EURASIP Journal on Wireless Communications and Networking

RESEARCH **Open Access**

Study on using design patterns to implement a simulation system for WiMAX network

Hsin-Hung Hsieh[1], Bih-Hwang Lee[1*], Huai-Kuei Wu[2] and Hung-Chi Chien[1]

Abstract

A network simulator always is an important tool to observe and evaluate the study concept for wireless networks. Considering the restriction of the limited budget, the non-commercial open-source simulation software often becomes the top choice of academia. Since the natures of academic study are innovation and excellence, a simulator frequently encounters that the existing modular functions are inadequate and need to be appended or modified. If a selected simulator inherently has poor architecture, the maintenance, recondition, and expansion of functions will become more difficult and more time-consuming in the future, while it is difficult to understand and reuse by the successors. Therefore, how to select a most suitable simulator is an important issue.

In order to make a simulator have flexible architecture and believable results, design patterns are proposed as the norms to design system architecture. To realize this idea, we surveyed six most used simulators, i.e., J-Sim, NS-2, NS-3, OMNet++, OPNET, and QualNet, to ponder their system architectures and design concepts from the source codes and the related literatures of the modular function expansion. We propose a network simulator architecture, named as CCGns, which is a discrete-event virtual network simulator and follows the IEEE 802.16-2009 standard. CCGns obeys the object-oriented design principles and is coded by the Java language. CCGns comprises eight packages for physical layer, medium access control layer, network layer, devices, topologies, events, scheduler, and reports, respectively. The main contribution includes three aspects which propose a scalable MAC messages management and the corresponding architecture, an applicable for multi-hop relay network topology architecture, and a two-stage minimum variance bandwidth allocation algorithm.

To the best of our knowledge, this article is not the first one to apply the design patterns for the simulator architectures of wireless network, but we use more design patterns and types than the others and also provide the unified modeling language figures to explain the system architectures. We particularly focus on how the management procedure of control messages influences the time-related performance evaluation metrics, e.g., how the amount and processing time of different control messages affect throughput, packet delay, and packet drop ratio. By using mathematic calculation to verify the simulation results, the proposed system architecture has been proven to possess excellent fidelity.

Keyword: Simulator, Design pattern, WiMAX, Object-oriented design, Unified modeling language

* Correspondence: bhlee@mail.ntust.edu.tw
[1]Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan
Full list of author information is available at the end of the article

1 Introduction

To study wireless network, a network simulator has been an important tool. By using software to construct a simulator, there have the advantages of having more convenience to build simulation scenarios and establish monitoring procedure. In order to quickly modify the scenario and revise the study concepts, simulation software has become the main method to fulfill the researchers' study concepts. There are two types of network simulation software: commercial and non-commercial. The commercial simulation software generally has the advantages of having the more complete modular architecture and systematization and providing better consultation service, but the cost is expansive for authorization. Conversely, the non-commercial simulator is cheaper or even free, but it has non-optimized codes and the poor compatibility among modules potentially. By considering the limited budget, the non-commercial simulation software has often become the top choice of academia, with which provides source codes.

Because of the natures of innovation and excellence in academic study, simulation software frequently encounters that the existing modular functions are inadequate and need to be appended or modified. How to find the most suitable simulation software always is a difficult and important issue, which may be determined by some evaluation metrics, such as fidelity, suitability, extensibility, scalability, user support, learning time, implementation-friendly, performance, and cost. The fidelity refers to the similarity between the real world and simulation results. The suitability refers to the communication protocol types that can be performed by simulator. The extensibility refers to the flexibility when the architecture is needed to append or modify the function modules. The scalability refers to the network size that can be simulated. The user support refers to the experience of the user-interface operation. The learning time refers to the time devoted to learn the simulation. The implementation-friendly refers to the convenience degree for implementation, configuration, maintenance, and the observation of simulation results. The performance includes the minimum requirements for CPU utilization, memory size, and storage capacity, as well as the real time needed to perform simulation. The cost means the money is spent on building a simulation.

In order to make the simulator's architecture to have the extensibility, we proposed to use the design patterns as the norm of system architecture design and implement. We surveyed the six most used simulators, i.e., J-Sim [1], network simulator-2 (NS-2) [2], network simulator-3 (NS-3) [3], OMNeT++ [4], optimized network engineering tool (OPNET) [5], and QualNet [6], to ponder their system architectures and design concepts from their source codes and the related literatures of the modular functions. We propose a network simulator architecture, named as CCGns, based on the IEEE 802.16-2009 standard. CCGns is a discrete-event virtual network simulator coded by Java language [7] and uses Eclipse as develop tool, which obeys the object-oriented design (OOD) principles to design the function modules. CCGns uses design paradigm to design system architecture and comprises of eight packages: physical (PHY) layer, medium access control (MAC) layer, network (NET) layer, devices (DEV), topology (TPY), events (EVT), scheduler (SCH), and report (RPT) packages. The main contribution of CCGns is to propose three aspects: a scalable MAC message management and processing architecture (SM^{3}PA), an applicable for multi-hop relay network architecture (AMRNA), and a two-stage minimum variance bandwidth allocation (TSMVBA) algorithm. SM^{3}PA includes three design concepts: firstly, it contains event class, event processing procedure, and event owners having individual architecture with the independent growing scale; secondly, it uses command flow to accomplish a series of events; and thirdly, the event owner certainly has the ability to handle event. In SM^{3}PA, users can arbitrarily define the management message types to focus on the interested messages and save the needed real simulation time. In AMRNA architecture, we can use a unified interface to simulate four different network topology architectures. TSMVBA is responsible to provide the optimal frame structure for the uplink (UL) and downlink (DL) subframe using the different subcarrier permutation mode.

To the best of our knowledge, this article is not the first one to apply the design patterns for the simulator architectures of wireless network, but we use more design patterns and types than the others and also provide the unified modeling language (UML) figures to explain the system architectures. We particularly focus on how the management procedure of control messages influences the time-related performance evaluation metrics, e.g., how the amount and processing time of different control messages influence throughput, packet delay, and packet drop ratio. By using mathematic calculation to verify the simulation results, the proposed system architecture has been proven to possess excellent fidelity.

This paper is organized as follows. Section 2 describes the related researches including the evaluation and comparison of some simulators and the introduction to WiMAX expansion module literatures and design patterns. Section 3 describes the CCGns system architecture and includes the design paradigms and function explanation used in the eight packages. Section 4 describes the simulation scenarios and demonstrates the simulation results, meanwhile using mathematic calculation to verify with each other. Section 5 is conclusions.

2 Related works

2.1 Comparisons of various network simulators

Several literatures have been proposed to evaluate the simulation software, which can be summarized into three categories: the first category is for overview and introduction [8, 9]. The second category is the evaluation and comparison for the inner system architecture, such as fidelity, suitability, scalability, and extensibility [10–14]. The third is the evaluation and comparison for the outer user experiences, such as learning time, user support, interface user-friendly, operate convenience, and performance [15–17]. In order to obtain the most suitable simulator, we evaluate the existing 22 simulators as mentioned in the above papers according to their license ways, programming languages, operating systems, and lifetime as shown in Table 1. After evaluation, we finally selects the six most often used simulation software as the study items, i.e., J-Sim, NS-2, NS-3, OMNeT++, OPNET, and QualNet. We give brief introductions and comparisons for the abovementioned six simulation software as follows.

A. Brief introduction to J-Sim
   JavaSim is the former name of J-Sim, which is a simulation combination environment based on the independent autonomous component programming model (ACPM), called autonomous component architecture (ACA). J-Sim uses Java to code the class program, and then, it uses Jacl interpreter to integrate the other script languages, such as Perl, Tcl, or Python. The advantages of J-Sim include lower coupling, the real-time process-driven simulation, the implementation of complete Internet protocol, automatic configuration, and online monitoring. The disadvantages include the lack of complete user manual, the need to learn the Jacl, no graphical user interface (GUI), and the wireless network kits only providing 802.11 and sensor network.
B. Brief introduction to NS-2
   NS-2 is the most widely used network simulator currently, which is an object-oriented based discrete-event simulator. The simulation steps include four stages: (1) using C++ to write function modules; (2) using OTcl to describe the simulate

Table 1 The comparisons of various network simulators

No.	Simulator	License	Language	Platform	Publications
1.	AKAROA [32]	Free for academic use	C++	Solaris 2/SunOS 4/Linux	1990~
2.	DIANEmu [33]	Free	Java	Based on JDK and Eclipse	1992~2002
3.	SSFNet [34]	Free for academic use	C++ and Java	Linux/Solaris/WinNT	1997~2003
4.	GloMoSim [35]	Open source	Parsec	Win/Linux/Solaris	1998~
5.	GTNetS [36]	Open source	C++	Linux/OSX/Solaris/Win	2003~2008
6.	J-Sim	Open source	Java/Jacl	Linux/Unix/Win	2004~2008
7.	JANE [37]	Free	Java	Based on JVM	2003~2007
8.	JiST/SWANS [38]	Free for academic use	Java	Based on JVM	2002~2005
9.	NAB	Open source	Object-Caml	Win/Linux/Unix/OSX for OCaml	2004
10.	NCTUns 6.0 [39]	Free	C/C++	Fedora 12	1999~2010
11.	NetSim [40]	Commercial free for academic use	C/C++/Java	Windows	2002~
12.	NS-2	Open source	C++/OTcl	Cross-platform	1996~
13.	NS-3	Open source	C++/Python	Windows/Unix/Linux	2006~
14.	OMNeT++	Free for academic use	C++	Cross-platform	1999~
15.	OPNET	Commercial free for academic use	C(C++)	Windows/Linux	1986
16.	P2P Realm [41]	Open source	Java	Based on JVM	2006
17.	QualNet	Commercial	Parsec	Cross-platform	2007~
18.	REAL [42]	Free	C	FreeBSD (for i386)	1997~
19.	Shunra VE [43]	Commercial	Hardware	Windows	1998~
20.	ShoX [44]	Open source	Java	Based on JVM & Eclipse	2008~
21.	SimPy [45]	Open source (MIT)	Python	Cross-platform	2002~
22.	TOTEM [46]	Open source (GPL)	C/C++	Linux	2003~2007

scripts, such as parameter definition, object behavior, traffic model, and network topology; (3) using a trace file to save simulation results, then using parser to analyze the original data, such as AWK or Perl; and (4) using plotter software to display the parsed data, such as Xgraph or Gnuplot. The reason of using two languages to construct simulation is that the simulation scripts and parameters definition can quickly be changed under different scenarios and without the need to recomplier the kernel programs. The advantages are having a large amount of free open-source code and more complete documentations [18]. The disadvantages are having a steep learning curve, less user-friendly for operation interface, not intuitive for using OTcl, and the heavy loading caused by the one-to-one architecture between C++ and OTcl.

C. Brief introduction to NS-3
The concepts of NS-3 architecture come from GTNetS, which also is a discrete-event network simulator using C++ and Python two languages to construct simulation primarily for research and educational use. The NS-3 simulation core supports for network study in both IP and non-IP networks. However, the large amount of users focuses on the wireless network simulation, which involves models for Wi-Fi, WiMAX, and LTE. NS-3 is not a successor of NS-2 but a substituent. In terms of overall performance, NS-3 is the best one among all present simulators. The latest version is the NS-3.24 released in September 2015. The advantage is to entirely refer the true definitions to design packet format, good extensibility, producing the pcap trace file which can be read and analyzed by tcpdump or Wireshark. The disadvantage is that NS-3 has less available modules than NS-2 and needs to convert NS-2 modules for NS-3 use.

D. Brief introduction to OMNeT++
OMNeT++ has a discrete-event simulation environment mainly to simulate communication networks, which has four primary simulation frameworks and two types of module. The simulation frameworks are INET Framework, INETMANET, MiXiM modeling framework, and Castalia simulator, while the two module types are the simple module and the compound module. The simple module is the basic active module in the model, while the compound module groups other modules into a larger module and has no active behavior. At the beginning of the simulation, the network description (NED) language is first used to describe the model structure, then the simple modules are grouped into the compound module by connection or combination, and, finally, the largest compound module performs the simulation. The advantages include having detail documentations and enormous module library, the only simulator having online visual interface, and more convenience to use than NS-2 and NS-3. The disadvantages include the need to install extra program package, the longer simulation time, high memory demand, the efforts of learning just less than NS-2, and the performance a little bit less than NS-3.

E. Brief introduction to OPNET
OPNET is one of the reputable commercial simulation software, which has three main module suites including basic ITDG, advanced Modeler, and a professional Modeler/Radio. OPNET uses object-oriented to construct data model and build its simulation environment layer by layer, having the most quick discrete-event simulation engine, hundreds of communication protocols and vendor equipment modules, and a complete parallel processing simulation core. The advantages include having a comprehensive model library, a fine model details, a user-friendly interface, and a customized output for simulation results. The disadvantages include expensive licensing fees, difficult to learn and needed professional training, lower performance for large-scale network, and non-transparent for the parameter definition.

F. Brief introduction to QualNet
QualNet uses GloMoSim as core and is commercialized visualization simulation software for mobile network, which can use different simulation protocols on several parallel architectures and can be one part of real network to perform simulation. QualNet has five main components: simulator, animator, scenario designer, real-time analyze, and tracer tools. The advantages include suitable for a large-scale network simulation, fast execution, providing accurate communication protocols and module models, allowing to modify or append customized codes, and a user-friendly GUI. The disadvantage is expensive and unaffordable for individuals.

2.2 Related literatures on WiMAX expansion module

In the previous studies, many researchers provide their contribution on how to add or modify the WiMAX module, e.g., some NS-2 WiMAX modules are proposed by Networks and Distributed System Laboratory (NDSL) [19, 20], National Institute of Standards and Technology (NIST) [21, 22], LRC [23], and Light WiMAX (LWX) [24], respectively. In addition, Freitag et al. propose NS-2 WiMAX service flow module [25]; Guo et al. propose NS-2 WiMAX path propagation module [26]; Farooq et al. propose NS-3 WiMAX module [27]; Ismail et al. propose NS-3 WiMAX module [28]; and Furlong et al. propose NS-3 WiMAX OFDMA expansion module [29].

In [30], the authors propose to use a strategy pattern to implement the scheduling module.

In those papers, all authors have referred to their customize simulators and code by using object-oriented language for expansion modules, which follow object-oriented design rules; therefore, those system architectures posses the convenience for modification and the flexibility for expansion. However, those properties are difficult to quantify and have a consistent assessment metric. Moreover, some papers have mentioned that none of simulation software is able to reproduce all of MAC layer control messages and processing procedures, as well as we found that any two simulation software do not implement the same control message types and processing procedure. Hence, it would be different for the performance expression of time-related evaluation metrics in different simulation software, e.g., the influence of packet delay time. In order to investigate how the management message processing procedure affects the time-related evaluation metrics, a system architecture should allow users to arbitrarily define his management message types and processing procedure, and then, we can obtain a more objective system performance evaluation results. Finally, to our knowledge, it often is incompatible for the operation interface of scheduling modules due to incorporation for the different scheduling factors and algorithms, which leads the studies inconvenience to evaluate the performance of various scheduling algorithms. Therefore, it would be helpful for the study of the scheduling algorithm selection, if the system architectures of schedule algorithm and operation interface can be independently developed.

2.3 Introduction to design patterns

A programming is a series of thinking process to convert an abstract problem description into a realistic code entity. Compare with the program-oriented design manner that based on the functions capability and executions order, the object-oriented design manner has capability to receive, process, and deliver independent data unit as design foundation, i.e., object; hence, the system architecture has more flexible and easier to maintain. The design patterns are exactly in the basis of object-oriented design manner to get rid of bad one and keep good one from the frequent occurrence solutions. The purpose of design patterns is to organize these solutions in a simple and easy way, and they make our programming more flexible, more modularized, more reusable, and easier to understand. Design patterns do not teach us how to code, but they are the discussion schemes to solve the problems under different situations. The pattern in design patterns means a useful solution has been proven that can be used to solve the recurring problems under a special scenario. Based on the Gof literatures [31], there are 23 types of design patterns and are divided into three categories, which are creational, structural, and behavioral introduced as follows.

Creational design patterns mainly abstract the instantiation process, which help us to make a system independent of how its objects are created, composed, and represented. A class creational pattern uses inheritance to vary the class instantiated, whereas an object creational pattern will delegate instantiation to another object. Creational patterns include abstract factory, builder, factory method, prototype, and singleton pattern. Structural patterns are concerned with how classes and objects are composed to form larger structures. Structural class patterns use inheritance to compose interfaces or implementations. Structural patterns include adapter, bridge, composite, decorator, façade, flyweight, and proxy pattern. Behavioral patterns are concerned with algorithms and the assignment of responsibilities between objects. Behavioral patterns describe not only just patterns of objects or classes but also the patterns of communication between them. These patterns characterize complex control flow that is difficult to follow at run-time. They shift your focus away from flow of control to let you concentrate just on the way that objects are interconnected. Behavioral patterns include a chain of responsibility, command, interpreter, iterator, mediator, memento, observe, state, strategy, and template method.

3 CCG network simulator

It is difficult to quantify and has a consistent assessment metric for the system architecture flexibility and expansion convenience; hence, we propose to use design patterns as the criteria of system architecture design. In this section, we introduce how CCGns applies design patterns in the system architecture of network simulation software. CCGns follows IEEE 802.16-2009 standard and consists of eight packages, namely PHY, MAC, NET, DEV, TPY, EVT, SCH, and RPT packages, whose functions, design ideas, and the used patterns are described as follows.

3.1 PHY package

PHY package is a program collection which is responsible for the physical layer functions corresponding to the real world, the main capability is using the frame structure form to provide link capacity, and the modular function includes the various wireless technical frame structures, such as orthogonal frequency division multiple access (OFDMA), and subcarrier permutation mechanism. There are three main modules in this package, i.e., slot factory module, frame builder module, and resource allocation module. In slot factory module, factory pattern is used to produce the simple structure object, such as basic slot structure, slot, and frame, because these objects use only one substructure type. In frame builder module, builder pattern is used to produce more complex objects, such as subframe, because the area types and amount on the inside of the subframe are dependent on the network topology and the device node-type. In resource allocation module, the order of nodes to obtain link resource is in first-come first-served (FCFS) manner. The system architecture will be implemented by the use of singleton if the real object only has one object entity to exist, such as subcarrier permutation mode or basic slot structure. Meanwhile, in order to simplify the needed parameters of frame production, we enumerate the original parameters in advance, such as system bandwidth and fast Fourier transform (FFT) sizes, and then, we calculate the derived parameters by using the original parameters, such as symbol time, useful transmission time, and the amount of symbol time of frame. According to the selected parameters, such as the ratio of bandwidth allocation and permutation mode for DL and UL, user can calculate the duration and subchannel amount of subframe finally to create the instance of frame object. There exist three conditions to have the best solution of the subframe duration allocation, i.e., to completely use the frame duration, to exactly be the integer multiple of slot time for the subframe duration, and to be identical for both the subframe duration ratio and the bandwidth allocation ratio; therefore, we proposed a two-stage minimum variance bandwidth allocation algorithm (TSMVBA) to deal with the bandwidth allocation problem for DL and UL subframes. The UML diagrams for slot factory and frame factory have been shown in Appendix 1: Figs. 8 and 9, respectively.

3.2 MAC package

MAC package is a program collection which is responsible for the MAC functions corresponding to the real world, whose main capability is to process and execute the management messages. Some specific purposes can be accomplished by the management messages, such as CDMA connection contention mechanism, bandwidth requirement and allocation mechanism, and the packet transmission and reception procedures. We proposed SM^{3}PA to let users arbitrarily define their management messages for their interested manage procedure, and the definitions and process procedures of management messages can be independently developed. SM^{3}PA have three main concepts: firstly, the system architecture of event types, event processing procedure, and event owner can be independently developed; secondly, the concept of command flow (CF) is used, where CF refers to a chain of events getting together to accomplish a specific purpose; thirdly, the event owner certainly has the ability to handle the event.

In the virtual simulation world, time is advanced in a non-continuous way, and the advanced interval precision determines the required reality time. For saving reality time, we focus on the occurrence time of the specific event, called monitor time. We list all the event types that possibly occur at monitor time, which are enumerated in the event package. The processing procedures of all event types define a common execution interface, which consist of many sub-procedures, while the operating interface of these sub-procedures is defined in the event processing procedure package. The different types of devices may have the same type of events, but their processing procedure may be different; therefore, the execution details and steps of the sub-procedures are provided by the device class, i.e., the concept that "the event owner certainly has the ability to handle event." We establish a chain of event types to accomplish a specific purpose to facilitate the management of the relationship among events and to mitigate the management problem caused by increasing the number of sub-procedures because the system quickly grows up, where the codes are independently in various sub-packages, that is how we implement the command flow way.

When the simulation starts to execute, system core will ask for the execution time of the next nearest event of all device instances, and then, the system time will advance to the next nearest event time and to notify all device instance sequentially to process events. Since the needed processing time of each event is different, the next system time of each device instance will be different. When the event execution finishes, the original event owner will create new event instance and store in the event queue of new event owner. A current event may trigger several new events if multiple events occur simultaneously, where the new events will comply with the event priority order to execute. If all events at this monitor time have been processed, the above procedure will be repeated until no new event creates or reaches the predefined simulation stop time.

MAC package and event package are complementary. An event package is responsible to define event types, while an MAC package uses various event types to create command flow and sequentially to execute the events in command flow. In order to prove practicable, we define three command flows, i.e., connection contention mechanism, bandwidth request and grant mechanism, and data transmit and receive mechanism. The design of command flow is based on hybrid multi-hop relay network (HMRN), and HMRN is described in topology package.

In Fig. 1, the red and green blocks represent DL and UL subframes, respectively, while Arv, Req, Gnt, Trs, Rcv, Rly, Dpt, and Ack denote packet arrival, bandwidth

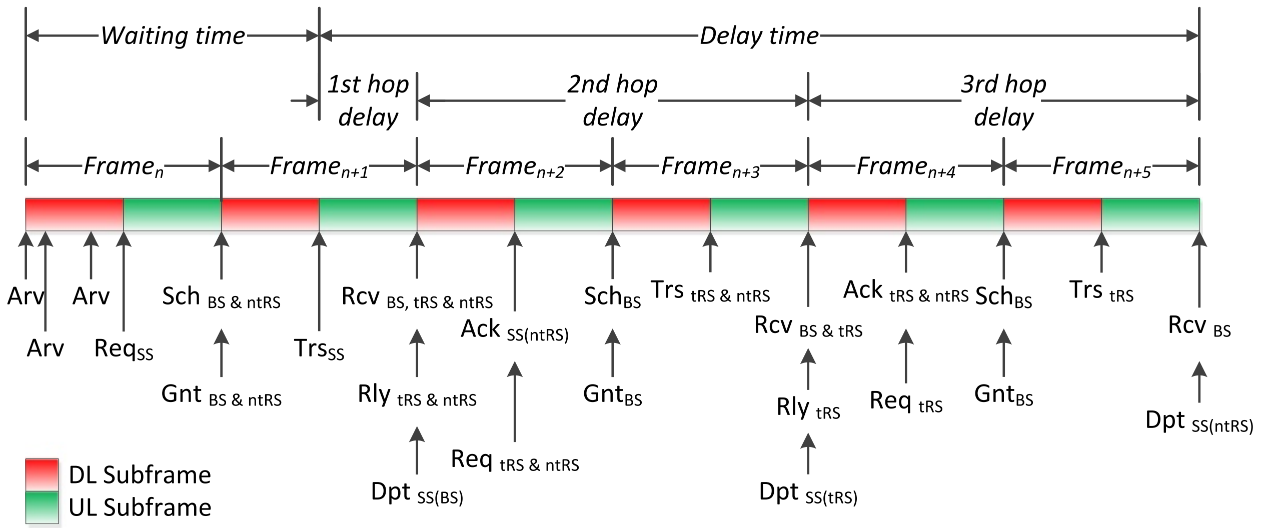


Fig. 1 MAC management message command flow—the view of system time

request, bandwidth grant, packet transmit, packet receive, packet relay, packet departure, and packet acknowledge, respectively, where the subscripts represent the device types of event owner. About the event relations in command flow, the relation of one-to-one is the simplest, e.g., a subscriber station (SS) bandwidth request event Req_{ss} triggers the scheduling event of base station (BS) and transparent relay station (tRS) Sch_{BS} & $_{ntRS}$ and further triggers the bandwidth grant event Gnt_{BS} & $_{ntRS}$. The relation of one-to-many is more complex, e.g., the packet receive events $Rcv_{BS,tRS}$ & $_{ntRS}$ occurred on the relay station (RS) triggers the packet relay event Rly_{tRS} & $_{ntRS}$, but the packet departure event Dpt_{BS} will be triggered if it occurs on BS, i.e., the identical type event occurred on the different device types may have different processing procedures. In Fig. 1, a packet needs about two frame time duration from entering system to departing system.

3.3 NET package

NET package is a program collection which is responsible for the network and upper layer functions corresponding to the real world, whose main capability is to implement the traffic generator (TG) and probability distribution model (PDM). PDM is based on linear congruential generator (LCG), which may have uniform, exponential, normal, lognormal, extreme, and geometric distributions for implementations. After a most suitable distribution model has been selected, the random number generator is used to generate random variant, e.g., traffic or service time, where the UML diagram for random number generator is shown in Appendix 1: Fig. 10.

3.4 Event package

Event package is a program collection which is responsible for the MAC layer management message corresponding to the real world, whose main capability is to define the event types and the sub-procedure operation interfaces inside the event process procedures, and the code entity of execution detail is provided by the device instance. The system architecture of event package includes interface class, abstract class, and object class program, and the use of patterns has factory, bridge, chain of responsibility, and command pattern, where the UML diagram for event package is shown in Appendix 1: Fig. 11.

This package uses the interface class as super class and defines five basic operate interfaces, namely event identity code (EID), event owner (EO), event type (ET), trigger time (TT), and execution method (EM). EID is used to facilitate tracing the footprints of a specific event instance and convenience to monitor and debug. EO is responsible to provide the code entity of the event processing sub-procedure. ET is used to define event priority and the relationship among events. TT is used to sequentially perform event on time. EM is used to make all event types have a common interface method to perform their processing procedure. Under the different types of network topology, the same type events may trigger the same new events but having different owners, e.g., in transparent relay network, the users send the bandwidth request information to BS. After BS finishes the scheduling procedure, it informs the users to send data to tRS. However, in non-transparent relay network, the ntRS is responsible for the user all the request, scheduling, and grant procedures. In order to let the same type event have the different processing procedures with the different owners, we proposed a system architecture to allow that event type, processing procedure, and event owner can be developed and grow up independently to increase flexibility.

In this package, we use factory pattern to create event instance. The use of bridge pattern is to separate execution sub-procedures defined by the device package role category from execution procedures defined by the event package processing procedure class, so they can evolve independently. The chain of responsibility pattern is to let all device categories have an opportunity to process event instances on behalf of management messages and lower the coupling relationship between the sender and the receiver. Finally, the design concepts of event class come from command pattern, i.e., event instance, to control system operation.

3.5 Device package

Device package is a program collection which is responsible for network device corresponding to the real world, whose main capability is to facilitate the definition and implementation of the network devices. This package divides simulation system network devices into two categories. The first category is responsible for describing the basic ability of real-world network devices, called basic device, e.g., UE, RS, and BS. The second category is responsible for describing the event process ability of virtual world network device, called role device, i.e., a temporary name of basic device during the process period of a specific event, e.g., packet arrival role and bandwidth request role. In basic device, we define and implement the basic function of real network device, such as PHY layer, MAC layer, packets manage, and events manage functions. In role device, we define the needs of sub-procedure operation interface of the event execution and use the one-to-one mapping manner to corresponding with event types. The identical type events in different type of basic devices possibly have different processing sub-procedure, e.g., packet receive event. After receiving packet, BS lets the packet depart; tRS forwards it to BS; non-transparent relay station (ntRS) adds it to the next bandwidth requirement; therefore, the details of sub-procedure execution, i.e., code entity, are provided by basic device. This design concept is inspired by the different individuals that the same work item should have the same execution procedure, but the execution details can be adapted according to the reality conditions. Moreover, to simplify the operation of event processing, we defined a common execution interface for all of event types. During the simulation, basic device will trigger the various type events along with simulation progress. When it is necessary to execute the event processing procedure, the basic device transforms into the corresponding role device according to the trigger event type and obtains the event processing ability, and the role device will transform back into basic device after event execution finished. The UML diagrams for basic device class and role device class have been shown in Appendix 1: Figs. 12 and 13, respectively.

3.6 Topology package

Topology package is a program collection which is responsible for network topology corresponding to the real world, whose main capability is conveniently to manage and use the network topology. A modular function includes two parts, namely topology establishment and user deployment. In order to establish the network topology in a simulation architecture, we proposed an applicable for multi-hop relay network topology architecture (AMRNTA). In AMRNTA, we use tree structure to establish the network topology, in which all nodes have two types of links, i.e., control link and data link. Control link and data link are responsible for delivering the management messages and data, respectively, and each link has both uplink and downlink directions. Before establishing network topology, each son node needs to define a parent node, and then, the son node first establishes a UL data link to his parent node, meanwhile the parent node adds this son node to his DL data link queue. Next, the son node sets his parent node as a control link target node (CLTN). If the CLTN has the wireless resource allocation ability, the son node establishes a UL control link to this CLTN, meanwhile the CLTN adds the son node to his DL control link queue; otherwise, the son node sets the CLTN parent node as a new CLTN if it does not have the wireless resource allocation ability and repeat the foregoing procedure until the new CTLN has the ability to allocate wireless resource. The establishment of network topology is in a point-to-multipoint (PMP) manner and has broadcast function, because a son node only has a parent node but the parent node may possibly have many son nodes. With this architecture, the procedure of establishing network topology can be simplified.

In order to avoid interference, each serving station needs to have independent resource, and its available resource is to satisfy the maximum user requirement and be proportionally allocated to the serviced users. We propose three slot allocation rules as follows. Firstly, based on the maximum number of hops in the network topology, a subframe is divided into the same number of zones. Secondly, the available slot amount of each zone is proportionally allocated to the serviced users. Thirdly, the number of serviced users should be the maximum amount if several zones reuse the same resource area.

About the user deployment modular functions, we design three user deploy mechanisms, i.e., sequence, random, and proportional modes. The sequence mode refers to the users which will be sequentially assigned to the serving station, which is suitable for system debug. The random mode refers to the user will be assigned to the serving station according to uniform distribution in random manner, which is most approximate to real situation. The proportional mode refers to the users which will be assigned to the serving station according to the ratio of their serviced users, which is suitable to evaluate the system performance under different loads.

3.7 Scheduler package

Scheduler package is a program collection which is responsible for scheduling mechanism corresponding to the real world, whose main capability is the implementation of scheduler algorithm, and the goal is to make the reuse of system scheduler codes to maximize. In order to let all scheduling algorithms share a common operating interface, to combine multiple parameters into a schedule metric to implement the more complex schedule concept, and to replace the schedule module more easily to evaluate the performance of various scheduling algorithms, we propose four design concepts; the first is the input of schedule module which is scheduling units having the ability to generate the schedule comparison unit; the second is schedule comparison unit having the abilities to compare with identical class and combine many parameters to form a comparison item; the third is the core of schedule module which is scheduling algorithm concentrating on the schedule mechanism; the fourth is the output of schedule module which is an ordered set of scheduling units. In the aspect of pattern use, we use a strategy pattern to encapsulate the schedule module to make them interchangeable, and the system architecture and the exchange of schedule algorithms are independent. The factory pattern simplifies the generate procedure of schedule unit. In this package, we implement uniform, random, and round robin scheduling algorithm to demonstrate the feasibility. The UML diagram for scheduler package is shown in Appendix 1: Fig. 14.

3.8 Report package

Report package is a program collection which is responsible for the output of system information and performance evaluation metrics, whose main capability is to make the system information output module independent of other packages. The output format of information can be arbitrarily changed under the premise of not affecting the other packages. The design idea of this package comes from the concept of the observation point in queueing theory, and a chain of responsibility patterns is used to design system architecture. The modular functions have two parts, namely system monitor and performance evaluation metrics. The former one includes system scenario and simulation parameters, as well as the important information at the time of system status changed. The latter one includes throughput, packet delay time, packet queue length, packet waiting time, and packet drop ratio. The UML diagram for report package is shown in Appendix 1: Fig. 15.

4 Simulation experiments

In order to demonstrate the abovementioned design patterns for simulation, we propose a scenario as shown in Fig. 2, which shows a BS surrounded by three tRSs and six ntRSs located at the corresponding positions as shown in the figure. In the hexagonal cell structure, the system service range is the inscribed circle of radius R, which is divided into the center region and the peripheral region. The center region is a circular area of radius $2/3R$; the users located in the center area are served by BS. The peripheral region is an annular area of width $1/3R$; every 120 degrees deploys one RS; the users located in the annular area are served by the RS. In hybrid relay network architecture, we deploy tRS and ntRS on the annular area and the vertex of hexagonal cell, respectively, where ntRS needs tRS's help to forward data to BS.

The relevant system parameters are listed in Table 2. Among them, the proportional user distribution type means the serviced user amount of a serving station is decided by the ration between its service area and overall service area. The uniform scheduling mechanism means the serving station assigns resources to a user according to this user's total amount of the assigned resource (TAAR) in the past, where smaller TAAR has higher priority, which is to let all users fairly use the bandwidth resource. The partial usage of subcarriers (PUSC) permutation mode is also used in both UL and DL directions.

4.1 System output data

Figure 3 shows the uplink subframe output data for various device types, in kilobit (kb), where the abscissa and Y-axis represent system time and output data, respectively. The uplink subframe output data of BS SS (bSS), tRS SS (tSS), ntRS SS (ntSS), RS, and BS have been shown by the curves in different colors of red, blue, green, pink, and orange, respectively. In Fig. 3, the uplink subframe output data of BS is 30.96 kb, which is composed of the BS user data 10.8 kb and the forwarded user data by RS 20.16 kb, while the latter part are from tSS and ntSS with the forwarded user data 12.96 and 7.2 kb, respectively. Because that the access zone (AZ) size for 1 hop users is 105 slots ($3T_{\text{slots}} \times 35\text{CHs}$), of which 75 slots are allocated to the BS to receive user data, the BS AZ output data is 10.8 kb ($75 \times 48 \times 4 \times 3/4 = 10.8$ kb). The size of transparent relay zone (tRZ) is 140 slots ($4T_{\text{slots}} \times 35\text{CHs}$). BS uses all slots to receive the data forwarded by tRS, so the output data is 20.16 kb. As to non-transparent relay zone (ntRZ), because it merely supplies forwarded data from ntRS to tRS, so the output data is 0 kb. It is worth mentioning that, in order to guarantee an RS can forward data to its destination as soon as

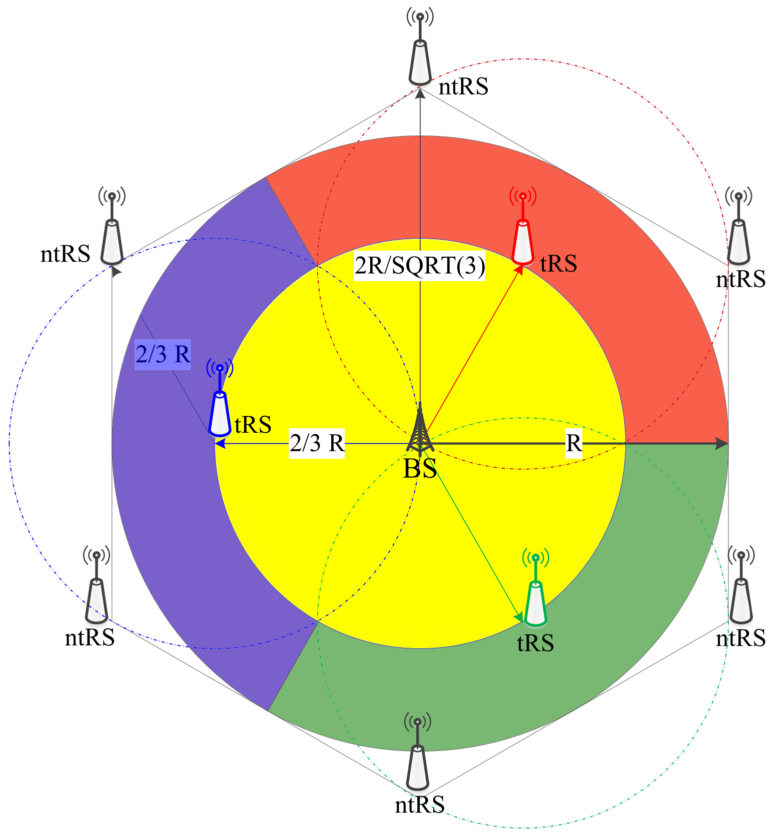


Fig. 2 Single cell simulation environment

Table 2 System parameters for simulation

Parameters	Values
Topology type	Hybrid
Node distribution type	Proportional
Schedule type	Uniform
PHY specification	SOFDMA 10 MHz
CP ratio	1/8
Permutation	DL PUSC and UL PUSC
Frame duration	5 ms
Modulation and coding scheme	16-QAM 3/4
Bandwidth resource ratio	DL:UL = 1:1
Packet size	5 slots
Packet mean interarrival time	1.0 ms
Monitor interval	5 ms
Length of packet queue	50 packages
Number of BS	1
Number of tRS	3
Number of ntRS	6
Total number of SS	20

possible, we adopted the relay first resource allocation strategy (RFRAS). RFRAS gives RS packets having higher priority when the service object includes RS and SS packets, and then, resources will be assigned according to their priority order. In the meantime, to prevent service objects with lower priority from being starved, each service object can only be assigned with one resource at a time.

4.2 Average delay time

Figure 4a shows the average packet delay times for various device types, where the abscissa and *Y*-axis represent system time and delay time, respectively. The delay times of bSS, tSS, ntSS, RS, and BS have been shown by the curves in different colors of red, blue, green, pink, and orange, respectively, i.e., the delay times of 1 hop, 2 hops, 3 hops, 1&2 hops, 1&2&3 hops with the values of 2.5, 7.5, 127, 49.1, and 32.8 ms, respectively. Meanwhile, Fig. 4b shows the detail delay times from tSS to BS (the blue line in Fig. 4a), including the delay times of the first hop (tSS$\rightarrow$tRS) and the second hop (tRS$\rightarrow$BS), while Fig. 4c shows the detail delay times from ntSS to BS. In Fig. 4a, the value of red line is 2.5 ms, because BS users can upload data to the BS directly. The packet transmitted during the uplink subframe will certainly be received before the

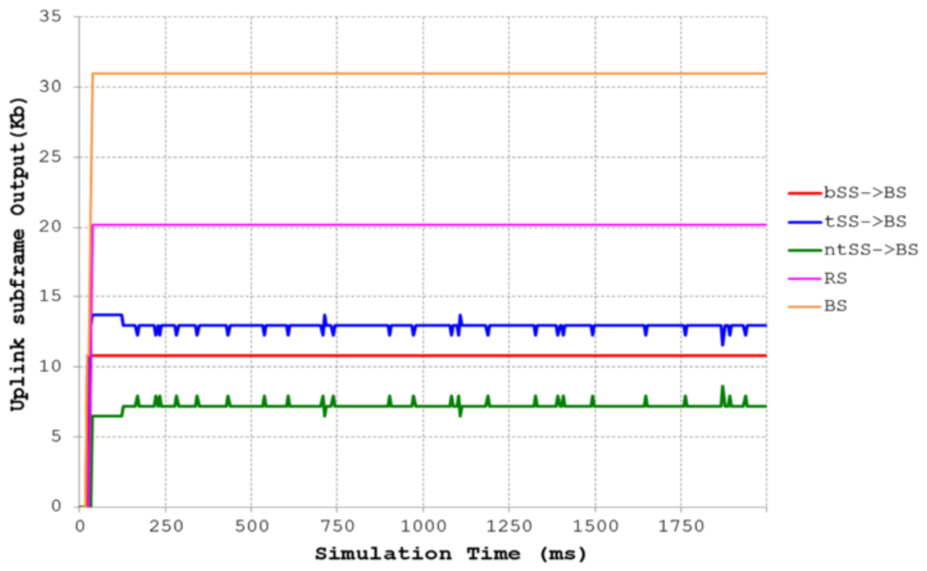


Fig. 3 The uplink subframe output data of various device types

uplink subframe is terminated; therefore, the maximum delay time is one uplink subframe duration. In addition, if the packet arrival rate is faster than the upload rate, the packets will be stored in queue or dropped. Since the packet queuing waiting time does not belong to the delay time, the delay time of 1 hop users will always be a fixed value of 2.5 ms.

4.3 Average packet queue length

Figure 5 shows the average packet queue lengths for all device types. In Fig. 5, we first see the growing up trends of bSS and tSS are similar, and both reach full-load at about 110 ms. In light of Section 4.1, AZ has 105 slots, of which 8 bSS shared 75 slots. Every bSS can be allocated with about 9.375 slots in average; the remaining 30 slots were shared by 3 tSS; and every tSS obtains 10 slots. Since they have identical packet arrival rate and approximately the same packet upload rate, their growing curves are similar. Secondly, we can see that at the initial stage, the number of ntSS queue packets maintained at about 10 packets. When system time is 190 ms, it starts to climb and then reached full loading at 400 ms. At the initial stage, because the tRS queue is under light loading, tRS has the maximum capability to receive serving user packets. However, the queue length of tRS will gradually increase, because the ability of BS receiving packets is lower than the ability of the whole tRSs transmitting packets. When the tRS queue is full, ntRS also gets slow because of the uploading speed, which causes the packet queue length to gradually increase until full.

4.4 Average waiting time

Figure 6 shows the average packet waiting times for various device types. The waiting times of bSS, tSS, and ntSS have been shown by the curves in different colors of red, blue, and green, respectively, i.e., with the values of 126, 120, and 68 ms, respectively. The waiting times can also be obtained by estimation as follows. In AZ, each bSS has 9.375 slots, i.e., 1.875 packets, usable in average; because bSS queue often keeps at 48 packets, the average waiting time of the last packet is 25.6 frame duration, i.e., 128 ms. Similarly, for the waiting times of tSS, since each tSS has 10 slots, i.e., 2 packets, usable in average, the average waiting time of the 48th packet is 24 frame duration, i.e., 120 ms. For the waiting times of tSS, every ntSS can upload 3.5 packets in average, and the queue length often keeps at 47 packets; therefore, the last packet needs to wait for 13.4 frame duration, i.e., 67 ms. Therefore, the estimated results are close to the simulation results.

4.5 Average drop ratio

Figure 7 shows the average packet drop ratios for all device types. In Fig. 7, in terms of bSS, the packet drop ratio is 58 % by simulation, and it is lower than 62.5 % by calculation with average manner, i.e., when the simulator calculates the drop ratio, the arriving and the dropping packets are accumulative from the beginning to the end of the simulation. However, when we calculate the average drop ratio, we only use average packet arrival rate and uplink subframe capacity. At the initial stage of simulation, there are plenty bandwidth resources, so the drop ratio is small. The smaller drop ratio dilutes the late

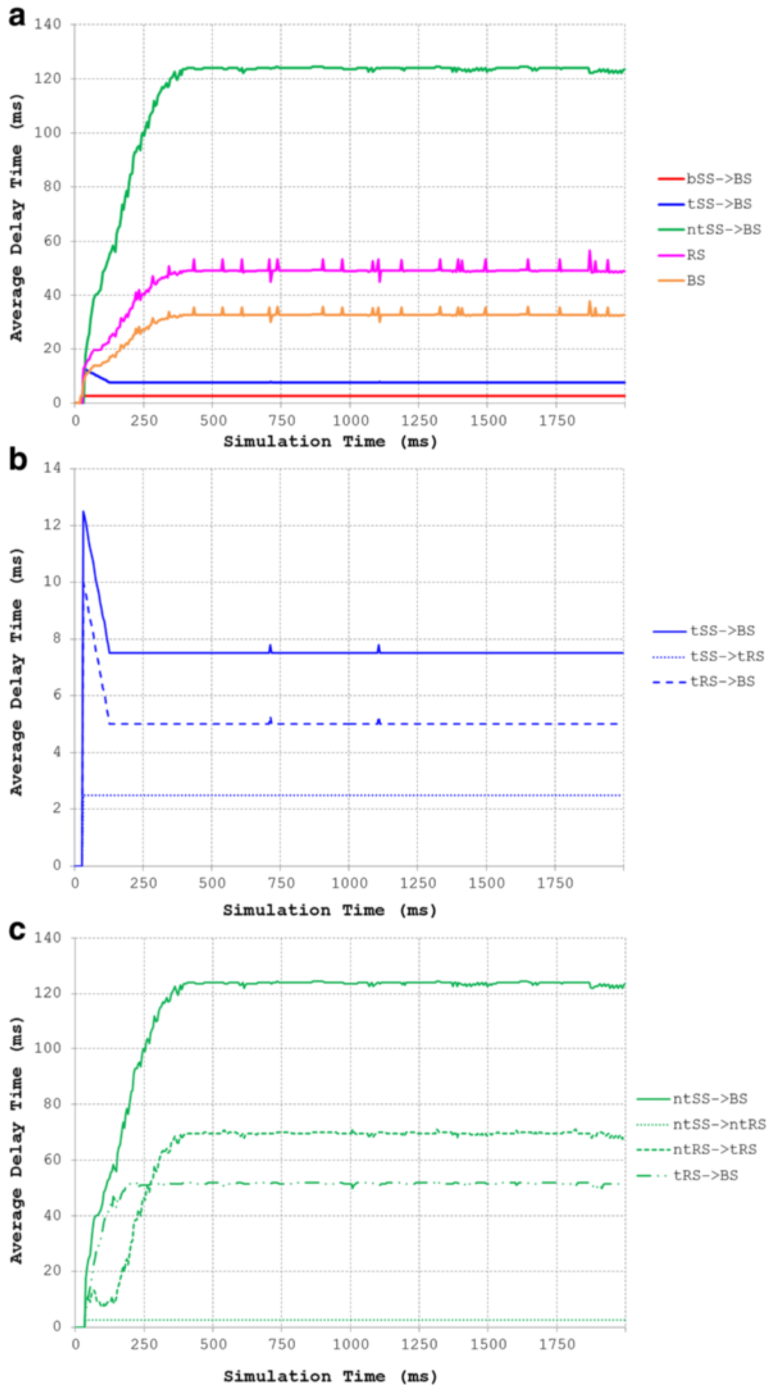


Fig. 4 a The delay time of all device types. **b** The delay time of tSS device types. **c** The delay time of ntSS device types

stage's larger drop ratio; therefore, the simulation drop ratio is lower than the average drop ratio. Similar situation also happens in tSS and ntSS. In terms of tSS, the simulation and average drop ratios are 55.4 and 60 %, respectively, and the ntSS simulation and average drop ratios are 22.5 and 30 %, respectively.

5 Conclusions

In the study domain of wireless network, the network simulator always is an important tool to observe and evaluate the study concept. In order to make the simulator architecture to have the flexibility and the simulation results are believable, we proposed to use the design patterns as the norms of system architecture design and the

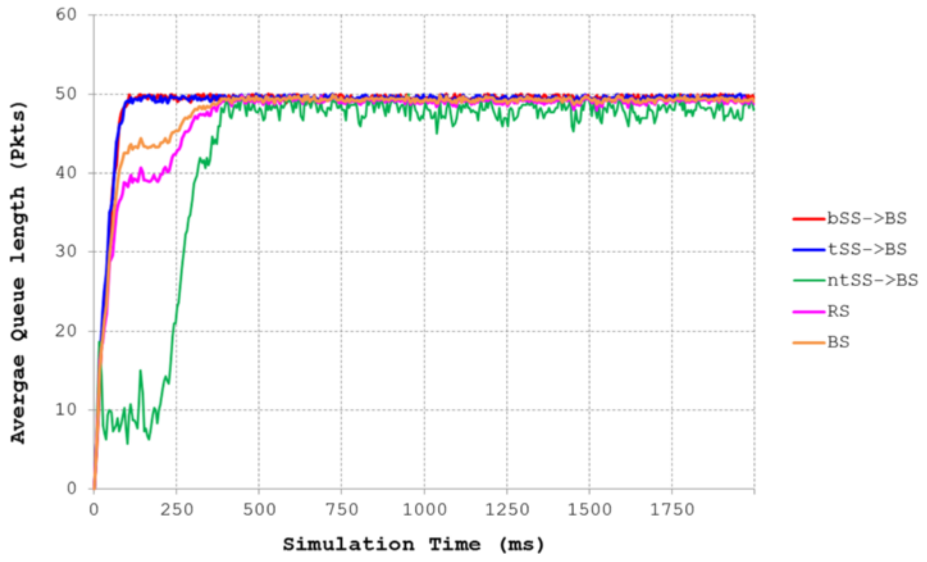


Fig. 5 The average queue length for all device types

creative ideas inspired by the most used simulator design ideas and related module expansion literatures to construct our system architecture. We proposed the CCGns, which follows the IEEE 802.16-2009 standard, coded by the Java language and using Eclipse as develop tool, follows the OOD principles, and refers to the design patterns paradigms. CCGns is a discrete-event virtual network simulator, totally consisting of PHY, MAC, NET, DEV, TPY, EVT, SCH, and RPT packages, whose main contribution is to propose SM^{3}PA, AMRNTA, and TSMVBA. In the architecture description, we provide the design concepts and the UML figures. In the simulation results, we use the most complex hybrid relay network topology as an example and also use the mutual verifying manner for the average calculated values and the simulation results to prove excellent fidelity for the system throughput, average packet delay time, average packet wait time, average packet queue length, and average packet drop ratio. Although we are not the first to propose the design patterns on the wireless network simulator architectures, both the amount and types of using design patterns are the most. In the future, we will also implement the LTE protocol simulation system by using this architecture.

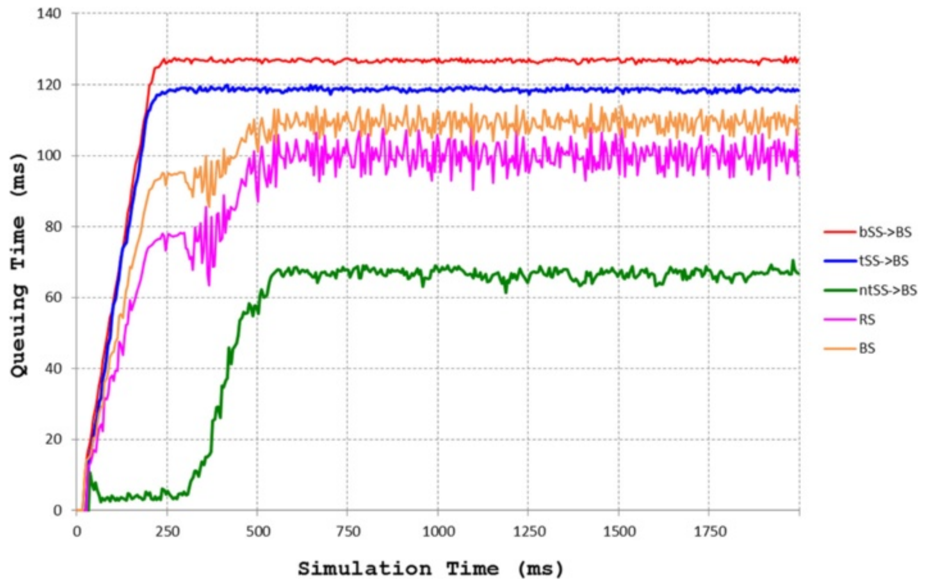


Fig. 6 The average waiting time

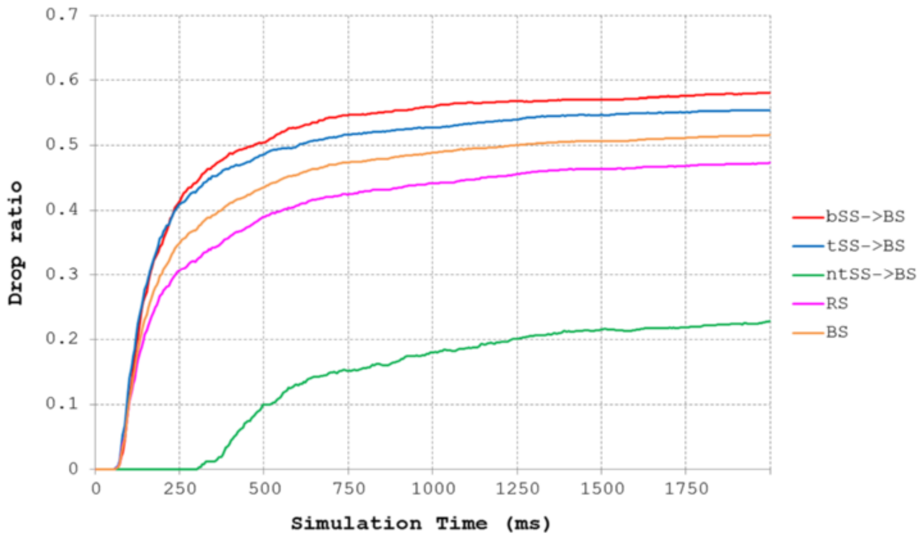


Fig. 7 The average packet drop ratio

6 Appendix 1

6.1 UML diagrams

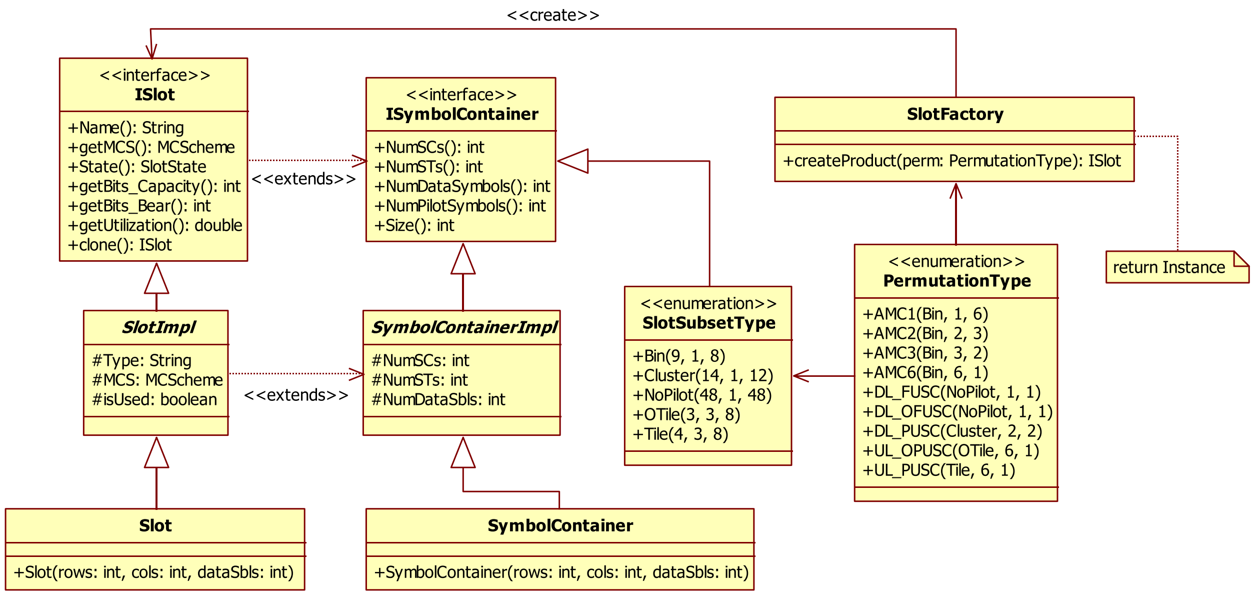


Fig. 8 Slot factory UML diagram

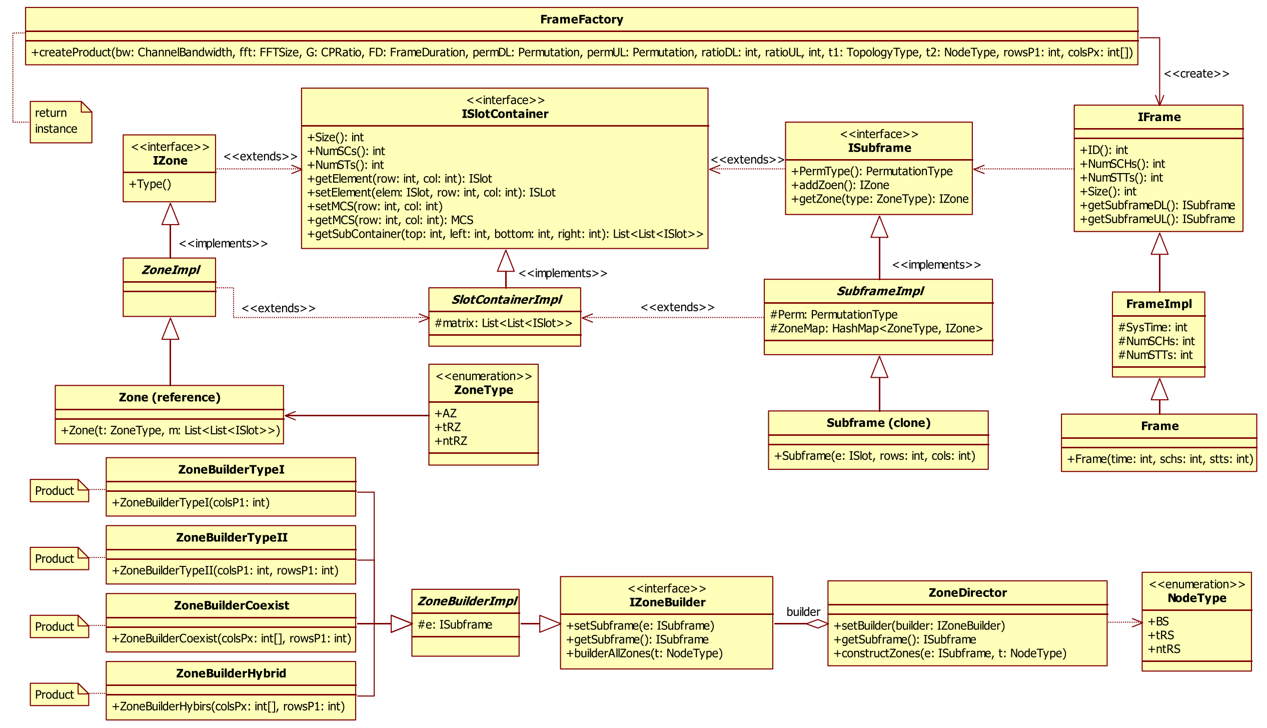
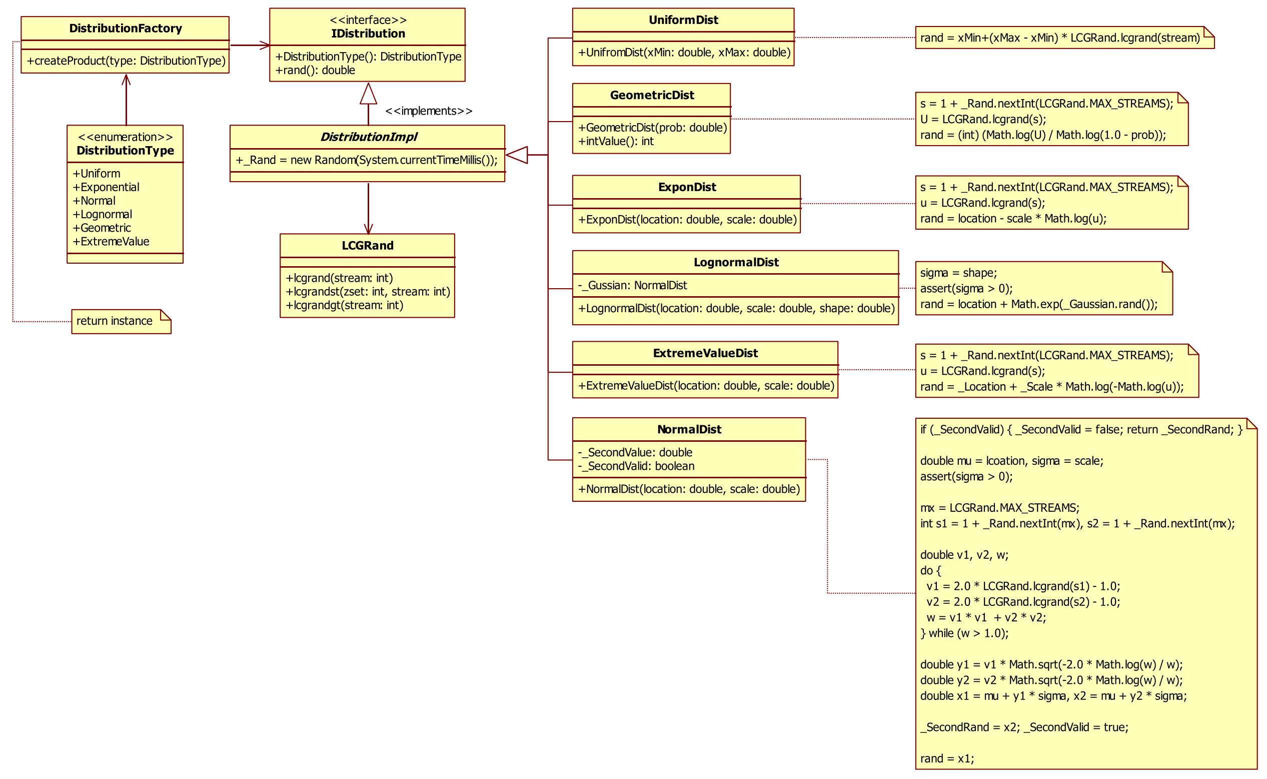

FrameFactory
+createProduct(bw: ChannelBandwidth, fft: FFTSize, G: CPRatio, FD: FrameDuration, permDL: Permutation, permUL: Permutation, ratioDL: int, ratioUL, int, t1: TopologyType, t2: NodeType, rowsP1: int, colsPx: int[])
<<create>>
return instance
<<interface>> IZone
+Type()
<<extends>>
<<interface>> ISlotContainer
+Size(): int
+NumSCs(): int
+NumSTs(): int
+getElement(row: int, col: int): ISlot
+setElement(elem: ISlot, row: int, col: int): ISLot
+setMCS(row: int, col: int)
+getMCS(row: int, col: int): MCS
+getSubContainer(top: int, left: int, bottom: int, right: int): List<List<ISlot>>
<<extends>>
<<interface>> ISubframe
+PermType(): PermutationType
+addZoen(): IZone
+getZone(type: ZoneType): IZone
IFrame
+ID(): int
+NumSCHs(): int
+NumSTTs(): int
+Size(): int
+getSubframeDL(): ISubframe
+getSubframeUL(): ISubframe
<<implements>>
ZoneImpl
<<extends>>
<<implements>>
SlotContainerImpl
#matrix: List<List<ISlot>>
<<extends>>
<<implements>>
SubframeImpl
#Perm: PermutationType
#ZoneMap: HashMap<ZoneType, IZone>
FrameImpl
#SysTime: int
#NumSCHs: int
#NumSTTs: int
Zone (reference)
+Zone(t: ZoneType, m: List<List<ISlot>>)
<<enumeration>> ZoneType
+AZ
+tRZ
+ntRZ
Subframe (clone)
+Subframe(e: ISlot, rows: int, cols: int)
Frame
+Frame(time: int, schs: int, stts: int)
Product
ZoneBuilderTypeI
+ZoneBuilderTypeI(colsP1: int)
Product
ZoneBuilderTypeII
+ZoneBuilderTypeII(colsP1: int, rowsP1: int)
Product
ZoneBuilderCoexist
+ZoneBuilderCoexist(colsPx: int[], rowsP1: int)
Product
ZoneBuilderHybrid
+ZoneBuilderHybirs(colsPx: int[], rowsP1: int)
ZoneBuilderImpl
#e: ISubframe
<<interface>> IZoneBuilder
+setSubframe(e: ISubframe)
+getSubframe(): ISubframe
+builderAllZones(t: NodeType)
builder
ZoneDirector
+setBuilder(builder: IZoneBuilder)
+getSubframe(): ISubframe
+constructZones(e: ISubframe, t: NodeType)
<<enumeration>> NodeType
+BS
+tRS
+ntRS

Fig. 9 Frame factory UML diagram


DistributionFactory
+createProduct(type: DistributionType)
<<interface>> IDistribution
+DistributionType(): DistributionType
+rand(): double
<<implements>>
<<enumeration>> DistributionType
+Uniform
+Exponential
+Normal
+Lognormal
+Geometric
+ExtremeValue
DistributionImpl
+_Rand = new Random(System.currentTimeMillis());
LCGRand
+lcgrand(stream: int)
+lcgrandst(zset: int, stream: int)
+lcgrandgt(stream: int)
return instance
UniformDist
+UnifromDist(xMin: double, xMax: double)
GeometricDist
+GeometricDist(prob: double)
+intValue(): int
ExponDist
+ExponDist(location: double, scale: double)
LognormalDist
-_Gussian: NormalDist
+LognormalDist(location: double, scale: double, shape: double)
ExtremeValueDist
+ExtremeValueDist(location: double, scale: double)
NormalDist
-_SecondValue: double
-_SecondValid: boolean
+NormalDist(location: double, scale: double)

```
rand = xMin+(xMax - xMin) * LCGRand.lcgrand(stream)
```

```
s = 1 + _Rand.nextInt(LCGRand.MAX_STREAMS);
U = LCGRand.lcgrand(s);
rand = (int) (Math.log(U) / Math.log(1.0 - prob));
```

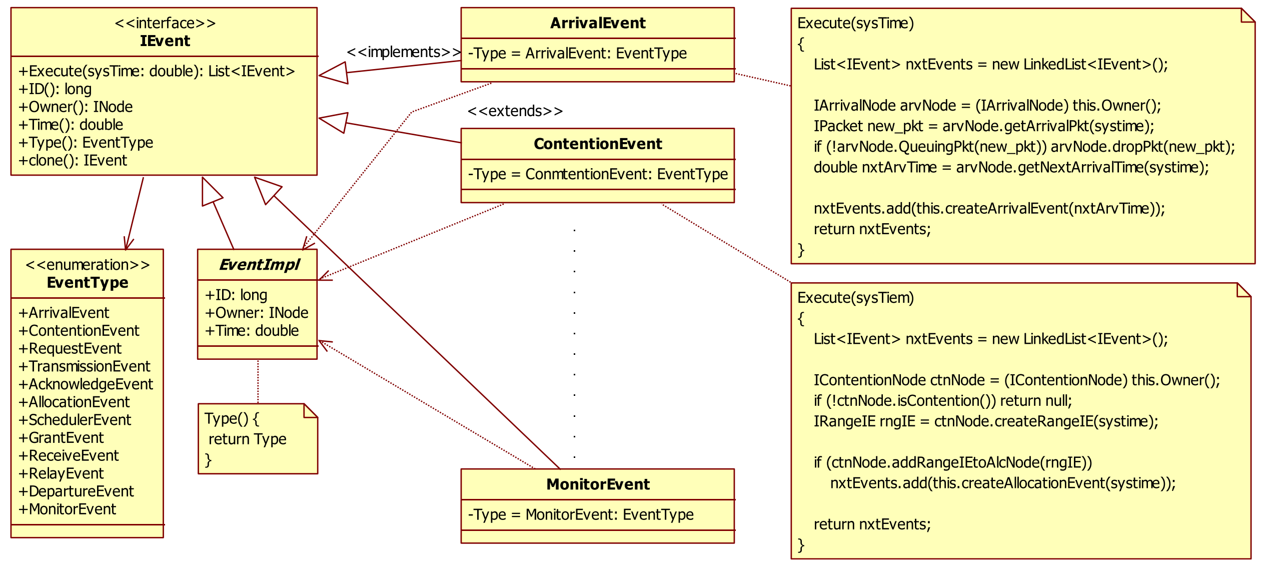
```
s = 1 + _Rand.nextInt(LCGRand.MAX_STREAMS);
u = LCGRand.lcgrand(s);
rand = location - scale * Math.log(u);
```

```
sigma = shape;
assert(sigma > 0);
rand = location + Math.exp(_Gaussian.rand());
```

```
s = 1 + _Rand.nextInt(LCGRand.MAX_STREAMS);
u = LCGRand.lcgrand(s);
rand = _Location + _Scale * Math.log(-Math.log(u));
```

```
if (_SecondValid) { _SecondValid = false; return _SecondRand; }

double mu = lcoation, sigma = scale;
assert(sigma > 0);

mx = LCGRand.MAX_STREAMS;
int s1 = 1 + _Rand.nextInt(mx), s2 = 1 + _Rand.nextInt(mx);

double v1, v2, w;
do {
  v1 = 2.0 * LCGRand.lcgrand(s1) - 1.0;
  v2 = 2.0 * LCGRand.lcgrand(s2) - 1.0;
  w = v1 * v1  + v2 * v2;
} while (w > 1.0);

double y1 = v1 * Math.sqrt(-2.0 * Math.log(w) / w);
double y2 = v2 * Math.sqrt(-2.0 * Math.log(w) / w);
double x1 = mu + y1 * sigma, x2 = mu + y2 * sigma;

_SecondRand = x2; _SecondValid = true;

rand = x1;
```

Fig. 10 The random number generator UML diagram



```
Execute(sysTime)
{
  List<IEvent> nxtEvents = new LinkedList<IEvent>();

  IArrivalNode arvNode = (IArrivalNode) this.Owner();
  IPacket new_pkt = arvNode.getArrivalPkt(systime);
  if (!arvNode.QueuingPkt(new_pkt)) arvNode.dropPkt(new_pkt);
  double nxtArvTime = arvNode.getNextArrivalTime(systime);

  nxtEvents.add(this.createArrivalEvent(nxtArvTime));
  return nxtEvents;
}
```

```
Execute(sysTiem)
{
  List<IEvent> nxtEvents = new LinkedList<IEvent>();

  IContentionNode ctnNode = (IContentionNode) this.Owner();
  if (!ctnNode.isContention()) return null;
  IRangeIE rngIE = ctnNode.createRangeIE(systime);

  if (ctnNode.addRangeIEtoAlcNode(rngIE))
     nxtEvents.add(this.createAllocationEvent(systime));

  return nxtEvents;
}
```

Fig. 11 The event package UML diagram

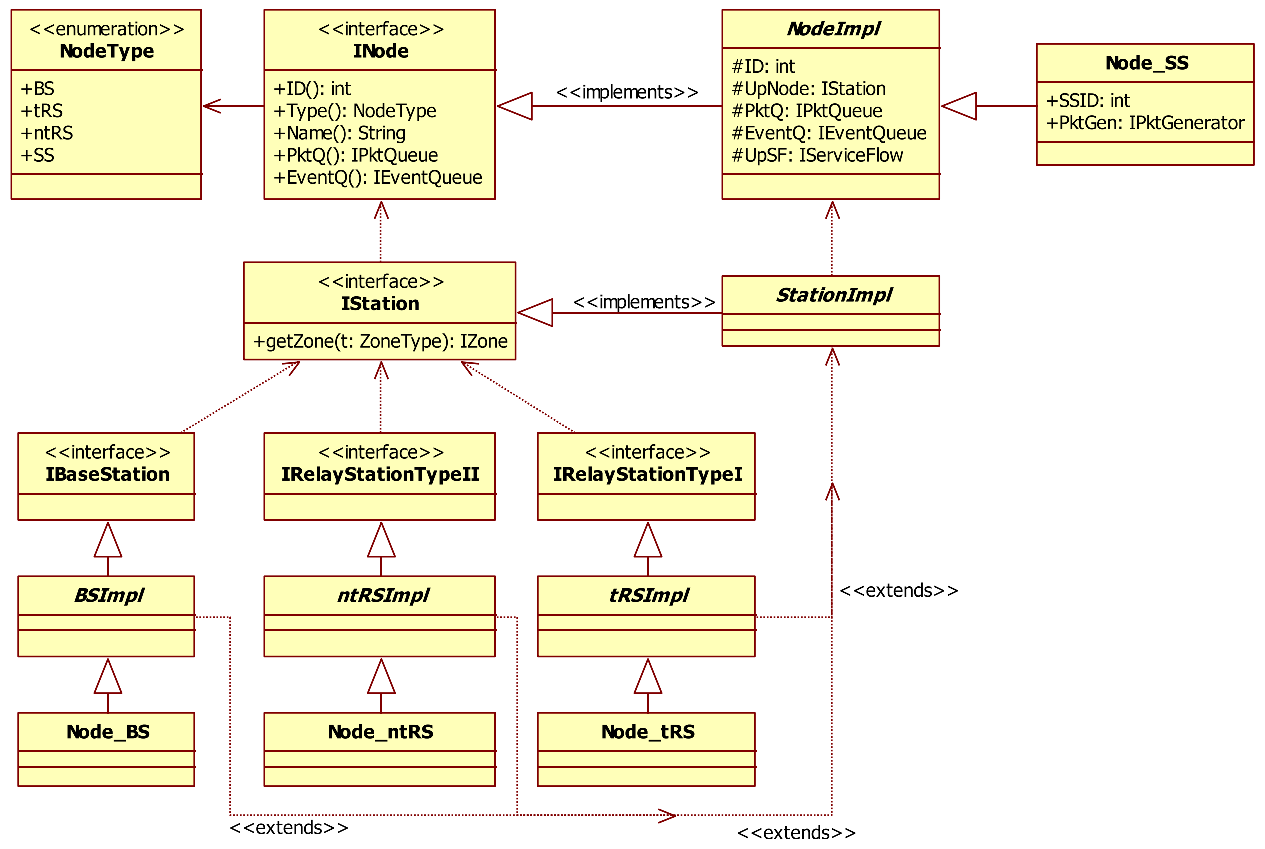
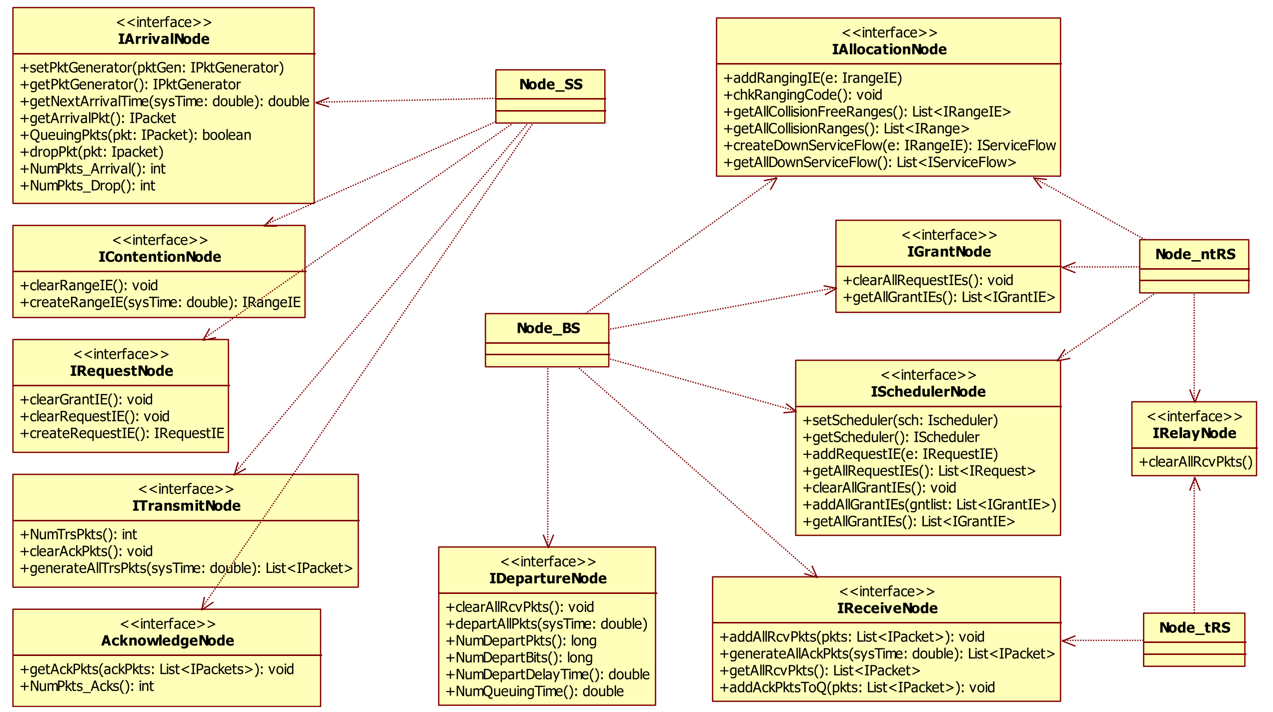
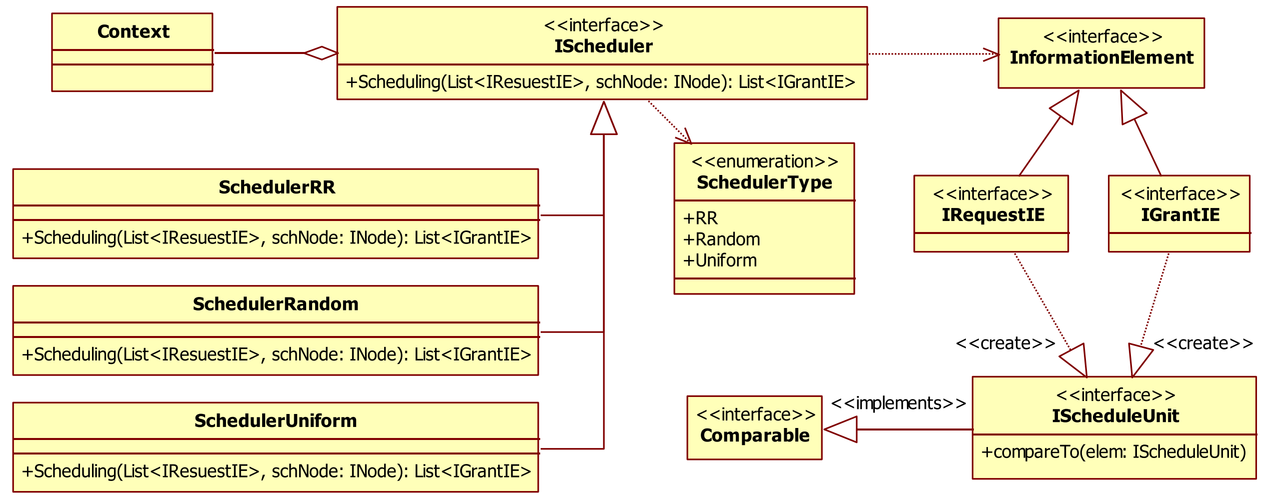


Fig. 12 The basic device class UML diagram



<<interface>> IArrivalNode
+setPktGenerator(pktGen: IPktGenerator)
+getPktGenerator(): IPktGenerator
+getNextArrivalTime(sysTime: double): double
+getArrivalPkt(): IPacket
+QueuingPkts(pkt: IPacket): boolean
+dropPkt(pkt: Ipacket)
+NumPkts_Arrival(): int
+NumPkts_Drop(): int
<<interface>> IContentionNode
+clearRangeIE(): void
+createRangeIE(sysTime: double): IRangeIE
<<interface>> IRequestNode
+clearGrantIE(): void
+clearRequestIE(): void
+createRequestIE(): IRequestIE
<<interface>> ITransmitNode
+NumTrsPkts(): int
+clearAckPkts(): void
+generateAllTrsPkts(sysTime: double): List<IPacket>
<<interface>> AcknowledgeNode
+getAckPkts(ackPkts: List<IPackets>): void
+NumPkts_Acks(): int
Node_SS
Node_BS
<<interface>> IDepartureNode
+clearAllRcvPkts(): void
+departAllPkts(sysTime: double)
+NumDepartPkts(): long
+NumDepartBits(): long
+NumDepartDelayTime(): double
+NumQueuingTime(): double
<<interface>> IAllocationNode
+addRangingIE(e: IrangeIE)
+chkRangingCode(): void
+getAllCollisionFreeRanges(): List<IRangeIE>
+getAllCollisionRanges(): List<IRange>
+createDownServiceFlow(e: IRangeIE): IServiceFlow
+getAllDownServiceFlow(): List<IServiceFlow>
<<interface>> IGrantNode
+clearAllRequestIEs(): void
+getAllGrantIEs(): List<IGrantIE>
Node_ntRS
<<interface>> ISchedulerNode
+setScheduler(sch: Ischeduler)
+getScheduler(): IScheduler
+addRequestIE(e: IRequestIE)
+getAllRequestIEs(): List<IRequest>
+clearAllGrantIEs(): void
+addAllGrantIEs(gntlist: List<IGrantIE>)
+getAllGrantIEs(): List<IGrantIE>
<<interface>> IRelayNode
+clearAllRcvPkts()
<<interface>> IReceiveNode
+addAllRcvPkts(pkts: List<IPacket>): void
+generateAllAckPkts(sysTime: double): List<IPacket>
+getAllRcvPkts(): List<IPacket>
+addAckPktsToQ(pkts: List<IPacket>): void
Node_tRS

Fig. 13 The role device class UML diagram



Context
<<interface>> IScheduler
+Scheduling(List<IResuestIE>, schNode: INode): List<IGrantIE>
<<interface>> InformationElement
SchedulerRR
+Scheduling(List<IResuestIE>, schNode: INode): List<IGrantIE>
SchedulerRandom
+Scheduling(List<IResuestIE>, schNode: INode): List<IGrantIE>
SchedulerUniform
+Scheduling(List<IResuestIE>, schNode: INode): List<IGrantIE>
<<enumeration>> SchedulerType
+RR
+Random
+Uniform
<<interface>> IRequestIE
<<interface>> IGrantIE
<<create>>
<<create>>
<<interface>> Comparable
<<implements>>
<<interface>> IScheduleUnit
+compareTo(elem: IScheduleUnit)

Fig. 14 The scheduler package UML diagram

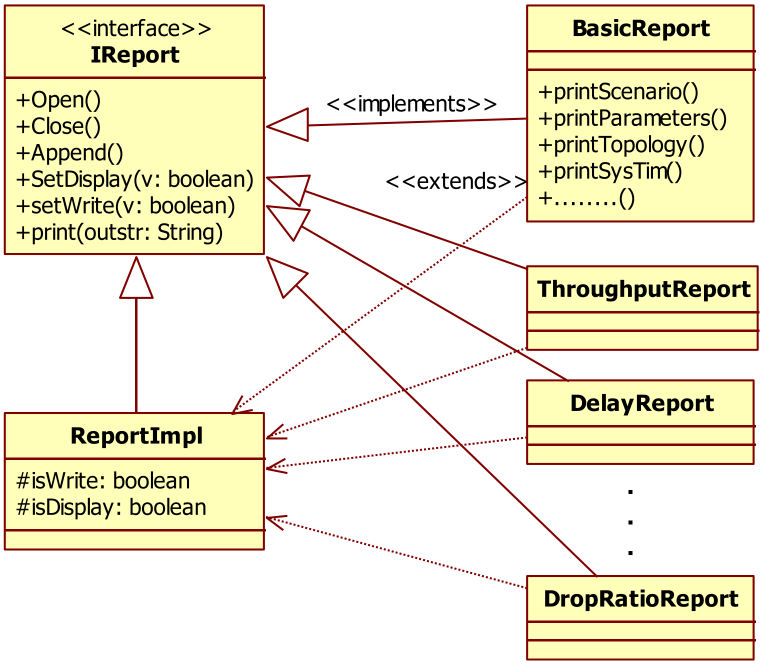


Fig. 15 The report package UML diagram

Acknowledgments
This work was supported in part by the Ministry of Science and Technology (MOST) of Taiwan under Grant No.: MOST 102-2221-E-011-069.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan. [2]Oriental Institute of Technology, New Taipei City, Taiwan.

Received: 25 August 2015 Accepted: 14 May 2016
Published online: 02 June 2016

References
1. J-Sim, [Online] Available: https://sites.google.com/site/jsimofficial/. Accessed 26 May 2016
2. NS-2, [Online] Available: http://www.isi.edu/nsnam/ns/. Accessed 26 May 2016
3. NS-3, [Online] Available: http://www.nsnam.org/. Accessed 26 May 2016
4. N Bagoria, A Garhwal, A Sharma, Simulation of Physical layer of WiMAX Network using OPNETModeller, International Journal of P2P Network Trends and Technology (IJPTT), 2013, **3**(4).
5. OPNET WiMAX, [Online] Available: http://www.opnet.com/WiMAX/index.html
6. QualNet [Online] Available: http://web.scalable-networks.com/content/qualnet. Accessed 26 May 2016
7. TIOBE [Online] Available: http://www.tiobe.com/tiobe_index?page=index. Accessed 26 May 2016
8. L Hogie, P Bouvry, F Guinand, An overview of MANETs simulation, in *Electronic Notes in Theoretical Computer Science, Proc. of 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord'05), LNCS* (Elsevier, Namur, Belgium, 2005), pp. 81–101
9. S Siraj, AK Gupta, R Badgujar, Network simulation tools survey. International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE) **1**(4), 201–210 (2012)
10. S Duflos, GL Grand, AA Diallo, C Chaudet, A Hecker, C Balducelli, F Flentge, C Schwaegerl, O Seifert, List of available and suitable simulation components, in *Technical report, Ecole Nationale Superieure des Telecommunications (ENST)*, 2006
11. L Begg, W Liu, L Pawlikowski, S Perera, H Sirisena, Survey of simulators of next generation networks for studying service availability and resilience, in *Technical Report TR-COSC 05/06, Department of Computer Science & Software Engineering* (University of Canterbury, Christchurch, New Zealand, 2006)
12. NI Sarkar, SA Halim, A review of simulation of telecommunication networks: simulators, classification, comparison, methodologies, and recommendations. Cyber Journals: Multidisciplinary Journals in Science and Technology. Special Issue, Journal of Selected Areas in Telecommunications (JSAT) **2**(3), 10–17 (2011)
13. A Kumar, SK Kaushik, R Sharma, P Raj, Simulators for wireless networks: a comparative study, in *International Conference on Computing Sciences (ICCS)*, 2012, pp. 338–342
14. S-M Huang, Y-C Sung, S-Y Wang, Y-B Lin, NCTUns simulation tool for WiMAX modeling, in *Third Annual International Wireless Internet Conference*, 2007, pp. 22–24
15. J Lessmann, P Janacik, L Lachev, D Orfanus, Comparative study of wireless network simulators, in *Seventh International Conference on Networking*, 2008, pp. 517–523
16. E Weingartner, H Vom Lehn, K Wehrle, A performance comparison of recent network simulators, in *IEEE International Conference on Communications (ICC)*, 2009, pp. 1–5
17. AR Khan, SM Bilal, M Othman, A performance comparison of open source network simulators for wireless networks, in *IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 2012, pp. 34–38
18. M. Greis, NS tutorial, [Online] Available: http://www.isi.edu/nsnam/ns/tutorial/. Accessed 26 May 2016
19. FC-D Tsai, J Chen, C-W Chang, W-J Lien, C-H Hung, J-H Sum, *The design and implementation of WiMAX module for NS-2 simulator, in Proc. of the Workshop on Ns-2: the IP Network Simulator*, 2006, pp. 1–8
20. J Chen, C Wang, F Tsai, C Chang, S Liu, J Guo, JSW Lien, C Hung, "Design and implementation of wimax module for ns-2 simulator," in *1st International Conference on Performance EvaluationMethodologies and Tools (VALUETOOLS'06) (ACM)*, 2006.
21. N. I. of Standards and Technology, *The network simulator NS-2 NIST add-on—IEEE 802.16 model (PHY + MAC), Technical Report*, 2009

22. NIST (National Institute of Standards and technology), [Online] Available: http://www.nist.gov/itl/antd/emntg/ssm_tools.cfm. Accessed 26 May 2016
23. LRC (Computer Networks Laboratory), [Online] Available: http://www.lrc.ic.unicamp.br/wimax_ns2/. Accessed 26 May 2016
24. Y-C Lai, Y-H Chen, *Designing and implementing an IEEE 802.16 network simulator for performance evaluation of bandwidth allocation algorithms, 2009 11th IEEE International Conference on High Performance Computing and Communications (HPCC'09)*, 2009, pp. 432–437
25. J Freitag, NLS Da Fonseca, WiMAX module for the ns-2 simulator, in *IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2007, pp. 1–6
26. X Guo, R Rouil, C Soin, S Parekh, B Sikdar, S Kalyanaraman, WiMAX system design and evaluation methodology using the NS-2 simulator, in *Communication Systems and Networks and Workshops (COMSNETS)*, 2009, pp. 1–10
27. J Farooq, T Turlett, An IEEE 802.16 WiMAX module for the NS-3 simulator, in *Proc. of the 2nd International Conference on Simulation tools and Techniques*, 2009, pp. 1–11
28. MA Ismail, G Piro, LA Grieco, T Turletti, An improved IEEE 802.16 WiMAX module for the NS-3 simulator. Proc. SIMUTools **63**, 1–63 (2010)
29. WP Furlong, R Guha, OFDMA extension of NS-3 WiMAX module, in *UKSim Fourth European Modeling Symposium on Computer Modeling and Simulation*, 2010, pp. 426–431
30. G Pedreno, JJ Alcaraz, F Cerdan, Using design patterns in a HSDPA system simulator, in *3rd International Symposium on Wireless Communication Systems (ISWCS)*, 2006, pp. 679–683
31. E Gamma, R Helm, R Johnson, J Vlissides, *Design patterns: elements of reusable object-oriented software*, 1995
32. K Pawlikowski, V Yau, AKAROA: a Package for Automatic Generation and Process Control of ParallelStochastic Simulation, Australian Computer Science Communications, **15**(1), 71-82 (1993)
33. Wiki, Project DIANE, [Online] Available: http://en.wikipedia.org/wiki/Project_DIANE. Accessed 26 May 2016
34. SSFNet, [Online] Available: http://www.ssfnet.org/homePage.html. Accessed 26 May 2016
35. Wiki, GloMoSim, [Online] Available: http://en.wikipedia.org/wiki/GloMoSim. Accessed 26 May 2016
36. GTNetS, [Online] Available: http://www2.ece.gatech.edu/research/labs/MANIACS/GTNetS/. Accessed 26 May 2016
37. D Gorgen, H Frey, C Hiedels, JANE—the Java ad hoc network development environment, in *40th Annual Simulation Symposium (ANSS)*, 2007, pp. 163–176
38. E Schoch, M Feiri, F Kargl, M Weber, Simulation of ad hoc networks: ns-2 compared to JiST/SWANS, in *Simutools '08 Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST), 2008
39. S-M Huang, Y-C Sung, S-Y Wang, Y-B Lin, Nctuns simulation tool for wimax modeling, *in WICON '07: Proceedings of the 3rd international conference on Wireless internet*, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST), 2007, pp. 1–6
40. BOSON NetSim, [Online] Available: http://www.tetcos.com/netsim_gen.html. Accessed 26 May 2016
41. N Kotilainen, M Vapa, T Keltanen, A Auvinen, J Vuori, P2PRealm—peer-to-peer network simulator, in *IEEE International Workshop on Computer-Aided Modeling, Analysis, and Design of Communication Links and Networks (CAMAD)*, 2006, pp. 93–99
42. REAL, [Online] Available: http://www.cs.cornell.edu/skeshav/real/overview.html. Accessed 26 May 2016
43. Shunra, Shunra Virtual Enterprise (VE), [Online] Available: https://en.wikipedia.org/wiki/Shunra. Accessed 26 May 2016
44. ShoX, [Online] Available: http://shox.sourceforge.net/. Accessed 26 May 2016
45. SimPy, [Online] Available: http://simpy.readthedocs.org/en/latest/. Accessed 26 May 2016
46. TOTEM, [Online] Available: http://totem.info.ucl.ac.be/. Accessed 26 May 2016