1 Introduction

This chapter introduces network modeling and simulation, and both OPNET and OPNET Modeler. If you already have relevant background, you can quickly read through this chapter and go to Chapter 2.

1.1 Network modeling and simulation

There are several feasible methods for investigating networking protocols and evaluating network performance (Leemis and Park 2006; www.opnet.com):

- Analysis and mathematical modeling
- Simulation – typically time-based simulation or discrete event-based simulation
- Hybrid simulation with both analysis and simulation
- Test-bed emulation

Analysis and mathematical modeling can provide quick insights and answers to the problems being studied. It is generally faster than simulation, but in many cases is inaccurate or inapplicable. Analytical models are not available for many situations. Even so, many of the available models lack accuracy and some are modeled through approximations. Especially for a network of queues, it can either be decomposed via the Kleinrock independence assumption or be solved using a hop-by-hop single system analysis, both of which lose accuracy. The modeling difficulties and loss of accuracy can be greatly exacerbated when the networking protocols become even slightly complex. It is often necessary to resort to approximation by reducing the general model to a typical and representative analytical path in order to reduce the analytical difficulties (Kleinrock, 1976).

Network simulation provides a way to model the network behaviors by calculating the interactions between modeling devices. Discrete event simulation (DES) is the typical method in large-scale simulation studies instead of a simpler time-based method. DES enables modeling in a more accurate and realistic way, and has broad applicability (Leemis and Park 2006). DES creates an extremely detailed, packet-by-packet model for the activities of network to be predicted. However, it often has significant requirements for computing power; in particular, for very large-scale simulation studies, the process can be time-consuming. It can take several hours or even days to complete. However, simulation can always provide accurate solutions for either a single-node queuing system or a network of queues, from simple algorithm to complex protocol.

One way to work around these issues in mathematical analysis and explicit simulation is combining the methods in the simulation in order to gain access to the advantages of both while overcoming their disadvantages. This combined method is typically called hybrid simulation, i.e., partially modeling in DES for accuracy and partially in mathematical analysis for faster speed and less computational burden (see www.opnet.com).

There are many network simulators like OPNET (see www.opnet.com), NS (www.isi.edu/nsnam/ns), and OMNeT++ (www.omnetpg.org) which are popular and widely used. Among them, OPNET is capable of simulating in both explicit DES and hybrid simulation modes, and supports other simulation features like co-simulation, parallel simulation, high-level architecture, and system-in-the-loop interactive simulations.

Test-bed emulation typically involves implementing the studied algorithms and protocols into real-world hardware but in a much smaller scale or size. Since test-bed emulation considers the aspects of both protocols and real-world situations, it is the best way to provide a benchmark estimating how feasible the algorithms and protocols are and how close they are to the actual situation. Also, it is a useful way demonstrate new networking concepts. The disadvantage is it will also deal with all other real-world difficulties and some unexpected engineering problems which can be completely irrelevant to the studied algorithms and protocols but can be significant in the overall emulation results. Further, the cost of building an emulation test-bed may be significant. Test-beds are not suitable for investigating large systems.

Accordingly, research methodologies for data traffic and networking can be a combination of some or all of these methods. These methods can be used to cross-check each other in order to capture the system in a more accurate, efficient, and cost-effective way.

1.2 Introduction to OPNET

OPNET stands for OPtimized Network Engineering Tools, and was created by OPNET Technologies, Inc., which was founded in 1986. OPNET is a network simulation tool set; its products and solutions address the following aspects of communications networks (see www.opnet.com):

- Application performance management
- Planning
- Engineering
- Operations
- Research and development

This tool set is powerful and can create and test large network environments via software. To address each of these aspects, OPNET provides corresponding product modules throughout its product line.

OPNET products for "application performance management" include ACE Analyst for analytics for networked applications, ACE Live for end-user experience monitoring and real-time network analytics, OPNET Panorama for real-time application monitoring and analytics, and IT Guru Systems Planner for systems capacity management for enterprises.

OPNET products for "network planning, engineering, and operations" include IT/SP Guru Network Planner for network planning and engineering for enterprises and service providers, SP Guru Transport Planner for transport network planning and engineering, NetMapper for automated up-to-date network diagramming, IT/SP Sentinel for network audit, security and policy-compliance for enterprises and service providers, SP Sentinel for network audit, security and policy-compliance for service providers, and OPNET nCompass for providing a unified, graphical visualization of large, heterogeneous production networks for enterprises and service providers.

OPNET products for "network research and development" include OPNET Modeler, OPNET Modeler Wireless Suite, and OPNET Modeler Wireless Suite for Defense.

The products applicable in this book are OPNET Modeler and OPNET Modeler Wireless Suite.

1.3 OPNET Modeler

OPNET Modeler is the foremost commercial product that provides network modeling and simulation software solution among the OPNET product family. It is used widely by researchers, engineers, university students, and the US military. OPNET Modeler is a dynamic discrete event simulator with a user-friendly graphic user interface (GUI), supported by object-oriented and hierarchical modeling, debugging, and analysis. OPNET Modeler is a discrete event simulator that has evolved to support hybrid simulation, analytical simulation, and 32-bit and 64-bit fully parallel simulation, as well as providing many other features. It has grid computing support for distributed simulation. Its System-in-the-Loop interface allows simulation with live systems which feed real-world data and information into the simulation environment. It provides an open interface for integrating external object files, libraries, and other simulators. It incorporates a broad suite of protocols and technologies, and includes a development environment to enable modeling of a very wide range of network types and technologies. With the ongoing release of updated versions, OPNET Modeler incorporates more and more features in order to keep up with the evolution of communication networks, devices, protocols, and applications. Hundreds of protocols and vendor device models with source code are already incorporated in the modeler. OPNET Modeler accelerates the research and development (R&D) process for analyzing and designing communication networks, devices, protocols, and applications (see www.opnet.com). OPNET Modeler GUI makes it user-friendly, and makes it easy for users to begin learning about it and working with it. However, when trying to progress beyond this initial phase, its full-featured functionalities and powerful programming interfaces make it difficult for people to grasp.

OPNET Modeler provides a comprehensive development environment with a full set of tools including model design, simulation, data collection, and data analysis and supporting the modeling of communication networks and distributed systems. OPNET Modeler can be used as a platform to develop models of a wide range of systems. These applications include: standard-based local area network (LAN) and wide area network (WAN) performance modeling, hierarchical internetwork planning, R&D of protocols and communication network architecture, mobile network, sensor network and satellite network. Other applications include resource sizing, outage and failure recovery, and so on.

OPNET Modeler is used in the case studies throughout this book. Readers of this book are assumed to have the license for this particular OPNET product to be able to go through the book content in practice. This book is based on OPNET Modeler 14.5 and later versions, but the modeling methodologies discussed are applicable to earlier versions.

1.4 Summary

This chapter discusses the methodologies for network modeling and simulation. OPNET and its products are introduced. Among OPNET products, OPNET Modeler is the one to address network research and development, and is the product to be used in the case studies throughout this book.

1.5 Theoretical background

1.5.1 Simulation and principles of simulator

Network simulations can be categorized into time-clocked simulation and discrete event simulation. In time-clocked simulation, simulation progresses through the iterative progressing of time slots. Events within the iterated time slot are executed while simulation is progressing. The flowchart of time-clocked simulation is shown in Figure 1.1.

In discrete event simulation, simulation progresses by the execution of the scheduled next event. Simulation time is updated after the next scheduled event is executed. The flowchart of DES simulation is shown in Figure 1.2.

Q1.1 What are the differences between the time slots in time-based simulation and simulation time?

The time slots in time-based simulation refer to the clock time in the real world. Simulation time refers to the time used in running the model. Thus, the simulation time and the time elapsed in a run of the simulation are two different concepts.

Compared with discrete event simulation, time-clocked simulation will iterate all time slots regardless of whether there are events within a particular time slot or not. For a burstlike system with long silent periods, i.e., there are no events in many continuous slots, the time-clocked simulation will be inefficient since it still needs to iterate all those time slots without events being processed. Instead, discrete event simulation only

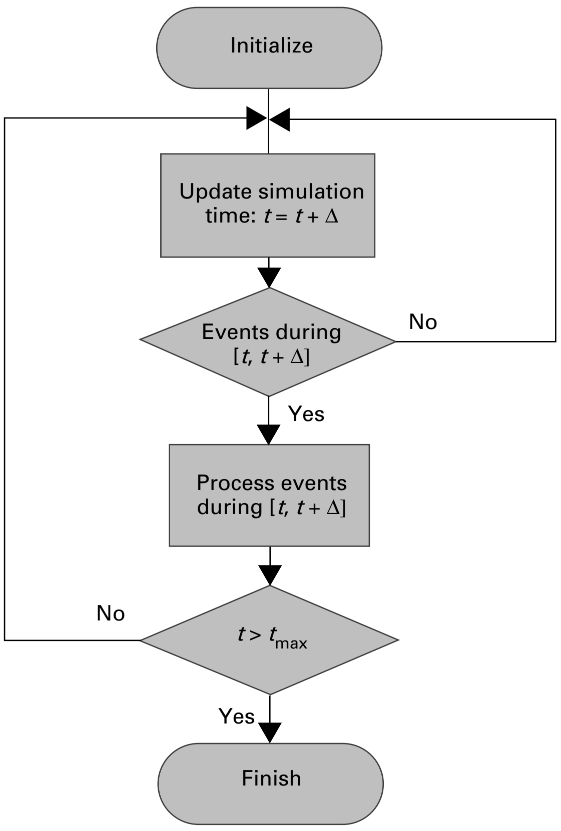
Initialize
Update simulation time: $t = t + \Delta$
Events during $[t, t + \Delta]$
No
Yes
Process events during $[t, t + \Delta]$
No
$t > t_{max}$
Yes
Finish

Figure 1.1 Time-based simulation (Robinson 2004, Hayes 2004)

iterates the scheduled events which must be processed in an ordered fashion, to avoid the inefficiency incurred in time-clocked simulation. For this reason, most modern simulators support the approach of discrete event simulation, i.e., DES.

To be able to execute DES, a basic DES simulator framework should have the following elements:

- The random generators representing different random variables as initial system inputs like packet size, packet interarrival times, system processing time and noise, etc.
- Simulation time which can be updated to allow simulation to progress
- Prioritized event lists to store events to be executed one by one
- Simulation finish conditions such as simulation duration, which is the normal way of finishing a simulation, and some other customized termination conditions.

Figure 1.3 shows the pseudo-code of the structure of the basic DES simulator. It has three phases: initialization, simulation, and cleanup. In initialization phase, all state variables are populated with initial values, like simulation time, event list, statistics, and memories. In simulation kernel phase, a main loop is used to run the simulation until a simulation termination condition is reached such as simulation finish time. If

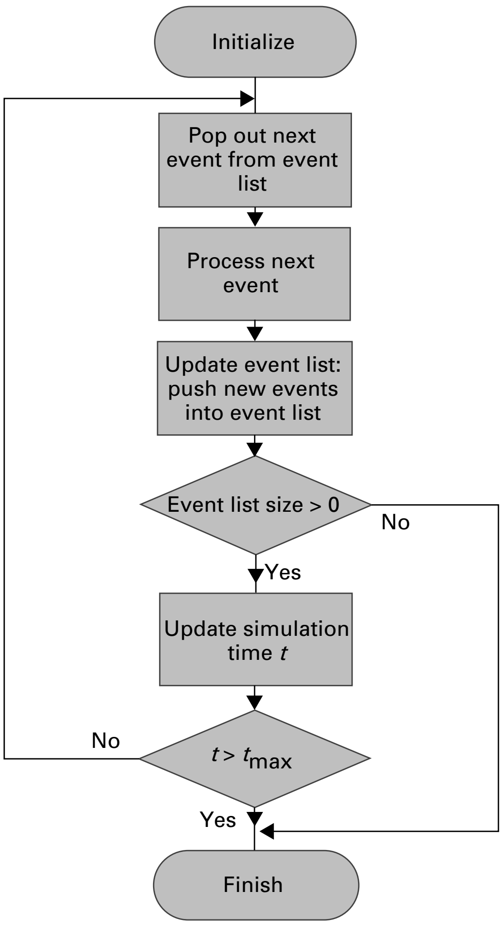

Figure 1.2 Event-based simulation (Robinson 2004, Hayes 2004)

the termination condition is not satisfied, the next scheduled event is popped out and processed. The processing of an event may include: calculating formulas, recording statistics, spawning more events and pushing them into event list, cleaning up invalid variables and free memories, creating new variables and memories, etc. After an event is processed, the simulation time will be updated. The updated value is calculated according to the particular event being executed. This process continues until termination conditions are reached. After leaving the simulation kernel phase, some cleanup work will be done before finishing the simulation, such as writing records into files, freeing memories, and so on.

The simple simulator framework demonstrated in Figure 1.3 is to model a single-process system. However, sometimes systems with concurrent behaviors need to be modeled such as TCP server applications. To model the concurrent behaviors, multiple-processes simulation support will be required inside the simulator's kernel.

```
void main()
{
  //~ initialization
  initialize_variables();
  alocate_memories();
  ...

  //~ simulation kernel operations
  while(simulation_time < finish_time)
  {
    current_event = pop_next_event_from_list();
    process_event(current_event);
    update_simulation_time();
    ...
  }

  //~ finishing up
  write_records_to_file();
  free_memories();
  ...
}
```

Figure 1.3 Pseudo-code for simulation

Q1.2 What are the differences between an operating system process and a simulation process?

An operating system can concurrently run multiple system processes, and a system process can have multiple system threads running within it. A system process must have at least one system thread as its main thread. However, a system thread is lighter than a system process from the perspectives of startup speed, resource occupance, and management burden. Different from a system process, a simulation process here refers to the simulated task such as "transfer of packets," which is an abstract concept used in simulation and is irrelevant to the operating system. In this book, process anywhere refers to a simulation process.

1.5.2 Hybrid simulation

Explicit DES provides accurate simulation results, while analytical methods generally take much less time to compute. Hybrid simulation is a methodology of combining both explicit DES methods and analytical methods in order to take advantages of both. Hybrid simulation may take different forms in actual implementations. In OPNET simulation, the simplest form of hybrid simulation is background traffic simulation. In simulation, traffic through a node can be divided into two parts: explicit traffic and background traffic. Explicit traffic is simulated accurately through the DES method while background traffic is derived analytically. For explicit traffic, each packet's arrival and departure times together with other data of interest are explicitly modeled and recorded. However, for background traffic, there is no tracking of individual packets. The traffic is generated by the workload of the modeled background traffic, and the workload is produced by analytical modeling. Background traffic represents high-level information and is collected over long periods of time. Background traffic is used to characterize and simulate part of a network at an abstracted level in contrast to explicit traffic, which is modeled at a detailed level. The objective in incorporating background traffic is to dramatically reduce the computing power and memory required, in order to save simulation run time.

2 Installation of OPNET Modeler and setting up environments

This chapter shows the steps for installing and configuring the OPNET Modeler and its related environment variables. Having followed this chapter, one should be able to run the OPNET Modeler correctly. If OPNET Modeler and relevant software have already been installed and environment variables have been configured on the target machine, this chapter can be skipped. If you have problems compiling OPNET models, especially, compiling standard OPNET models which should have no compilation and linking errors, please check this chapter to make sure your software is properly installed and environment variables are correctly configured, since many OPNET model compilation and linking errors come from incorrect configuration of the C/C++ compiler's environment variables.

This chapter first describes the system requirements for using OPNET Modeler, including both hardware and software requirements. Then it shows the steps for installing and configuring OPNET Modeler on both Windows and Linux operating systems respectively.

2.1 System requirements for using OPNET Modeler

This section lists the requirements for using OPNET Modeler 14.5 and later versions, and also highlights the relevant key points. For other versions of OPNET Modeler, please check the system requirements datasheet and installation manual shipped with corresponding products, or visit the OPNET website for more information (www.opnet.com). Tables 2.1–2.3 list the system support and hardware and software requirements for using OPNET Modeler.

2.2 Installation on Windows

On Windows, you need to install OPNET Modeler and Microsoft Visual Studio or Microsoft Visual C++ for OPNET Modeler to compile C/C++ based simulation models. The order of installing Visual Studio and OPNET Modeler is irrelevant. However, after finishing installing both of them, you need to check the relevant environment variables to make sure they are correctly configured. If not, you can go through the following steps to make sure everything is installed and configured correctly and ready for modeling and simulation.

Table 2.1 Supported operating systems and processors

Operating system	Processor
Windows 2000 Professional and Server Windows Server 2003 (32-bit and 64-bit) Windows XP Professional (32-bit and 64-bit) Windows Vista Business (32-bit and 64-bit) Red Hat Enterprise Linux 3 and 4 Fedora Linux 3 and 4	x86 or EM64T (Intel Pentium III, 4, Xeon, or compatible), 1.5 GHz or better x86 AMD or AMD64, 1.5 GHz or better

For Windows XP Professional and Windows Vista Business, at least Service Pack 1 is required for OPNET Modeler to work correctly. Since simulation is a computation-intensive process, powerful processors can generally help accelerate the simulation speed.

Table 2.2 General hardware requirements

RAM	Disk space	Display
512 MB is minimum RAM requirement; 1–2 GB RAM recommended	Up to 3–5GB free disk space required for installation	Minimum resolution is 1024 × 768

For simulating complex models generating large amount of events, more RAM may be required. Running out of memory is quite common in simulation. For disk space, at least several GB free disk space is required for storing simulation files. Sometimes, single simulation scenarios can generate temporary files with several GB. The display resolution is required to allow a simulation graphic user interface (GUI) to be presented in an appropriate form.

Table 2.3 Other requirements

C/C++ compiler	Internet browser	Others
For Linux, gcc 3.4 or higher For Windows, Visual C++ 6.0 or higher	Internet Explorer 5.0 or higher Netscape 7.0 or higher Mozilla Firefox 1.06 or higher	TCP/IP networking protocol support

C/C++ compiler is required to build and debug OPNET models. An internet browser is used for viewing OPNET documentation. The browser should be configured to allow viewing pages with HTML frames and JavaScript. TCP/IP networking software is required to perform some network communication-related tasks like accessing a remote licensing server or serving license for remote clients, and communicating with real network devices. The TCP/IP networking software is generally shipped with the operating system as part of the networking protocol stack.

2.2.1 Installation of OPNET Modeler

The following steps demonstrate how to install OPNET Modeler on a Windows system.

- Log in to Windows as Administrator.