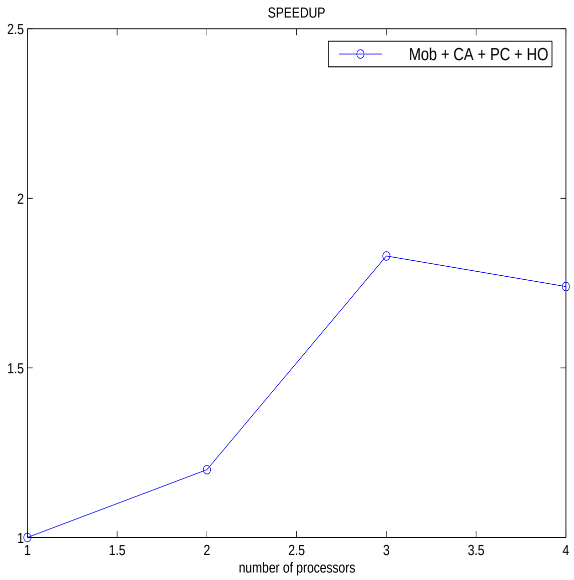


Figure 5.4: Speedup of one zone parallelization. See text for more detail.

5.2.4 Parallelization of the Radio Channels

Instead of dividing the geography into multiple `zone`s, we divide the radio channels into multiple `zone`s and map these `zone`s on different processors. This method is called *radio channel parallelization*. In radio channel parallelization for multi-channel networks, each `zone` contains the entire geography and the associated path loss matrix, but only a fraction of the radio channels. Figure 5.5 shows, as an example, WiPPET overview with four `zone`s for radio channel parallelization.

Load Balancing

In radio channel parallelization, the total radio channels in the system are divided almost equally and alloted to each `zone`. The number of `zone` entities equal to the number of processors used for the simulation. The sharing of the simulation load among processors for a uniprocessor and four processor cases are shown in Figure 5.6. One can extend the same mapping philosophy to N processor simulations.

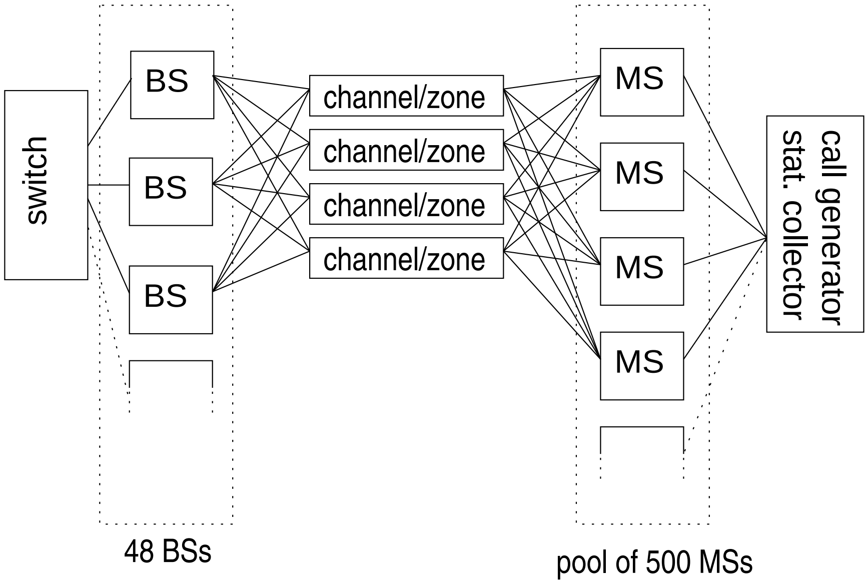


Figure 5.5: WiPPET overview – partition of radio channels into four radio channel `zones`. It has a `callgen`, 500 `MS`s in the pool, 4 `zone`s each contains one fourth of radio channels and the whole geography with associated path loss matrix, 48 `BS`s and, a `switch`. This configuration is used for four processor simulation. Similarly, one can divide the radio channels into N groups and assign each `zone` a group and run N processor simulation in which each `zone` is mapped on one processor.

Experiments and Analysis

Let us first begin with analysis. To get better speedup, the number of excess events should be reduced in the multi-processor simulation. The processes which introduce the excess events, are as follows:

1. **zone transfer – mobility**: A `MS` may leave or enter a `zone` in case of the channel assignment or reassignment. That is, transferring the `MS` from one radio channel `zone` to a new radio channel `zone`. Hence, a zone transfer occurs in the call admission channel allocation and the handoff channel reallocation. These zone transfers generate excess events in multi-processor simulation.

2. **Interference calculation on every channel – channel allocation**: The channel allocation of the LI-DCA algorithm at a parent `BS` collects a list of interference values on every channel in the call admission and the handoff protocols. Then, the algorithm finds the least interference channel, which is assigned to the new `MS`. In

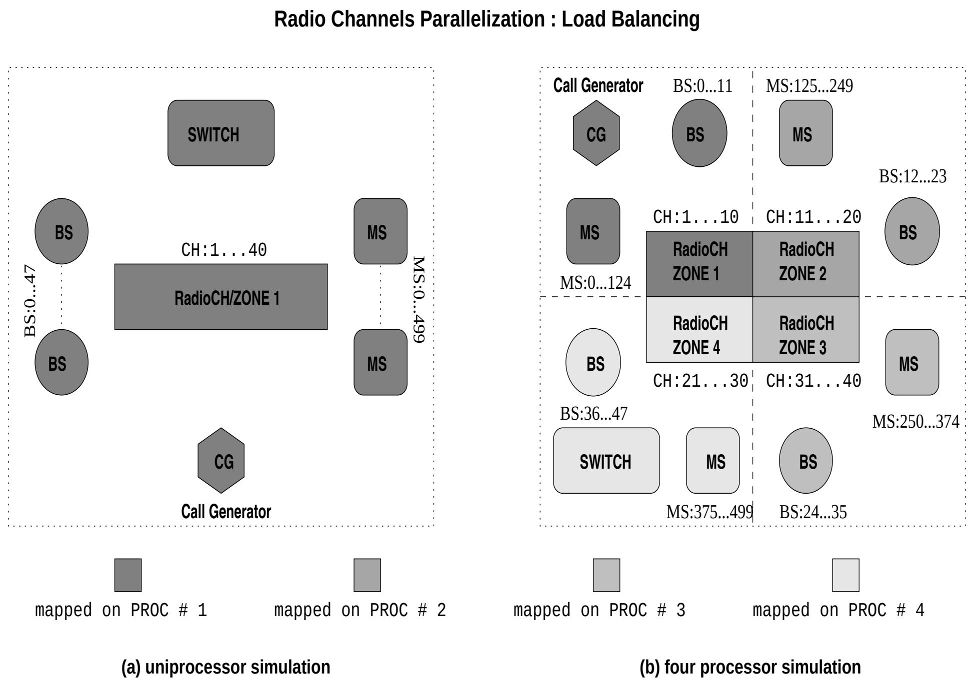


Figure 5.6: Load balancing in radio channels parallelization (a) uniprocessor simulation. a `callgen`, 500 `MS`s in the pool, a `zone` contains all radio channels, the whole geography and associated path loss matrix, 48 `BS`s and, a `switch` are mapped on one processor (b) four processor simulation. The `callgen`and the `switch` are mapped on processor 1 and processor 2 respectively. One fourth of `MS` pool, i.e. $500/4 = 125$ `MS`s, are mapped on each processor. Similarly, one fourth of `BS`s, i.e. $48/4 = 12$ `BS`s, are mapped on each processor. A `zone` including the whole geography with associated path loss matrix and one fourth of radio channels, i.e $40/4 = 10$ radio channels, is mapped on each processor. This uniform load balancing among processors are done before starting of a simulation. one can extend philosophy of partitioning the radio channels into N groups and assign each `zone` a group of radio channels, divide `MS`s and `BS`s equally among N processors and run N processor simulation in which each `zone` is mapped on one processor.

radio channel zone parallelization, a `zone` in a multi-processor simulation contains a fraction of channels. To collect the interference values on every channel, the `BS` requests every `zone` for the interference values on every channel. These request and result events are the excess events in the multi-processor simulation.

3. **Interference calculation on a channel – power updates and handoff check**: In the power control algorithm and handoff check procedures, the parent `BS` requires to calculate the interference on the `MS` uplink channel. Since there is only one `zone`, which owns the given `MS` uplink channel, the parent `BS` request only that particular `zone` to find the interference on the `MS` uplink channel. Hence, there are no excess events in the power control algorithm and handoff check procedures in the multi-processor simulation.

From the above discussion, we conclude that in radio channel zone parallelization the excess events are generated by the zone transfer and the channel allocation.

Now, let us conduct experiments to find the effect of these excess events on speedup in multi-processor simulation. The experiments were conducted on 1,2,4,6, and 8 processors simulating the entire WiPPET protocol stack (Mob+CA+PC+HO) in the normal mode. Figure 5.7(a) and Figure 5.7(b) show speedup and percentage of excess events for 1,2,4,6, and 8 processor simulations respectively.

The simulation results show the desired speedup and no significant increase in excess events as number of processors increases. Hence, the excess events due to the zone transfer and the channel allocation are not prominent in radio channel parallelization, which matches with the conclusion made for geography parallelization.

Based on previous two experiments 1) geography parallelization with Mob+CA only; 2) radio channel parallelization, the excess events due to the zone transfer and the channel allocation are not prominent in the multi-processor simulation and, hence, do not effect the speedup. On the other hand, the excess events generated by frequent (every 0.5 second) interference calculation in the power control update and the handoff check procedure dominate the multi-processor simulation causing the insignificant speedup in

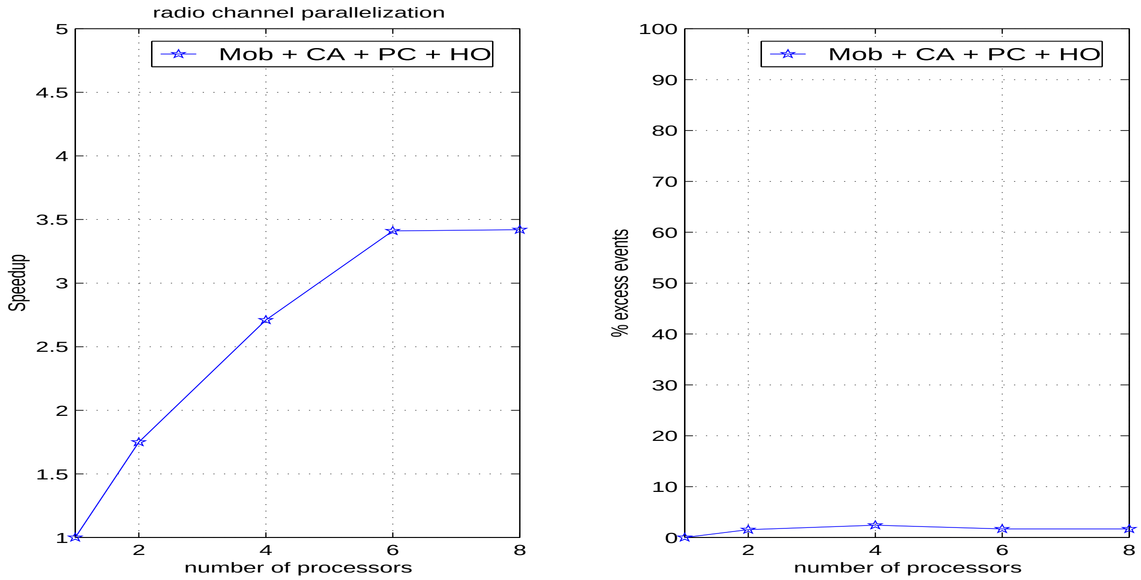


Figure 5.7: Radio channel parallelization (a)[left] speedup (b)[right] percentage of excess events. See text for analysis.

geography parallelization.

5.3 Sensitivity of Performance to Parameters

Previously, we introduced three methods of parallelizing WiPPET, geography parallelization, one zone parallelization, and, radio channel parallelization. We also concluded that by parallelizing the simulation, the model network remains the same but the simulation problem no longer remains the same because of the introduction of excess events.

The purpose of this section is to understand the sensitivity of the performance to simulation parameters and to summarize the parallel simulation performance of geography and radio channel partitioning.

5.3.1 Experiments

Table 5.1 shows seven experimental scenarios to evaluate geographic and radio channel partitioning. It specifies the Poisson call arrival rate, the geographic resolution, the

parameter	**scenario**							
[units]	Normal	A	B	C	D	E	F	G
traffic [calls/s]	2	2	6	2	2	6	2	2
resolution [m]	10	10	10	2	10	10	10	10
PWR period [s]	0.5	–	–	–	0.5	0.5	0.1	0.5
speed [m/s]	25	25	25	25	25	25	25	2.2

Table 5.1: Seven experimental scenarios to evaluate geographic and channel `zone` partitioning.

power control update period and the traveling speed of the `MS` for seven scenarios that were tested with geographic and channel partitioning on up to 8 processors.

Scenarios A, B, and C, simulate only mobility and call admission, called mobility (dominating activity) scenarios. A is the normal configuration mode among the mobility scenarios, B has 3× offered traffic, and C has 5× finer geographic resolution than normal mode (see Tabel 5.1). In scenario B, the average number of `MS`s in the system will increase due to increase in the offered traffic. In scenario C, the `MS` position updates occur more frequently for the finer geographic grid resolution.

Scenarios D,E,F and G, simulate mobility, call admission, power control and handoff, called interference (dominating activity) scenarios. D is the normal configuration case in the interference scenarios, E has 3× offered traffic, F has 5× more frequent power updates (more frequent interference calculation), and G has 11× slower moving `MS`s (less frequent position updates).

Model partitioning depends on the number of processors used. The mapping strategy is the same as shown in Figures 5.2 and 5.5 for geography parallelization and channel parallelization respectively.

5.3.2 Discussion and Summary

The experimental results appear in Figures 5.8 and 5.9. Figures 5.8 and 5.9 share a similar two-by-two panel layout such that the left column refers to geographic `zone` partitioning, the right column refers to channel `zone` partitioning, the top row refers to mobility scenarios (A, B, C), and the bottom row refers to interference scenarios (D,

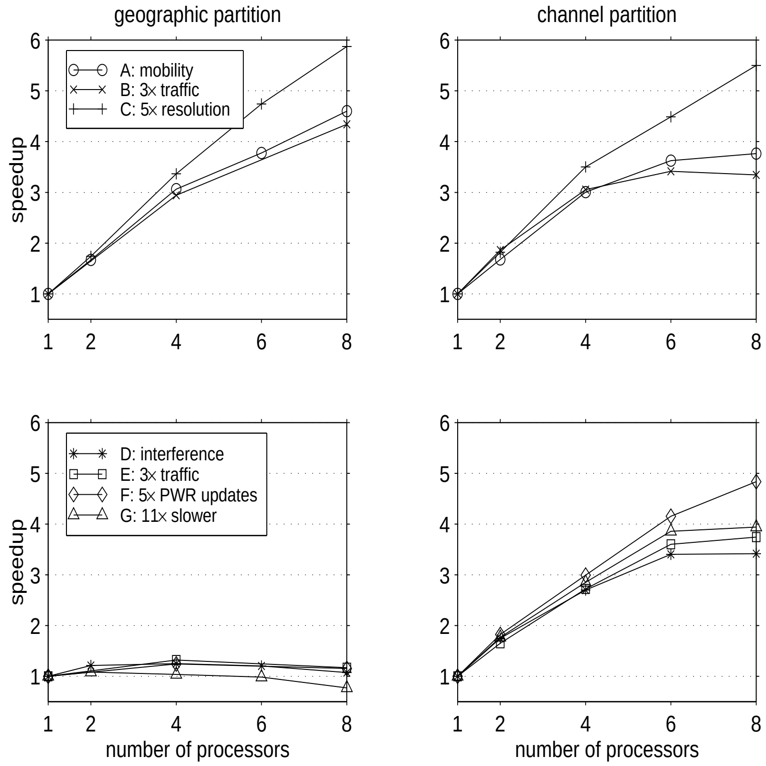


Figure 5.8: The speedup for N processors is defined as the ratio of computing time on one processor to the computing time on N processors.

E, F, G).

The results in Figure 5.8 show a strong speedup in some but not all of the scenarios. It is interesting to analyze how this happens. As we discussed earlier, that parallelization introduces the excess events and causes undesired speedup. From a simulation standpoint, however, that is not enough to consider the costs of partitioning; we must also consider the mechanisms that force synchronization among processors and lead to rollbacks.

It is helpful to think of each scenario as a blend of pure mobility activities and pure interference calculation activities. The next four paragraphs consider partitioning costs and synchronization mechanisms for the four combinations of dominating/major activity and partitioning method.

- Mobility/geographic: Mobility is an autonomous activity of each `MS` that is managed by its `zone`. Partitioning introduces zone transfer (inter-processor) events.

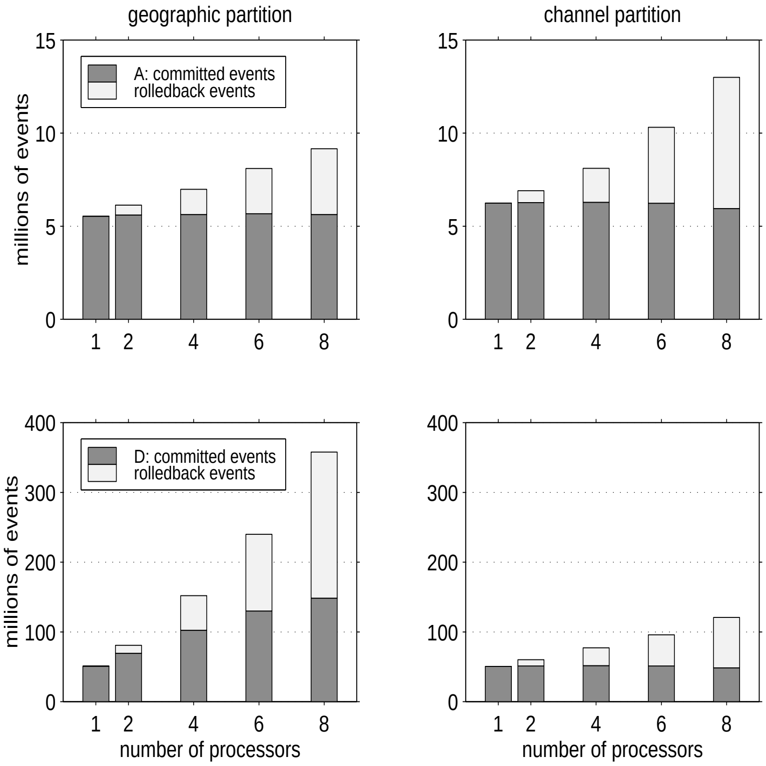


Figure 5.9: The figures show numbers of committed and rollbacked events to perform the same 10000 second simulation for the nominal mobility scenario (A) and the nominal interference scenario (D). Subdividing `zone`s requires additional events to communicate results among `zone`s and for zone-transfer of mobiles. The effect is especially evident in geographically parallelized interference calculations.

Synchronization is forced when an `MS` transfers from one `zone` to another by moving.

- Mobility/channel: In channel partitioning, zone transfers are also introduced and synchronization is forced when an `MS` transfers from one `zone` to another by changing a channel. In both channel and geographic partitioning, zone transfers happens at most a handful of times per call. The low cost and rarity of synchronization required for pure mobility is supported by the strong speedup curves in the top panels of Figure 5.8.

  In those panels, note especially scenario C, which has the $5\times$ fine geography grid (i.e. $5\times$ frequent position updates). Since the position update is an independent and a local activity (it limited to the particular `zone`), it doesn't require synchronization among the processors except in zone transfer (nearly ideal situation). In this, so called nearly ideal situation, the workload of position updates is divided equally among the processors, so, the simulation should give almost linear speedup (see equation 2.1), which is substantiated by the speedup curves in Figure 5.8 for both parallelization methods.

- Interference/channel: For orthogonal radio channels, interference in one channel is completely independent of interference on other channels (local activity). Zone transfer is the only event introduced by partitioning. Here again, synchronization is forced only when the `MS` transfers from one `zone` to another by changing a channel. The rare synchronization required with orthogonal radio channels is supported by the strong speedup curves in the lower right panel of Figure 5.8.

- Interference/geographic: As we discussed earlier in geography parallelization section, within a radio channel, interference is a simultaneous phenomenon across the geographical extent of the network. Therefore the cost of geographic partitioning is excess events due to the hundreds of interference calculation requests. Since these excess events are inter-processor events, can we blame this extensive synchronization requirement for the dramatic failure of geographic parallelism seen in the lower left panel of Figure 5.8 (which shows virtually no speedup with increasing number of processors)? Surprisingly not. In fact the lower panels of Figure 5.9 show that the efficiency of the optimistic simulator (the ratio of net/committed events to total events) is nearly identical for interference scenarios under geographic versus channel partitioning. The culprit is the total *number* of events.

We have touched on the fact in previous sections that partitioning of a model into multiple `zone`s necessarily introduces additional events into the simulation. These additions do not change the behavior of the modeled system, but act to coordinate calculations involving two or more `zones`. Figure 5.9 quantifies the effects of partitioning on event counts. Total events processed is the sum of net/committed events and rolled back events. The number of committed events is a measure of the size of the specified simulation, whereas the number of rolled back events is a measure of the overhead incurred by use of optimistic simulation. The lower left panel shows that even without rollbacks, for partitioning of geography, the committed events increases with number of processors.

That investment can only be recovered if either the amount of computing per synchronized interference calculation (across geographically partitioned `zone`s) is much larger than event processing overhead, or if the geographical fan-out of messages is reduced. The interference calculations in the current experiment are lightweight with respect to the event processing overhead in the current TeD/C++.

One way of reducing overhead event count is by limiting the effective radio range so that the parent `BS` will pool only the `zone`s, which are in the radio range of the `BS`, to fulfill each interference calculation request. Preliminary experiments in reducing event counts, and in artificially increasing the work of interference calculation show that both mechanisms serve to improve the speedup seen in geographical parallelization. There is also a scope to reduce the parallelization costs by simple programming techniques that reduce event usage, such as event aggregation or subscriber notifications.

In summary, this and related work clearly indicates that once the mechanisms for parallelizing the simulations of wireless networks are implemented in the design of primary simulation objects (which is not easy), modelers can routinely reuse them to create models exhibiting significant speedups: even when a four processor workstation is used, speeding the simulations by a factor of 3, which means a lot in engineering design! Orthogonal channels and limited radio range are the primary opportunities for exploitable concurrency in wireless networks.

Chapter 6

Summary

TED/C++, a software for parallel simulation framework based on C++, was reviewed. A methodology of modeling a wireless network simulation for session/call traffic (WIPPET$_{\text{session}}$) and packet/data traffic (WIPPET$_{\text{packet}}$) in TED/C++ framework was discussed. The differentiating elements of these simulators were shown to be following:

1. The designing and modeling of various WIPPET entities - `callgen`, `MS`, `zone`, `BS`, and `switch`.
2. The design of various radio resource management algorithms and protocols distributed among different WIPPET entities - channel allocation (LI-DCA, ACP-DCA), handoff standard based (AMPS, IS-136, GSM) and non-standard (MADRAS), etc.
3. The modeling radio propagation with distance loss, shadow/long-scale fading and Rayleigh/short-scale fading.

This thesis also proposed two ways to parallelize the simulation and these two methods are evaluated based on the parallel simulation results in terms of simulation efficiency and execution time speed.

The integrity of both the methodology and the WIPPET$_{\text{session}}$ simulator is verified by comparing WIPPET simulation results (blocking and dropping versus offered traffic) with previously obtained simulation results of the uniprocessor MADRAS (Mobility And Dynamic Resource Allocation Simulation). These results were obtained as an example of an application of the testbed to simulate the effect of various new radio resource management algorithm on wireless system.

The performance of WiPPET$_{\mathrm{packet}}$ simulator is also evaluated by comparing the simulation results (PER versus fade margin for CDPD uplink) with published analytical results.

Appendix A

TeD example: ATM Multiplexer

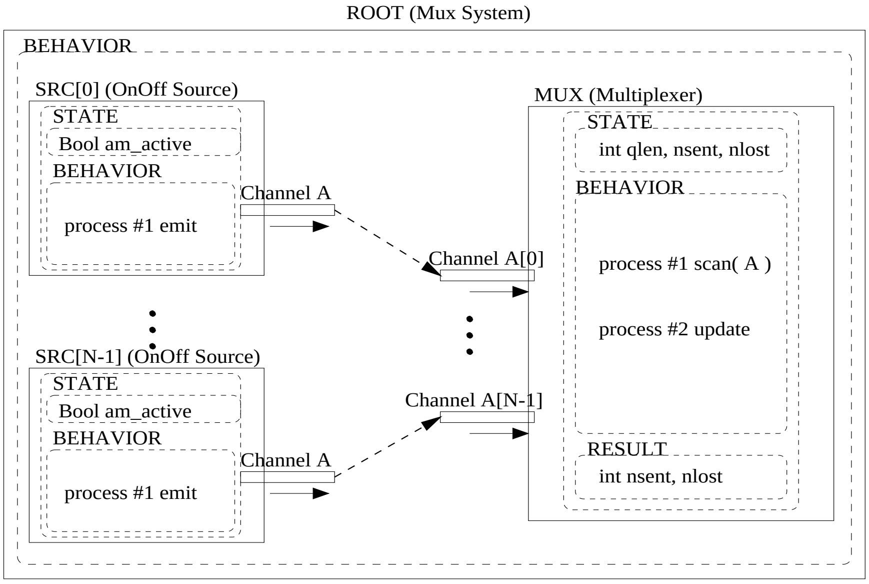


Figure A.1: Multiplexing system example

Consider the description of a simple ATM multiplexer operating on a set of input sources as shown in figure A.1. The sources emit ATM cells, and the multiplexer multiplexes the cells coming from the sources onto a single output link. The multiplexer has an internal buffer in order to hold cells under heavy input traffic conditions. The measures of interest are the number of cells transferred to the output link, and the number of cells lost (dropped) in the multiplexer due to buffer overflows.

Cell Event and Cell Channel

First, a channel type is defined that specifies all the types of data that can flow across any channel of that type. One type of data unit – `ATMCell` – is declared, and a channel of type `ATMChannel` is declared that allows `ATMCell`s to flow through it. These declarations are placed in a file named `call.the`.

```
event ATMCell
{\{
    // No data variables used (No payload).
    CONSTRUCTOR( void ) {}
\}}
```

```
channel ATMChannel { ATMCell }
```

Multiplexer

The multiplexer entity is defined, in a file named `mux.the`, as one with `N` `in`put channels, each of type `ATMChannel` (in this example, we will not model the generation of output cell stream from the multiplexer; however, it can be added in a straightforward way to the code in this example):

```
entity Mux( int N )
{
    channels
    {
        in ATMCellChannel A[ $PARAM(N)$ ];
    }
}
```

Note that `N` is a parameter to the entity, implying that the value of `N` has to be specified at the time of instantiating the entity.

One (of several possible) descriptions of the behavior for the multiplexer is declared in `mux.the`. This architecture keeps track of the number of transferred cells and the number of lost cells. It uses two *process*es to model the multiplexer's behavior.

```
architecture Mux of Mux( int N )
{
    dconst  { $int B, C; double T;$ }
    state   { $int qlen; int nsent; int nlost;$ }
    lstate  { $INT *qfreq;$ }
    behavior
    {
        process #1 scan( A );
        process #2 update;
    }
    result { $int nsent; int nlost;$ }
}
```

Note that `B`, denoting the buffer size, and `C` denoting the factor by which the output link of the multiplexer is faster as compared to each of the input links, and `T` denoting the time for one cell emission on any input link, are declared as deferred constants of this architecture. Also, note the use of **lstate** containing a variable `qfreq` that will be used to keep track of cell-delay distributions.

The deferred constants are initialized in `mux.ted` as follows:

```
dconst Mux : Mux : init
{\{
    //Specify the number of processor on which this entity instance
    //will be mapped.
    int ppe;
    GET_RSRC_INT("PPE", &ppe);

    // TeD macro used to mapped entity to processor.
    // ppe = # processor.
    SET_PREFERRED_PE(ppe);

    B = 5; // default value
    GET_RSRC_INT( "BufferSize", &B );
    cout << INAME() << ":BufferSize=" << B << endl;

    C = 5; // default value
    GET_RSRC_INT( "LinkCapacity", &C );
    cout << INAME() << ":LinkCapacity=" << C << endl;

    T = 1.0; // default value
    GET_RSRC_DOUBLE( "EmissionTime", &T );
    cout << INAME() << ":Emissiontime=" << T << endl;
\}}
```

As one notices that the `Mux` entity instance is mapped on processor *ppe* in the deferred constants initialization. As we see later that the configuration file is used to set the processor number (*ppe*) on which the entity is mapped.

The state variables are initialized in `mux.ted` as follows:

```
state Mux : Mux : init
{\{
    qlen = 0; nsent = 0; nlost = 0;
\}}
```

The lstate items are allocated and initialized in `mux.ted` as follows:

```
lstate Mux : Mux : init
{\{
    qfreq = new INT[ DCONST(B)+1 ];
    for( int i = 0; i < DCONST(B)+1; i++ ) qfreq[i] = 0;
\}}
```

Note the use of dynamic memory allocation to set the **lstate** variables.

The two processes of the multiplexer are defined in a file named `mux.ted` as follows.

```
process #1 Mux : Mux : scan( A[ $0$ to $PARAM(N)-1$ ] )
{\{
    for( int i = 0, n = ASETSZ(A); i < n; i++ )
    {
        assert( ACTIVE( ASETCH(A,i), ATMCell ) );

        int curr_len = STATE(qlen);

        if( curr_len >= DCONST(B) )
        {
            STATE(nlost)++;
            LSTATE(qfreq[DCONST(B)])++;
        }
        else
        {
            LSTATE(qfreq[curr_len])++;
            STATE(qlen)++;
        }
    }
\}}
```

The first process `scan` is executed whenever at least one of the input channels has a data arrival on it.

```
process #2 Mux : Mux : update
{
    \{ if( STATE(qlen) > 0 )
       {
           STATE(qlen)--;
           STATE(nsent)++;
       }
    \}
    wait for $DCONST(T)/DCONST(C)$;
}
```

The second process `update` is scheduled to be executed every one cell emission time period. This process calculates and updates the state information defining the queue length and the number of cells lost so far

The result variables are initialized and used in `mux.ted` as follows:

```
result Mux : Mux : init
{\{
    nsent = 0; nlost = 0;
\}}

result Mux : Mux : wrapup
{\{
    nsent = STATE(nsent); nlost = STATE(nlost);
    cout << "MUX: nsent=" << nsent << ",nlost=" << nlost << endl;
    cout << "MUX: Delay distribution = {" << endl;
    for( int i = 0; i < DCONST(B)+1; i++ )
    {
        double prob = LSTATE(qfreq[i]) / double( nsent+nlost );
        cout << "        " << prob << endl;
    }
    cout << "    }" << endl;
\}}
```

Cell Source

A data source entity is declared that emits cells as its output stream. Instances of this type of entity will be used as inputs to the multiplexer. The `BurstySource` entity is defined, in a file named `source.the`, to have zero input channels, and exactly one output channel of type `ATMChannel`:

```
entity BurstySource
{
    channels { out ATMChannel A; }
}
```

One interesting type of data source is an ON/OFF data source, which continuously emits cells if it is in an ON state, and does not emit any cells if it is in an OFF state. The source switches between the ON and OFF states with some given probabilities, which are specified as parameters to this type of behavior.

```
architecture OnOffSource of BurstySource
{
    dconst{ $double T; long seed; double p_on, p_off;$ }
    state{\{ CBool am_active;
             CUniformRNG active;
             CUniformRNG silent; \}}
    behavior
    {
        process #1 emit;
    }
}
```

The preceding behavior declaration, in `source.the`, uses one process named `emit` that periodically updates the source's state every one cell emission time. For every update, the source calculates its next state, using the ON/OFF probabilities. If it turns out to be in an ON state, then it emits one cell on its output channel.

The deferred constants are initialized in `source.ted` as follows:

```
dconst BurstySource : OnOffSource : init
{\{
    //Specify the number of processor on which this entity instance
    //will be mapped.
    int ppe;
    GET_RSRC_INT("PPE", &ppe);

    // TeD macro used to mapped entity to processor.
    // ppe = # processor.
    SET_PREFERRED_PE(ppe);

    // Default values
    T = 1.0;
    seed = PARAM(I)*10 + 10;
    p_on = 0.5;
    p_off = 0.9;

    GET_RSRC_DOUBLE( "EmissionTime", &T );
    cout << INAME() << ":EmissionTime=" << T << endl;

    int int_seed;
    if( GET_RSRC_INT( "Seed", &int_seed ) ) seed = int_seed;
    cout << INAME() << ":Seed=" << int_seed << endl;

    GET_RSRC_DOUBLE( "P_on", &p_on );
    cout << INAME() << ":P_on=" << p_on << endl;

    GET_RSRC_DOUBLE( "P_off", &p_off );
    cout << INAME() << ":P_off=" << p_off << endl;
\}}
```

The state is initialized in `source.ted` as follows:

```
state BurstySource : OnOffSource : init
{\{
    am_active = FALSE;
    CRNGSeed s( DCONST(seed), 2*DCONST(seed) );
    active.init( s ); silent.init( s );
\}}
```

The `emit` process of the `OnOffSource` is defined in a file named `source.ted` as follows:

```
process #1 BurstySource : OnOffSource : emit
{
    \{
     if( STATE(am_active) )
         STATE(am_active) = STATE(active).next() > DCONST(p_on);
     else
         STATE(am_active) = STATE(silent).next() > DCONST(p_off);

     if( STATE(am_active) )
         CHANNEL(A) << DELAY(DCONST(T)) << EVENT(ATMCell, ());
    \}
    wait for $DCONST(T)$;
}
```

Multiplexing System

We proceed to using the preceding two types of entities, and create one aggregate entity that consists of the sources as well as the multiplexer:

```
entity MuxSystem
{
    channels {}
}

architecture MuxSystem of MuxSystem
{
    dconst { $int N;$ }
    behavior
    {
        component elems;
    }
}
```

The preceding multiplexing system entity and its architecture declarations are placed in a file named `system.the`. The deferred constants are initialized in `system.ted` as follows:

```
dconst MuxSystem : MuxSystem : init
{\{
    //Specify the number of processor on which this entity instance
    //will be mapped.
    int ppe;
    GET_RSRC_INT("PPE", &ppe);

    // TeD macro used to mapped entity to processor.
    // ppe = # processor.
    SET_PREFERRED_PE(ppe);

    //Initialize number of bursty sources in the Mux system
    GET_RSRC_INT("NumInputs",&N);
    cout << INAME() << ":NumInputs=" << N << endl;

\}}
```

The architecture of the multiplexing system is defined in terms of its components. The components of the multiplexing system are defined in a file named `system.ted` as follows.

```
component MuxSystem : MuxSystem : elems
{
    entities
    {
        mux => Mux %1( $DCONST(N)$ ) : Mux;
        src => BurstySource : OnOffSource [ $DCONST(N)$ ];
    }

    map
    {\{
        for( int i = 0; i < DCONST(N); i++ )
            MAP( mux, A[i], src[i], A );
    \}}

    wrapup
    {\{
        cout << "MuxSystem: nsent = " << SUBRESULT(mux,nsent);
        cout << ", nlost = " << SUBRESULT(mux,nlost) << endl;
    \}}
}
```

Note that, the behavior of the multiplexing system is described in terms of its constituent parts – `N` sources, and one multiplexer.

Finally, the startup configuration is defined in a file named `model.cfg` as follows.

```
int MAX_N = 15;
int N = 10;
double gain = 3.0;
double T = 1.0;

for root use MuxSystem() : MuxSystem()
{
    resource int NumInputs = N;
    resource int PPE = 1;//MuxSystem is mapped on processor# = 1
    for mux
    {
        int K = int( double( N ) / gain + 0.5 );

        resource int BufferSize = 2*K;
        resource int LinkCapacity = K;
        resource double EmissionTime = T;

        //Mux is mapped on processor# = 1
        resource int PPE = 1;
    }

    name allsources = src[0,N-1]; // For readability
    for @allsources
    {
        resource double EmissionTime = T;
        resource double P_on = 0.6;
        resource double P_off = 0.8;
        meta resource int Seed[MAX_N] =
            { 12, 29, 340, 978, 7634, 13, 09, 872,
              82, 987, 23, 897, 623, 734, 8973 };

        //All the bursty sources are mapped on processor# = 0
        meta resource int PPE[MAX_N] =
            { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
    }
}
```

This specifies the root entity of the application is the multiplexing system, with some specified values as parameters for the system. Note the use of the `meta` tag to specify different seed values and processor numbers for each of the sources. As we notice that how it is easy to do entity-to-processor mapping: specify the number of processor on which you want to map the entity or entity instance. TED has a general purpose `Makefile`, which can be easily customized for the particular simulation model. By typing `make`, a complete TED compilation of the application is initiated. Upon successful compilation, the mode configuration is executed by typing the following in the model directory: `OBJ/ARCH/runmodel` p T `-A model.rdb`, where ARCH is the architecture code of the computer, p is the number of processors to use for simulation, and T is the total amount of simulated time to simulate. More complete, detailed documentation and examples can be found in the language manuals [16, 17].

Appendix B

Zone States and Constants

B.1 Constants

As we discussed in section 3.2, the `zone` contains a path loss matrix $g_{ds}(\mathrm{MS}_{\mathrm{location}}, \mathrm{BS})$ in which stores the gain from any possible `MS` location to all `BS`s, the lists of four strongest pilot `BS`s from all possible `MS` location, which are used in base station assignment algorithm to find a strongest `BS` for a given `MS` position and, the lists of neighboring point IDs of all possible `MS` location, which are used in mobility protocol to find a next position of a `MS`. These deferred constants are stored in one data structure called `zonePT` as shown below.

```
struct zonePT
{
  int    ptID;   // point ID.
  double x,y;    // (x,y) location of the point.
  double *gdS;   // path loss or gain to all BS from the point.
  int    *iBS;   // Index of four strongest pilot BS from the point.
  int    *nghpt; // Neighboring point IDs of the point.
  int    *nghchznID;// zoneID of the neighboring points.
}
```

The `zone` has an array of structure `zonePT` and the length of the array is equal to number of point in the `zone` geography. Hence, each point in the geography is represented by a structure `zonePT` called *point structure*. Each point structure has a unique point ID and x and y locations. The point structure contains an array of gain to all `BS`s from the point (possible `MS` location), an array of four strongest pilot `BS` ID, and an array of neighboring point IDs of the point (length of the array is 4).

As we discussed in subsection 3.7.5 that in multiple `zone` simulation, a `MS` can travel from one `zone` to the other `zone` during a simulation. It is called a `MS` *zone transfer*. To detect the zone transfer of the `MS`, the point structure also contains an array of`zone` ID of the neighboring points. Hence, for a given `MS` location (point), the mobility architecture finds the `zone` ID of the next point of the `MS` position. The mobility architecture detects a zone transfer, if the `zone` ID is not the same as current `zone` ID.

B.2 States

The `zone` contains a `MS` state structure (`ZN_MSS`) for each active mobile in which it stores information related to the `MS`: the status, i.e active or idle, the parent `BS` ID, the channel ID, the `MS` uplink transmitter power (msTxPWR), the parent `BS` downlink transmitter power (bsTxPWR), and the `MS` call duration.

```
struct ZN_MSS
{
  int status; // MS is active or idle
  int bsID;   // parent BS ID of the MS
  int chnl;   // channel ID of the MS
  double msTxPWR; // MS transmitter power
  double bsTxPWR; // parent BS transmitter power
  double calldur; // MS call duration

  int TXpacket; // sequence number of the current or last
                // packet transmitted by the MS.
  int TXstatus; // MS transmitter is on or off.
  double TXON;  // starting time of the current or last packet
  double TXOFF; // ending time of the current or last packet
}
```

In packet level simulation, `ZN_MSS` also contains the transmitter status of the `MS` (TXstatus), i.e. currently, the `MS` transmitter is on and transmitting packets, the starting and ending times of the current packet, which is transmitted, (TXON) and (TXOFF) respectively, and the sequence number of the last packet transmitted (TXpakcet).

For each MS, the zone contains a mobility state structure (MOB_STATE) for each active mobile. The MOB_STATE structure contains the current MS position or point ID (ptID), the parent zone ID (zoneID), the direction of the MS in which it is moving (Dir), and the MS speed as shown below:

```
struct MOB_STATE
{
  int    ptID;  // current position or point ID of MS
  int    zoneID; // parent zone ID
  int    Dir;    // Direction of the MS motion
  double speed;  // MS speed
}
```

The zone contains a uplink and a downlink channel state structures UPChnl and DWChnl for each uplink and downlink. The UPChnl structure contains a list of MS IDs, which are active on that uplink, and their uplink transmitter powers. The list of MS IDs and their related uplink powers are stored in array of msID and msTxPWR respectively. The length of array is equal to the total number of the MSs in a simulation.

```
struct UPChnl
{
  int           *msID;   // array of MS ID active on the uplink
  double        *msTxPWR;// MS transmitter power on the uplink
}
```

Similarly, the DWChnl structure contains a list of BS IDs, which are active on that downlink, and their downlink transmitter powers. The list of BS IDs and their related downlink powers are stored in array of bsID and bsTxPWR respectively. The length of array is equal to the total number of the BSs in a simulation.

```
struct DWChnl
{
  int    *bsID; // array of BS ID active on the uplink
  double *bsTxPWR; //  BS transmitter power on the uplink
}
```

References

[1] F. Babich, O. E. Kelly, and G. Lombardi. A variable-order discrete model for the mobile propagation channel. In M. Luise and S. Pupolin, editors, *Broadband Wireless Communications (Proc. 9th Tyrrhenian Internat. Workshop on Digital Comm., Lerici, Italy, 1997)*, pages 259–266. Springer–Verlag, London, 1998.

[2] K. G. Chen. Integrated dynamic radio resource management of wireless communication systems. Masters thesis, Rutgers, The State University of New Jersey, New Brunswick, NJ, May 1996.

[3] C.N. Chuah and R. Yates. Evaluation of a minimum power handoff algorithm. In *Sixth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications PIMRC'95*, pages 623–627, 1995.

[4] C.N. Chuah, R. Yates, and D. Goodman. Integrated dynamic radio resource managemant. In *Proc. IEEE Vehicular Technology Conference, VTC-95*, pages 584–588, 1995.

[5] R. H. Clarke. A statistical theory of mobile-radio reception. *Bell System Technical Journal*, 47:957–1000, 1968.

[6] S. Das, R. M. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. GTW: A time warp system for shared memory multiprocessors. In *Winter Simulation Conference Proceedings*, pages 1332–1339, 1994.

[7] R. M. Fujimoto. Parallel discrete event simulation. *Communication of the ACM*, 33(10):30–53, October 1990.

[8] D. J. Goodman. *Wireless Personal Communication Systems.* Addison-Wesley, first edition, 1997. pages 350–355.

[9] M. Gudmundson. Correlation model for shadow fading in mobile radio systems. *Electron. Lett.*, 27(23):2145–2146, 1991.

[10] W. C. Jakes, editor. *Microwave Mobile Communications.* Wiley, 1974.

[11] P. F. Reynolds Jr. A spectrum of options for parallel simulation. In *Proceedings of the 1988 Winter Simulation Conference, WSC'88*, pages 325–332, December 1988.

[12] J. Lai and N. B. Mandayam. Packet error rate for burst-error-correcting code in Rayleigh fading channels. In *Proc. IEEE Vehicular Technology Conference, VTC-98*, volume 2, pages 1568–1572, May 1998.

[13] D. M. Nicol and R. M. Fujimoto. Parallel simulation today. *Annals of Operations Research*, 53(3):249–286, 1994.

[14] J. Panchal, O. Kelly, J. Lai, N. Mandayam, A. T. Ogielski, and R. Yates. Parallel simulations of wireless networks with TED: Radio propagation, mobility and protocols. *Performance Evaluation Review*, 25(4):30–39, Mar 1998.

[15] J. Panchal, O. Kelly, J. Lai, N. Mandayam, A. T. Ogielski, and R. Yates. Wippet, a virtual testbed for parallel simulations of wireless networks. In *Parallel And Distributed Simulations - PADS*, 1998.

[16] K. Perumalla and R. Fujimoto. A C++ instance of TeD. TR GIT-CC-96-33, College of Computing, Georgia Institute of Technology, 1996.

[17] K. Perumalla, A. Ogielski, and R. Fujimoto. MetaTeD: A meta language for modeling telecommunication networks. TR GIT-CC-96-32, College of Computing, Georgia Institute of Technology, 1996.

[18] T. S. Rappaport. *Wireless Communications.* Prentice Hall, first edition, 1996. pages 69–192.

[19] T. S. Rappaport, Scott Y. Seidel, and Koichiro Takamizawa. Statistical channel impulse response models for factory and open plan building radio communication system design. *IEEE Transactions on Communications*, 39(5):794–807, May 1991.

[20] V. Sali. Simulation of Rayleigh fading clarke's model using spectrum shaping. WINLAB-WiPPET internal report.

[21] V. Sali. Simulation of Rayleigh fading jake's implementation of clarke's model. WINLAB-WiPPET internal report.

[22] B. Sklar. Rayleigh fading channels in mobile digital communication systems part 1. *Communications Magazine*, 35(9):136–146, 1997.

[23] R. Yates, S. Gupta, C. Rose, and S. Sohn. Soft dropping power control. In *Proceedings of The IEEE Vehicular Technology Conference VTC 97*, pages 1694–1698, May 1997.

[24] R. Yates and C.Y. Huang. Integrated power control and base station assignment. *IEEE Transactions on Vehicular Technology*, 44(3):638–644, August 1995.

[25] R. D. Yates. A framework for uplink power control in cellular radio systems. *IEEE J. Sel. Areas Commun.*, 13(7):1341–1346, Sep 1995.