CHAPTER 3: SIMULATION OVERVIEW FOR CONGESTION 
CONTROL IN ATM NETWORKS

This chapter provides an overview into the simulation approach that will be used in the development of the ATM network simulator components. The aim is to provide an explanation of the model that will be used to simulate the network.

This simulation model will first explain the entire context of the simulator. Then, it will explain the link and buffer model that will be applied for the congestion control mechanism of the simulator.

3.1 Overview of Simulation Model

The simulation model determines how the simulator imitates the real-world behaviour of networks. Since object-oriented programming approach is used, the model will consist of objects that will simulate the behaviour and states of the real-world objects.

This section will provide an overview on the simulation model that will guide through the development of the ATM network simulator components, which emphasises on link and buffer management.

3.1.1 Simulation Architecture

The simulation system is designed and developed with the resource-executor modelling concept. This concept technique uses of the object-oriented approach where flexibility, reusability and extensibility of the developed simulation environment is an important features. The concept explains the objects behaviour and simulates their states in the simulator.

The executor classes posse transaction processes to be executed. Within the network simulator, there are many executor classes to be executed simultaneously. In order to provide these simultaneous executions, the Java multithreading operation is used. Primary executor classes are each being processed in an individual thread.
Resource class is used for the executor classes to perform their tasks. This class represents a component for the executor class to run upon. In this simulation, the resource class is the ATM cell class that drives the executor classes to execute. The resource class would act as a data flowing through the simulator. Classes in the simulator that execute processes would be the executor class. This concept enables the executor class to control the resource objects by knowing their data sources and information throughout the simulator.
Besides, related object classes are grouped into packages. This method is used in Java programming language. This allows individual modules to be developed separately and later can be integrated to form a complete simulation environment.

3.1.2 The Resource and Executor Classes

The simulator consists of many classes and they are categorised into two types: the resources class and the executor class. The resources class is a class, which provides resources to execute the simulator. In this case, ATM cell is a resource class. It generates cells flowing through the network simulator. An executor class is a class that execute and run a process using the resource classes during the simulation period. The examples of an executor class are link class, link end point class, buffer class, switch class and operation class.

An important feature of this simulator design is the use of inter-object communication for the execution of the simulation logic. Communication between resource and executor classes during simulation period is possible - when the executor classes drive the resource class as an object in their processing. Therefore, the simulator is extensible to build another complex simulator as its components can be easily communicated with additional classes and other external components.

Beside the inter-communication between classes, an important feature desired for the ATM network simulator is the multithreading operation. The following section will discuss it in detail.

3.1.3 Multithreaded Operation

This network simulation model will use of parallel operation, as the real world simulation does not anyhow simulate in a sequential way. Sequential models make use of an event list where each component is scheduled to simulate at a certain clock time. After the clock time, the event manager will triggers appropriate components to execute.
This model does not use an event manager to trigger the scheduled events for each component. In this model, each object is responsible to determine when its own transaction will be performed. Therefore, the objects are independent in executing their transactions.

But problem will arise if the objects keep on executing independently without knowing which object to start processing first. Therefore, objects are allowed to process in a parallel but concurrently in a multithreaded operation.

By the use of parallel processing with multithreaded operation, each primary executor object within the simulator seems to execute at the same time. This is because the in single process environment, only one thread is executing at one given time. Since Java programming language handles execution of threads within a processing time and the availability of switching process is quick, therefore it appears that the threads execute concurrently.

With the use of multithreading operation, the simulation model is acting much more like the real world simulator. Each object can handle their own events, at the same time dealing concurrent execution with other objects.

Since the objects execute independently, the synchronisation between the objects and the binding to the global time is an important issue in this operation. Therefore, such binding to the global time needs a clock object to synchronise the timing. This issue will be discuss in the following section.

### 3.1.4 Simulator Timing and Synchronisation

Timing is an important aspect of the simulator. It will reflect the real world network speed and data transfer rate as well as delay in every network portion. Therefore, a proper timing mechanism is needed for the object classes to perform their execution.

The timing mechanism is important in synchronising the objects execution in the simulator, binding them to the global time clock. The clock controls the processes throughout the simulator. Synchronisation is meaningful when all the executor classes are bind to the clock. The clock is an executor class and also acts as a control in the simulator.
Time in the simulator is measures as *ticks*. Each tick can represent specific real world time. The time in simulator is modelled in discrete units, which can be as small as 0.01 microseconds in the real world time.

Each executor class will use the discrete tick time to synchronise their execution in the simulator. Each executor class will perform execution at that tick time and waits for other executor classes to complete their transactions.

The executor classes in the simulator have to confirm upon their complete of transaction to the global clock. Once the global clock has received confirmation upon every executor class, the clock will increment the tick time to the next tick for the simulator. The clock will then inform all the executor classes that a new tick has started.

Besides global synchronisation between objects, the synchronisation between neighbouring objects is needed. The use of control messaging and signalling while transferring the resources into other executor classes. This will appear in the link management architecture where the link will send signals upon the confirmation of sending and fetching cells.

### 3.1.5 Simulator Components

The simulation model consists of two manageable entities. There are the link and buffer. A link object class consists of two link end point classes. Each link end point class consists of two types of buffers – input queue buffer and output queue buffer. A switch composed of many buffers. Therefore, buffer class is a component for a switch class.

Every entity in a simulation model is an object by itself, holding its own characteristics and behaviour. Interactions between objects can be performed using message passing. It has been request to call a function of another object. Objects are self contained and only provides an interface to the outside world where communication with other objects could take place.
Based on Figure 3.2, the simulator is composed of four major components that are listed as below:

- global clock
- link
- buffer
- ATM cell

Firstly, the global clock is the most important object class that determine the universal clock for the simulator. When the clock increments by one tick, the processes of each executor objects would be executed simultaneously. Each object that binds to the thread has its own object ID. They must register themselves when simulation starts. Once every objects has ends their execution their process, they will signal the clock that they have completed their task and waits for next tasks. Next task will be performed only when the new tick increments.

Secondly, the link is an executor object that holds the process of connecting the end systems and switches. At the same tick time, the process of link would be executing. By the end of the
Chapter 3

process, the clock would receive an indicator signal stating the readiness for getting into the next tick.

Thirdly, the buffer is another executor object that fetch and send cells from both the link and switch synchronously. Since buffer is an entity of switch, the switch will indicate the signal of next tick to the clock.

Finally, the ATM cell in the simulator is a resource class. It carries information and data of itself, while will be used by the executor classes during simulation.

The simulator uses multithreading operation to synchronise its executor's execution. Each primary executor objects of the simulator would be bind to the thread. The operation, the link and the buffer driver (switch) are each primary executor object. This thread enables the objects to execute concurrently at the same tick time.

The following section will discuss the link and buffer architecture in detail.

3.2 Link Architecture

The link architecture consists of the link class, link end point class and the queue class. The link is modelled with a simple single server queuing model and the first in first out (FIFO) queuing discipline. Further discussion on the link model will be discussed in the following section.

3.2.1 Link Model

From the model, the link would act in a passive way. Both the receiver and sender would wait for a signal from the input or output queue of the link. With a positive signal, the process of inserting and getting cells from the buffer should be activated. Therefore, signal indication from the link is made whenever the link is ready to transfer the cells from the sender and to the receiver.
Each link has two end points. The end point indicates either it is the left or right side of that link. For the simulation module, the TCP application end system is indicated by the default Link End Point 0 and the other end would be Link End Point 1. If the link were used to connect between switches, it would have either Link End Point 0 or Link End Point 1 at each end.

A transferring method is used to transfer cells in between the input and output queues. ATM cells would be transferred from the input queue to the output queue of another link end point, where it is ready to be fetched by the receiver.

The propagation delay in the link is the time value that the ATM cell would stay before it is fetched by the receiver end-system. The link speed and its link length will determine the delay time. Basically, a link speed would be around 100 Mbps between a TCP end system with the switch and 155 Mbps between switches. The length of the link will be defined in KiloMeter (KM).
3.2.2 Implementing FIFO queuing in Link

In this project, ATM cells in the queues are queued-up using First In First Out (FIFO) queuing discipline. A single queue is maintained at each input and output queue of the link. When a new cell arrives and is transferred to the input queue, it is placed at the end of the queue. As long as the queue is not empty, the link would transmit the cells from one queue to another and finally to the receiver, taking the oldest remaining cell next.

![Diagram of FIFO queuing model in link module](image)

FIFO queuing is a simple queuing discipline. The queuing discipline does not affect critically to the link, as the link does not need priority checking on the ATM cells that flows in. Link would accept cells from the end systems or switch buffer only when the link is not full. If the queues in the link are empty, ATM cells are possible to be fetched in.

3.2.3 Link Traffic Flowing

ATM cells that enter the link are highly depending on the global clock and the link speed. Cell rate is calculated when the global clock time (in tick) and the link rate (in Mbps) are determined. The link will determine at what ticks a cell would enter the link input and output queue.

A cell credit remaining variable is used to determine upon what tick a cell can be sent to the link. Initially, the cell credit remaining variable is set to zero. The cell credit is determined by the cell rate calculated below.

\[
Cell Rate = \frac{(\text{UsecPerTick} \times \text{Link rate in Mbps})}{(53 \text{ bytes} \times 8 \text{ bit})}
\]
Where $UsecPerTick$ is the value of tick time relatively in microseconds, link rate is calculated in Megabits per second. 53 bytes determine a size of an ATM cell, which has 53 X 8 bits in fixed size. Therefore, the cell rate could be calculated in cells per tick.

![Flowchart](image)

*Figure 3.5: Cells flow in link architecture*

Upon executing the simulator, every tick is used to complete a link’s transactions. For the first tick, there might no cells entered the link as there is no cells transferred to the link yet. But the ticks would keep adding on until the integer value of the cell credit remaining hits an integer
value above zero. Until a cell has successfully entered the link input queue, the tick will continue increased by the global clock and the process keeps on (Figure 3.5).

By assuming the propagation time delay for a cell in the link from the sender to the receiver is at the speed of light [1], therefore the delay of each cell in the link is determined as,

\[ \text{Delay in Link} = (\text{int}) \text{ UsecToTicks} \left( \left( \frac{\text{length in Meter}}{\text{speed of light}} \right) / \text{Usecs} \right) \]

The delay is cast into integer value in ticks. The output queue would determine for how long the cell would stay in the queue until it is ready for the receiver to fetch the cell over. UsecToTicks is a function that converts the value of microseconds relative to the number of ticks. The value for speed of light is determined as 3 X 10^8 meter per second. And the value for Usecs is the value for microseconds of a tick, which can be as small as 0.01 microseconds for a tick.

### 3.2.4 Link Algorithm

![Link Algorithm Diagram](image)

*Figure 3.6: Basic Link Components*

Based to the basic link components in Figure 3.6, a link maintains two link end points, which indicates each ends of a link. In this simulation, link end point 0 is used to connect with the TCP end systems and link end point 1 is used to connect with the switch. In the case that the link is connecting between two switches instead of an end systems and a switch, there would be another overloaded function for the case, where the link end points of a link should be initialised at runtime.
Therefore, in object-oriented concept, a link class is composed of two link end point classes. At the link end point class would maintain the functions of both input and output queue together. The ATM cells would flows into and from the input and output queue with the algorithm below:

\[
\text{While not done getting cell from the attached system at link end point }
\]
\[
\text{If the input queue is not full }
\]
\[
\text{Fetch cell according to the cell rate at each tick;}
\text{Transfer cells to another end point's output queue according to the number of cells contain in input queue at that tick;}
\text{Send a cell out from the output buffer after a period of delay time in the link;}
\]
\[
\text{Else}
\]
\[
\text{refuse cells from entering the input queue of the link;}
\]

### 3.3 Buffer Architecture

The buffer architecture determined the buffering model used in this simulation and methods used for congestion control in the buffer management. The methods used includes FIFO queuing in buffer, early packet discard for ATM cells of TCP packets, fair buffer allocation to retain the fairness in early packet discard, RM cell management and explicit rate control in buffer management.

#### 3.3.1 Buffer Model

![Switch architecture with buffer model](image)

*Figure 3.7: Switch architecture with buffer model*
Buffer is a component of a switch. Switch is an object that communicates with the thread of global clock. Therefore, buffer is drives by the switch execution, acts as one of the switch’s transactions. Besides, buffer is attached to link object. Each buffer will have one link to integrate. A switch is extendable from having four buffers to as many as wished, as the input and output buffer has the same functions. Congestion control will take affect in the buffer when an ATM cell is fetched into the buffer.

Figure 3.8: Switch with buffer model and congestion control mechanism

The congestion control mechanism is written as methods that stored in the buffer class object. Congestion will be checked while each cells being fetched to be buffer. Congestion will be alerted when the buffer queue size is above the threshold value for the buffer. The early packet discard (EPD) method will be invoked at the same time. Cells of a new packet will be discarded but cells of the remaining packets in the queue will still be fetched. But all the incoming cells will be discarded when the buffer is full, which means that the queue capacity has reached the maximum buffer size. Congestion would occur at the both the input buffer and output buffer, depending on the queue size of the buffer after cells being fetched into it.

3.3.2 Implementing FIFO Queuing in Buffer

As the same method as in link management, ATM cells are queued-up in the buffer queue. A single server queuing model with first come first serve discipline is used to queue up the cells.
Single server queue is the simplest queuing system, which acts as a centre to provide service to an item. In this simulation, the item is either a switch or a link. ATM cells will queue up at the buffer before being fetched by the switch for processing, or being sent to the link system.

The cells being fetched from the buffer is the head-of-line cell of that buffer. Meaning that the buffer uses the first in first out queue to store them in a waiting line before the switch does its dispatching discipline. This FIFO queuing theory is an easiest method as the cells are fetched to the buffer without any priority checking since each link is connected to each buffer. ATM cells flow from the link to the buffer in a continuous mode without any blocking from other connection. Therefore, this queuing discipline is enough to determine the queue of cells.

![Diagram of Single Server Queuing with FIFO](image)

*Figure 3.9: Single Server Queuing with FIFO*

### 3.3.3 Implementing Early Packet Discard

By using early packet discard (EPD) scheme, an EPD threshold is set at the buffer (that is lower than the total buffer capacity). This scheme is based on the observation on transporting IP traffic over an ATM network that usually requires fragmenting large IP packets into smaller ATM cells [53]. After IP packets are fragmented into fixed sized ATM cells, the connection will uses the ATM network to transmit cells.

If an ATM cell corresponding to the first fragment of an IP packet arrives when the occupancy of the buffer exceeds the EPD threshold, the cell is discarded includes all succeeding fragments. However, an ATM cell corresponding to a non-initial fragments is still accepted as long as there are any free buffer capacity. If the buffer were full, all cells that attempt to enter would be discarded. In other word, if the total number of cells in the buffer exceeds the EPD threshold, the policy would prevent all new packets' cells from entering the

86
buffer, but continues to accept the cells for those packets that have already begun to enter [54].

In this simulation, besides using EPD scheme, the explicit rate control policy is used. Besides, a RM cell would carries information about the maximum transmission rate a source would like to go at. When a switch on the path of a VC receives an RM cell, it computes the rate it is willing to allocate for this VC, and overwrites the existing ER value in the RM cell if the computed rate is lower [53]. In this simulation, for the first two attempts, only those RM cells with using bandwidth above their fair share would be notified. After that attempts, all RM cells available in the buffer would be alerted to slow down the cell rate by their sources. This is because the congestion might enter a critical point if all the sources do not slow down their cells transferring speed.

Despite the effectiveness of EPD, the fairness problem can be very serious. Therefore, an enhancement of fair buffer allocation (FBA) is used to increase fairness.

### 3.3.4 Implementing Fair Buffer Allocation

Another algorithm for buffer management algorithm is the fair buffer allocation (FBA), which was initially proposed by Heinanen and Kiki and extensively studied by Jain et. al. [55]. This algorithm is used to achieve fairness properties in ABR's max-min fair allocation of bandwidth without keeping a lot of the per-connection state required by the ABR rate computation [53].

To explain the FBA algorithm, definition of the following parameters for a given ATM switch:

\[
B = \text{Buffer Capacity of the switch, in cells}
\]

\[
R = \text{Threshold parameter which triggers packet discard; } R < B
\]

\[
N = \text{The current number of cells in the buffer; } N \leq B
\]

\[
N(i) = \text{The current number of cells in the buffer for VC I}
\]

\[
V = \text{Number of active VCs; that is, the number of VCs that have at least one cell in the buffer}
\]
The following relationship is satisfied:

\[ N = \sum_{i=1}^{\nu} N(i) \]

EPD would begin to drop cells when a EPD threshold is reached. FBA adopts a policy of more aggressive dropping of packets as congestion increases. The rule for FBA is based on the following condition:

\[ (N > R) \quad AND \quad W(i) > Z \times \left( \frac{B - R}{N - R} \right) \]

Where parameter \( Z \) is given a value little less than one. For FBA, \( W(i) \) is compared to a value that increases as congestion increases. If were to think that the \( (B - R) \) cell slots in the buffer has a safely zone, the greater of the fraction of the safety zone that is occupied, the smaller the value to which \( W(i) \) is compared. As congestion increases, the switch’s buffer will begin to drop packets from more and more VCs, exempting only those that have a few buffered cells [1].

In this simulation, whenever the aggregate buffer occupancy is below a specific EPD threshold, connections that occupy more than their fair share would be discarded. Connections that are below their fair share would be safe from discarding. The fair share of a connection is given by [55]:

\[ \text{Buffer Fair Share} = Z \left( \frac{K - R}{X - R} \right) \frac{X}{N_a} \]

Where, \( K \) is the buffer capacity, \( N_a \) is the number of active connections, \( X \) is the current buffer occupancy, and \( Z = 0.9 \), and \( R = 0.9K \) [42]. Fair share for each buffer can be compute using the formula given above.

Besides controlling the cells fetching attempt into the buffer queue, the switch should alert the connection source to slow down its cells transmission. This would happen in TCP application where they would have their own congestion control scheme such as, the slow-start mechanism. The source would realise the congestion happen when receiving RM cells that has indications in it.
3.3.5 Resource Management (RM) Cell

In ABR traffic management framework, the source and end systems limit their data transmission to rates allowed by the network. These rates are sent to the source as feedback via resource management (RM) cells. The RM cells are generated by the sources and travel along the data path to the destination and end systems. The components of the ABR traffic management framework as in [2] are shown as in Figure 3.9.

![Diagram showing RM cells and their flow](image)

*Figure 3.10: ABR Traffic Management Model: Source, Switch, Destination and RM cells*

In this simulation model, the RM cells would carry Congestion Indication (CI) bit, the No Increase (NI) bit and Explicit Rate (ER) that can be set and reduced by the congested switches to any value. Therefore, the source would determine its new value by reading the value in the RM cell. The Cells Loss Priority (CLP) of the RM cells are always zero, which means it is not prompt to discard when congested. The switch would only have the authority to change the value of CI, NI and ER of forward RM cells when congestion occurs according to a specific rule. CI would be set to one (1) if congestion occurs and NI would only be change to one (1) if the switch would like to indicate that no additive increase is allowed for the source. Besides, the ER value is changed based on the specified explicit rate control.

If congestion occurs, the source could react according to the ABR feedback mechanism with the rules given in the following algorithm,

```python
if CI == 1
    Reduce ACR by an amount proportional to the current ACR but not less than MCR
else if NI == 0 increase ACR by an amount proportional to PCR but not more than PCR
```
if ACR > ER set ACR <- max [ER, MCR]

The effect of feedback cells would adjust the ACR to vary between MCR and PCR. The indication of bits would determine the adjustment of the ACR.

The switch's buffer could modify the explicit rate of an RM cell when congestion occurs. Thus, the modification shall be based on specific rules in controlling the explicit rate.

3.3.6 Explicit Rate Control on RM cell

In this simulation, the new Explicit Rate (ER) value is calculated based on bandwidth demand estimate algorithm where the Mean Allowed Cell Rate (MACR) is computed at the switch based on a running exponential average of the ACR value read from each connection's forward RM cells [2]. MACR is given by

\[
MACR = MACR + (ACR - MACR) \times AVF
\]

Where AVF = 1/16. If the load factor (which is the Input Rate / Target Rate) is < 1, the leftover bandwidth is reallocated according to

\[
MACR = MACR + MAIR
\]

Where MAIR = 0.5 Mbps, is the MACR Additive Increase Rate.

A positive queue derivative indicates an increasing bandwidth demand that causes the switch to enter a congestion state; the ER value computed is then

\[
ER = MACR + MRF
\]

Where, MRF = 0.95, if the switch has detected congestion. Else, the ER value is set as

\[
ER = MACR
\]
3.3.7 Buffer Management Algorithm

![Diagram of a network link with a buffer driver]

*Figure 3.11: Basic Buffer Components*

Link has immediate connection with the buffer. Emerging cells from the link to the buffer, the ATM cells would be fetched to the buffer when the link’s output queue indicated that there are available cells to transfer into the buffer component. Therefore, the buffer class would fetch the cells over.

While fetching the cells over to the buffer, the capacity of the buffer is observed. In case of congestion, the early packet discard scheme will be useful. Capacity of the buffer will determine whether it exceed the EPD threshold or not. Once the capacity exceeds the threshold, the following algorithm would be in process:

```
Fetch cells from link's output queue or fetch cells from switch processing;
If buffer is full{
    Discard cell that fetched;
}
Else if the buffer is above EPD threshold {
    If the incoming cell is a new packet's first cell with SDU = 0 {
```
Fetch cell into the buffer;
}
Else {
    Determine on that cell's connection VC;
    If the cell is the remaining cell of a segment entered the buffer before congestion {
        If the cell’s connection uses exceeds the fair buffer allocation
            Discard cell that fetched;
        Else
            Fetch cell into the buffer;
    }
    Else {
        Discard cell that fetched;
    }
}
Look through the buffer queue for forward RM cells;
Check for the connection uses exceeds the fair buffer allocation;
If connection exceeds fair buffer allocation is found {
    Check in the queue If (it is Forward RM cells of that connection & & attempt is 1) {
        Calculate the new ER value for it;
        Change the ER value of that forward RM cells;
    }
    Check in the queue Else if (it is Forward RM cell & & attempt is > 1) {
        Change the ER value for every forward RM cells;
    }
} // end of if
} // end of else if

3.4 Chapter Summary

The context and architecture of the entire simulation model was described in this chapter. The
test has covered a broad view of the general architecture with its components fitting in, including the global clock timing and multithreading operation.

Secondly, the model is then applied to the link and buffer management. The components of link and buffer are defined. The description on the systems functions was discussed further.

The next chapter will deals with the designs and implementation of the relevant classes, based on the simulation model discussed in this chapter.