Chapter 1
Simulation of Computer Networks

People communicate. One way or another, they exchange some information among themselves all the times. In the past several decades, many electronic technologies have been invented to aid this process of exchanging information in an efficient and creative way. Among these are the creation of fixed telephone networks, the broadcasting of television and radio, the advent of computers, the rise of the Internet, and the emergence of wireless sensation. Originally, these technologies existed and operated independently, serving their very own purposes. Not until recently that these technological wonders have started to converge, and it is a well-known fact that a computer communication network is a result of this convergence.

This chapter presents an overview of computer communication networks and the basics of simulation of such a network. Section 1.1 introduces a computer network along with the reference model which is used for describing the architecture of a computer communication network. A brief discussion on designing and modeling a complex system such as a computer network is then given in Sect. 1.2. In Sect. 1.3, the basics of computer network simulation are discussed. Section 1.4 presents one of the most common type of network simulation, namely, the time-dependent simulation. An example simulation is given in Sect. 1.5. Finally, Sect. 1.6 summarizes the chapter.

1.1 Computer Networks and the Layering Concept

A computer network is usually defined as a collection of computers interconnected for gathering, processing, and distributing information. *Computer* is used as a broad term here to include devices such as workstations, servers, routers, modems, base stations, and wireless extension points. These computers are connected by communication links such as copper cables, fiber optic cables, and microwave/satellite/radio links. A computer network can be built as a nesting and/or interconnection of several networks. The Internet is a good example of computer

T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*,
DOI 10.1007/978-1-4614-1406-3_1, © Springer Science+Business Media, LLC 2012

networks. In fact, it is a network of networks, within which tens of thousands of networks interconnect millions of computers worldwide.

1.1.1 Layering Concept

A computer network is a complex system. To facilitate design and flexible implementation of such a system, the concept of *layering* is introduced. Using a layered structure, the functionalities of a computer network can be organized as a stack of layers.

Logically, each layer communicates to its peer (a logical entity on the same layer) on the other communication node. However, the actual data transmission occurs through the lowest layer, namely, the physical layer. Therefore, data at the source node always move down the layers until reaching the physical layer. Then, it is transmitted via a physical link to a neighboring node or the destination node. At the destination node, the data are passed to the layers until reaching the corresponding peer.

Representing a well-defined and specific part of the system, each layer provides certain *services* to the above layer. When performing a task (e.g., transmit a packet), an upper layer asks its lower layer to do more specific job. Accessible (by the upper layers) through so-called interfaces, these services usually define *what* should be done in terms of network operations or primitives, but do not specifically define *how* such things are implemented. The details of how a service is implemented are defined in a so-called protocol.

A protocol is a set of rules that *multiple* peers comply with when communicating to each other.[1] As long as the peers abide to a protocol, the communication performance would be consistent and predictable. As an example, consider an error detection protocol. When a transmitter sends out a data packet, it may wait for an acknowledgment from the receiver. The receiver, on the other hand, may be responsible for acknowledging to the transmitter that the transmitted packets are received successfully.

The beauty of this layering concept is the layer independency. That is, a change in a protocol of a certain layer does not affect the rest of the system as long as the interfaces remain unchanged. Here, we highlight the words *services*, *protocol*, and *interface* to emphasize that it is the interaction among these components that makes up the layering concept.

Figure 1.1 graphically shows an overall view of the layering concept used for communication between two computer hosts: a source host and a destination host. In this figure, the functionality of each computer host is divided into four

[1] Unlike a protocol, an algorithm is a set of steps to get things done (either with or without communications).

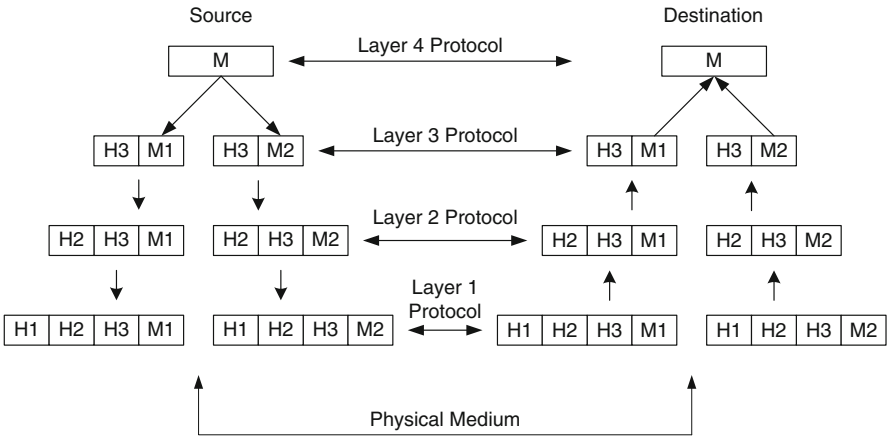

Fig. 1.1 Data flow in a layered network architecture

layers.[2] When logically linked with the same layer on another host, these layers are called *peers*.[3] Although not directly connected to each other, these peers logically communicate with one another using a protocol represented by an arrow. As mentioned earlier, the actual communication needs to propagate down the stack and use the above layering concept.

Suppose an application process running on Layer 4 of the source generates data or messages destined for the destination. The communication starts by passing a generated message `M` down to Layer 3, where the data are segmented into two chunks (`M1` and `M2`), and control information called *header* (`H3`) specific to Layer 3 is appended to `M1` and `M2`. The control information are, for example, sequence numbers, packet sizes, and error checking information. These information are understandable and used only by the peering layer on the destination to recover the data (`M`). The resulting data (e.g., `H3+M1`) are called a "protocol data unit (PDU)" and are handed to the next lower layer, where some protocol-specific control information is again added to the message. This process continues until the message reaches the lowest layer, where transmission of information is actually performed over a physical medium. Note that, along the line of these processes, it might be necessary to further segment the data from upper layers into smaller segments for various purposes. When the message reaches the destination, the reverse process takes place. That is, as the message is moving up the stack, its headers are ripped off layer by layer. If necessary, several messages are put together before being passed to the upper layer. The process continues until the original message (`M`) is recovered at Layer 4.

[2]For the sake of illustration only four layers are shown. In the real-world systems, the number of layers may vary, depending on the functionality and objectives of the networks.

[3]A peering host of a source and a destination are the destination and the source, respectively.

1.1.2 OSI and TCP/IP Reference Models

The Open Systems Interconnection (OSI) model was the first reference model developed by International Standards Organization (ISO) to provide a standard framework to describe the protocol stacks in a computer network. Its consists of seven layers, where each layer is intended to perform a well-defined function [1]. These are physical layer, data link layer, network layer, transport layer, session layer, presentation layer, and application layer. The OSI model only specifies what each layer should do; it does not specify the exact services and protocols to be used in each layer.

The Transmission Control Protocol (TCP)/Internet Protocol (IP) reference model [1], which is based on the two primary protocols, namely, TCP and IP, is used in the current Internet. These protocols have proven very powerful, and as a result have experienced widespread use and implementation in the existing computer networks. It was developed for ARPANET, a research network sponsored by the US Department of Defense, which is considered as the grandparent of all computer networks. In the TCP/IP model, the protocol stack consists of five layers – physical, data link, network, transport, and application – each of which is responsible for certain services as will be discussed shortly. Note that the application layer in the TCP/IP model can be considered as the combination of session, presentation, and application layers of the OSI model.

1.1.2.1 Application Layer

The application layer sits on top of the stack and uses services from the transport layer (discussed below). This layer supports several higher-level protocols such as Hypertext Transfer Protocol (HTTP) for World Wide Web applications, Simple Mail Transfer Protocol (SMTP) for electronic mail, TELNET for remote virtual terminal, Domain Name Service (DNS) for mapping comprehensible host names to their network addresses, and File Transfer Protocol (FTP) for file transfer.

1.1.2.2 Transport Layer

The objective of a transport layer is to perform flow control and error control for message transportation. Flow control ensures that the end-to-end transmission speed is neither too fast to make the network congested nor too slow to underutilize the network. Error control ensures that the packets are delivered to the destination properly.

Generally, when a transport protocol receives a message from the higher layer, it breaks down the message into smaller pieces. Then it generates a PDU – called a *segment* – by attaching necessary error and flow control information, and passes the segment to the lower layer.

Two well-known transport protocols, namely, TCP and User Datagram Protocol (UDP), are defined in this layer. While TCP is responsible for a reliable and connection-oriented communication between two hosts, UDP supports an unreliable

connectionless transport. TCP is ideal for applications that prefer accuracy over prompt delivery and the reverse is true for UDP.

1.1.2.3 Network Layer

This layer determines the route through which a packet is delivered from a source node to a destination node. A PDU for the network layer is called a *packet*.

1.1.2.4 Link Layer

A link layer protocol has three main responsibilities. First, flow control regulates the transmission speed in a communication link. Second, error control ensures the integrity of data transmission. Third, flow multiplexing/demultiplexing combines multiple data flows into and extracts data flows from a communication link. Choices of link layer protocols may vary from host to host and network to network. Examples of widely used link layer protocols/technologies include Ethernet, Point-to-Point Protocol (PPP), IEEE 802.11 (i.e., Wi-Fi), and Asynchronous Transfer Mode (ATM).

Link layer protocols are different from transport layer protocol as follows. The former deals with a single communication link. On the other hand, the latter does the same job for an end-to-end flow which may traverse multiple links.

1.1.2.5 Physical Layer

The physical layer deals with the transmission of data bits across a communication link. Its primary goal is to ensure that the transmission parameters (e.g., transmission power) are set appropriately to achieve the required transmission performance (e.g., to achieve the target bit error rate performance).

Finally, we point out that the five layers discussed above are common to the OSI layer. As has been mentioned already, the OSI model contains two other layers sitting on top of the transport layer, namely, session and presentation layers. The session layer simply allows users on different computers to create communication sessions among themselves. The presentation layer basically takes care of different data presentations existing across the network. For example, a unified network management system gathers data with different format from different computers and converts their format into a uniform format.

1.2 System Modeling

System modeling is an act of formulating a simple representation for an actual system. It allows investigators to look closely into the system without having to actually implement it. During the investigation, various parameters can be applied

to study system behavior. After the system is well understood, investigators can decide whether the actual system should be implemented.

System modeling often requires simplification assumptions. These assumptions exclude irrelevant details of the actual system, hence making the model cleaner and easier to implement. However, excessive assumptions may lead to inaccurate representation of the system. Design engineers need to use their discretion to achieve the best modeling trade-off.

Traditionally, there are two modeling approaches: Analytical approach and simulation approach.

1.2.1 Analytical Approach

The general concept of analytical modeling approach is to come up with a way to describe the system mathematically, and apply numerical methods to gain insight from the developed mathematical model. Examples of widely used mathematical tools are queuing and probability theories. Since analytical results derive mainly from mathematical proofs, they are true as long as the underlying conditions hold. If properly used, analytical modeling can be a cost-effective way to provide a general view of the system.

1.2.2 Simulation Approach

Simulation recreates real-world scenarios using computer programs. It is used in various applications ranging from operations research, business analysis, manufacturing planning, and biological experimentation, just to name a few. Compared to analytical modeling, simulation usually requires fewer simplification assumptions, since almost every possible detail of system specifications can be incorporated in a simulation model. When the system is rather large and complex, a straightforward mathematical formulation may not be feasible. In this case, the simulation approach is usually preferred to the analytical approach. The essence of simulation is to perform extensive experiment and make convincing argument for generalization. Due to the generalization, simulation results are usually considered not as strong as the analytical results.

1.3 Basics of Computer Network Simulation

A simulation is, more or less, a combination of art and science. That is, while the expertise in computer programming and the applied mathematics accounts for the science part, the very skills in analysis and conceptual model formulation usually represent the art portion. A long list of steps in executing a simulation process, as given in [2], seems to reflect this popular claim.

A simulation of general computer networks consists of three main parts:

- *Part 1 – Planning*: This part includes defining the problem, designing the corresponding model, and devising a set of experiments for the formulated simulation model. It is recommended that 40% of time and effort be spent on planning.
- *Part 2 – Implementing*: Implementation of simulation programs consists of three steps:
  - *Step 1 – Initialization*: This step establishes initial conditions (e.g., resetting simulation clocks and variables) so that the simulation always starts from a known state.
  - *Step 2 – Result generation*: The simulation creates and executes events, and collects necessary data generated by the created events.
  - *Step 3 – Postsimulation processing*: The raw data collected from simulation are processed and translated into performance measures of interest.

  It is recommended that 20% of time should be used for implementation.
- *Part 3 – Testing*: This part includes verifying/validating the simulation model, experimenting on the scenarios defined in Part 1 and possibly fine-tuning the experiments themselves, and analyzing the results. The remaining 40% of time should be used in this part.

This formula is in no way a strict one. Any actual simulation may require more or less time and effort, depending on the context of interest, and definitely on the modeler himself/herself.

1.3.1 Simulation Components

A computer network simulation can be thought of as a flow of interaction among network entities (e.g., nodes, packets). These entities move through the system, interact with other entities, join activities, trigger events, cause some changes to state of the system, and terminate themselves. From time to time, they contend or wait for some type of resources. This implies that there must be a logical execution sequence to cause all these actions to happen in a comprehensible and manageable way.

According to Ingalls [4], the key components of a simulation include the following:

1.3.1.1 Entities

Entities are objects that interact with one another in a simulation program. They cause some changes to the states of the system. In the context of a computer network, entities may include computer nodes, packets, flows of packets, or nonphysical objects such as simulation clocks.

1.3.1.2 Resources

Resources are limited virtual assets shared by entities such as bandwidth or power budget.

1.3.1.3 Activities and Events

From time to time, entities engage in some activities. The engagement creates events and triggers changes in the system state. Common examples of events are packet reception and route update events.

1.3.1.4 Scheduler

A scheduler maintains a list of events and their execution time. During a simulation, it moves along a simulation clock and executes events in the list chronologically.

1.3.1.5 State and Global Variables

State variables keep track of the system state. They can be classified as local variables and global variables based on their scope of operation. Local variables are valid under a limited range, while global variables are understandable globally by all program entities.

In general, global state variables hold general information shared by several entities such as the total number of nodes, the geographical area information, the reference to the scheduler, and so on.

1.3.1.6 Random Number Generator

A Random Number Generator (RNG) introduces randomness in a simulation model. It generates random numbers by sequentially picking numbers from a deterministic sequence of pseudo-random numbers [5], yet the numbers picked from this sequence appear to be random.

Without randomness, the results for every run would be exactly the same. To generate a set of different results, we may initialize the RNG for different runs with different seeds. A *seed* identifies the first location where the RNG starts picking random numbers. Two simulations whose RNG picks different initial positions would generate different simulation results.

In a computer network simulation, for example, a packet arrival process and a service process are usually modeled as random processes. These random processes are usually implemented with the aid of an RNG. The readers are referred to [6, 7] for a comprehensive treatment on random process implementation.

1.3.1.7 Statistics Gatherer

Statistics gatherers use variables to collect relevant data (e.g., packet arrival and departure time). These data can later be used to compute the performance measures such as average and standard deviation of the queuing delay for data packets traversing a network.

1.3.2 Simulation Performance

Performance of a simulation is measured by the following metrics [3]:

- *Execution speed*: How fast a simulation can be completed
- *Cost*: Expense paid to procure software/hardware, develop simulation programs, and obtain simulation results. Generally, commercial tools have more features and easier to work with at the expense of increasing cost.
- *Fidelity*: How reliable the simulation results are. Fidelity can be increased by incorporating more details (i.e., making less assumptions) into the simulation.
- *Repeatability*: An assurance that if the experiment was to be repeated, the results would be the same. Repeatability can be quantified using *confidence interval* (see the details in Sect. 1.3.3).
- *Scalability*: The impact of the size of the problem (e.g., the number of node, the input traffic) on other simulation performance measures.

1.3.3 Confidence Interval

Confidence interval [6] is a useful mathematical tool which helps quantify the level of repeatability. A confidence interval is a range between which one has confidence in finding data points. It is characterized by the width and the confidence coefficient (i.e., probability) to find data points. Figure 1.2 shows an example of confidence interval of $[-1,1]$ with the confidence coefficient is 80%. The interpretation of this example is that the probability of finding a data point within an interval $[-1,1]$ is 0.8.

It is fairly impossible/impractical to have perfectly repeatable results. Confidence interval measures such imperfection. As long as the imperfection is well defined (e.g., by confidence interval) and reasonable, the simulation results are usually deemed sufficiently reliable.

1.3.4 Choices for Network Simulation Tools

There is a wide variety of network simulation tools in the market. Each has its own strengths and weaknesses. To choose the most appropriate one, the following factors might be considered [3]:

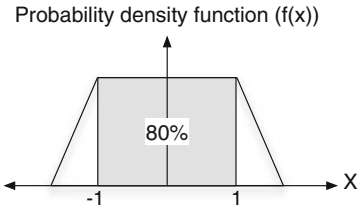

Fig. 1.2 A confidence interval of [−1,1] with confidence coefficient of 80%

1.3.4.1 Simulation Platform

The simulation platform can be software, hardware, or hybrid. Software simulation platforms can be very flexible and economical. In most cases, it can be installed in personal computers or servers. Therefore, it can be upgraded very easily. The hardware simulation platform (e.g., those using very high-level design language (VHDL)) can be very fast to run and is more suitable for computationally intensive simulation. It is also essential when input parameters need to be collected from surrounding environment. The major drawback of hardware simulation platform is that they can be prohibitively expensive to be implemented and modified. Hybrid simulation platforms combine the benefits of both the software and the hardware platforms. An example of hybrid simulation platforms is Hardware In the Loop (HIL) simulation, which is usually used to test complex real-time embedded systems [3].

1.3.4.2 Types of Simulation Tools

Simulation tools can also be classified based on how they are developed:

- *Open-source or closed-source*: By revealing its source code, open-source software opens itself for investigation. Users/programmers can find and report problems, modify the source codes, incorporate new features, and redistribute the software. The drawback of the open-source software is the lack of accountability. A lot of open-source software projects is run by volunteers. Since they can be modified by anyone at any time, they might behave differently from users' expectation. Closed-source software, on the other hand, can be modified only by the software developers. Therefore, these developers are fully accountable for the software quality.
- *Free or commercial*: Although free of charge, free software may lack the support and accountability. Commercial software, on the other hand, is usually well documented and has better technical support.
- *Publicly available or in-house*: Publicly available software can be open-source or close-source, and can be free or commercial. It can help save substantial effort

required to develop simulation software. When appropriately chosen, it can be fairly trustworthy, since well-developed software would have been extensively examined by the public. Developed internally, in-house software has greater flexibility. When the software needs to be changed or updated, the developers know what, where, and how to make the changes rapidly.

1.3.4.3 User Interface

The user interfaces of a simulation program can be Command Line Interface (CLI) or Graphic User Interface (GUI). Aiming at obtaining statistical results, a large number of academic works use CLI-based simulation tools, since these tools use most computational power for simulations. GUI-based simulation tools, on the other hand, allocate a part of computational power to improve user interface. They usually provide user-friendly network configuration interfaces, and have graphical and animation-based simulation result presentation.

1.3.4.4 Examples of Simulation Tools

The following are some of the widely used network simulation tools:

- `NS2`: An open-source software written in C++ and OTcl programming languages
- `GloMoSim`: An open-source software developed at University of California, Los Angeles (UCLA)
- `QualNet`: A commercialized version of `GloMoSim`. It supports a wider variety of protocols, has better documentation, and provides customizable simulation modules.
- `Opnet`: A commercial network simulation tool which offers several features – including HIL, parallel computing, detailed documentation, and technical support.
- `MATLAB`: A commercial multi-purpose software that can be used for network simulation and complex numerical evaluation.

1.4 Time-Dependent Simulation

As its name suggested, time-dependent simulation proceeds chronologically. It maintains a simulation clock to keep track of simulation time. Based on how events are handled, time-dependent simulation can be classified into two categories: time-driven simulation and event-driven simulation.

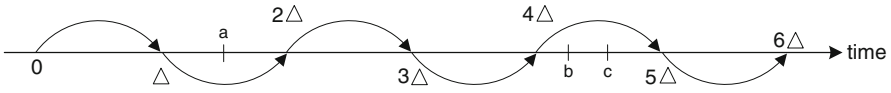

Fig. 1.3 Clock advancement in a time-driven simulation

1.4.1 Time-Driven Simulation

Time-driven simulation induces and executes events for every fixed time interval of Δ time units. In particular, it looks for events that may have occurred during this fixed interval. If found, such events would be executed as if they occurred at the end of this interval. After the execution, it advances the simulation clock by Δ time units and repeats the process. The simulation proceeds until the simulation time reaches a predefined termination time.

Figure 1.3 shows the basic idea behind time advancement in a time-driven simulation. The curved arrows represent such advances, and `a`, `b`, and `c` mark the occurrences of events. During the first interval, no event occurs. The second interval contains event `a`, which is not handled until the end of the interval.

Time interval (Δ) is an important parameter of time-driven simulation. While a large interval can lead to loss of information, a small interval can cause unnecessary waste of computational effort. Suppose, in Fig. 1.3, events `b` and `c` in the fifth interval are packet arrival and departure events, respectively. Since these two events are considered to occur exactly at the end of the interval (i.e., at 5Δ), the system state would be as if there is no packet arrival or departure events during $[4\Delta, 5\Delta]$. This is considered a loss of information. An example of waste of computational effort occurs between 2Δ and 4Δ. Although no event occurs in this interval, the simulation wastes the computational resource to stop and process events at 3Δ and 4Δ. In time-driven simulation, programmers need to use their discretion to optimize the time interval value Δ.

Example 1.1. Program 1.1 shows the pseudo codes for a time-driven simulation. Lines 1 and 2 initialize the system state variables and the simulation clock, respectively. Line 3 specifies the stopping criterion. Lines 4–6 are run as long as the simulation clock (i.e., `sim_clock`) is less than the predefined threshold `stopTime`. These lines execute events, collect statistics, and advance the simulation. □

Program 1.1 Skeleton of a time-driven simulation

```
1 initialize {system states}
2 sim_clock := startTime;
3 while {sim_clock < stopTime}
4     collect statistics from current state;
5     execute all events that occurred during
                          [sim_clock, sim_clock + step];
6     sim_clock := sim_clock + step;
7 end while
```

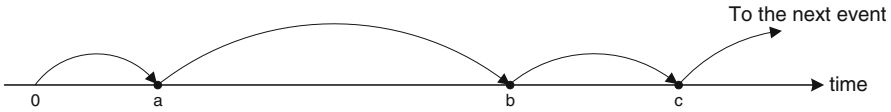

Fig. 1.4 Clock advancement in an event-driven simulation

Program 1.2 Skeleton of an event-driven simulation

```
1 initialize {system states}
2 initialize {event list}
3 while {Event list != NULL}
4     retrieve and remove an event
          whose timestamp is smallest from the event list
5     execute the retrieved event
6     set sim_clock := time corresponding to the retrieved
          event
7 end while
```

1.4.2 Event-Driven Simulation

An event-driven simulation does not proceed according to fixed time interval. Rather, it induces and executes events at any arbitrary time. Event-driven simulation has four important characteristics:

- Every event is stamped with its occurrence time and is stored in a so-called event list.
- Simulation proceeds by retrieving and removing an event with the smallest timestamp from the event list, executing it, and advancing the simulation clock to the timestamp associated with the retrieved event.
- At the execution, an event may induce one or more events. The induced events are stamped with the time when the event occurs and again are stored in the event list. The timestamp of the induced events must not be less than the simulation clock. This is to ensure that the simulation would never go backward in time.
- An event-driven simulation starts with a set of initial events in the event list. It runs until the list is empty or another stopping criterion is satisfied.

Figure 1.4 shows a graphical representation of event-driven simulation. Here, events `a`, `b`, and `c` are executed in order. The time gap between any pair of events is not fixed. The simulation advances from one event to another, as opposed to one interval to another in time-driven simulation. In event-driven simulation, programmers do not have to worry about optimizing time interval.

Example 1.2. Program 1.2 shows the skeleton of a typical event-driven simulation program. Lines 1 and 2 initialize the system state variables and the event list, respectively. Line 3 specifies a stopping criterion. Lines 4–6 are executed as along

as Line 3 returns `true`. Within this loop, the event whose timestamp is smallest is retrieved, executed, and removed from the list. Then, the simulation clock is set to the time associated with the retrieved event. □

1.5 A Simulation Example: A Single-Channel Queuing System

As an example, this section demonstrates a simulation of a single-channel queuing system shown in Fig. 1.5. Here, we have one communication link connecting Node A to Node B. Applications at Node A create packets according to underlying distributions for inter-arrival time and service time. After the creation, the packets are placed into a transmission buffer. When the communication link is free, the head of the line packet is transmitted to Node B, and the head of the line server fetch another packet from the buffer in a First-In-First-Out (FIFO) manner.

We now define the components of the event-driven simulation based on the framework discussed in Sect. 1.3.

1.5.1 Entities

The primary entities in this simulation include the following:

- *Applications* which generate traffic whose inter-arrival time and service time follow certain distributions,
- *A server* which stores the packet being transmitted (its state can be either `idle` or `busy`),

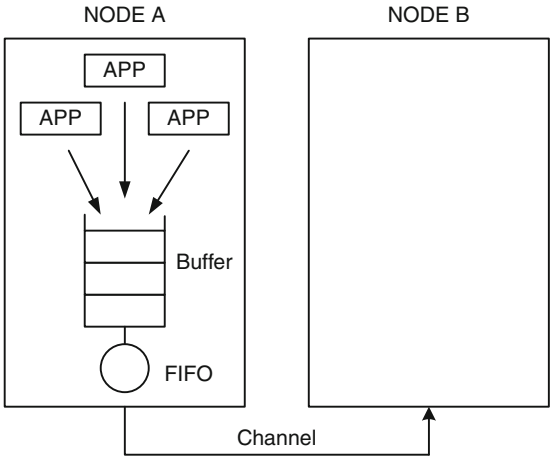

Fig. 1.5 Illustration of a single-channel queuing system

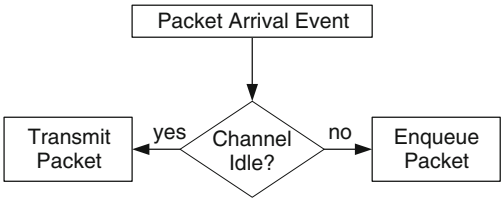

Fig. 1.6 Packet arrival event

- *A queue* which stores packets waiting to be transmitted (its state consists of the size and the current occupancy), and
- *Communication link* which carries packets from Node A to Node B.

1.5.2 State Global Variables

For simplicity, we make all variables global so that they can be accessed from anywhere in the simulation program:

- `num_pkts`: The number of packets in the systems – one in the head-of-the-line server plus all packets in the buffer.
- `link_status`: The current status of the communication link (its state can be either `idle` or `busy`).

1.5.3 Resource

Obviously, the only resource in this example is the transmission time in the channel.

1.5.4 Events

Main events in this simulation include the following:

1. `pkt_arrival` corresponds to a *packet arrival* event. This event occurs when an application generate a packet. As shown in Fig. 1.6, the packet may either be immediately transmitted or stored in the queue, depending on whether the channel is busy or idle.
2. `srv_complete` corresponds to a *successful packet transmission* event. This event indicates that a packet has been received successfully by node B. At the completion, node A begins to transmit (serve) another packet waiting in the queue. If there is no more packet to be sent, the channel becomes idle. The flow diagram of the process is shown in Fig. 1.7.

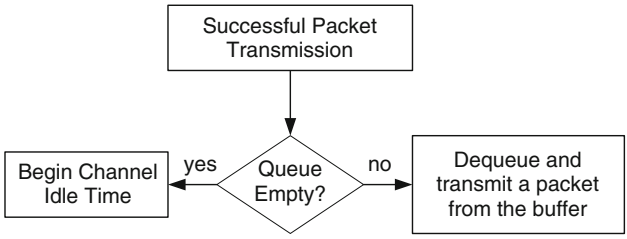

Fig. 1.7 Successful packet transmission (service completion) event

1.5.5 Simulation Performance Measures and Statistics Gatherers

Here, we consider the two following performance measures:

- *Average packet transmission latency* is the average time that a packet spends (from its arrival to its departure) in the system.
- *Average server utilization* is the percentage time where the server is busy.

It is important to note that all the above measures are the average values taken over time. The simulation time should be sufficiently long to ensure statistical accuracy of the simulation result.

In order to compute the above two performance measures, arrival time and service times of all the packets must be gathered. The computation of other performance measures from these two data will be shown later in this section.

1.5.6 Simulation Program

Program 1.3 shows the skeleton of a program which simulates the single-channel queuing system described above. It proceeds according to the three-step simulation implementation defined in Sect. 1.3:

Step 1 – Initialization (Lines 1–4): Lines 1 and 2 initialize the status of the communication link (`link_status`) to `idle` and number of packets in the systems (`num_pkts`) to zero. Line 3 sets the simulation clock to start at zero. Line 4 creates an event list (`event_list`) by invoking the procedure `create_list()`. The event list contains all events in the simulation. Again, the scheduler moves along this list and executes the events chronologically. From within the procedure `create_list()`, the initial packet arrival events created by applications are placed on the event list.

Step 2 – Result generation (Lines 5–10): This is the main part of the program where the loop runs as long as the two following conditions satisfied: (1) the event list is nonempty and (2) the simulation clock has not reached a predefined threshold.

Program 1.3 Simulation skeleton of a single-channel queuing system

```
% Initialize system states
1  link_status = idle; %The initial link status is idle
2  num_pkts = 0;       %Number of packets in system
3  sim_clock = 0;      %Current time of simulation

   %Generate packets and schedule their arrivals
4  event_list = create_list();

   % Main loop
5  while {event_list != empty} & {sim_clock < stop_time}
6    if the application creates events, insert them to the list
7    expunge the previous event from event list;
8    set sim_clock = time of current event;
9    execute the current event;
10 end while

   %Define events
11 pkt_arrival(){
12     if(link_status)
13      link_status = busy;
14      num_pkts = num_pkts + 1;
        % Update "event_list": Put "successful packet tx event"
        % into "event_list," T is random service time.
15      schedule event "srv_complete" at sim_clock + T;
16     else
17         num_queue = num_queue + 1; %Place packet in queue
18     num_pkts = num_queue + 1;
19 }

20 srv_complete(){
21     num_pkts = num_pkts - 1;
22     if(num_pkts > 0)
23         schedule event "srv_complete" at sim_clock + T;
24     else
25         link_status = idle;
26         num_pkts = 0;
27 }
```

Within the loop, Line 6 takes arrival/departure events (if any) created by applications and placed them in the event list. Lines 7–10 execute the next event on the event list by invoking either the procedure `pkt_arrival()` in Lines 11–19 or the procedure `srv_complete()` in Lines 20–27.

The procedure `pkt_arrival()` (Lines 11–19) checks whether the communication link is idle when a packet arrives. If so, the link is set to *busy*, and a *service completion* event is inserted into the `event_list` for future execution. The timestamp associated with the event is equal to the current clock time (`sim_clock`) plus the packet's randomly generated service time (`T`). If the link is

Table 1.1 Probability mass functions of inter-arrival time and service time

Time unit	Inter-arrival (probability mass)	Service (probability mass)
1	0.2	0.5
2	0.2	0.3
3	0.2	0.1
4	0.2	0.05
5	0.1	0.05
6	0.05	
7	0.05	

Table 1.2 Simulation result of a single-channel queuing system

Packet	Interarr. time	Service time	Arrival time	Service starts	Packet waiting time	Packet transmission latency
1	–	5	0	0	0	5
2	2	4	2	5	3	7
3	4	1	6	9	3	4
4	1	1	7	10	3	4
5	6	3	13	13	0	3
6	7	1	20	20	0	1
7	2	1	22	22	0	1
8	1	4	23	23	0	4
9	3	3	26	27	1	4
10	5	2	31	31	0	2
					1.0	3.5

busy, on the other hand, the packet will be enqueued into the buffer, and the packet counter (`num_pkts`) is incremented by one unit.

The procedure `srv_complete()` (Lines 20–27) first updates the number of packets in the system (`num_pkts`). Then, it checks whether the system contains any packet. If so, the head-of-the-line packet will be served. This is done by inserting another *service completion* event at time `sim_clock + T`. However, if the queue is empty, the channel is set to `idle` and the number of packets in the system is set to zero.

Step 3 – Postsimulation processing: This step collects and computes the performance measures based on the simulation results. Suppose that the inter-arrival time and the service time comply with the probability mass functions specified in Table 1.1. Table 1.2 shows the simulation results for ten packets.

In Table 1.2, the second and third columns represent the inter-arrival time and service time, respectively, of each packet. These two columns contain raw information. Data shown in other columns derive from these two columns.

The fourth and fifth columns specify the time where the packets arrive and start to be served in the head-of-the-line server, respectively. The sixth column represents the packet waiting time – the time that a packet spends in the queue. It is computed as the time difference between when the service starts and when the packet arrives. Finally, the seventh column represents the packet transmission latency – the time

Table 1.3 Evolution of number of packets in the system over time

Event	Packet no.	Simulation clock
Arrival	1	0
Arrival	2	2
Completion	1	5
Arrival	3	6
Arrival	4	7
Completion	2	9
Completion	3	10
Completion	4	11
Arrival	5	13
Completion	5	16

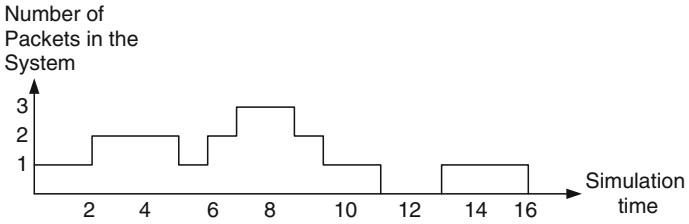

Fig. 1.8 Number of packets in the system at various instances

that a packet spends in both the queue and the channel. It is computed as the summation of the waiting time and the service time.

Based on the results in Table 1.2, we compute the average waiting time and the average packet transmission latency by averaging the sixth and seventh columns (i.e., adding all the values and dividing the result by 10). The results are therefore 1.0 and 3.5 time units, respectively.

Based on the information in Table 1.2, we also show a series of events and the dynamics of buffer occupancy with respect to the *Simulation Clock* (`sim_clock`) in Table 1.3 and Fig. 1.8, respectively. Based on Fig 1.8, the mean server utilization can be computed from the ratio of the time when the server is in use to the simulation time, which is $14/16 = 0.875$ in this case.

1.6 Chapter Summary

A computer network is a complex system. Design analysis, and optimization of a computer network can be a comprehensive task. Simulation, regarded as one of the most powerful performance analysis tools, is usually used in carrying out this task to complement the analytical tools.

This chapter focuses mainly on time-dependent simulation, which advances in a time domain. The time-dependent simulation can be classified into two categories. Time-driven simulation advances the simulation by fixed time intervals, while event-driven simulation proceeds from one event to another. NS2 is an event-driven simulation tool. Designing event-driven simulation models using NS2 is the theme of the rest of the book.

1.7 Exercises

1. What are the differences between OSI model and TCP/IP model. Draw a diagram to emphasize the differences.
2. What are the key steps in simulating a computer communication network?
3. Draw a probability density function with confidence interval $[-7, +7]$ and confidence coefficient is 95%.
4. You are given a text file. Each line of this text file contains a number representing a data point. Write a program which computes the average value and the standard deviation for the data points along with the confidence level when a confidence interval is given as an input parameter.
5. What are the two types of time-dependent simulations? Write down their main features, strengths, and weaknesses.
6. Write a sub-routine which prints out the current time slot. In a time-slotted system, write a program which executed the sub-routine at time slot 1, 7, 13, 24, and 47 by

   a. Using time-driven simulation
   b. Using event-driven simulation