Module-10:

Simulation of Computer Systems- Simulation Tools: Process Orientation and Event Orientation. Model Input: Modulated Poisson Process and Virtual-Memory Referencing. High-Level Simulation. CPU and Memory Simulations. Simulation of Computer Networks- Traffic Modeling, Media Access Control: Token- Passing Protocols and Ethernet, Data Link Layer, TCP, Model Construction.

Computer simulation is the process of mathematical modelling, performed on a computer, which is designed to predict the behaviour of or the outcome of a real-world or physical system, A simulation uses a mathematical description, or model, of a real system in the form of a computer program. This model is composed of equations that duplicate the functional relationships within the real system. When the program is run, the resulting mathematical dynamics form an analog of the behavior of the real system, with the results presented in the form of data. A simulation can also take the form of a computer-graphics image that represents dynamic processes in an animated sequence.

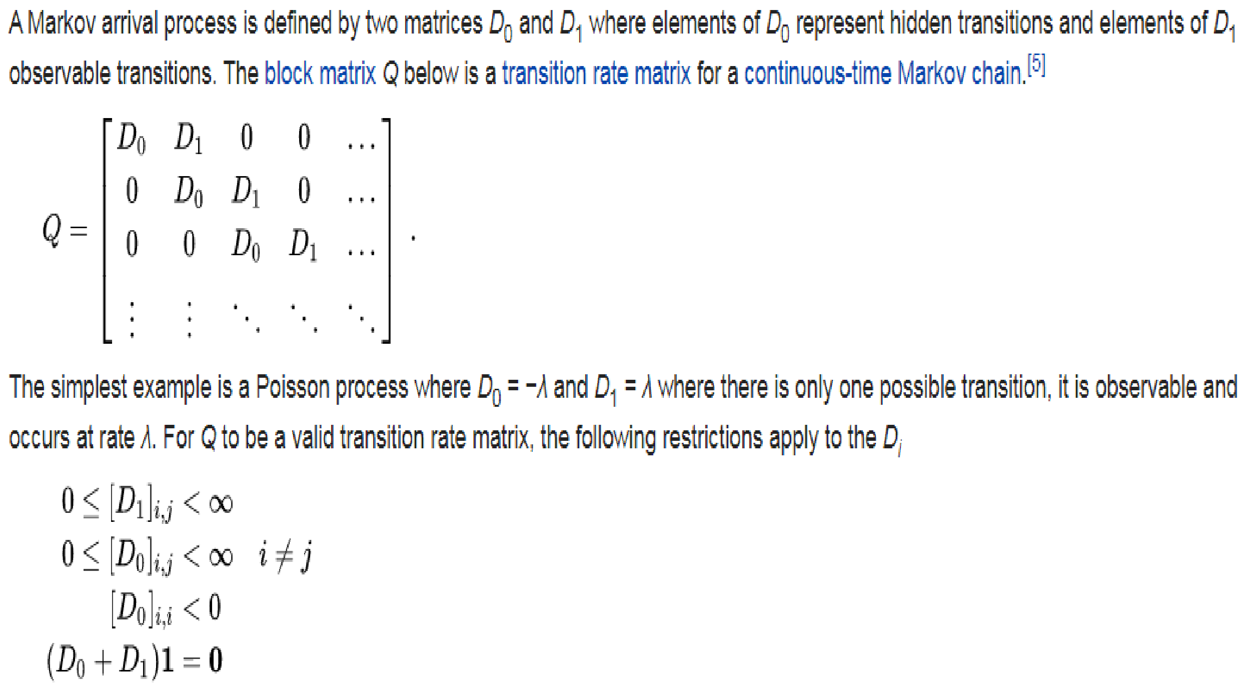
A **process oriented** type person is that emphasizes or focuses on processes, systems, or procedures rather than results or underlying causes. The Process Oriented simulation model is like a recurring thread of execution that expires time during its execution. The simulation modeler concentrates on the life cycle of entitities within a system.

Event oriented thinking or a person sees the world as a complex succession of events rather than as a system as a whole. An event is behavior that happened or will happen. The Event Oriented simulation model is one that is essentially an event driven finite state machine. The simulation modeler concentrates on events and how they affect the state of the simulated system.

The Event Oriented simulation model incurs very little overhead above the Future Event List (FEL). The Process Oriented simulation model requires a bit of overhead above the FEL and this is one of its cons. This overhead is to enable the CPU of the computer used for the simulation to perform context switching between processes.

Input modeling is the practice of selecting probability distributions (i.e., **input models**) to represent such random **input** processes. ... When data are available, we fit a probability distribution to the data. Otherwise, all information available is to be used for constructing an **input model**.

A process, belonging to the class of markov renewal processes, where arrivals occur according to a state dependent poisson process with different rates governed by a continuous-time markov chain. A Markovian arrival process is a mathematical model for the time between job arrivals to a system. The simplest such process is a Poisson process where the time between each arrival is exponentially distributed.



A Markov arrival process is defined by two matrices D_0 and D_1 where elements of D_0 represent hidden transitions and elements of D_1 observable transitions. The block matrix Q below is a transition rate matrix for a continuous-time Markov chain.[5]

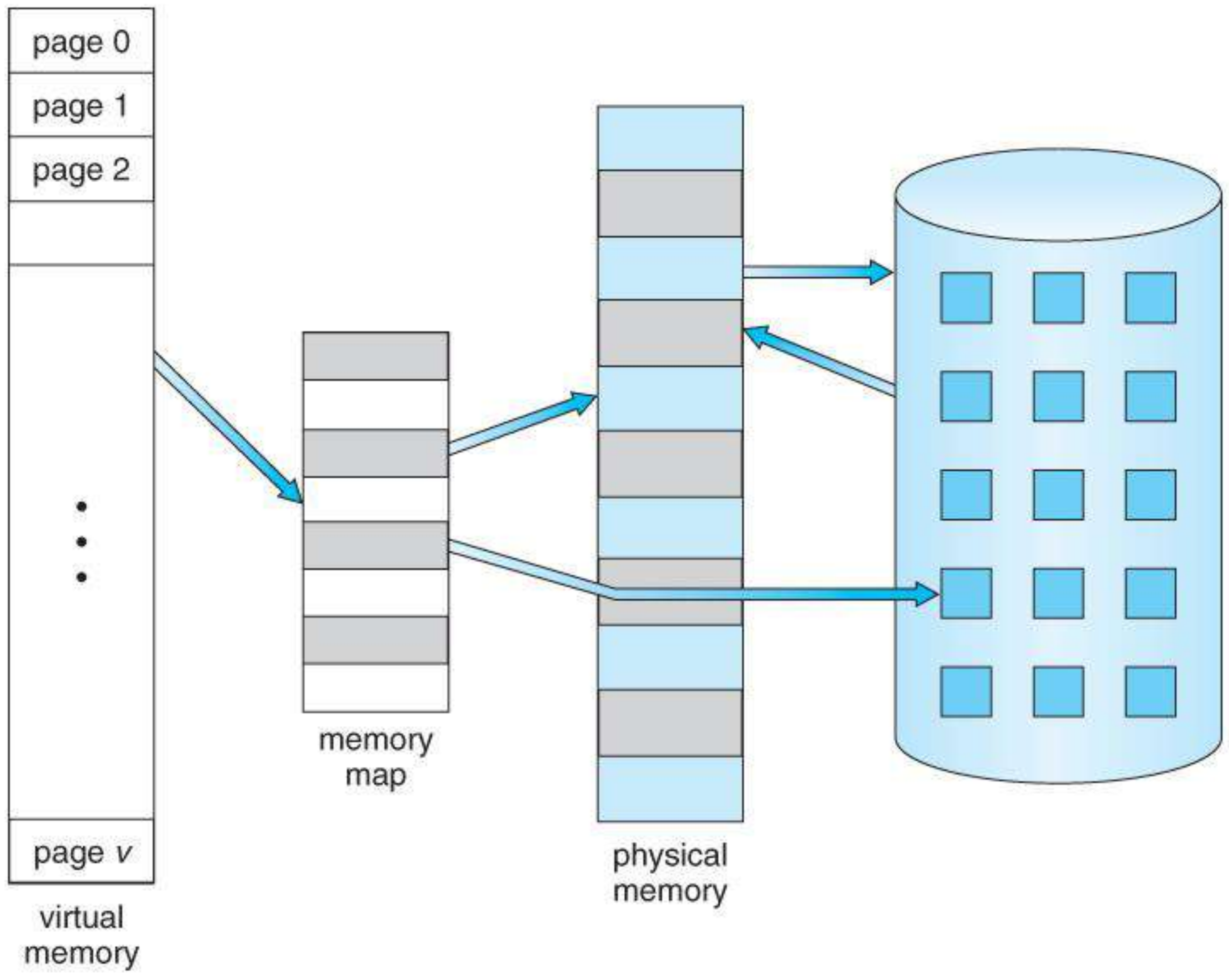
$$Q = \begin{bmatrix} D_0 & D_1 & 0 & 0 & \dots \\ 0 & D_0 & D_1 & 0 & \dots \\ 0 & 0 & D_0 & D_1 & \dots \\ \vdots & \vdots & \ddots & \ddots & \ddots \end{bmatrix}.$$

The simplest example is a Poisson process where $D_0 = -\lambda$ and $D_1 = \lambda$ where there is only one possible transition, it is observable and occurs at rate λ. For Q to be a valid transition rate matrix, the following restrictions apply to the D_i

$$\begin{aligned} 0 \leq [D_1]_{i,j} &< \infty \\ 0 \leq [D_0]_{i,j} &< \infty \quad i \neq j \\ [D_0]_{i,i} &< 0 \\ (D_0 + D_1)\mathbf{1} &= \mathbf{0} \end{aligned}$$

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.



Virtual memory allows a process of P pages to run in F frames, even if F < P. This is achieved by use of a page table, which records which pages are in RAM in which frames, and a page fault mechanism by which the memory management unit (MMU) can ask the operating system (OS) to bring in a page from disk. The page table must be accessible by both the MMU and the OS, and an IPC facility is needed for communication between the MMU and OS. Given the MMU simulator below, write a program that does the work of the OS to maintain a process' use of RAM and the page table. The page table is held in shared memory, and signals are used for IPC. You do not need to simulate the RAM, only the page table maintenance. However, to make things realistic, your OS process must sleep(1) whenever a disk access (write out page to disk, read in page from disk) would be necessary in a real implementation.

The Page Table

The page table has four fields in each page table entry:

- A Boolean Valid indicating if the page of that index is in RAM.
- An integer Frame giving the frame number of the page in RAM.
- A Boolean Dirty indicating if the page has been written to.
- An integer Requested which is non-zero only if that page is not in RAM and has been requested by the MMU. In this case it's value is the PID of the MMU.

The OS process must create the page table in shared memory, and initialize it to indicate that no pages are loaded (all Valid fields set to 0). You may need to add more fields for your OS, with corresponding changes in the MMU.

The MMU

This memory management unit simulator takes several arguments:

- The number of pages in the process.
- A reference string of memory accesses, each of the form <mode><page>, e.g., W3 is a write to page 3.
- The PID of the OS process.

The MMU attaches to the shared memory (using the OS PID on the command line as the key), then runs through the reference string. For each memory access, the MMU:

1. Checks if the page is in RAM.
2. If not in RAM, writes its PID into the Requested field for that page.
3. Simluates a page fault by signalling the OS with a SIGUSR1.
4. Blocks until it receives a SIGCONT signal from the OS to indicate that the page has been loaded (well, as mentioned above, the load is not done in this project, just simulated by sleep(1) delays).
5. If the access is a write access, sets the Dirty bit.
6. Prints the updated page table.

When all memory accesses have been processed, the MMU detaches from the shared memory and signals the OS one last time, but without placing its PID in any Requested field. That must be detected by the OS, at which point it can destroy the shared memory and exit.

The OS

The OS simulator must take two arguments:

- The number of pages in the process.
- The number of frames allocated to the process.

For simplicity, assume that the pages and frames are numbered 0, 1, 2, ...

After creating and initializing the page table in the shared memory, the OS must sit in a loop waiting for a SIGUSR1 signal from the MMU. When it receives a signal, it must:

1. Scan through the page table looking for a non-zero value in the Requested field.
2. If a non-zero value is found, it's the PID of the MMU, and indicates that the MMU wants the page at that index loaded.
3. If there is a free frame allocate the next one to the page.
4. If there are no free frames, choose a victim page.
    - If the victim page is dirty, simulate writing the page to disk by sleep(1) and increment the counter of disk accesses.
    - Update the page table to indicate that the victim page is no longer Valid.
5. Simulate the page load by sleep(1) and increment the counter of disk accesses.
6. Update the page table to indicate that the page is Valid, in the allocated Frame, not Dirty, and clear the Requested field.
7. Print the updated page table.
8. Send a SIGCONT signal to the MMU to indicate that the page is now loaded.
9. If no non-zero Requested field was found, the OS exits the loop.

You can use whatever algorithm you want for choosing a victim page. You may need to add extra fields to the page table, and make *minor* modifications to the MMU code (ask me if you are unsure if what you are doing is *minor*). Before terminating the OS must print out the total number of disk accesses, and destroy the shared memory.

High-Level Simulation

It is a standard for distributed simulation, used when building a simulation for a larger purpose by combining (federating) several simulations. The purpose of HLA is to enable interoperability and reuse. Key properties of HLA are:

- The ability to connect simulations running on different computers, locally or widely distributed, independent of their operating system and implementation language, into one Federation.
- Ability to specify and use information exchange data models, Federation Object Models (FOMs), for different application domains.
- Services for exchanging information using a publish-subscribe mechanism, based on the FOM, and with additional filtering options.
- Services for coordinating logical (simulation) time and time-stamped data exchange.
- Management services for inspecting and adjusting the state of a Federation.

HLA is composed of three (3) parts: [2]

1. **Rules:** that simulations must obey in order to be compliant to the standard. Federates and Federations which define relationships among federating compliant simulations.

1. Federate: an HLA compliant simulation entity.

2. Federation: multiple simulation entities connected via the RTI using a common OMT.

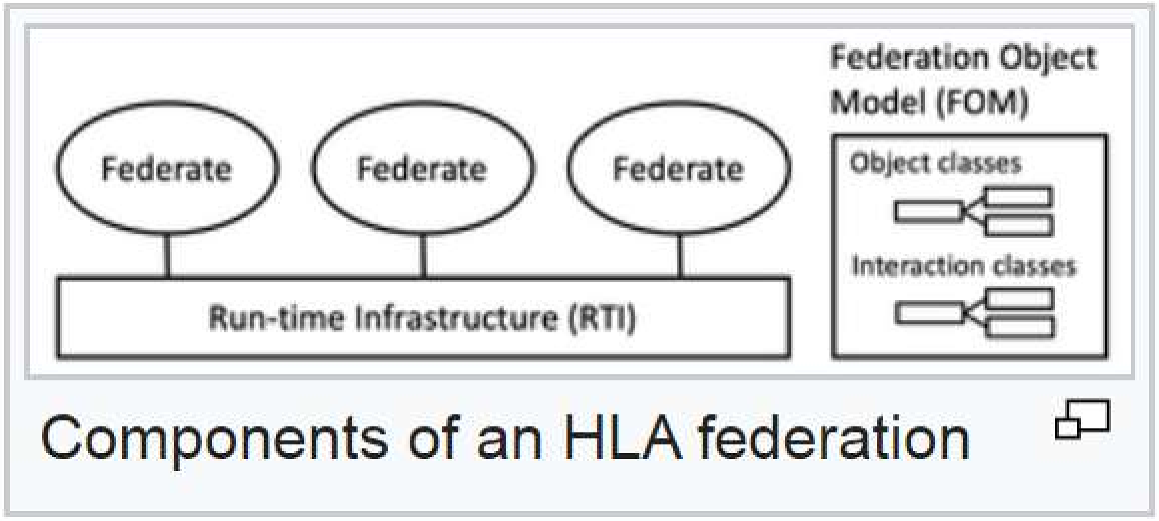
Interface Specification: Describes the way compliant simulations interact during operation and with the Run-Time Infrastructure (RTI). The RTI provides a programming library and an Application Programming Interface (API) (API) compliant to the interface specification.

Object Model Template: Specifies the form in which simulation elements are described

0. Federation Object Model (FOM). The FOM describes the shared object, attributes and interactions for the whole federation.

1. Simulation Object Model (SOM). A SOM describes the shared object, attributes and interactions used for a single federate.

The architecture specifies the following components.


Federation Object Model (FOM)
Federate
Federate
Federate
Object classes
Interaction classes
Run-time Infrastructure (RTI)
Components of an HLA federation

Components of an HLA federation

- A **Run-time Infrastructure** (RTI) that provides a standardized set of services through different programming languages. These services include information exchange, synchronization and federation management
- **Federates** that are individual simulation systems using RTI services.
- A **Federation Object Model** (FOM) that specifies the Object Classes and Interaction Classes used to exchange data. The FOM can describe information for any domain.

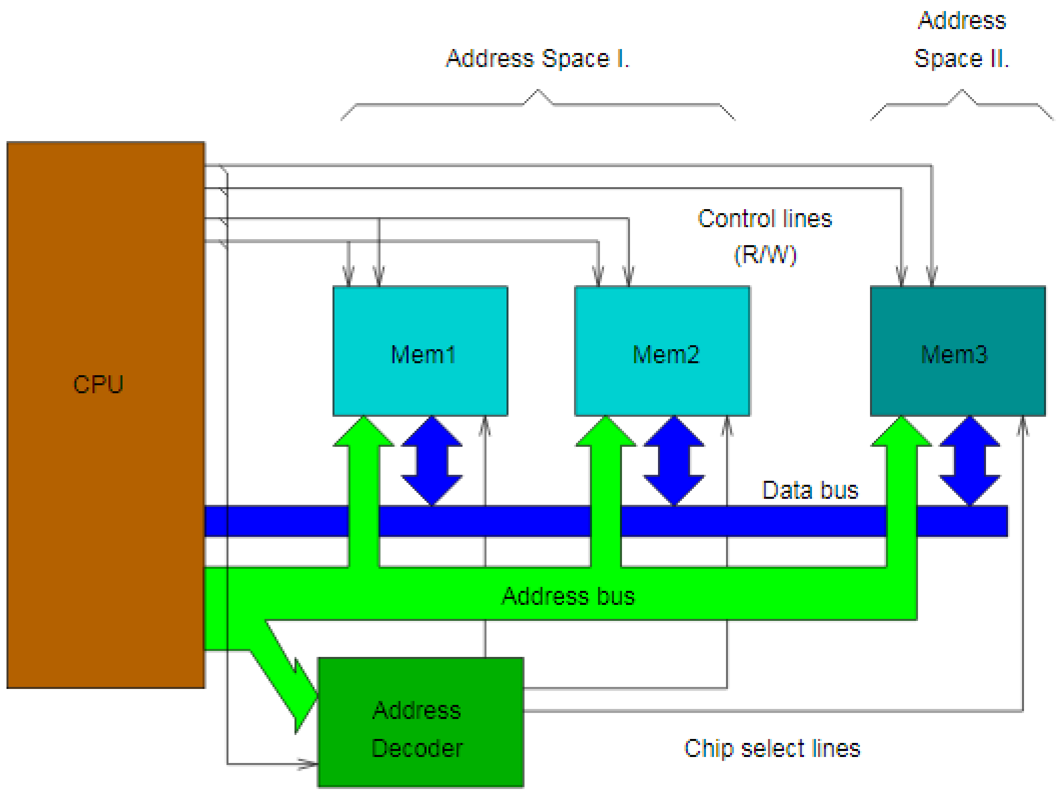
Together the above components form a **Federation**.

The HLA standard consists of three parts:

1. **IEEE Std 1516-2010 Framework and Rules**, which specifies ten architectural rules that the components or the entire federation shall adhere to.
2. **IEEE Std 1516.1-2010 Federate Interface Specification**, which specifies the services that shall be provided by the RTI. The services are provided as C++ and Java APIs as well as Web Services.
3. **IEEE Std 1516.2-2010 Object Model Template Specification** , which specifies the format that HLA object models, such as the FOM, shall use.

CPU and Memory Simulations

The CPU and Memory will be simulated by separate processes that communicate



Address space

In our terminology "**address space**" means a set of:

- address bus lines
- data bus lines
- control (read/write) lines

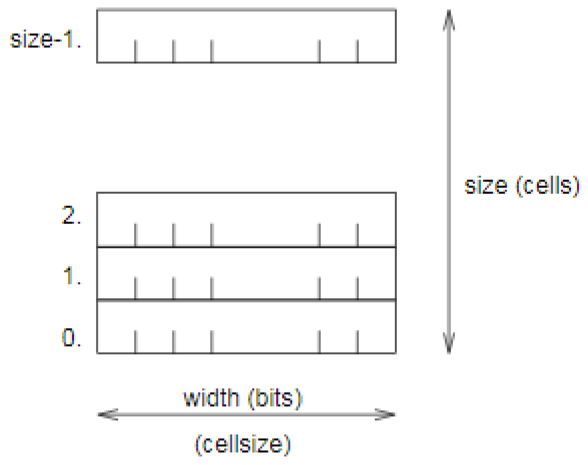
Bus lines can be shared by address spaces, in this case number of control line sets specifies the address spaces.

Microcontrollers usually have more address spaces. Some is used inside only, some can be used for external memories.

Address space does not store any value. It just specifies range of addresses by *start address* (which is not necessarily zero) and *size* which the CPU can provide when it tries to access a memory location.

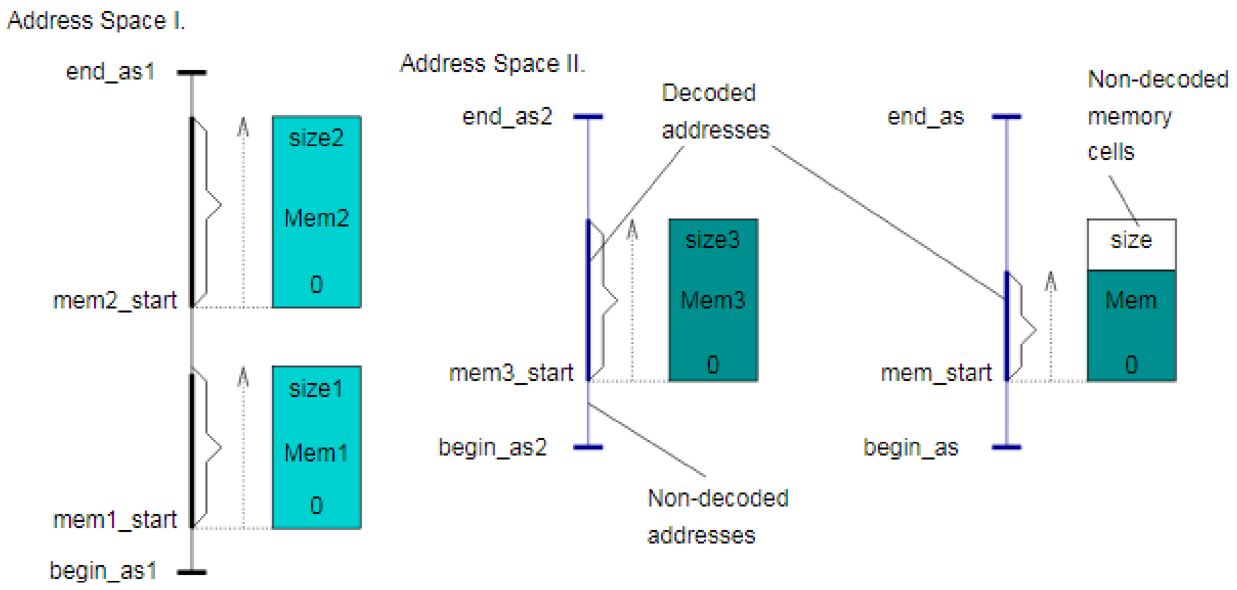
Memory chip

"**Memory chip**" is a circuit which can hold values in cells. Cells are indexed from 0 up to size-1. Each cell stores some (usually 8) bits.



Address decoder

Addresses coming from an address space via address bus must be routed to memory and translated to cell indexes. This is done by the "**address decoder**". It listens addresses on the bus and control lines and enables exactly one memory chip. This way cells of the memory chip appear in the address space.

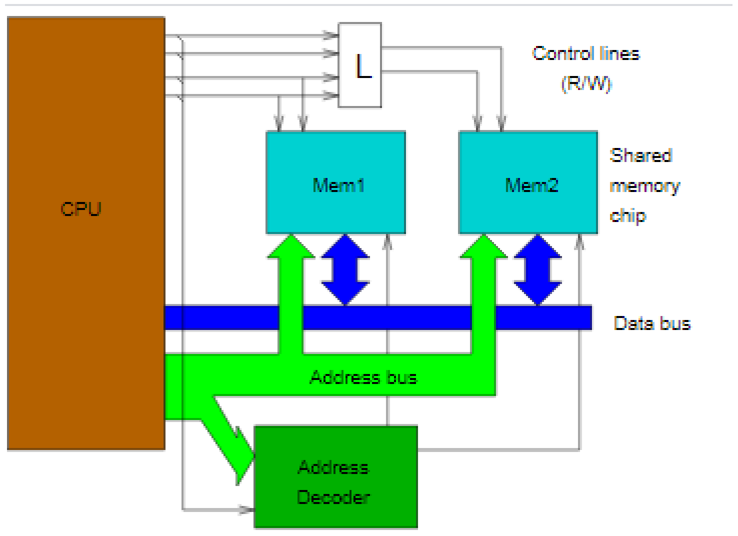


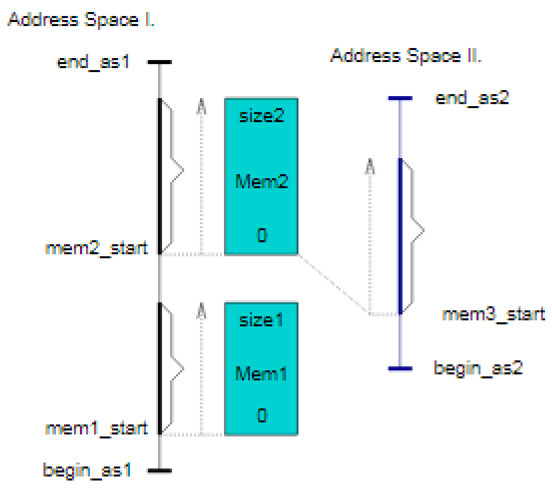
It is possible that some addresses are not decoded. Writing to such an address results data to be lost. Reading of a non-decoded address results random value. Pullup or pulldown resistors can be applied to data bus lines to provide a specific value for read operations.

As it shown above, it is also possible that some cells of a chip is not mapped to any address.

Mixed address spaces (shared chips)

Decoder connects an address of an address space to a cell of a memory chip. Only one cell can be connected to each addresses but same cell can be connected to more than one address spaces. This is done by routing different control lines to the same chip through logic **L**:





Because each cell of a memory chip uses same read/write control, in real world it is not possible to share individual cells of a chip between address spaces. In µCsim simulator we can define as many address decoders as we want so it is possible to map any cell to any address.

Simulation of Computer Networks

It is a technique whereby a software program models the behavior of a network by calculating the interaction between the different network entities (routers, switches, nodes, access points, links etc.). Most simulators use discrete event simulation - the modeling of systems in which state variables change at discrete points in time. The behavior of the network and the various applications and services it supports can then be observed in a test lab; various attributes of the environment can also be modified in a controlled manner to assess how the network / protocols would behave under different conditions.

A **network simulator** is software that predicts the behavior of a computer network. Since communication networks have become too complex for traditional analytical methods to provide an accurate understanding of system behavior, network simulators are used. In simulators, the computer network is modeled with devices,links, applications etc. and the network performance is reported. Simulators come with support for the most popular technologies and networks in use today such as 5G, Internet of Things (IoT), Wireless LANs, mobile ad hoc networks, wireless sensor networks, vehicular ad hoc networks, cognitive radio networks, LTE etc.

Uses of network simulators

Network simulators provide a cost-effective method for

- Network design validation for enterprises / data centers / sensor networks etc.
- Network R & D (More than 70% of all Network Research paper reference a network simulator)[*citation needed*]
- Defense applications such as HF / UHF / VHF Radio based MANET Radios, Tactical data links etc.
- LTE, LTE Advanced, IOT, VANET simulations
- Satellite communication

- Optimization of cache placement on large virtual content distribution networks
- Education - Online courses, Lab experimentation and R & D. Most universities use a network simulator for teaching / R & D since its too expensive to buy hardware equipment

There are a wide variety of network simulators, ranging from the very simple to the very complex. Minimally, a network simulator must enable a user to

- Model the network topology specifying the nodes on the network and the links between those nodes
- Model the application flow (traffic) between the nodes
- Providing network performance metrics as output
- Visualization of the packet flow
- Technology / protocol evaluation and device designs
- Logging of packet/events for drill down analyses / debugging

Traffic Modeling

A **traffic model** is a mathematical model of real-world traffic, usually, but not restricted to, road traffic. Traffic modeling draws heavily on theoretical foundations like network theory and certain theories from physics like the kinematic wave model.

Traffic simulation or the simulation of transportation systems is the mathematical modeling of transportation systems (e.g., freeway junctions, arterial routes, roundabouts, downtown grid systems, etc.) through the application of computer software to better help plan, design, and operate transportation systems.[1] Simulation of transportation systems started over forty years ago, and is an important area of discipline in traffic engineering and transportation planning today. Various national and local transportation agencies, academic institutions and consulting firms use simulation to aid in their management of transportation networks.

Simulation in transportation is important because it can study models too complicated for analytical or numerical treatment, can be used for experimental studies, can study detailed relations that might be lost in analytical or numerical treatment and can produce attractive visual demonstrations of present and future scenarios.

To understand simulation, it is important to understand the concept of system state, which is a set of variables that contains enough information to describe the evolution of the system over time. System state can be either discrete or continuous. Traffic simulation models are classified according to discrete and continuous time, state, and space

Traffic models

Simulation methods in transportation can employ a selection of theories, including probability and statistics, differential equations and numerical methods.

- Monte Carlo method

One of the earliest discrete event simulation models is the Monte Carlo simulation, where a series of random numbers are used to synthesise traffic conditions.

- Cellular automata model

This was followed by the cellular automata model that generates randomness from deterministic rules.

- Discrete event and continuous-time simulation

More recent methods use either discrete event simulation or continuous-time simulation. Discrete event simulation models are both stochastic (with random components) and dynamic (time is a variable). Single server queues for instance can be modeled very well using discrete event simulation, as servers are usually at a single location and so are discrete (e.g. traffic lights). Continuous time simulation, on the other hand, can solve the shortcoming of discrete event simulation where the model is required to have input, state and output trajectories within a time interval. The method requires the use of differential equations, specifically numerical integration methods

- Car-following models

A class of microscopic continuous-time models, known as car-following models, are also based on differential equations.. They model the behavior of each individual vehicle ("microscopic") in order to see its implications on the whole traffic system ("macroscopic"). Employing a numerical method with a car-following model can generate important information for traffic conditions, such as system delays and identification of bottlenecks.

Systems planning

The methods noted above are generally used to model the behavior of an existing system, and are often focused around specific areas of interest under a range of conditions (such as a change in layout, lane closures, and different levels of traffic flow). Transport planning and forecasting can be used to develop a wider understanding of traffic demands over a broad geographic area, and predicting future traffic levels at different links (sections) in the network, incorporating different growth scenarios, with feedback loops to incorporate the effect of congestion on the distribution of trips.

What is Media Access Control?

A media access control is a network data transfer policy that determines how data is transmitted between two computer terminals through a network cable. The media access control policy involves sub-layers of the data link layer 2 in the OSI reference model.

The essence of the MAC protocol is to ensure non-collision and eases the transfer of data packets between two computer terminals. A collision takes place when two or more terminals transmit data/information simultaneously. This leads to a breakdown of communication, which can prove costly for organizations that lean heavily on data transmission.

Media Access Control Methods

This network channel through which data is transmitted between terminal nodes to avoid collision has three various ways of accomplishing this purpose. They include:

- Carrier sense multiple access with collision avoidance (CSMA/CA)
- Carrier sense multiple access with collision detection (CSMA/CD)
- Demand priority
- Token passing

Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)

Carrier sense multiple access with collision avoidance (CSMA/CA) is a media access control policy that regulates how data packets are transmitted between two computer nodes. This method avoids collision by configuring each computer terminal to make a signal before transmission. The signal is carried out by the transmitting computer to avoid a collision.

Multiple access implies that many computers are attempting to transmit data. Collision avoidance means that when a computer node transmitting data states its intention, the other waits at a specific length of time before resending the data.

CSMA/CA is data traffic regulation is slow and adds cost in having each computer node signal its intention before transmitting data. It used only on Apple networks.

Carrier Sense Multiple Access with Collision Detection (CSMA/CD)

Carrier sense multiple access with collision detection (CSMA/CD) is the opposite of CSMA/CA. Instead of detecting data to transmit signal intention to prevent a collision, it observes the cable to detect the signal before transmitting.

Collision detection means that when a collision is detected by the media access control policy, transmitting by the network stations stops at a random length of time before transmitting starts again.

It is faster than CSMA/CA as it functions in a network station that involves fewer data frames being transmitted. CSMA/CD is not as efficient as CSMA/CA in preventing network collisions. This is because it only detects huge data traffic in the network cable. Huge data traffic increases the possibility of a collision taking place. It is used on the Ethernet network.

Demand Priority

The demand priority is an improved version of the Carrier sense multiple access with collision detection (CSMA/CD). This data control policy uses an ‘active hub’ in regulating how a network is accessed. Demand priority requires that the network terminals obtain authorization from the active hub before data can be transmitted.

Another distinct feature of this MAC control policy is that data can be transmitted between the two network terminals at the same time without collision. In the Ethernet media, demand priority directs that data is transmitted directly to the receiving network terminal.

Token Passing

This media access control method uses free token passing to prevent a collision. Only a computer that possesses a free token, which is a small data frame, is authorized to transmit. Transmission occurs from a network terminal that has a higher priority

The **Token-Passing Protocol** relies on a control signal called the token. A token is a 24-bit packet that circulates throughout the network from NIC to NIC in an orderly fashion. If a workstation wants to transmit a message, first it must seize the token. At that point, the workstation has complete control over the communications channel. The existence of only one token eliminates the possibility of signal collisions. This means that only one station can speak at a time.

Logical Ring Physical Star topology for Token-Passing Standard.

It is sure that any break in the ring at any point will interrupt communications for all machines. To solve this problem, IBM developed a modified ring topology, which they called the logical ring physical star. The central point of the physical star configuration is Token Ring hub called the multi-station access unit (MSAU, pronounced as masow).

Workstations and servers attached to the MSAU through special STP adapter cables. IBM converted stars into a logical ring by connecting all MSAU hubs together through special ring-in (RI) and ring-out (RO) ports.

Advantages of Token Ring.

Here are Token ring's most useful advantages:

a. It offers excellent throughput under high-load conditions.

b. Token Ring facilitates LAN-to-LAN mainframe connections especially for interfacing with IBM's broader connectivity strategies.

Study And Simulation of a Data Link Layer Protocol

The Data Link Layer is the protocol layer which transfers data between adjacent networks. The Data Link Layer provides the functional and procedural means to transfer data between network entities and might provide the means to detect and possibly correct errors that may occur in the Physical Layer. Study and simulate protocol to analyze and find the advantages and disadvantages and to improve the quality of the protocol such that it handles the network data transmission properly. The protocol is studied so that the details of the protocols are revealed and the limitations of the protocol can be overcome later. During simulation process a protocol is taken and it is examined properly. The frame is designed and the input frames from the physical layer is taken and is combined together to obtain segments. And the output from the network layer as segments are divided into frames and are sent to the physical layer. Thus it performs dual function from top to bottom and from bottom to top.

Various issues that need to be addressed during Data Link Layer design are Framing, Services Provided to the Network Layer, Error and Flow Control. Various functions of the Data Link Layer are

- Providing service interface to the network layer
- Dealing with transmission errors
- Regulating data flow which can control Slow receivers, so that it's not overwhelmed by fast senders

Service provided by link layer to Network Layers above it are

- Interested in getting messages to the corresponding network layer module on an adjacent machine.
- Remote Network Layer peer should receive the identical message generated by the sender
- Make sure that all messages it sends, will be delivered correctly . Arbitrary errors may result in the loss of both data and control frames.
- Network Layer wants messages to be delivered to the remote peer in the exact same order as they are sent.

Error Detection and Correction

Transmission errors are caused by distortion, noises, cross talk and losing synchronization. Errors can be rectified by methodologies like Parity Checks (Adding extra bit to a string of bits to make total number of 1's even) and Cycle redundancy (String of bits as coefficients of a polynomial that uses module 2 arithmetic) codes. Error correction is done by sending redundant information along with the data or by making guess about original data using Hamming Code. In Hamming code power of 2 positions are check bits to find out the incorrect bit and to determine if check bits are correct.

DataLink Simulator

The **Data Link Layer** is Layer 2 of the seven-layer OSI model of computer networking. It corresponds to or is part of the link layer of the TCP/IP reference model.

Data Link Layer provides the functional and procedural means to transfer data between network entities and might provide the means to detect and possibly correct errors that may occur in the Physical Layer.

This C# simulation will cover data link layer's flow control techniques which are:

1. Stop and Wait
2. Go Back N
3. Selective Repeat

Simulation of TCP

Shah Asaduzzaman and Zaki Hasnain Patel have built a TCP model for SVM. This model simulates communication via the TCP network protocol.

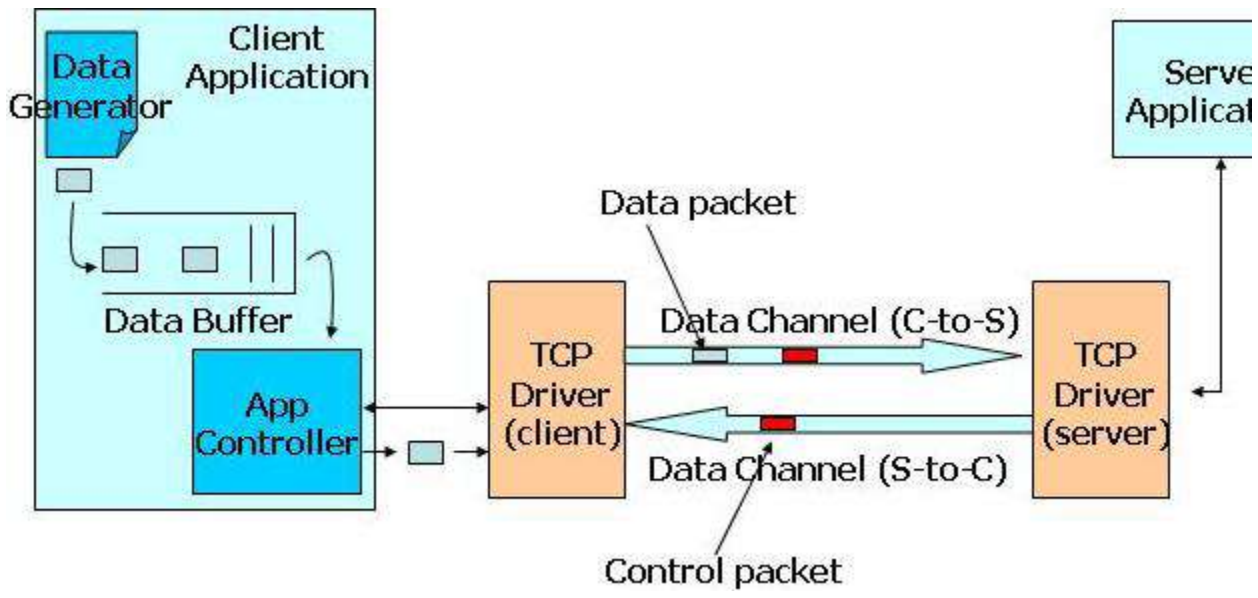


Figure 9.3: The TCP system

The communication process of the system is shown in Figure 9.3. There are 6 parts in the system:

- The client application generates data with a data generator. The data enter a buffer (FIFO queue). They are sent by the application controller one by one. The client application also listens to the data coming from the TCP driver.
- The TCP driver on the client side accepts data packets from the application controller. It sends messages via a network. It has no buffer. The messages must be sent immediately. It also listens to the incoming data channel.
- The data channel transfers data packets from the client side to the server side.
- The TCP driver on the server side accepts data from the data channel from the client side to the server side. It sends control information to the outgoing data channel.
- The server application computes with the received data packets. It generates control packets. Because the control packets are generated one at a time, buffer is not necessary for the server. The generated control packets are sent to the TCP driver on the server side.
- The data channel transfers control packets from the server side to the client side. Those packets are received by the client TCP driver.

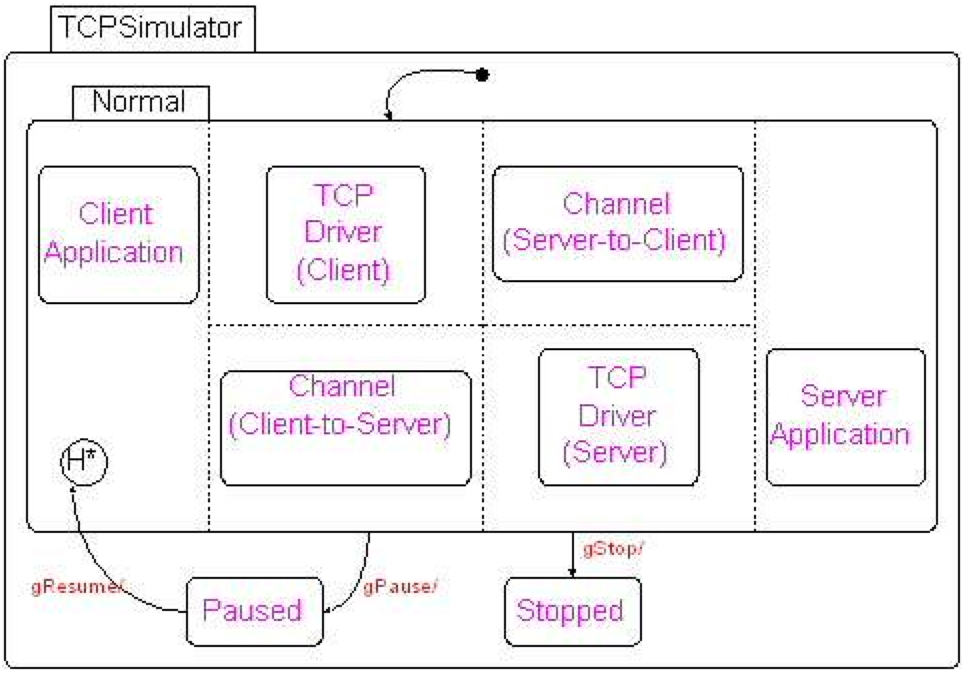


Figure 9.4: Overview of the TCP simulator

Each part of the system is modeled with a DCharts orthogonal component. The whole system is a combination of those orthogonal components by means of importation (Figure 9.4).

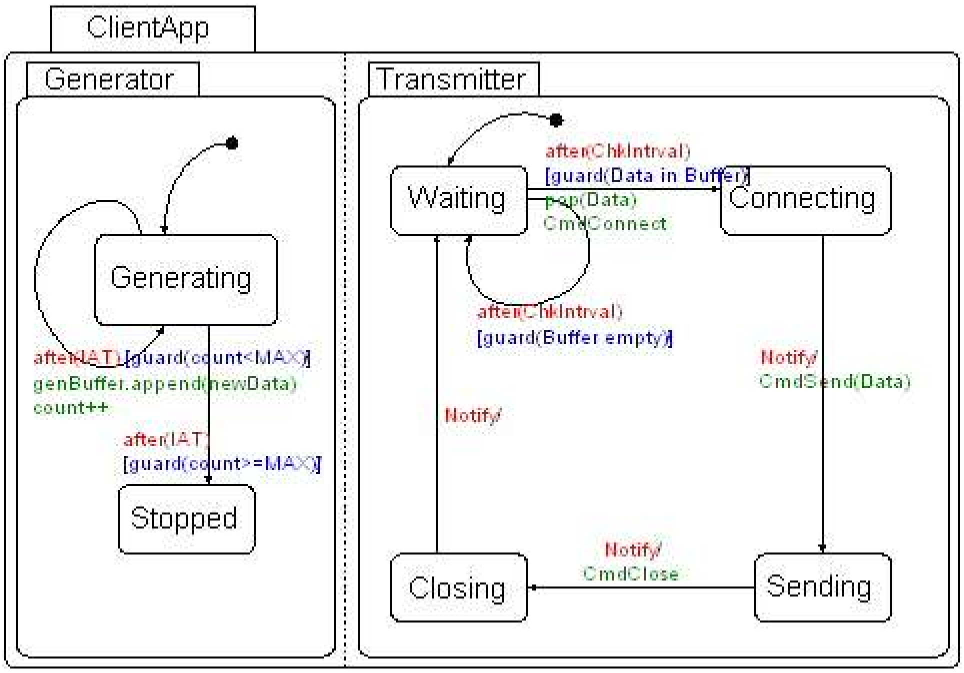


Figure 9.5: The submodel of the client application

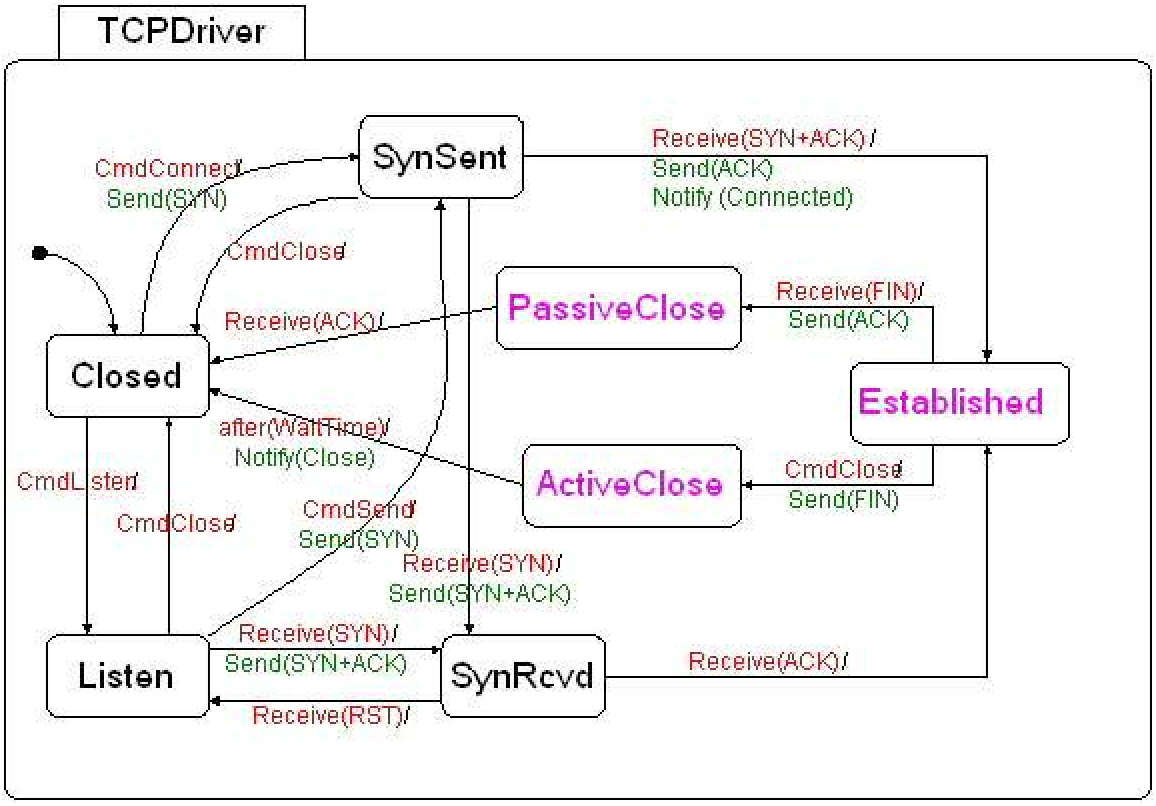


Figure 9.6: The submodel of the TCP driver (for both client side and server side)

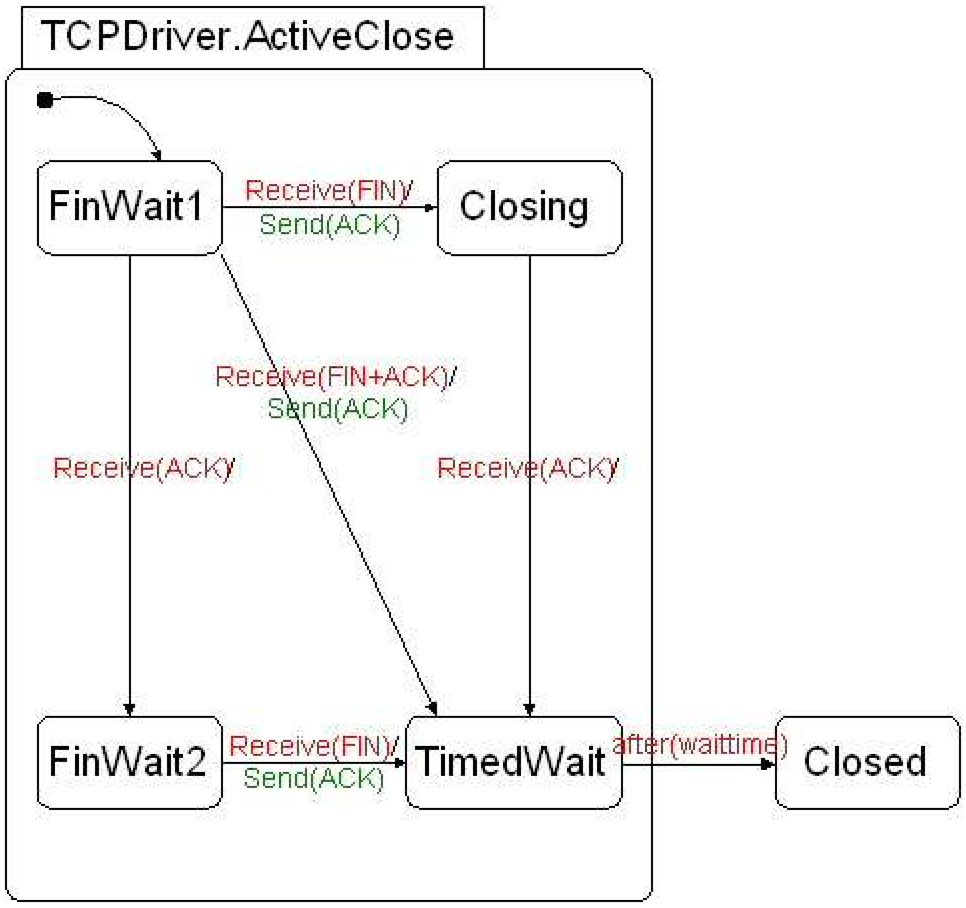


Figure 9.7: The ActiveClose state of the TCP driver

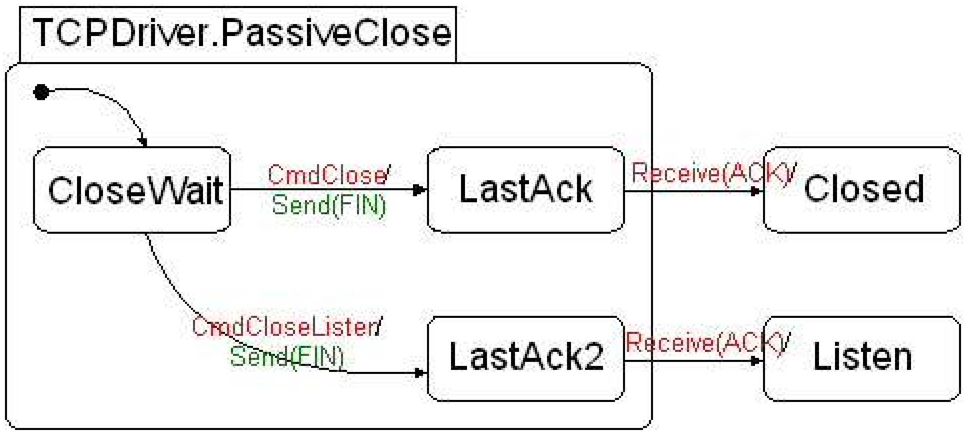


Figure 9.8: The PassiveClose state of the TCP driver

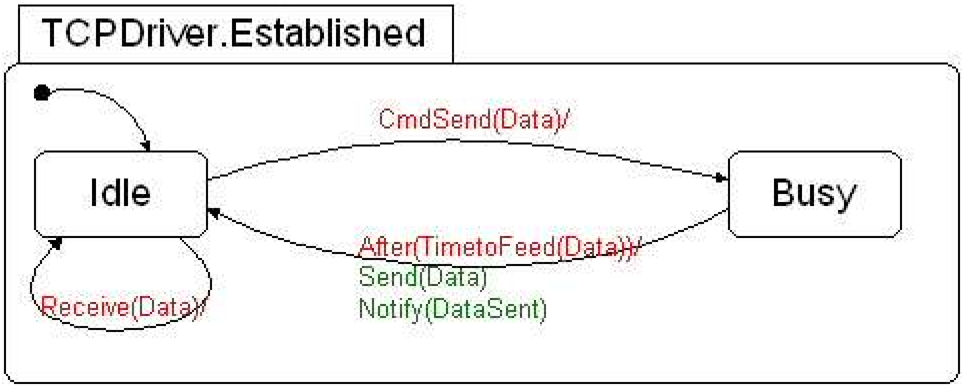


Figure 9.9: The Established state of the TCP driver

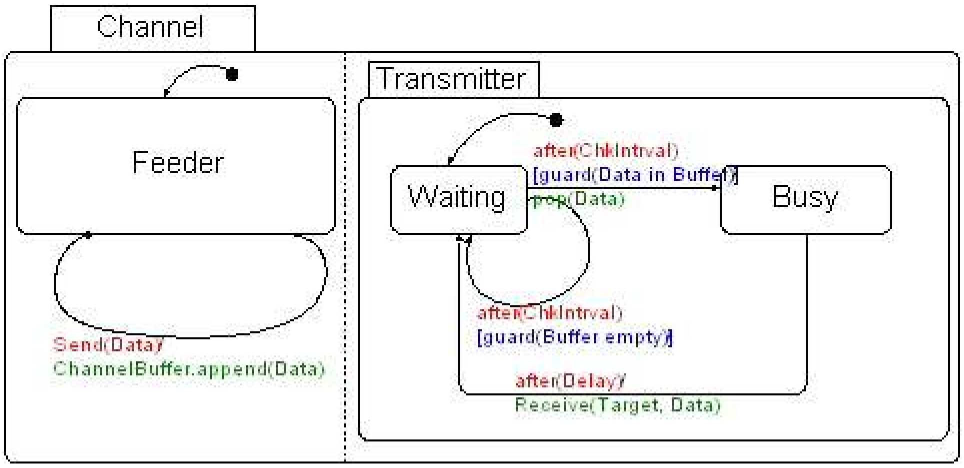


Figure 9.10: The submodel of the communication channel

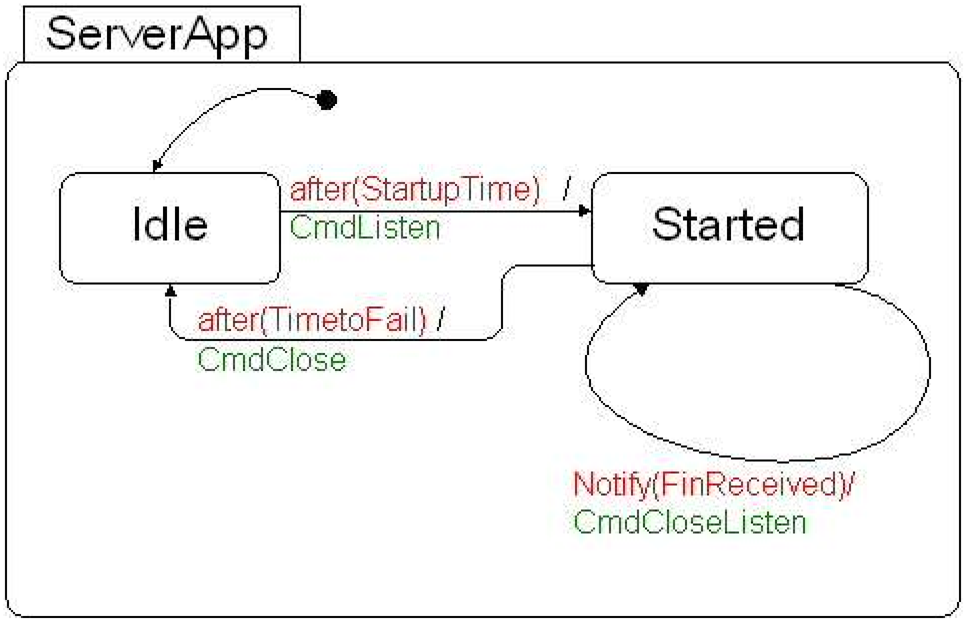

ServerApp
Idle
Started
after(StartupTime) / CmdListen
after(TimetoFail) / CmdClose
Notify(FinReceived)/ CmdCloseListen

Figure 9.11: The submodel of the server application

The 6 parts of the system is modeled with submodels imported into the total system:

- The client application is modeled in submodel ClientApp (Figure 9.5).
- The TCP driver is modeled in submodel TCPDriver. It is for both the client side and the server side, because the API of the TCP protocol on both sides is exactly the same. The ActiveClose, PassiveClose and Established states of the submodel are abstracted. Figures 9.7, 9.8 and 9.9 show the internal structure of those states, respectively.
- The data channels are modeled in submodel Channel (Figure 9.10). Both channels in the system are implemented with the same submodel.
- The server application is modeled in submodel ServerApp (Figure 9.11).

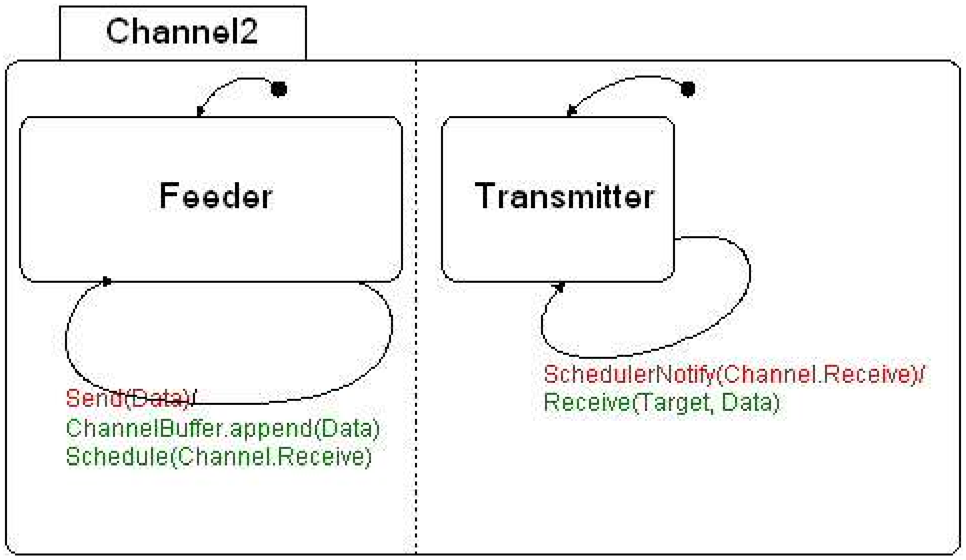

Channel2
Feeder
Transmitter
Send(Data)/
ChannelBuffer.append(Data)
Schedule(Channel.Receive)
SchedulerNotify(Channel.Receive)/
Receive(Target, Data)

Figure 9.12: The virtual-time version of the communication channel

The channel in Figure 9.10 uses the $after$ special event to simulate delay in the network and the time interval between two subsequent inquiries to the buffer. As a result, this model is a real-time model. To convert it into a virtual-time model, Shah Asaduzzaman and Zaki Hasnain Patel have provided another version of the channel submodel Channel2 (Figure 9.12). The clock component is imported as a top-level orthogonal component in the system. The new data channel schedules events by sending the Schedule event to the clock component. When the virtual time becomes equal to the scheduled time, the clock component sends back a SchedulerNotify event. (The schedule event and the notify event discussed in section 9.2 are renamed to Schedule and SchedulerNotify, respectively.)

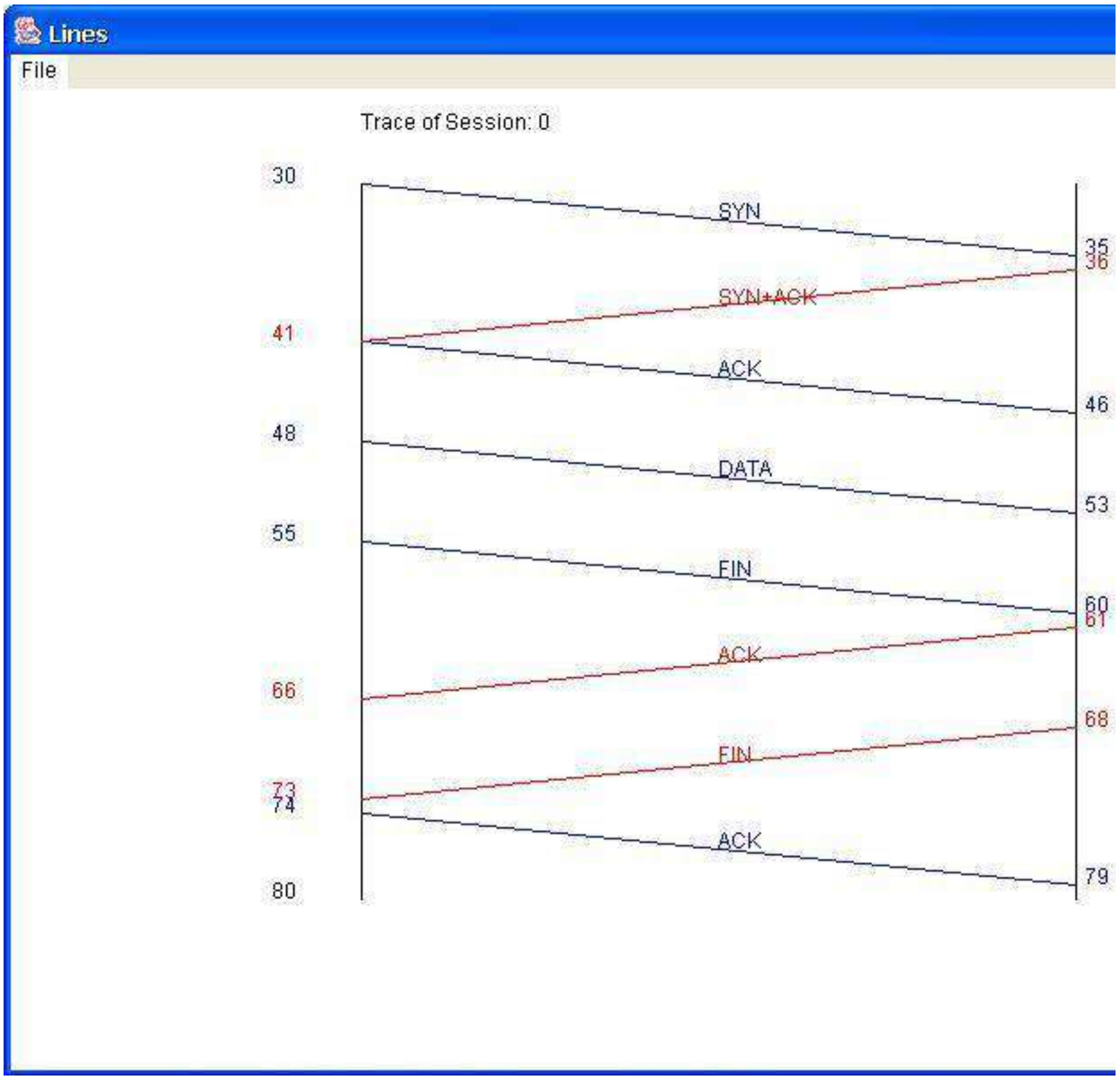


Figure 9.13: The plot of the simulation result of the TCP model

The results of the simulation are gathered and plotted in Figure 9.13. For more information about the TCP model, the readers are referred to Shah Asaduzzaman's on-line report for the Modeling and Simulation course at McGill University:

Simulation Modeling Steps

A simulation of a system is the operation of a model of the system; “Simulation Model”. The steps involved in developing a simulation model, designing a simulation experiment, and performing simulation analysis are: [1]

- **Step 1. Identify the Problem:** Enumerate problems with an existing system. Produce requirements for a proposed system.
- **Step 2. Formulate the Problem:** Select the bounds of the system, the problem or a part thereof, to be studied. Define overall objective of the study and a few specific issues to be addressed. Define performance measures – quantitative criteria on the basis of which different system configurations will be compared and ranked. Identify, briefly at this stage, the configurations of interest and formulate hypotheses about system performance. Decide the time frame of the study. Identify the end user of the simulation model.
- **Step 3. Collect and Process Real System Data:** Collect data on system specifications, input variables, as well as performance of the existing system.
- **Step 4. Formulate and Develop a Model:** Develop schematics and network diagrams of the system. Translate these conceptual models to simulation software acceptable form. Verify that the simulation model executes as intended. Verification techniques include traces, varying input parameters over their acceptable range and checking the output, substituting constants for random variables and manually checking results, and animation.
- **Step 5. Validate the Model:** Compare the model’s performance under known conditions with the performance of the real system. Perform statistical inference tests and get the model examined by system experts. Assess the confidence that the end user places on the model and address problems if any.
- **Step 6. Document Model for Future Use:** Document objectives, assumptions and input variables in detail. Document the experimental design.
- **Step 7. Select Appropriate Experimental Design:** Select a performance measure, a few input variables that are likely to influence it, and the levels of each input variable. Generally, in stationary systems, steady-state behavior of the response variable is of interest. Ascertain

whether a terminating or a nonterminating simulation run is appropriate. Select the run length. Select appropriate starting conditions. Select the length of the warm-up period, if required. Decide the number of independent runs – each run uses a different random number stream and the same starting conditions – by considering output data sample size. Sample size must be large enough (at least 3-5 runs for each configuration) to provide the required confidence in the performance measure estimates. Alternately, use common random numbers to compare alternative configurations by using a separate random number stream for each sampling process in a configuration. Identify output data most likely to be correlated.

- **Step 8. Establish Experimental Conditions for Runs:** Address the question of obtaining accurate information and the most information from each run. Determine if the system is stationary (performance measure does not change over time) or non-stationary (performance measure changes over time).
- **Step 9. Perform Simulation Runs:** Perform runs according to steps 7-8 above.
- **Step 10. Interpret and Present Results:** Compute numerical estimates (e.g., mean, confidence intervals) of the desired performance measure for each configuration of interest. Test hypotheses about system performance. Construct graphical displays (e.g., pie charts, histograms) of the output data. Document results and conclusions.
- **Step 11. Recommend Further Courses of Action:** This may include further experiments to increase the precision and reduce the bias of estimators, to perform sensitivity analyses, etc. Although this is a logical ordering of steps in a simulation study, many iterations at various sub-stages may be required before the objectives of a simulation study are achieved. Not all the steps may be possible and/or required. On the other hand, additional steps may have to be performed. [1]