

# Shikav extensions to support networking animation

Dissertation

submitted in partial fulfillment of the requirements  
for the degree of

**Master of Technology**

by

**Moniphal Say**

(Roll no. 05329201)

under the guidance of

**Prof. Sridhar Iyer**

**Prof. Abhiram Ranade**



Department of Computer Science and Engineering (KReSIT)

Indian Institute of Technology Bombay

July-2007



# Dissertation Approval Sheet

This is to certify that the dissertation entitled  
**Shikav extensions to support networking animation**

by

**Moniphal Say**

(Roll no. 05329201)

is approved for the degree of **Master of Technology**.

---

Prof. Sridhar Iyer

(Supervisor)

---

Prof. Abhiram Ranade

(Co-supervisor)

---

Prof. Anirudha Sahoo

(Internal Examiner)

---

Dr. Vijay T. Raisinghani

(External Examiner)

---

Prof. V. M. Gadre

(Chairperson)

Date: \_\_\_\_\_

Place: \_\_\_\_\_



# INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

## CERTIFICATE OF COURSE WORK

This is to certify that **Ms. Moniphal Say** was admitted to the candidacy of the M.Tech. Degree and has successfully completed all the courses required for the M.Tech. Programme. The details of the course work done are given below.

Sr.No.	Course No.	Course Name	Credits
<b>Semester 1 (Jul – Nov 2005)</b>			
1.	IT601	Mobile Computing	6
2.	HS699	Communication and Presentation Skills (P/NP)	4
3.	IT603	Implementation Techniques in DBMS	6
4.	IT619	IT Foundation Laboratory	8
5.	IT623	Foundation course of IT - Part II	6
6.	IT605	Computer Networks	6
<b>Semester 2 (Jan – Apr 2006)</b>			
7.	CS686	Object Oriented Systems	6
8.	HS700	Applied Economics (Institute Elective)	6
9.	CS640	Natural Computing (DNA Computing)	6
10.	IT680	Systems Lab.	6
11.	IT694	Seminar	4
<b>Semester 3 (Jul – Nov 2006)</b>			
12.	CS601	Algorithms and Complexity	6
13.	CS681	Performance Evaluation of Computer Systems and Networks (Audit)	6
<b>Semester 4 (Jan – Apr 2007)</b>			
14.	IT610	Quality of Service in Networks (Audit)	6
<b>M.Tech. Project</b>			
15.	IT696	M.Tech. Project Stage - I (Jul 2006)	18
16.	IT697	M.Tech. Project Stage - II (Jan 2007)	30
17.	IT698	M.Tech. Project Stage - III (Jul 2007)	42

I.I.T. Bombay

Dy. Registrar (Academic)

Dated:



# Acknowledgements

I would like to express my sincere gratitude toward my guides **Prof. Sridhar Iyer**, and **Prof Abhiram Ranade** for their constant support and encouragement. Without their invaluable guidance, this work would never have been a successful one.

I would like to thank **Kushal** for always being a supportive friend, providing constructive ideas and help me to fix bugs in my code. I also would like to thank **Janak, Ramdas** for their helpful discussions and thanks to **Ramanand** for his help in reviewing and correcting my English in this thesis.

I would like to thank my **family** for their love and support which always have been a source of my inner strength. I would also like to thank to the whole family of Prof. Sridhar Iyer for their kindness and love that make me feel like home. Last but not the least, I would like to thank my **friends** and the **KReSIT family** for making my 2 years in IITB memorable.

**Moniphal Say**

I. I. T. Bombay

July 16<sup>th</sup>, 2007





# Abstract

There is a growing interest in e-learning tools. One category of e-learning tools is electronic lesson. In addition to the lectures in classroom, electronic lessons can also be used, which help students in understanding the topics more completely by the use of animation as interactive visual aids. For this purpose, many authoring tools have been developed. Shikav 2.0 is one among them.

Shikav is a framework for constructing interactive electronic lessons, including animations using its own high level script language. The framework lets us create lessons in the field of Mathematics, Algorithms, Biology, Physics, and Economics. Shikav is developed in Java and has support for constructs such as *point, line, arc, circle, triangle, rectangle, anglecurve, edge, vertex*. However these are not sufficient to easily create animations of network protocols. Additional constructs need to be supported such as *node, packet, beacon, repeat loops, if else conditional statement*.

We have added modules, and made modifications to Shikav to support lessons explaining the basic concept of network protocols. In this thesis, a detailed description of the concepts, design, features of Shikav are presented. The design and the newly added features to Shikav have also been thoroughly described. We also explain a new defined script called “network script” language.



# Contents

<b>Acknowledgements</b>	<b>vii</b>
<b>Abstract</b>	<b>ix</b>
<b>List of figures</b>	<b>xiii</b>
<b>Abbreviations</b>	<b>xv</b>
<b>1 Introduction and Motivation</b>	<b>1</b>
1.1 Advantages of electronic lessons . . . . .	1
1.2 Shikav . . . . .	2
1.3 Motivation . . . . .	3
1.4 Problem statement . . . . .	4
1.5 Accomplishments . . . . .	5
1.5.1 Features added into Shikav script language . . . . .	5
1.5.2 Network script language . . . . .	6
1.6 Thesis outline . . . . .	8
<b>2 Shikav and Related work</b>	<b>11</b>
2.1 Design of Shikav framework . . . . .	11
2.1.1 Compile, Issue, Update . . . . .	11
2.1.2 Features of Shikav . . . . .	13
2.2 Components of Shikav . . . . .	14
2.3 Shikav Script language . . . . .	15
2.4 Related tools . . . . .	20
2.4.1 JAWAA . . . . .	20
2.4.2 Samba and Polka . . . . .	20

2.4.3	Compare with Shikav . . . . .	20
2.4.4	Tool for creating networking lesson - JSIM . . . . .	21
2.4.5	Summary . . . . .	23
<b>3</b>	<b>Examples for Shikav Extension</b>	<b>25</b>
3.1	TCP 3 way handshake . . . . .	25
3.2	TCP slow start . . . . .	26
3.3	WiFiRe . . . . .	27
3.4	Summary . . . . .	30
<b>4</b>	<b>Implementation details of Shikav Extension</b>	<b>33</b>
4.1	Class Diagram of new added classes . . . . .	33
4.2	Enhancement in Shikav . . . . .	36
4.2.1	Object creation . . . . .	36
4.2.2	Shikav script . . . . .	36
4.3	Network script language . . . . .	37
4.3.1	Integration of new script language into Shikav . . . . .	40
4.3.2	How does “NetworkParser” work? . . . . .	41
4.4	Examples . . . . .	41
4.4.1	TCP 3 way handshake . . . . .	43
4.4.2	TCP slowstart . . . . .	43
4.4.3	WiFiRe . . . . .	44
<b>5</b>	<b>Conclusion and Future extensions</b>	<b>49</b>
5.1	Conclusion . . . . .	49
5.2	Future extensions . . . . .	49
<b>A</b>	<b>Examples of lecture files</b>	<b>51</b>
A.1	TCP 3 way handshake . . . . .	51
A.2	TCP slowstart . . . . .	53
A.3	WiFiRe . . . . .	55
	<b>Bibliography</b>	<b>59</b>

# List of Figures

1.1	The work flow of Shikav . . . . .	2
1.2	Screen shot of lesson created in Shikav . . . . .	3
1.3	The screenshot of animation of node transferring packets in Shikav . . . . .	8
2.1	Use case diagram of Shikav . . . . .	12
2.2	Components of Shikav . . . . .	14
2.3	Layer of packages in JSIM . . . . .	21
2.4	The screen shot of simulation of customers in MC Donald . . . . .	23
3.1	The process of TCP 3 way handshake (TCP initial connection set up) . . .	26
3.2	High level network configuration for WiFiRe . . . . .	28
3.3	Sequence diagram of initial ranging steps . . . . .	29
3.4	Sequence diagram of registration steps . . . . .	30
3.5	Sequence diagram of UGS data connection . . . . .	31
3.6	Sequence diagram of rtPS and nrtPS data connection . . . . .	31
4.1	Relation between newly added classes . . . . .	34
4.2	The flow of Shikav . . . . .	42
4.3	Screenshot of animation showing the working of WiFiRe protocol . . . . .	47
A.1	Screenshot of animation showing the working of TCP 3 way handshake . .	52
A.2	Screenshot of animation showing the working of TCP Slow Start . . . . .	55
A.3	Screenshot of animation showing the working of WiFiRe protocol . . . . .	58



# Abbreviations

## Abbreviations

- BS : Base Station
- BS ID : Base Station Identifier
- JSIM : Java Simulation
- IRRe : Initial Ranging Request
- IRRes : Initial Ranging Response
- nrtPS : non-real time Polling Service
- rtPS : real time Polling Service
- RSSI : Received Signal Strength Indication
- ST : Subscriber Terminal
- ST ID : Subscriber Terminal Identifier
- UGS : Unsolicited Grant Service





# Chapter 1

## Introduction and Motivation

There are various ways for students to learn a particular subject, by text book, by visual aid, by animated representation, and by audio representation. Books provide textual and visual presentations. E-learning becomes popular topic of interests. Authoring tools help non-professional programmers to create electronic lessons. Some of these tools provide the facility to create interaction between the lessons and the end-users while some provide the graphical interface to facilitate the creation of lessons. Students learn best by interacting with the lessons. For the same purpose, many authoring tools such as Shikav [8], JAWAA [9], Samba [2], JSIM [6] etc. have been developed. These provided different sets of features.

### 1.1 Advantages of electronic lessons

Electronic lessons or courseware have many advantages, which are listed below:

- **Multimedia content:** courseware can contain multimedia features such as images, animation, audio and video clips. These enrich the contents of the topic and make their study more interesting.
- **Interaction:** students understand the studied topics better by varying the input parameters, moving objects.
- **Accessibility:** courseware can be made available to more users by putting it online on the Internet, Intranet, or by distributing it on CDs.

## 1.2 Shikav

Shikav [8] is a framework for developing interactive electronic lessons. Its development was initiated by Professor Abhiram Ranade at IIT Bombay. Shikav is written completely in Java. Shikav has its own *high level script language* which is described in section 2.3. This language enables the author to create electronic lessons in various topics in fields such as Mathematics, Biology, Algorithms, Physics, Economics.

Shikav can be run as a stand alone application or can be run on the Web. It takes the script file called “lecture file” with “.lec” extension as argument.

### How does it work?

The figure 1.1 shows the work flow of Shikav.

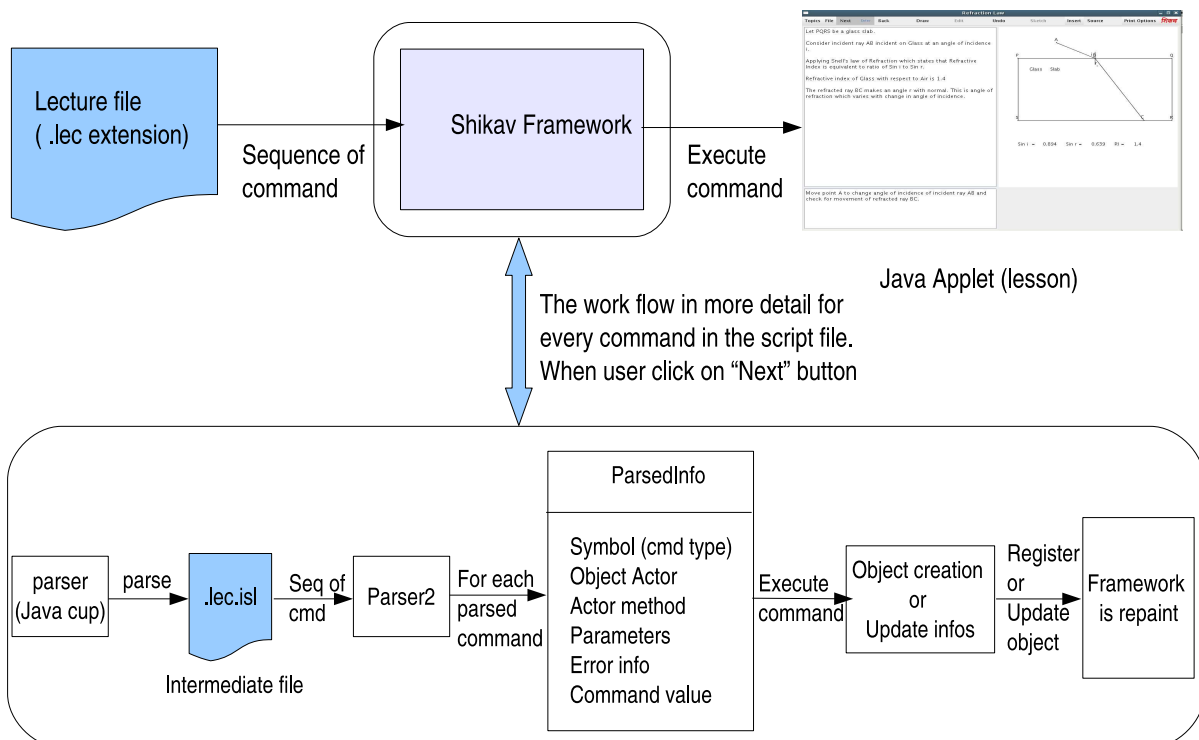


Figure 1.1: The work flow of Shikav

- Shikav first checks for a valid Java version installed (from version 1.3 upward) in the system. If the version is valid, the default framework will be created with various menu options as shown in the figure 1.2.

- The script file (“.lec” extension) will first be compiled and then parsed sequentially. The *ParsedInfo* object will be created corresponding to each command.
- The command gets executed with the creation of object based on the necessary information in *ParsedInfo* object. The panel object gets registered in the panel queue of the framework and the geometry object gets registered with the *Geometry* panel. The method *draw(Graphic g)* of the geometry objects gets called and the object gets display in the panel. Every time the *Geometry* panel gets repaint, the method *update()* of its registered objects get called and objects get updated.

Figure 1.2 is the screen shot of a sample lesson created in Shikav.

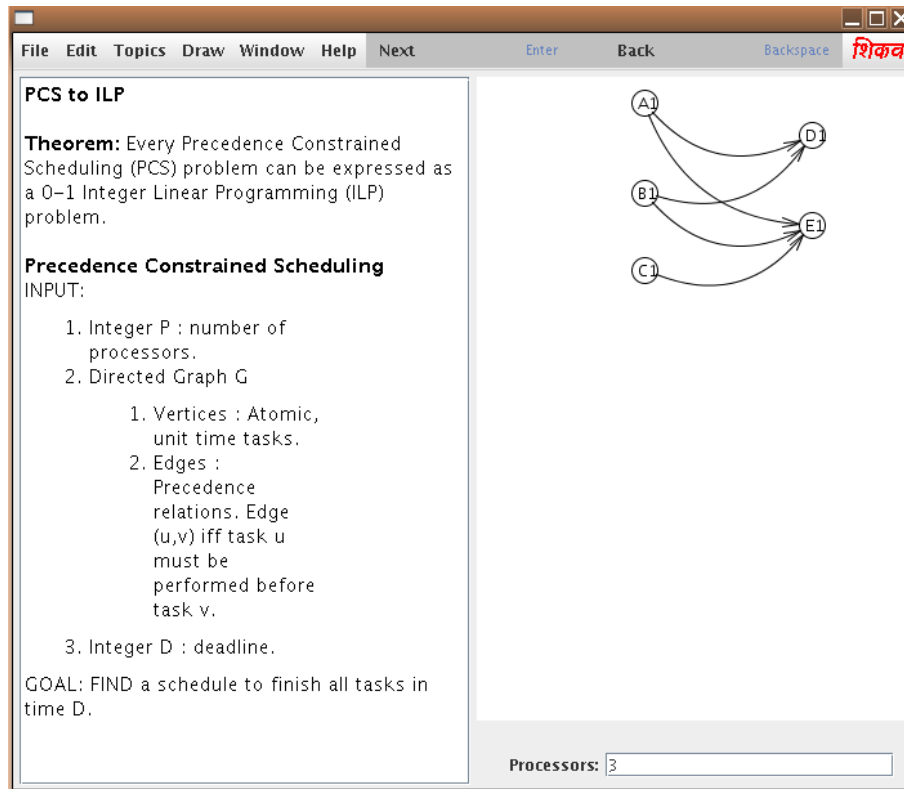


Figure 1.2: Screen shot of lesson created in Shikav

## 1.3 Motivation

An answer to a question like “*how does a particular protocol work?*” would be simple and easy to understand if it contains animation along with the explanation. As described in section 1.1, interactive electronic lessons play an important role in helping students

to understand the basic concepts of the lessons, so choosing an authoring tool becomes a crucial decision for the authors (lesson content's creators). Similar to other authoring tools, Shikav [8] has been created with its own high level script language to enable the authors to create lessons in the field of Mathematics, Algorithms, Biology, Physics, and Economics. In comparison to other tools which are briefly described in Section 2.4, Shikav has advantages such as it provides scope to extend its source code, it uses the "reflection" technique of Java in which the designed was made in such a way that the behaviors of new added classes into Shikav codebase could be accessed from the script, moreover electronic lesson created in Shikav has the classroom board interface in which the frame can be divided into multiples panels.

However the following capabilities are missing such as:

- Creation of nodes
- Creation of packets
- Creation of control packet (beacon)
- Loops
- If-Else conditional statement

In short Shikav doesn't have intrinsic support for networking animations. For instance, one can understand or experiment with various protocols in networking by using simulations, but to understand its basic concepts and steps, the animated representation can really be useful.

## 1.4 Problem statement

Our goal is to extend the Shikav framework to allows authors to create electronic lessons in the topic of networking. Shikav has been created with many features which enable authors to create lessons but the facility to create electronic lessons explaining network protocols has not yet been provided. Hence, building the lessons in networking by creating nodes, defining the behavior of each node to send, receive packets, etc. became a difficult task for the authors. The goal of this project is:

1. Adding necessary features into Shikav for supporting creation of networking animations
2. Defining and building a script language one level higher than Shikav script language for the sole aim of creating networking animations.

Note that the purpose was not to build a simulation in Shikav i.e. the electronic lesson created will not be able to show the analysis, the statistics of the packets which are available in other simulations. It is only the representation of how the packets have been transferred.

## 1.5 Accomplishments

Various network protocols and simulation tool [6] have been studied. The significant work done is as follows:

1. One module of code has been implemented and added into the existing Shikav codebase. This module enables the authors to create lessons explaining networking which means it completes the missing features required for creating networking animations in Shikav.
2. One higher level script for the unique purpose of creating animation explaining network protocols has been defined and implemented.

### 1.5.1 Features added into Shikav script language

The newly added module allows authors to create objects of type *Node*, *Packet*, *Beacon* explaining the basic working flow of any network protocols. *For loop* has also been added, which is used in the cases where the same action repeats. The features added are:

- *Entitynode* enables the author to create nodes providing their name and coordinates, which can be displayed in the *Geometry* panel
- *Packet* enables the author to create packets providing the content, source, and destination. The display of packet can be seen only during its transfer from one node to another node. The initial point of the packet is set to be the coordinate of the source node.

- *Beacon* enables the author to create beacon providing the source point and the points of boundary.
- *If-Else conditional statement* allows the author to test condition and define statement based on the result of the tested condition.

### 1.5.2 Network script language

A higher level script has been defined and implemented for creating animations in networking. The terms used in this script are:

1. Node
2. send
3. receive
4. ConstructPacket
5. ConstructBeacon
6. repeat (this is being used in the place of For loops)
7. if *< condition >* else *< statement >* then *< statement >* endif

Authors who want to create animations explaining the work flow of network protocols have the following choices:

- writing a script file by following Shikav script syntax by using the new added features as described above and more in detail in Section 4.2.
- writing a script file by following the new defined script as described in Section 4.3 and passed into Shikav framework

Both are still named with “.lec” extension as usual Shikav lecture file.

The new scripts are defined to help the author in creating the animation in networking as the terms used are really simple. Below is an example of a few commands in this newly defined script language:

```
Node A
Node B
repeat 2 i
ConstructPacket(A, B, "ACK", p)
send(p, A, B, 2*i)
end
```

With these simple commands, the author can create the animation with two nodes, named *A*, and *B*. One packet named *p* is constructed with *ACK* as content, *A* as source and *B* as destination. This file is passed as an argument to the Shikav framework with one more argument as 0 to differentiate between Shikav lecture file and networking script. It gets parsed with *NetworkParser* which translates into Shikav scripts language as:

```
reset("0","0","990","680","networkscriptexample")
text = TextPanel("5","5","460","610","blue","white","green")
g = Geometry("470","5","500","610")
timer = TimerPanel("0.2")
packetBuffer = PacketQueue()
A = EntityNode("A",325,22)
B = EntityNode("B",384,403)
i=ArithElement()
i.setValue(0)
p = Packet("ACK")
p.setSource(A)
p.setDestination(B)
text : A has packet "ACK" to send to B
text : A is sending packets to B
text : <br>
A.send(p,2*i)
i=i+1
p = Packet("ACK")
p.setSource(A)
p.setDestination(B)
text : A has packet "ACK" to send to B
```

```

text : A is sending packets to B
text : <br>
A.send(p,2*i)
i=i+1
text : <br> The explanation ends here !

```

The nodes are placed randomly in the *Geometry* panel as its coordinates get generated randomly between the width and height of the geometry panel. In this case, the authors need not worry about defining the points to place the nodes. Figure 1.3 illustrates the animation of this script.

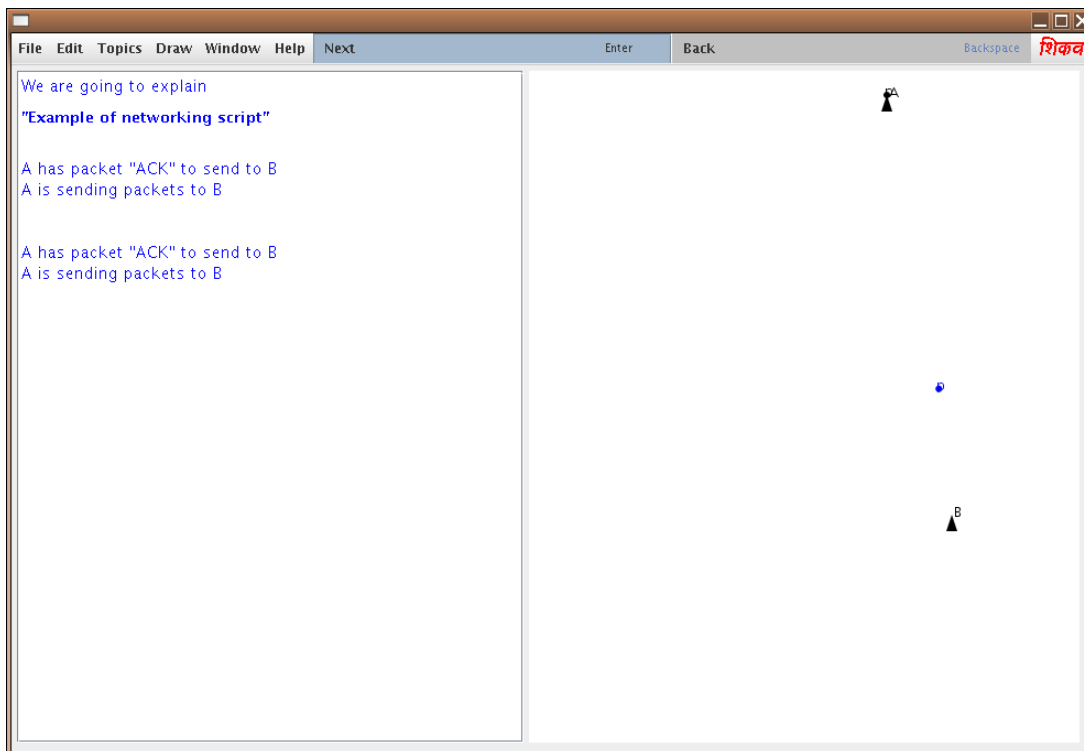


Figure 1.3: The screenshot of animation of node transferring packets in Shikav

More details of this newly defined script language are described in Section 4.3 of Chapter 4.

## 1.6 Thesis outline

In order to help in understanding the Shikav framework in more detail, we describe its description in Chapter 2. This also presents the related tools studied for creating



---

networking electronic lessons. The chapter is followed by the description of case study of a few network protocols such as TCP 3 way handshake[7], TCP slow start[10] and WiFiRe[3] in Chapter 3. Next, chapter 4 explains the implementation details by class diagrams. The enhancement of Shikav script, and a new defined script, and the example of lecture files created for each network protocol are studied in the Case Study. This thesis ends with the conclusion and future extensions.



# Chapter 2

## Shikav and Related work

### 2.1 Design of Shikav framework

There are 3 main actors or users in Shikav, *Teacher or author*, *Student* and *Enhancer* which has different requirements.

1. **Teacher/author:** uses Shikav to create the lesson. They will need to know the script of Shikav which is a high level language description consisting of sequences of commands. There have been many efforts put into Shikav in order to try to make the teacher's task less difficult by making Shikav script richer and easier.
2. **Student:** acts as the end user of the system who will interact with the lesson created by the teacher/author. Their basic requirement has been studied.
3. **Enhancer:** can make extensions to Shikav by adding new modules. Considering this, the system has been designed to have modularity.

The figure 2.1 shows the *use case diagram* of Shikav.

#### 2.1.1 Compile, Issue, Update

1. **Compile time :** Parser operates on each command in the *.lec* script file and creates the *ParsedInfo* object corresponding to each command. All the *ParsedInfo* objects are stored in *ExecutionQueue*. The *ParsedInfo* class contains:

**String symbol** Type of action to be taken. Example: “=” for assignment or creation of new object, “:” for appending text, and “.” for method call.

**String objectName** The Actor object (identify by its name)

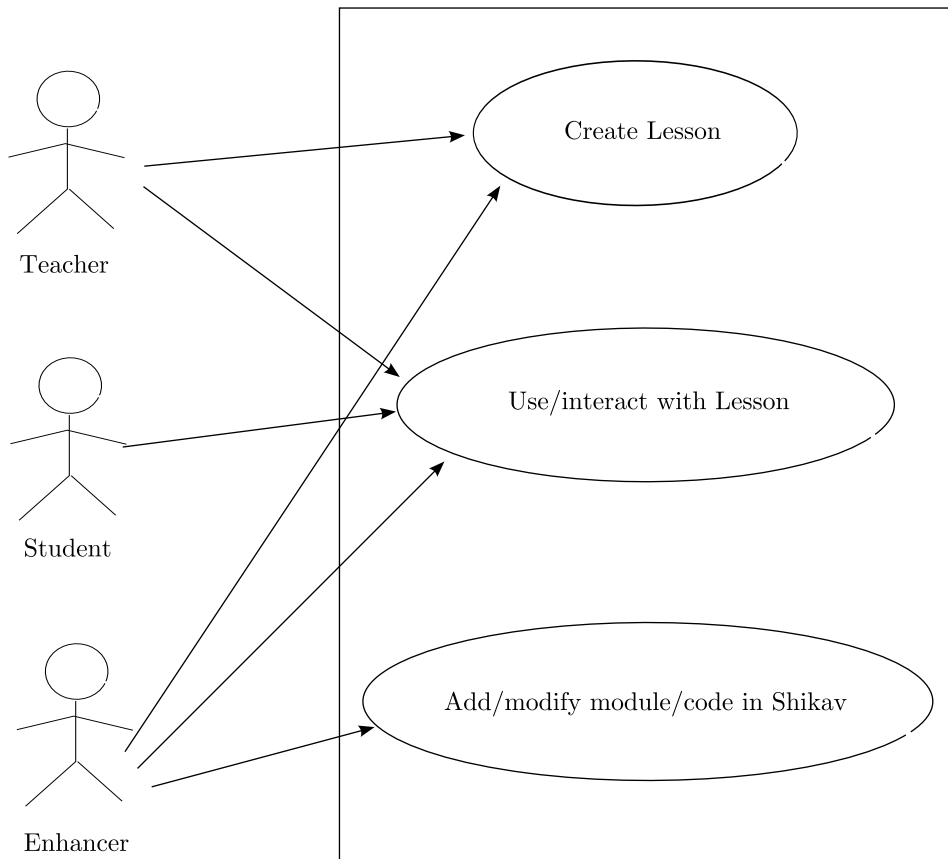


Figure 2.1: Use case diagram of Shikav

**String actorMethod** Action to be taken, depends on the symbol defined above.

Example : “=” for creation of new object or call function, “:” for appending the the parameters to the actor, and “.” for calling the method of the actor.

**String printvalue** Intermediate command.

**String orgLine** Original command (previous command).

**Vector parameters** Parameters passed as arguments.

**int lineNumber** Command line number for error handling.

2. **Issue time:** Each time the users hit the *Next* button, the instruction (one command line in the script file) gets executed. Then the objects get created and registered, and the objects are stored in the hash table data structure as variable *store* that has the pair **{name, object}** as **{key,value}**. The objects are referred by their unique names.
3. **Update time:** During execution of the lesson, the attributes of the created ob-

jects could be changed dynamically. The *update* method of geometrical objects gets called, then the *draw* method gets called for redrawing the objects which get displayed on the *Geometry* panel.

### 2.1.2 Features of Shikav

Shikav has basic inbuilt geometric objects such as Point, Line, Circle, Rectangle, etc. New features have been integrated in Shikav.

1. Double View Editor - It consists of “Code Editor” and “Debugging Editor” which author can write, modify, or view Shikav script for purpose of debugging while the lesson is running. End user can see the source code (script file with highlighted current executed command line) while running the lesson. The source code can also be edited and saved.
2. GeoShikav - (Freehand drawing tool): It is a free hand drawing tool in which end user can do free hand drawing in the panel such as line, point, and circle. The proper shape will be generated and the command script related to those drawing will be added and can be saved in the source script file.
3. Mathematical Expression Parser - Parses complex mathematical expressions such as  $a*(b+c)/(a-c)*d$ . These expressions will be parsed and the equivalent script commands are generated.
4. New features in Shikav language which were added by Aditya Kelkar[5] :
  - List: Act as a group of arithmetic or geometric elements which will be stored in a vector. Some basic operations of list are supported such as add, delete, extract element. With operations such as
    - (a) Map: Given a list of elements, we can apply the function (user defined function or inbuilt function) to the list which affects all elements in the list.
    - (b) Reduce: Given the list of elements, a specific function will be applied to each element of the list and the final result will contain only 1 object in the list.

- (c) Filter: Given the list of elements, a specific function will be applied on each element. Normally, the condition will apply to the elements with the result true or false.
- User defined function - The lesson creator can use inbuilt functions or write their own functions and call them for use at a later stage.
  - “If *condition* Else *statement*” in user defined function - The *condition* and *statement* can only be arithmetic expressions.

## 2.2 Components of Shikav

Figure 2.2 illustrates the main components of Shikav,

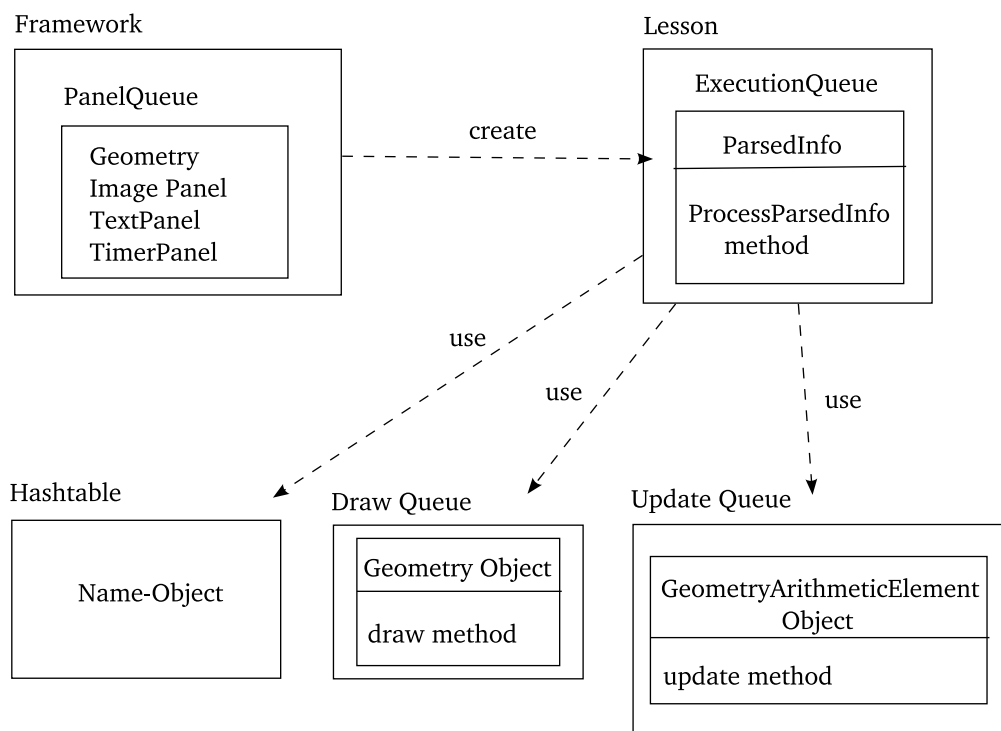


Figure 2.2: Components of Shikav

- *Framework*: The basic part of Shikav, it contains the panel queue, which consists of different panels created such as :
  - TextPanel: for displaying text
  - Geometry: for displaying/running animation

- ImagePanel: for displaying images
- TimerPanel: logical panel for keeping the timer of the framework
- *Lesson*: the main part of Shikav, for each lecture file, an *Lesson* object is created which interacts with Framework to execute each command in the lecture file sequentially. The *Lesson* contains *ExecutionQueue*.
- *ExecutionQueue*: stores the *ParsedInfo* object created corresponding to each command in the script file during the compile time. The current index (*PC - Program Counter*) is used to point to the position of *ParsedInfo* object in the *ExecutionQueue*.
- *Hashtable (store variable)*: stores all objects created, including the GUI object (the actual Framework itself), the registered objects to the Framework. Each object is identified by its unique name as the “key”
- *Vector (work variable) act as Draw Queue*: store all the *GeometryElement* objects that are registered to the *Geometry* panel.
- *Vector (order variable) act as Update Queue*: store all the *GeometryArithmeticElement* objects that are registered to the *Geometry* panel.

## 2.3 Shikav Script language

Shikav has defined its own high level script called “lecture file” with “.lec” in which the syntax has to be followed properly. The teacher/author needs to write a lecture file for each Lesson. Each command in the script file is called as “**Instruction**” which is explained below. The user can interact with the Lesson by using “Next” and “Back” buttons. Pressing the “Next” button executes one complete frame and displays results in the framework. The user can interact with keyboard as well as mouse with the lesson. Various events are captured by application and various handler functions are called.

### Classification of Instructions

**Set the lesson layout** The instruction is of form :

```
reset("0", "0", "990", "670", "Testing Shikav script")
```

This *reset* method applies to the Framework to set the layout of the lesson to be displayed. The Framework will start from coordinates (0,0) (the top left corner), width=990, height=670 and the title of the lesson is “Testing Shikav script”.

**Creation of Panels** New panel objects can be created by giving panel class name along with the parameters such as the coordinates and colors. The name assigned to the object will be used as its identity which can be referenced later.

- **Geometry:** used to display various geometrical objects such as points, lines. User interaction can also be done in this panel due to its mouse handlers event. The option for freehand drawing has also been provided. The textual data can be displayed in this panel with help of *Text* objects. Below is the example on how the instruction should be.

```
G = Geometry("500", "5", "400", "150")
```

During issue time, a *Geometry* object is created and stored in name table (Hashtable) as *G*. The panel would be with width of 400, height of 150 and starting at coordinates (500,5). The panel is also inserted in panel queue of the Framework. At update time, the *GeometryArithElement* object that are registered with update and draw queue of this panel are displayed with *paint* method.

- **ImagePanel:** for displaying various images. The existing image filename is given as parameter. The instruction would be :

```
I = ImagePanel("5", "500", "400", "150")
```

```
I : icon.jpg
```

During issue time, *ImagePanel* object gets created and stored in name table as *I*. The panel would be with width of 400, height of 150 and starting at coordinates (500, 5). This panel also gets inserted in panel queue of framework so that paint gets called at update time.

- **TextPanel:** for displaying various textual contents as the description. Various HTML tags can be used inside the textual data for bold, italic, break lines, headers etc.



```

S = TextPanel("5","5","500","150","black","white","red")
S ! This is an example of TextPanel
S : This is for appending this text to the previous text
- and this is for continuation line

```

During issue time, the *TextPanel* object with name *S* is created with coordinates (5, 5), width=500, height=150, font color=“black”, background color=“white”, and highlighted color=“red”. The text following the special symbol “!” will be displayed in *S* and any existing contents will get overwritten. The “:” is used to append the text contents in the panel. The symbol “-” is used for specifying the continuation of previous instruction. During issue time, the *TextPanel* object with parameters passed is created and is stored in name table as *S*. It is registered in panel queue of Framework in which the *repaint* will get called. This object is retrieved from store for next instruction and the text contents passed get attached to String component of object *S*. During update time, the *paint* method displays textual contents of object *S*.

- *TimerPanel*: for keeping track of clock which is used to specify time delays in the Framework. It is used to give delay in animations. The symbol “:” followed by value parameters specified the delay in seconds.

```

T = TimerPanel(".3")
T : ".2"

```

Same as other panel, during issue time, the *TimerPanel* is created identified by its name *T* and registered in the name table.

**Creation of Geometric objects** : geometric objects such as points, lines, and circles can be created with simple instructions, which get displayed in *Geometry* panel. In the same way as the previous case, new objects can be generated by calling the class names along with the coordinates and identified by its unique name. Example set of instructions:

```

P1 = Point(50,50) // creation of Point with coordinates (50,50)
P2 = Point(10,100) // creation of Point with coordinates (10,100)
text = Text("BeginPoint",P1) // Textual data at specified Point position

```

```

line = Line(P1,P2) // creation of line with 2 points
C = Circle(P1,50) // creation of circle with P1 as center point and radius=50
C.H("blue") // high light the circle with "blue" color (fill the circle color)

```

During issue time, objects are created and stored in name table identified with their name. These objects are also registered with update and draw queues of *Geometry* panel. The pointers to parameters passed are set from the created objects. At update time, the coordinates of points and lines gets updated with the values of pointed parameters. The objects are redrawn with new values in the Geometry panel.

**Movement of objects** Geometrical objects can be moved providing the new values of parameters. The *TimerPanel* is needed for the delay in second of movement.

```

T = TimerPanel(".3")
P1 = Point(50,50) // creation of Point with coordinates (x1=50,y1=50)
C = Circle(P1,50) // creation of circle with P1 as center point and radius=50
T : ".1"
P1.moveby(5,5) // move the point P1 to (x2= x1+5= 55, y2= y1+5= 55)
T : ".1" // give delay of "1" second
P1.moveby(5,5) // move the point P1 to (x3= x2+5= 60, y3= y2+5= 60)

```

During update time, as the point move, the circle *C* is also moved accordingly.

**Arithmetic Instructions** Examples of standard symbols provided in the expression are addition (+), multiplication (\*), subtraction (-), division (/) along with comparison operators (==, <, >), which can be used to performed basic arithmetic operations. It also supports trigonometric functionalities.

```

a = b + c
p = cos a

```

All these arithmetic operations have two operands: ArithElements and an operator as String. At issue time, these objects get created with operand pointers to the parameter ArithElements. During update, the result value of actual operation is

calculated with updated operand values. So the value of result object is consistent with operand values of the operation.

**Groups** These are used to bind various objects as a single set and operations can be performed on all these objects at same time. This can be a set of heterogeneous objects. Simple methods like move, hide, highlight etc can be performed on all the component objects of Group.

```
grp = Group{P,L,T} //P is Point, L is Line, T is Text
```

At issue time, the parameter objects are extracted from name table. In case they are not created earlier, the default creation as doubleconstants or String object is performed. These objects get inserted in component Vector of Group object and is stored in name table as grp. At update time, the update method of Group calls update on all its components.

**Macros** These are used to define various values with simple names which can be used further in instructions. These gets preprocessed and replaced in actual instructions.

```
\macro(red,<font color=red>)
```

These commands get stored in a table with the mapping of symbol and actual values. All the commands are preprocessed to figure out any of the macro symbols and get replaced with values from this macro table.

**Scenes** Scenes are the topics which are going to be discussed in the lesson. Every lecture file should have a scene which will act as title of lesson.

```
\scene(Name of scene)
```

For example:

```
\scene(Theorem of Pythagoras)
```

**Frames** User interaction is achieved by specifying a blank line command which indicates termination of one frame where the lesson execution stops for input from user. The user can play with the geometric diagrams with help of mouse or navigate for next or previous frames. At issue time, when such blank command is detected, further execution stops and control is passed to user. Whenever Next interrupt is obtained from user, the issue cycle for next frame begins.

## 2.4 Related tools

### 2.4.1 JAWAA

JAWAA [9] stands for **Java And Web-based Algorithm Animation**, developed at Duke University by Professor Susan Rodger and Duke undergraduates. It is a scripting language enables authors to create animations that can be able to run on web. JAWAA provides graphical editor, the authors can drag and drop the objects to be displayed in the animation, then the corresponding JAWAA script gets generated and the users can run it as Java Applet.

### 2.4.2 Samba and Polka

POLKA [1] is an open source animation system which provides the possibility for users to build algorithms and program animations. The main aim for the system was to build animations of parallel programs and computations. POLKA is written in C++. Along with Polka, SAMBA [2] is an interactive front-end which is an animation interpreter that reads one ascii command per line, then performs that animation directive. The Java version of Samba called, Jsamba which is an in progress project.

### 2.4.3 Compare with Shikav

Shikav has some additional features such as :

- Stage Metaphor: Shikav provides the facility of creating electronic lessons with the blackboard-like interface as taught by the teacher in classroom. Each lesson can be divided in to various panels which can be used for various purpose such as a text panel for explaining in text, a geometry panel for explanations using animation.
- Cross linking and referencing: Shikav has handled this feature effectively. The students can link from one lesson to another.
- Advanced Patterns: These features are being developed by some students in IIT-Bombay as their B-Tech projects, as exempld, they built the recursion tree for any algorithm that use recursion.

#### 2.4.4 Tool for creating networking lesson - JSIM

JSIM [6] is web-based simulation developed purely in Java by John A. Miller, Andrew F. Seila and Xuewei Xiang at University of Georgia. The simulations created include animation and statistical data. JSIM provide the graphical interface editor which allows authors to create the simulation. Once the author saves the file, the correspondent Java files needed to run the simulation are generated. There are three foundation layers of group of packages in JSIM as shown in Figure 2.3.

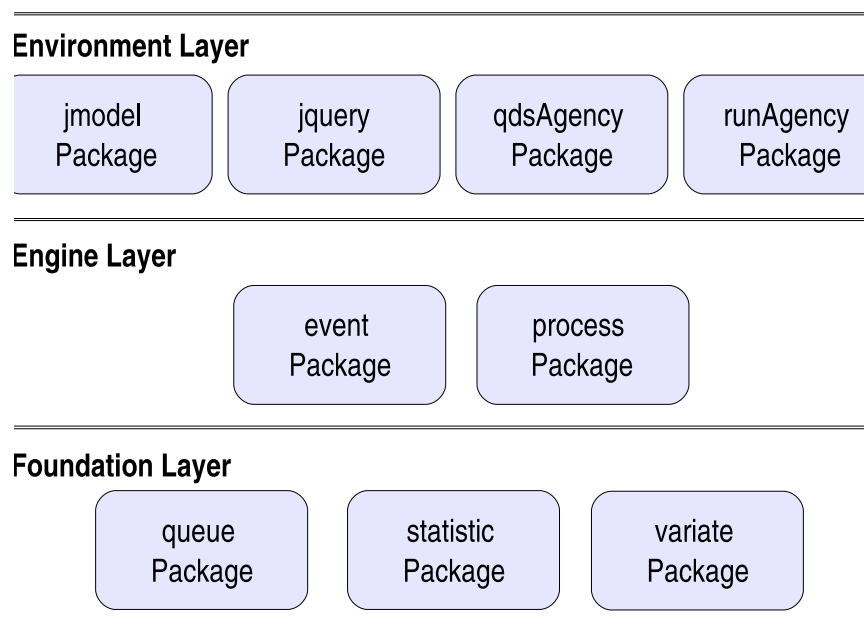


Figure 2.3: Layer of packages in JSIM

- The bottom layer, **Foundation layer**, is for collecting statistics of the simulation, providing the difference variety of queues such as FIFO, LIFO, and a variety of random variates, for different distribution functions for service time, arrival rate, etc.
- The middle layer, **Engine layer**, the *Event* package provide the event-scheduling simulation model and the *Process* package giving the model of interactive-process paradigm.

- The top layer, **Environment layer**, provides the flexibility to the simulation, for creating the visual designer (graphical interface) for the simulation, and for connecting to the database.

The simulations created in JSIM are mainly for queueing networks, the model has various types of nodes such as *Source*, *Server*, *Facility*, *Sink*, *Signal*. From the JSIM graphical interface, the authors can select:

- **Source**: for generating entities (SimObject, it is defined in JSIM as a single simulation entity such as customer of the system or requests) depending on the random inter-arrival time (number of entities to be created per unit time). The distribution function is defined by authors, for example, the inter-arrival time is Poisson with mean  $\lambda = 3$ .
- **Server**: for providing the service units to the entities arriving at the node depending on service time distribution defined by authors (e.g: the service time is exponential with mean  $\mu = 4$ ). If all the service units are busy, the client entities will be lost as the server doesn't allow the entities to be queued.
- **Facility**: derives from the Server class which has similar features but provides for the entities to be queued. This means the entities will be enqueued if all the service units are busy. The authors can set the queue type to be FIFO, LIFO, PriorityQueue but presently, only FIFO (using LinkedList) is supported.
- **Sink**: for consuming the entities, in contrast of *Source*. Generally, it is the exit of the system.
- **Signal**: for decreasing or increasing the service units in the server/facility. For example, when the queue length of the facility is large, the signal will increase the service units in the Facility in order to restraint the queue overflow.

Nodes are connected with the *Transport* which act as the edges in the graph. It sends entities from one end to another end, the speed of the transport is controlled by the *adjustSpeed* method and the entities are moved by the *move* method.

### Example of simulation in JSIM

Here is an example of a simulation in JSIM for a food court in McDonalds, the customers

will wait in the queue if the servers are all busy. The figure 2.4 is the screen shot of the simulation.

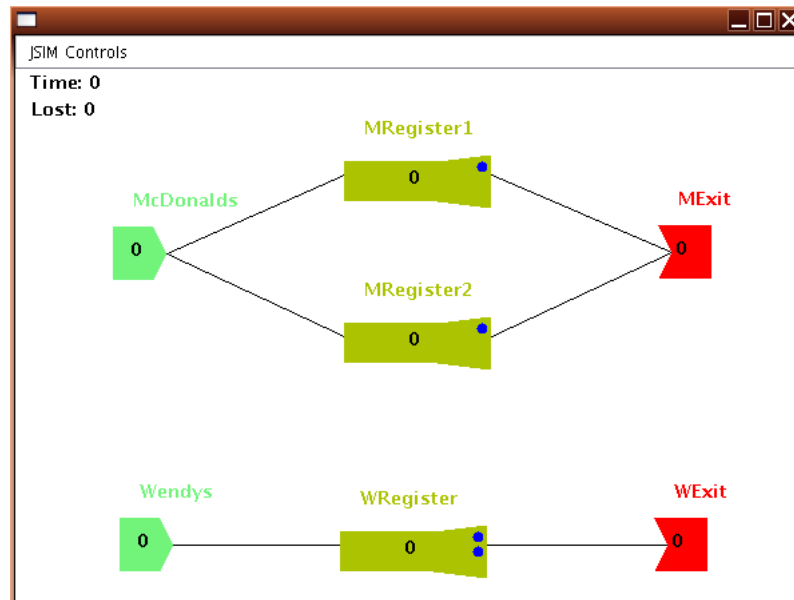


Figure 2.4: The screen shot of simulation of customers in MC Donald

We have thoroughly seen the details of Shikav, its infrastructure, its working phases and its features. The study of JSIM shows that the tool was mainly created for the purpose of creating simulation showing queueing networks. Our purpose was not to show the simulation of queueing networks but to explain the working flow of a particular network protocol. Hence JSIM has not been taken into the solution to our problem.

### 2.4.5 Summary

We have chosen Shikav since it has the following advantages:

- Shikav has been designed with modularity, which provides the scope to extend its source code.
- Shikav uses the *reflection* technique of Java. The classes added into Shikav code can be used from the script file and its methods can also be called within the script file itself.
- Shikav also provides the facility of creating electronic lessons with the blackboard-like interface as taught by the teacher in classroom. Each lesson can be divided in

to various panels which can be used for various purpose such as a text panel for explaining in text, a geometry panel for explanations using animation.

- Cross linking and referencing has been handled effectively by Shikav. The students can link from one lesson to another.

In order to understand the new features required to be added into Shikav enabling creating of animations explaining network protocols, a few network protocols such as TCP 3 way handshake, TCP slow start and WiFiRe have been studied and presented in the succeeding chapter.



# Chapter 3

## Examples for Shikav Extension

Various basic protocol steps in networking have been thoroughly studied in order to understand various features and constructs that need to be incorporated into Shikav. Some examples are given in this Chapter. The first two are simple examples while the third describes a more complex protocol.

### 3.1 TCP 3 way handshake

To create the connection in TCP (Transmission Control Protocol) [7], the initial step is called “TCP 3 way handshake”. The client host requests connection to the server host by sending *SYN*. When server host grants the connection, it sends *SYN\_ACK* back to client host, the latter sends *ACK* back to inform the former that the connection is established.

- 1) A --> B SYN my sequence number is X
- 2) A <-- B ACK your sequence number is X
- 3) A <-- B SYN my sequence number is Y
- 4) A --> B ACK your sequence number is Y

As step 2) and step 3) can be combined into a single step, hence called “TCP 3 way handshake”. Figure 3.1 illustrates this processes. In this case, the features needed are:

- Node: represents the Client host or Server host. The method required are *receive* and *send* which needed packet, source and destination as its parameters.
- Packet: represents the packet to be sent between nodes, the packet will have source, destination, and content

Section 4.4.1 has the lecture file created for this case.

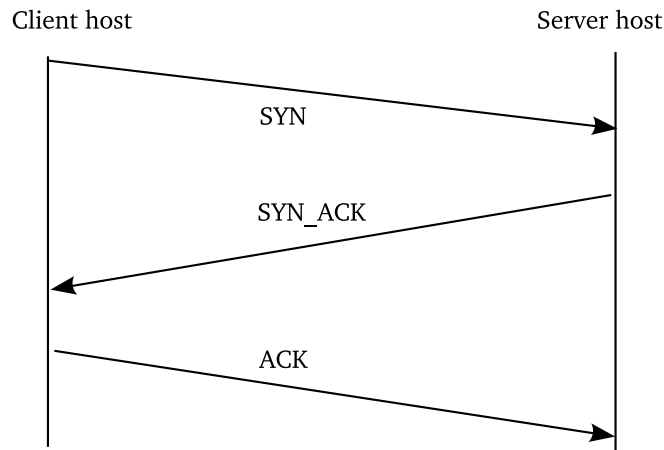


Figure 3.1: The process of TCP 3 way handshake (TCP initial connection set up)

## 3.2 TCP slow start

Also known as “Fast retransmission, fast recovery” [10] algorithm used as one part for congestion avoidance (congestion control, collision avoidance) phase in TCP. It works together with congestion avoidance to avoid sending data more than the capability of network transmission. There are two variables required to be maintain for each connections, “*congestion window, cwnd*” and “*slow start threshold size, ssthresh*”. A slow-start algorithm consists of two distinct phases: the exponential growth phase, and the linear growth phase. Assume there are 2 hosts *A* and *B*, the basic slow start algorithm works as follows:

```

set cwnd = IW (Initial Window) = 1 or 2
set ssthresh = 65535
send 1 packet
receive ACK
cwnd = cwnd * 2 = 2
send 2 packets
receive ACK
cwnd = cwnd * 2 = 4
send 4 packets
receive ACK
...
cwnd = cwnd * 2 = ssthresh
  
```

Entering “congestion avoidance” phase in which  $\text{cwnd} = \text{cwnd} + 1$ , if time out occurs

```
set ssthres = cwnd/2
```

```
set cwnd = IW
```

and then re-entering slow start phase. The slow start algorithm begins in the exponential growth phase initially with 1 or 2 segments (depending on the variant) for the congestion window size, and increases this size exponentially ( $\text{cwnd} = \text{cwnd} * 2$ ) until a predefined slow-start threshold is reached. Once the threshold is reached, the congestion window size is increased linearly ( $\text{cwnd} = \text{cwnd} + 1$ ). When segment acknowledgements are not received, it means time out, the slow-start threshold is set to half of the current congestion window size, and the algorithm restarts.

In this case, the features needed are :

- Node: represents the network hosts. The methods required are *receive* and *send* which needed packet, source and destination as its parameters.
- Packet: represents the packet to be sent between nodes, the packet will have source, destination, and content
- For loop: as the congestion window size keeps on increasing till the threshold, hence many rounds of sending packets are needed.
- If-Then-Else conditional statement: is used to test on the content of received packets in order to define corresponding replied action.

Section 4.4.2 has the lecture file created for this case.

### 3.3 WiFiRe

WiFiRe (Wireless Fidelity for Rural Extension) [3] is an approach that use most of the features of WiMAX (802.16) as MAC layer with some modification and it continues to use the existing Physical layer of WiFi (802.11b), but unlike WiFi, WiFiRe will be able to provide long range communications with high bandwidth by replacing the MAC mechanisms of existing WiFi (802.11b), so that the range of communication can be extended up to 15–20km, with low cost and easy availability of the chipsets. WiFiRe divides

the whole area into sectors, each has its own PHY layer which will be controlled by one single MAC layer. Each sector will have one Base Station. Only one frequency will be used in WiFiRe for uplink and downlink. Each sector is allocated frequency based on Time Division Duplex- Multi-sector TDM (TDD-MSTDM). Figure 3.2 shows the high level network configuration of WiFiRe.

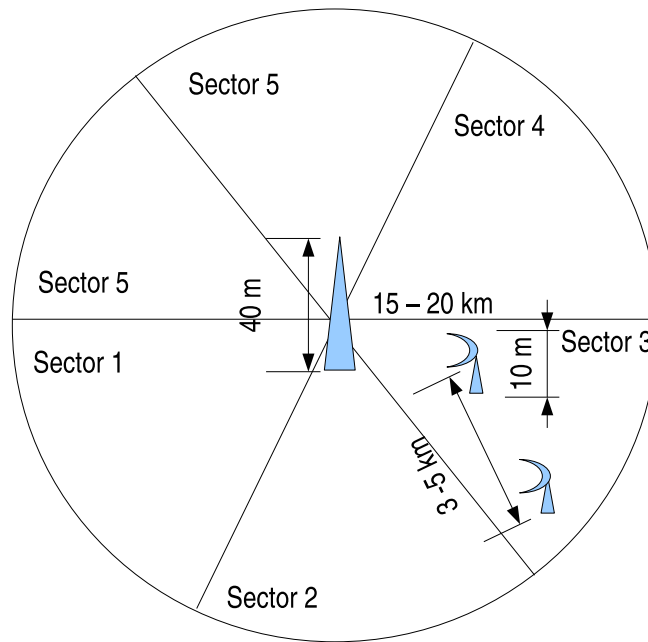


Figure 3.2: High level network configuration for WiFiRe

## Basic steps in WiFiRe

Below are the basic protocol steps in WiFiRe:

1. **Network Initialization:** The association between a ST and a System S is static. But deciding on which BS to use for communication is done through ranging and registration. An ST should communicate with only one S.
2. **Ranging:** New and Unsynchronized STs are allowed to range and register. On power on, ST will listen for beacon sending periodically from system S, with some specific ranging slots defined in the beacon. ST notes down all necessary information and sends ranging requested at the defined slots. Once ST get the ranging reply from system S, it goes into *Ranging* in order to synchronize the clock and other

physical parameters with the system S. It is also performed periodically to keep in synchronization with S. In this process S assigns ST two connection-ID's (CIDs) called the *Primary CID*, which will be used further for exchanging management services, and *Basic CID*, which will be used further for periodic ranging requests. Figure 3.3 illustrate the sequence diagram of ranging.

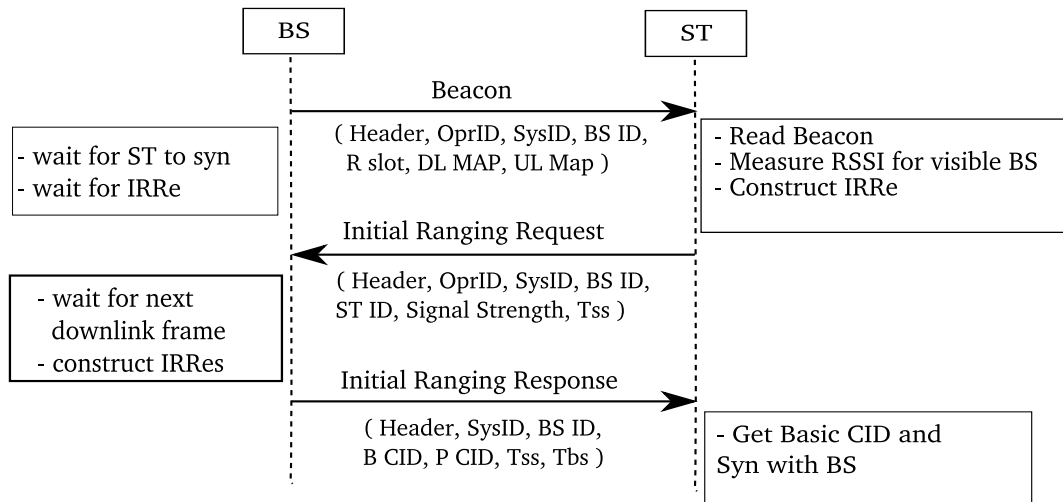


Figure 3.3: Sequence diagram of initial ranging steps

3. **Registration:** ST will send registration request to the system S which will send a reply back. Once this is done, it means that ST has been added to system S and ST has been assigned one IP address. During this process, ST and S exchanges operational parameters and capabilities. Registration process enable ST to get IP address in order to set-up provisioned connections. Figure 3.4 illustrate the sequence diagram of registration.
4. **Connection Management:** Once the registration is done, ST can request for any number of further connections. The MAC is connection-oriented and data flow between BS and ST occurs as per the service flow type associated with that particular data flow. Service can be added, modified or deleted. There are three types of data flow, **UGS** (Unsolicited Grant Service), the Figure 3.5 illustrates the sequence diagram of UGS data connection. **rtPS**(real time Polling Service) and **nrtPS**(non-real time Polling Service), Figure 3.6 is the sequence diagram of rtPS and nrtPS data flow.

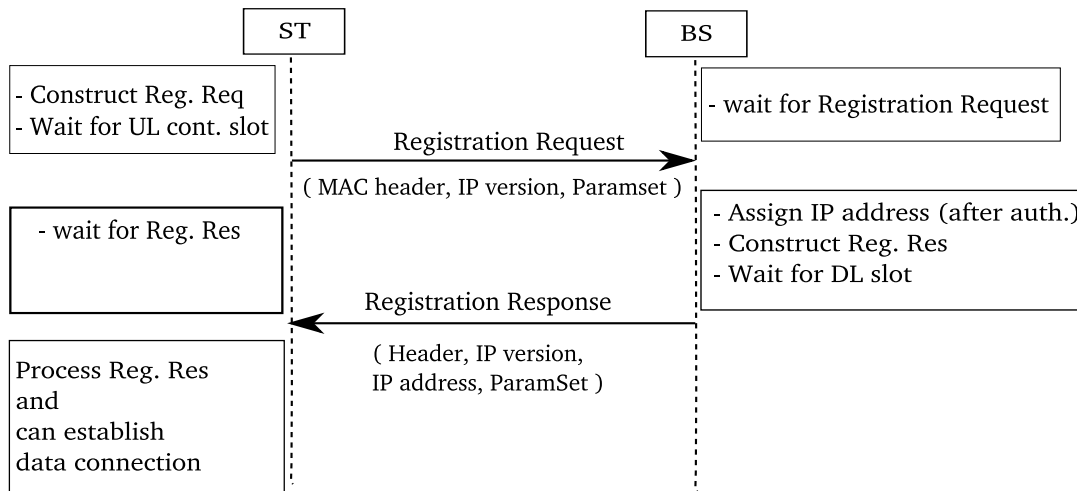


Figure 3.4: Sequence diagram of registration steps

In this case, the features needed are:

- Node: represents the Base Station or Subscriber Station which has method *receive* and *send* which needed source and destination as its parameters.
- Packet: represents the packet to be sent between nodes. The packet will have source, destination, and content
- Beacon: represents the signal broadcasting from BS to ST. The animation would show from one starting point spread to the sector.
- If-Then-Else conditional statement: is used to test on the content of received packets in order to define corresponding replied action.

Section 4.4.3 has the lecture file created for this case.

### 3.4 Summary

After studying these protocols, it was observed that it would take a lot of effort if we were to create the animations explaining the above protocols using only the current features of Shikav. As packets have to be sent from source to destination, the authors need to carefully calculate the coordinates of the packets to move and also they have to repeat the same step of moving the packet until it reaches the destination. In the case of

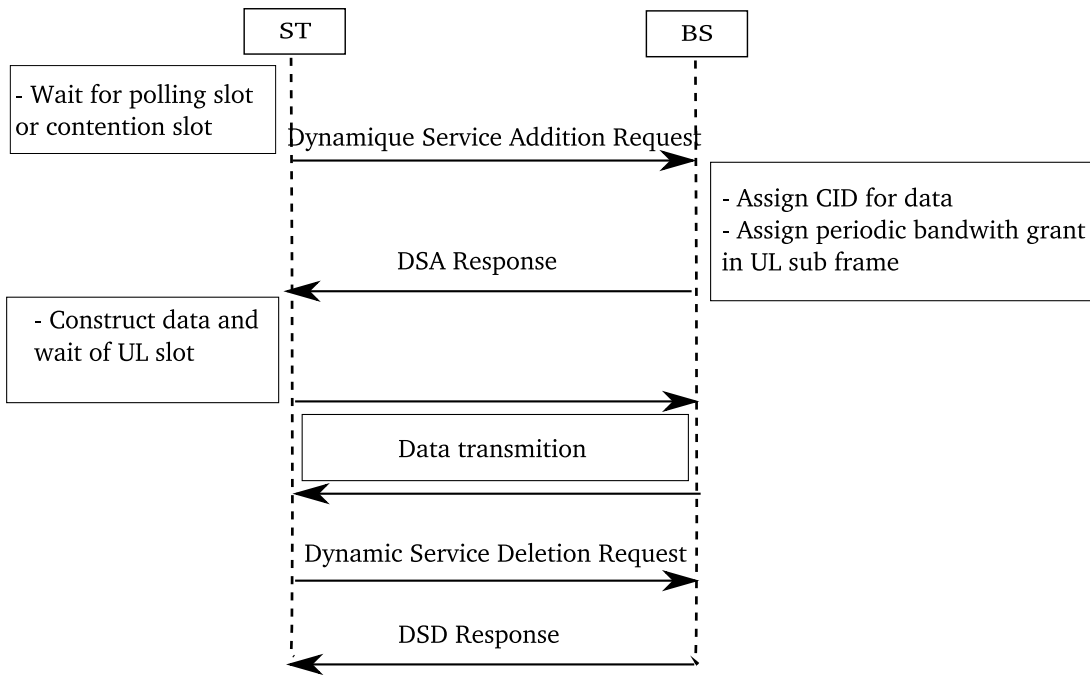


Figure 3.5: Sequence diagram of UGS data connection

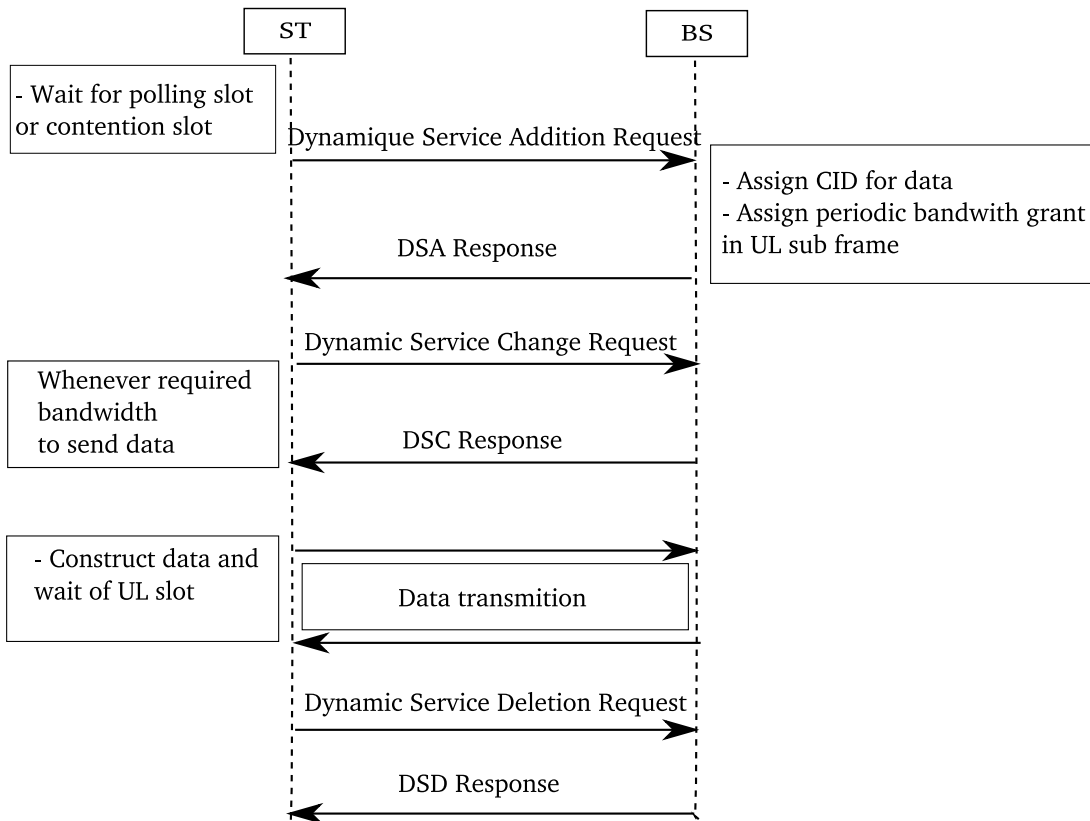


Figure 3.6: Sequence diagram of rtPS and nrtPS data connection

TCP slow start, the number of packets to be sent keep on increasing exponentially, and it would take a lot of effort to write the Shikav script language. Hence, after the study of

the above protocols, the features to be added into Shikav have been determined as:

1. Node
2. Packet
3. Beacon (control packet)
4. Repeat (for loops)
5. If-Then-Else conditional statement

We believe that these are sufficient to support the creation of networking animations. The designs and implementation of these features are clearly explained in next chapter. Also, a newly defined script language at one more level higher than Shikav script for the purpose of creating animations in networking is also described in succeeding chapter.



# Chapter 4

## Implementation details of Shikav Extension

Referring to the various protocols presented in Chapter 3, the features *Node*, *Packet*, *Beacon*, *repeat loops*, *if-then-else conditional statement* are needed. *Node*, *Packet* are commonly required objects that are used in explaining a particular network protocol. The *Node* is required to be added to represent each node in the network that has the behavior as *send*, *receive*. The *Packet* is needed for creating packets that can have content, source and destination. The *Beacon* is used for creating a beacon starting from a source and spreading to an area limited by the range and the angle. The *repeat loops* is required in case an author wants to create many numbers of packets or send many number of packets. The loop is defined and implemented in the new script as detailed in Section 4.3 and not with Shikav script language.

### 4.1 Class Diagram of new added classes

Figure 4.1 illustrates the relations between the new classes added to existing code of Shikav.

- **Entity**: an abstract class derives from *GeometryElement* class in order to be able to get displayed and gets registered in *Geometry* panel. It is the base class for representing node which provide the basic definitions of the behaviors of node. Below are the description of few important method:

**draw(Graphic g)** Defining the shape of the node. The shape to be used as for now is “triangle”.

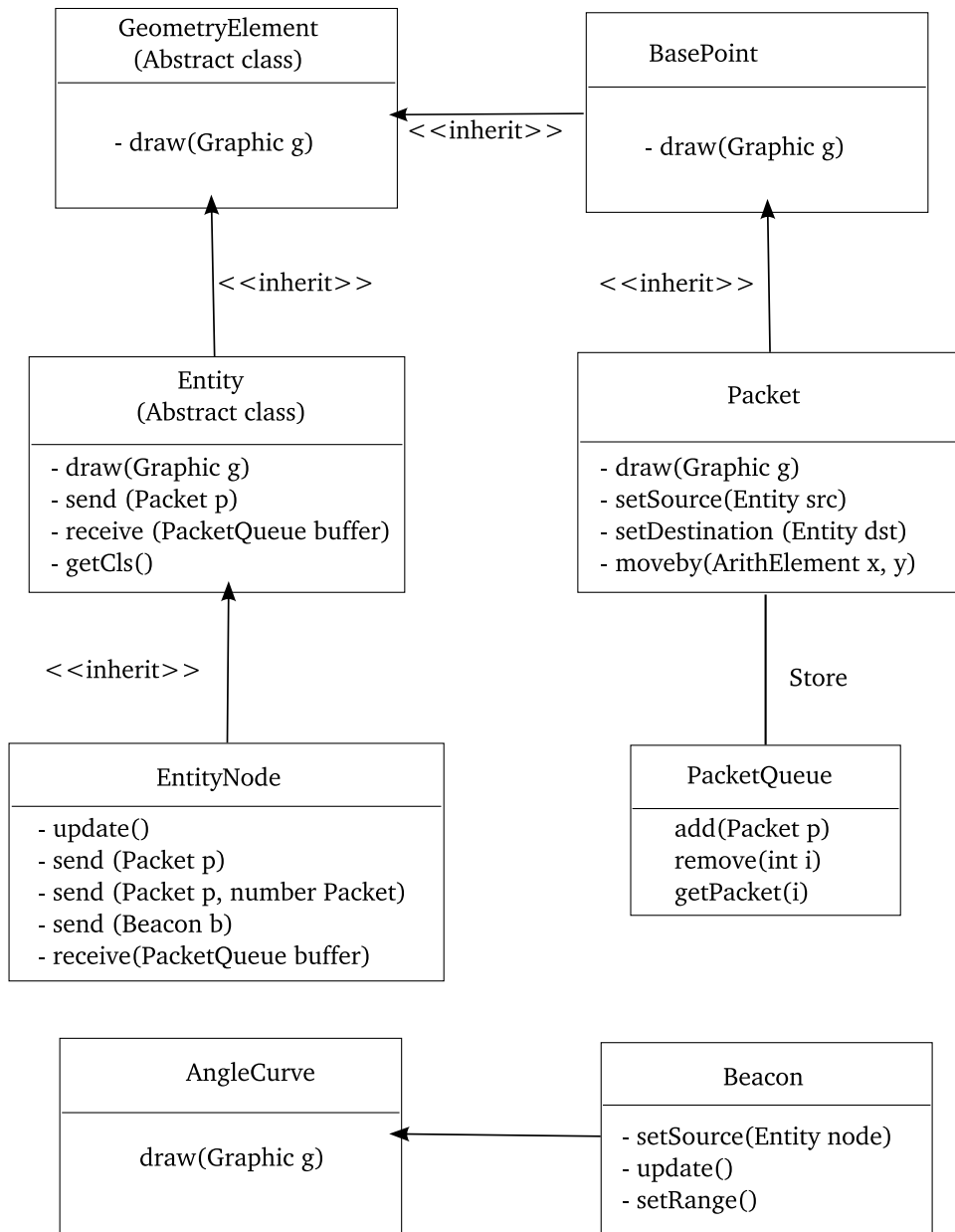


Figure 4.1: Relation between newly added classes

**send(Packet p)** An abstract method to be defined by the derived classes.

**receive(PacketQueue buffer)** An abstract method to be defined by the derived classes.

**getCls()** returns “Entity” as its class name. This is for identifying the type of class when the object is retrieved from the name table.

- **EntityNode**: derives from *Entity* class. It represents the node in the network and it implements the abstract methods defined by the based class.

**EntityNode(String name, ArithElement x, ArithElement y)** The constructor of the class.

**send (Packet p)** is the method to send packets from one node to another node.

The source of the packet is the node who provoke this method, and the destination is defined in the packet itself.

The animation of the packet transferring is done by calling the method *moveby()*.

**receive(PacketQueue buffer)** The method to receive the packet. It returns the *Packet* object, if there is no packet received, the object to be returned is **null**.

**send(Beacon b, String radius)** The method to send the beacon. The radius of the beacon are needed to be specify because the beacon is sending as spreading signal in a sector or a particular area defined by a source point and the points of boundary

**update** The method to be called during update time of Shikav, it is used to updating the value of (x,y). The coordinate (x,y) taken as parameters from the constructor are used as the top vertex of the triangle. By default, it takes (x - 5, y + 15) for bottom left vertex and (x + 5, y + 15) for the bottom right vertex.

- **Packet**: derives from *BasePoint* class as it is considered to be displayed in the form of ellipse with given radius. The source and destination of the packet are taken as the parameters of this class constructor. It also has “type” of the packet.

**Packet(Entity src, Entity dst, String payload)** The class constructor.

**setSource(Entity src)** set the source of the packet.

**setDestination(Entity dst)** set the destination of the packet.

**setRadius(String radius)** set the radius of the ellipse.

**update(ArithElement xnew, ArithElement ynew)** the method to be called during the update time when the framework get repaint.

- **Beacon(Entity src, String content, ArithElement range, ArithElement angle)**: derives from *AngleCurve* class in order to get the angle curve shape. The range parameter is needed for spreading the range of the Beacon for the purpose of animation, the angle is counting from the horizontal line.

## 4.2 Enhancement in Shikav

We have seen the features of Shikav in Section 2.1.2, more to those, new features have been added as described in following sections.

### 4.2.1 Object creation

With the new added classes as described above, the nodes, packets can be created by calling the class name provided the necessary parameters. For example:

```
Client = EntityNode("ClientNode", 50, 100) // Creating a node named ClientNode
                                           // with coordinates (50, 100)
                                           // in the Geometry panel

Server = EntityNode("ServerNode", 90, 100)

p = Packet("SYN") // Creating a packet with content "SYN"
p.setSource(Client) // set Client as source,
p.setDestination(Server) // set Server as destination
Client.send(p) // send a packet p
```

With this script, two nodes has been created, with the name ClientNode and ServerNode with (50,100), (90,100) coordinates respectively. A packet with content SYN, Client as source and Server as destination. The Client node send the packet p to Server node.

### 4.2.2 Shikav script

#### IF-ELSE conditional statement

In order to differentiate the action to do corresponding to the packet received, the condition statement has to be defined and add into Shikav script language. As described in Section 2.3, the IF-ELSE condition was defined only in user defined function, and only with arithmetic expressions as example `f = func(x,y){if x<y then x+5*y else x-2*y}` the IF-ELSE condition was not yet defined to be used outside user defined function.

The IF-ELSE conditional statement is defined to be used outside user defined function and the statement of the condition are defined to accept not only arithmetic expressions but also other command such as creation of object, invoking method. The grammar rule of the IF-ELSE conditional statement is as following:

```
IF condition THEN expr_list ELSE expr_list ENDIF
```

The *condition* is of arithmetic expression. The *expr\_list* is a set of expressions separated by **semicolon** “;”. Following is the example of the use of this conditional statement:

```
p = nodeA.receive()
a = p.content("SYN")
if a==1 then p1=Packet(nodeA, nodeB, "ACK"); nodeA.send(p1);
else timer:"0.2"; endif
```

The above fragment is to test if the content of received packet is "SYN" then construct packet with content "ACK" and send to *nodeB* else delay the timer of the animation by 2 seconds.

### Detailed implementation

The grammar rules for If-Then-Else conditional statement are added into Java cup grammar file named “inter.cup”, the grammar added are as follow:

- `expr_part ::= state SCOLON | copy SCOLON { :Symboltable.store("\n",1); : } | SCOLON`
- `state ::= IF cond_name:l THEN expr_list:e1 ELSE expr_list:e2 ENDIF`

In the symbol file “inter.lec”, the definition of *DSTR* has been modified to

```
[a-zA-Z][a-zA-Z0-9]*[ ]*:[^\r\n;]*[\r\n]* {
    return sym(sym.DSTR,yytext());
}
```

## 4.3 Network script language

A higher level of script has been defined providing facility to the authors who want to create lessons explaining networking. The authors can create a node and define its behavior using simple networking terms. The details of the key words of the script is described below:

**Node**  $\langle name \rangle$  This command is used to create node. It gets translated into Shikav script as `EntityNode("name", x, y)`, the value of  $x$  and  $y$  is generated randomly between the value of width and height of the *Geometry* panel.

**Node**  $\langle name \rangle \langle x \rangle \langle y \rangle$  This command is same as previous command but give an option to author to set the coordinates of the node in the display panel. It gets translated into Shikav script as `EntityNode("name", x, y)`, but in this case, the value of  $x$  and  $y$  is fixed by the author.

**ConstructPacket** ( $\langle content \rangle$ ,  $\langle packetName \rangle$ ) It defines the packet construction that gets translated into Shikav script as

```
packetName = Packet ("content")
text : "source" has packet "content" to send to "destination"
```

**send** ( $\langle packet \rangle$ ,  $\langle source \rangle$ ,  $\langle destination \rangle$ ) is used to send the packet that gets translated into Shikav script as

```
packetName.setSource(source)
packetName.setDestination(destination)
source.send(packet)
text : "source" sends packet to "packet.destination"
```

**send** ( $\langle packet \rangle$ ,  $\langle source \rangle$ ,  $\langle destination \rangle$ ,  $\langle number\_of\_packet\_to\_send \rangle$ ) is used to send the packet with the last parameter as the number of packet needed to be sent. This command gets translated into Shikav script as

```
packetName.setSource(source)
packetName.setDestination(destination)
source.send(packet, number_of_packet_to_send)
text : "source" sends packets to "packet.destination"
```

**ConstructBeacon** ( $\langle source \rangle$ ,  $\langle content \rangle$ ,  $\langle range \rangle$ ,  $\langle angle \rangle$ ,  $\langle beaconName \rangle$ )

It defines the beacon construction that gets translated into Shikav script as

```

beaconName = Beacon (source, range, angle)
beaconName.setContent("content")
text : "source" has beacon to send

```

**send** (*< beacon >*, *< source >*) is used to send the beacon from the source node with the range specified when the beacon is constructed, this command gets translated into Shikav script as

```

source.send(beacon)

```

**delay** *< seconds >* is used to set the delay in the animation. This will get translated into Shikav script as

```

timer : "seconds"

```

**Repeat** *< constant >* *< variable >* *< loopbody >* **end** This loop works as for loops which is used to eliminate the effort in writing same command script so repeatedly, the commands inside this loops are defined of above type. The example of the use of this command is as following

```

repeat 2 i
    ConstructPacket(source, destination, "content", p1)
    send(p1, source, destination, 2^i)
end

```

The corresponding Shikav script is as following

```

i=ArithElement()
i.setValue(0)
p1 = Packet("content")
p1.setSource(source)
p1.setDestination(destination)
text : source has packet "content" to send to destination
text : source is sending packets to destination
text : <br>

```

```

source.send(p1,2^i)
i=i+1
p1 = Packet("content")
p1.setSource(source)
p1.setDestination(destination)
text : source has packet "content" to send to destination
text : source is sending packets to destination
text : <br>
source.send(p1,2^i)
i=i+1

```

**IF** *< condition >* **THEN** *< statement >* **ELSE** *< statement >* **ENDIF** is used to test the content of the packet receive in which the **condition** is of the form “*< nodename > RECEIVE < content – of – packet >*”. And the **statement** can be command of the above type. This will get translated into Shikav script as

```

pcnt = nodename.receive(buffer) \\ the buffer is the packet buffer
bcnt = pcnt.isPayload(content) \\ this returns the value 1 or 0
if bcnt ==1 then [process statements]
else [process statements] endif

```

**Title** “*< lesson\_title >*” This command is used to give the title of the lesson in an introduction line of the animation. It gets translated into Shikav script as

```

text : we are going to explain lesson_title

```

In Shikav framework, geometrical objects and text get displayed only on *Geometry* panel and *TextPanel* respectively, hence these two panels need to be created. The *Timer-Panel* also need to be created for setting the delay in the animation. For this purpose, following commands are added by default into the lecture file “.lec.net” translated from the network script language file.

```

reset("0","0","990","680","lesson title")
text = TextPanel("5","5","460","610","blue","white","green")
g = Geometry("470","5","500","610")
timer = TimerPanel("0.2")

```



### 4.3.1 Integration of new script language into Shikav

This higher script will remain with “.lec” extension and is passed as an argument to Shikav framework but the user needs to provide the second argument as number “0”. In the *main* method of *shikav* class, test if there are two arguments, and if the second argument is number “0”, set type of file to “network”. In the *executeFramework* method of the *Framework* object, test if the passed file type is “network” then call *NetworkParser* which writes the translated command into normal Shikav script and write into intermediate file with “.lec.net” extension. Then set the file type to “org” and changed the file pointer to the current “.lec.net” extension and the processing of this file will be done according to usual Shikav working phases and the lesson get generated. Figure 4.2 illustrates this process.

### 4.3.2 How does “NetworkParser” work?

Referring to the description in Section 4.3.1, Shikav can differentiate the file type depending on the second argument, if the file type is of network script language then *NetworkParser* is called before passing to the normal working phase of Shikav. The detailed working flow of *NetworkParser* is as following:

- Read from the input file
- If the command starts with one among the predefined key word such as “Node, ConstructPacket, send, receive, delay, title, repeat”, tests if the arguments are properly given, if yes, for each case, translates the command into corresponding Shikav script language as described in Section 4.3 and write them into “.lec.net” file.
- There is a special case for the key word *repeat* as it has to read a block of command. In this case, the parser will read each line from the file buffer until it reaches the key word *end* and store in a vector as repeat body and gets parsed as normal command of the above type.
- Set the file type to “org” and passed it to the existing phase parser of Shikav.

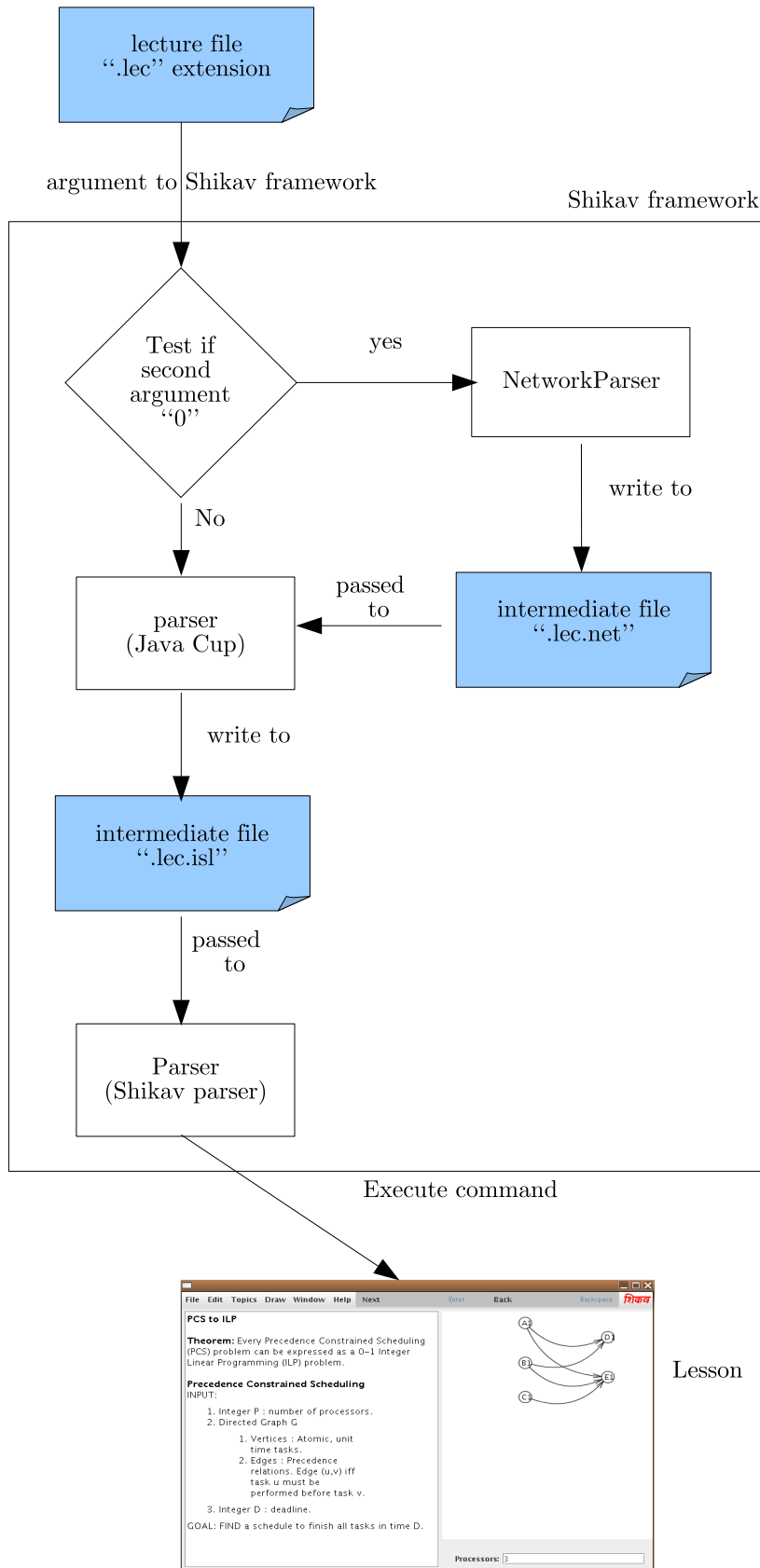


Figure 4.2: The flow of Shikav

## 4.4 Examples

The lecture files created corresponding to each network protocol described in Chapter 3 are given below. The file is first written in the newly defined “network script language” and then gets translated into the Shikav script language. Appendix A gives the resulting Shikav script after the lecture file is processed by *NetworkParser*.

### 4.4.1 TCP 3 way handshake

The procedure has been already explained in Chapter 3, the animation created would comprised of two hosts, named *HostA*, *HostB* and three packets are to be created with content of *SYN*, *SYN\_ACK*, *ACK*. The lecture file created in new defined language (Networking script) is as follows:

```
Title "TCP 3 way handshake"
Node HostA
Node HostB
ConstructPacket(HostA , HostB , "SYN" , p1)
send(p1, HostA, HostB)
if (HostA receive "SYN") then
ConstructPacket(HostB,HostA,"SYN_ACK",p2)
send(p2, HostB, HostA)
else
delay(2)
endif
if (HostB receive "SYN_ACK") then
ConstructPacket(HostA, HostB, "ACK", p3)
send(p3, HostA, HostB)
else
delay(2)
endif
```

The title of the animation is “TCP 3 way handshake”, there are two nodes *HostA*, *HostB*. *HostA* constructs a packet *p1* with *SYN* as content, *HostB* as destination, and then send *p1* to *HostB*. *HostB* constructs a packet *p2* with *SYN\_ACK* as content and sends back

to *HostA*. Finally *HostA* constructs a packet with *ACK* as content and sends to *HostB*. The animation ends here. The resulting Shikav script language is shown in Appendix A.

#### 4.4.2 TCP slowstart

The procedure has been already explained in Chapter 3, the animation created would comprised of two hosts, named *HostA*, *HostB* and three packets are to be created with content of *SYN*, *SYN\_ACK*, *ACK*. The lecture file created in new defined language (Networking script) is as follows:

```

title "TCP Slow Start"
node A
node B
constructPacket(A, B, "payload", p)
send(p, A, B)
repeat 15 i
if (A receive "ACK") then
constructpacket(A,B,"payload",p1)
send(p1, A, B, 2^i)
else
endif
end

```

The title of the animation is “TCP Slow Start”, there are two nodes *A*, *B*. *A* constructs a packet *p* with *payload* as content, *B* as destination, and send packet *p* to *B*. The procedure of sending is defined as in the case study, the number of packets to send increases exponentially till the threshold  $65536=2^{16}$  in our case. As written in the above script, the command `constructPacket(A , B , "payload" , p)` and `send(p, A, B, 2^i)` repeat 16 times which is get translated into over a hundred lines of commands in Shikav lecture file as shown in Appendix A.

#### 4.4.3 WiFiRe

There is one BS and there are many STs, and EU connects to ST, these are all created as *EntityNode* objects. Then Beacon object is created as BS needs to send Beacon that

is spread as a signal to the entire sector. Then one ST start the communication with the procedure that has been already explained in Section 3.3.

The network script language for this example is as follows:

```
Title "WiFiRe"
Node BS 20 30
Node ST 150 200
Node ST1 200 250
Node ST2 170 150
Node EU 50 200
ConstructBeacon(BS, "beacon", 400, -60, b)
send(b, BS)
ConstructPacket(ST , BS , "IRRe" , irre)
send(irre, ST, BS)
if (BS receive "irre") then
ConstructPacket(BS,ST,"IRRes",irres)
send(irres, BS, ST)
else
delay(1)
endif
if (ST receive "IRRes") then
ConstructPacket(ST, BS, "RegReq", regreq)
send(regreq, ST, BS)
else
delay(1)
endif
if (BS receive "RegReq") then
ConstructPacket(BS,ST,"RegRes",regres)
send(regres, BS, ST)
else
delay(1)
endif
ConstructPacket(EU, ST, "SYN", syn)
```

```
send(syn, EU, ST)
if (ST receive "SYN") then
ConstructPacket(ST, BS, "DSA-Req", dsareq)
send(dsareq, ST, BS)
else
delay(1)
endif
if (BS receive "DSA-Req") then
ConstructPacket(BS,ST,"DSA-Res",dsares)
send(dsares, BS, ST)
else
delay(1)
endif
if (ST receive "DSA-Res") then
ConstructPacket(ST, EU, "SYN_ACK", synack)
send(synack, ST, EU)
else
delay(1)
endif
ConstructPacket(EU, ST, "SYN_ACK_ACK", ack2)
send(ack2, EU, ST)
ConstructPacket(EU, ST, "data", data)
send(data, EU, ST)
if (ST receive "data") then
ConstructPacket(ST, BS, "data", data1)
send(data1, ST, BS)
else
endif
```

The resulting Shikav script language is shown in Appendix A. Figure A.3 is the screenshot of the animation.

These examples show that the newly added features and the newly defined script (Network script language), as described in Section 4.2 and Section 4.3 respectively, work

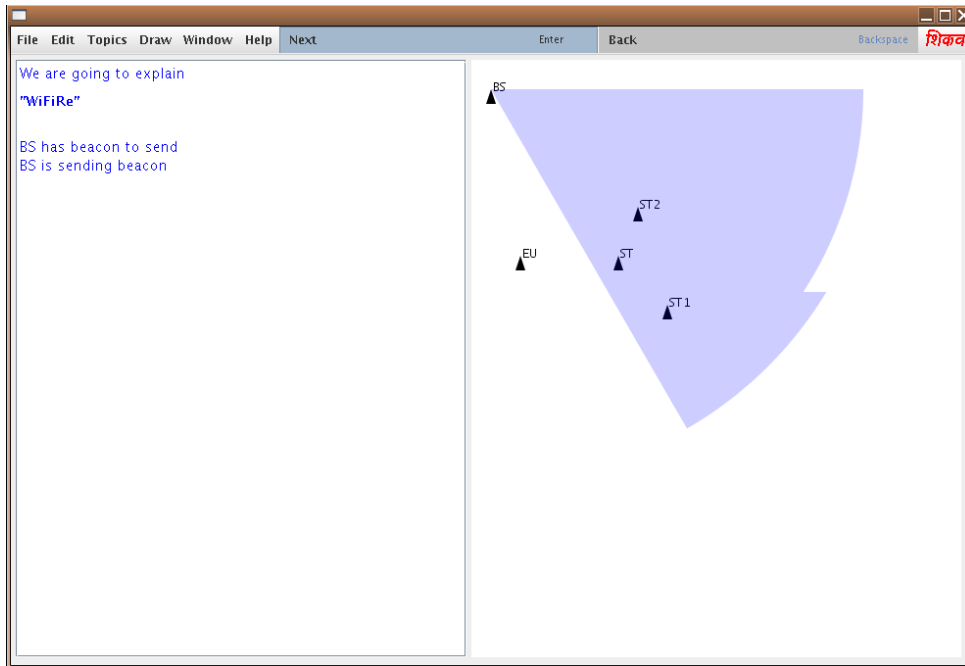


Figure 4.3: Screenshot of animation showing the working of WiFiRe protocol

as expected and can be used to explain network protocols in general.





# Chapter 5

## Conclusion and Future extensions

### 5.1 Conclusion

We have seen a detailed description of the Shikav framework, and a few network protocols. The basic features required to be added into Shikav enabling the creation of lessons explaining network protocols have been defined and implemented as described in Chapter 4. It enables authors to create lessons explaining the working flow of network protocols. This is shown by using various examples in Chapter 3. Above this, a higher level script with networking terms as key words has been defined, implemented and successfully integrated into Shikav. The script facilitates the author in creating animations explaining network protocols. Without this, the authors would take a lot of effort to write the Shikav script language to create electronic lessons explaining network protocols. This is not because Shikav does not have an effective script language but because Shikav has no intrinsic support for animations explaining network protocols.

However, further enhancements in Shikav and the newly defined script could be made. A few of them are discussed in the succeeding section.

### 5.2 Future extensions

This enhancement into Shikav for supporting animation in networking is mainly for visual representation. The actual simulation of packets has not been included.

- Integration with simulation: recently simulation capabilities have been added into Shikav by Ankit [4], but that version does not support packet animation. This could be done by integrating the new enhancement of representing the animation with the

simulation. The option of showing animation can be added into the Shikav menu using which the user can see the animation of packets transferring.

- Drag/Drop object: the user would be given an option to drag and drop the *EntityNode*.
- Enrich networking script: the script has been defined with basic functionalities. More features can be added to enrich the script language such as allowing the node to randomly send the packets. This means that author can create many nodes, then call send, and the node starts to send packet in random order so that the author need not to specify the order of node to send packet.

## Scope for other improvements in Shikav

After a detailed studied of Shikav framework, it has been observed that some improvement need to be done into Shikav, such as listed belows:

- The grammar rules which are defined in file “inter.cup” are complicated, this could be simplified and reduce some expressions and make the rule generic enough to use. The documentation on the grammar and especially the *Symboltable* class which stores all the commands successfully parsed by Java cup parser (first parser of Shikav) would be very helpful because it is not quite trivial to understand why *Symboltable* class is needed and how things have been done there. Also It would reduce the work load of the enhancer of Shikav by providing the documentation on the code, especially the relation between classes because as for now ones need to actually look into the code in almost each and everyline in order to figure out how it works.
- The code could be classified into packets depending upon their functionalities. As lately, many enhancements has been added into Shikav from different enhancer and mixed it all together into the Framework for which the source code become a bit unorganized.

# Appendix A

## Examples of lecture files

In this appendix, we present the resulting Shikav script language lecture files corresponding to each of the protocol example studied in Chapter 3

### A.1 TCP 3 way handshake

The procedure has been already explained in Chapter 3, the lecture file in networking script language is shown in Section 4.4.1 and its resulting Shikav script language is as follows:

```
reset("0", "0", "990", "680", "tcp3wayHandshake")
text = TextPanel("5", "5", "460", "610", "blue", "white", "green")
g = Geometry("470", "5", "500", "610")
timer = TimerPanel("0.2")
packetBuffer = PacketQueue()
text ! We are going to explain <h3>"TCP 3 way handshake"</h3>
HostA = EntityNode("HostA", 259, 487)
HostB = EntityNode("HostB", 14, 289)
p1 = Packet("SYN")
p1.setSource(HostA)
p1.setDestination(HostB)
text : HostA has packet "SYN" to send to HostB
text : HostA is sending packet to HostB
text : <br>
HostA.send(p1)
packetBuffer.add(p1)
```

```

p0 = HostA.receive(packetBuffer)
b0 = p0.isPayload("SYN")
if b0 == 1 then p2 = Packet("SYN_ACK"); p2.setSource(HostB);
p2.setDestination(HostA); text : HostB has packet "SYN_ACK" to send to HostA;
text : HostB is sending packet to HostA; text : <br> ;
HostB.send(p2); packetBuffer.add(p2); else timer : "0.2"; endif
p1 = HostB.receive(packetBuffer)
b1 = p1.isPayload("SYN_ACK")
if b1 == 1 then p3 = Packet("ACK"); p3.setSource(HostA);
p3.setDestination(HostB); text : HostA has packet "ACK" to send to HostB;
text : HostA is sending packet to HostB; text : <br> ;
HostA.send(p3); packetBuffer.add(p3); else timer : "0.2"; endif
text : <br> The explanation ends here !

```

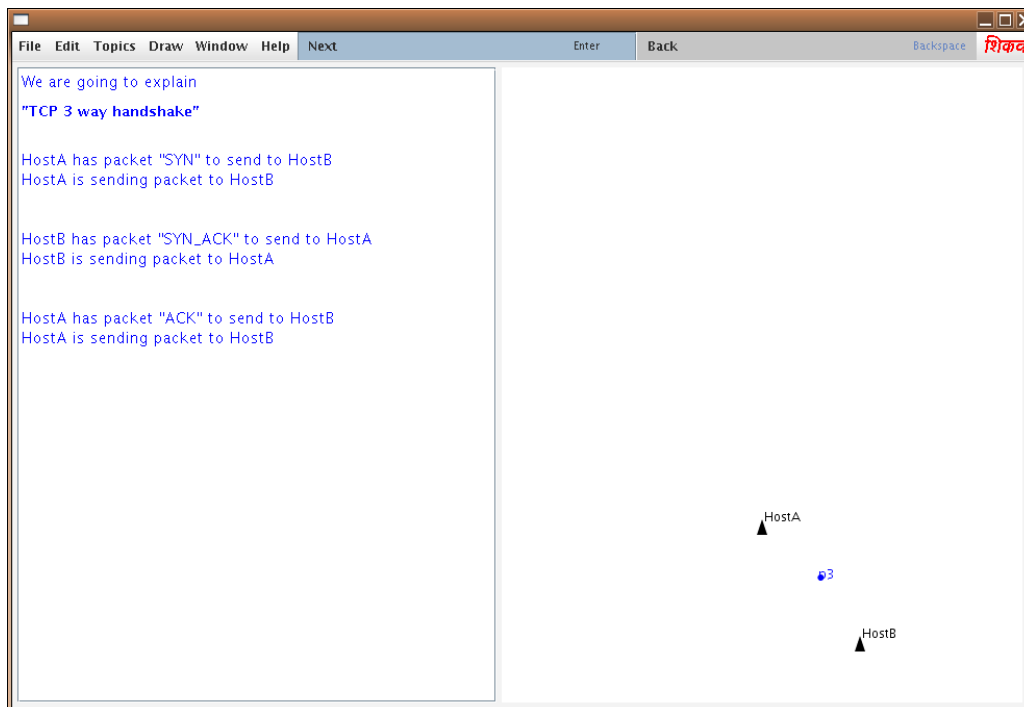


Figure A.1: Screenshot of animation showing the working of TCP 3 way handshake

## A.2 TCP slowstart

The procedure has been already explained in Chapter 3, the lecture file in networking script language is shown in Section 4.4.2 and its resulting Shikav script language is as follows:

```

reset("0","0","990","680","tcpSlowStart")
text = TextPanel("5","5","460","610","blue","white","green")
g = Geometry("470","5","500","610")
timer = TimerPanel("0.2")
packetBuffer = PacketQueue()
text ! We are going to explain <h3>"TCP Slow Start"</h3>
A = EntityNode("A",472,299)
B = EntityNode("B",191,558)
p = Packet("payload")
p.setSource(A)
p.setDestination(B)
text : A has packet "payload" to send to B
text : A is sending packet to B
text : <br>
A.send(p)
packetBuffer.add(p)
i=ArithElement()
i.setValue(0)
p0 = A.receive(packetBuffer)
b0 = p0.isPayload("ACK")
if b0 == 1 then p1 = Packet("payload"); p1.setSource(A);
p1.setDestination(B); text : A has packet "payload" to send to B;
text : A is sending packets to B; text : <br>; A.send(p1,2^i);
else ; endif
i=i+1
p1 = A.receive(packetBuffer)
b1 = p1.isPayload("ACK")

```

```
if b1 == 1 then p1 = Packet("payload"); p1.setSource(A);
p1.setDestination(B); text : A has packet "payload" to send to B;
text : A is sending packets to B; text : <br>; A.send(p1,2^i);
else ; endif
i=i+1
p2 = A.receive(packetBuffer)
b2 = p2.isPayload("ACK")
if b2 == 1 then p1 = Packet("payload"); p1.setSource(A);
p1.setDestination(B); text : A has packet "payload" to send to B;
text : A is sending packets to B; text : <br>; A.send(p1,2^i);
else ; endif
i=i+1
....
....
p13 = A.receive(packetBuffer)
b13 = p13.isPayload("ACK")
if b13 == 1 then p1 = Packet("payload"); p1.setSource(A);
p1.setDestination(B); text : A has packet "payload" to send to B;
text : A is sending packets to B; text : <br>; A.send(p1,2^i);
else ; endif
i=i+1
p14 = A.receive(packetBuffer)
b14 = p14.isPayload("ACK")
if b14 == 1 then p1 = Packet("payload"); p1.setSource(A);
p1.setDestination(B); text : A has packet "payload" to send to B;
text : A is sending packets to B; text : <br>; A.send(p1,2^i);
else ; endif
i=i+1
text : <br> The explanation ends here !
```

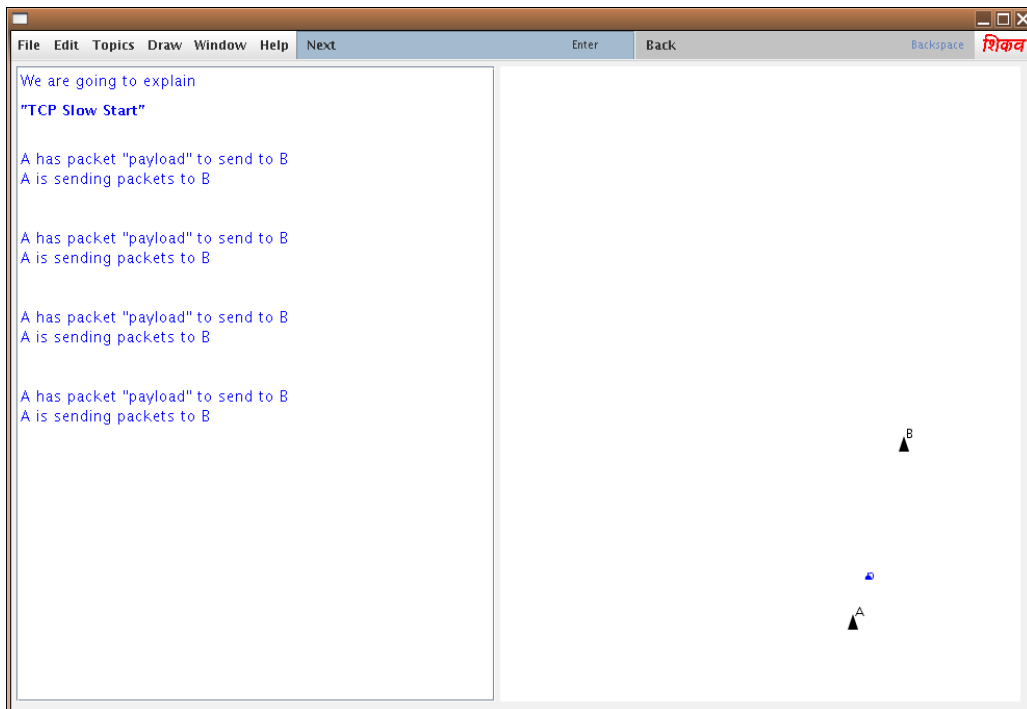


Figure A.2: Screenshot of animation showing the working of TCP Slow Start

## A.3 WiFiRe

The procedure has been already explained in Chapter 3, the lecture file in networking script language is shown in Section 4.4.3 and its resulting Shikav script language is as follows: The resulting lecture file created in Shikav script language is as follows:

```

reset("0","0","990","680","wifire")
text = TextPanel("5","5","460","610","blue","white","green")
g = Geometry("470","5","500","610")
timer = TimerPanel("0.2")
packetBuffer = PacketQueue()
text ! We are going to explain <h3>"WiFiRe"</h3>
BS = EntityNode("BS",20,30)
ST = EntityNode("ST",150,200)
ST1 = EntityNode("ST1",200,250)
ST2 = EntityNode("ST2",170,150)
EU = EntityNode("EU",50,200)
b = Beacon(BS, 20, -60)
b.setContent("beacon")

```

```
b.setRange(400)
b.H("blue")
text : BS has beacon to send
text : BS is sending beacon
text : <br>
BS.send(b)
irre = Packet("IRRe")
irre.setSource(ST)
irre.setDestination(BS)
text : ST has packet "IRRe" to send to BS
text : ST is sending packet to BS
text : <br>
ST.send(irre)
packetBuffer.add(irre)
p0 = BS.receive(packetBuffer)
b0 = p0.isPayload("irre")
if b0 == 1 then irres = Packet("IRRes"); irres.setSource(BS);
irres.setDestination(ST); text : BS has packet "IRRes" to send to ST;
text : BS is sending packet to ST; text : <br> ;
BS.send(irres); packetBuffer.add(irres); else timer : "0.1"; endif
p1 = ST.receive(packetBuffer)
b1 = p1.isPayload("IRRes")
if b1 == 1 then regreq = Packet("RegReq"); regreq.setSource(ST);
regreq.setDestination(BS); text : ST has packet "RegReq" to send to BS;
text : ST is sending packet to BS; text : <br> ; ST.send(regreq);
packetBuffer.add(regreq); else timer : "0.1"; endif
p2 = BS.receive(packetBuffer)
b2 = p2.isPayload("RegReq")
if b2 == 1 then regres = Packet("RegRes"); regres.setSource(BS);
regres.setDestination(ST); text : BS has packet "RegRes" to send to ST;
text : BS is sending packet to ST; text : <br> ; BS.send(regres);
packetBuffer.add(regres); else timer : "0.1"; endif
```



```
syn = Packet("SYN")
syn.setSource(EU)
syn.setDestination(ST)
text : EU has packet "SYN" to send to ST
text : EU is sending packet to ST
text : <br>
EU.send(syn)
packetBuffer.add(syn)
p3 = ST.receive(packetBuffer)
b3 = p3.isPayload("SYN")
if b3 == 1 then dsareq = Packet("DSA-Req"); dsareq.setSource(ST);
dsareq.setDestination(BS); text : ST has packet "DSA-Req" to send to BS;
text : ST is sending packet to BS; text : <br> ; ST.send(dsareq);
packetBuffer.add(dsareq); else timer : "0.1"; endif
p4 = BS.receive(packetBuffer)
b4 = p4.isPayload("DSA-Req")
if b4 == 1 then dsares = Packet("DSA-Res"); dsares.setSource(BS);
dsares.setDestination(ST); text : BS has packet "DSA-Res" to send to ST;
text : BS is sending packet to ST; text : <br> ; BS.send(dsares);
packetBuffer.add(dsares); else timer : "0.1"; endif
p5 = ST.receive(packetBuffer)
b5 = p5.isPayload("DSA-Res")
if b5 == 1 then synack = Packet("SYN_ACK"); synack.setSource(ST);
synack.setDestination(EU); text : ST has packet "SYN_ACK" to send to EU;
text : ST is sending packet to EU; text : <br> ; ST.send(synack);
packetBuffer.add(synack); else timer : "0.1"; endif
ack2 = Packet("SYN_ACK_ACK")
ack2.setSource(EU)
ack2.setDestination(ST)
text : EU has packet "SYN_ACK_ACK" to send to ST
text : EU is sending packet to ST
text : <br>
```

```

EU.send(ack2)
packetBuffer.add(ack2)
data = Packet("data")
data.setSource(EU)
data.setDestination(ST)
text : EU has packet "data" to send to ST
text : EU is sending packet to ST
text : <br>
EU.send(data)
packetBuffer.add(data)
p6 = ST.receive(packetBuffer)
b6 = p6.isPayload("data")
if b6 == 1 then data1 = Packet("data"); data1.setSource(ST);
data1.setDestination(BS); text : ST has packet "data" to send to BS;
text : ST is sending packet to BS; text : <br> ; ST.send(data1);
packetBuffer.add(data1); else ; endif
text : <br> The explanation ends here !

```

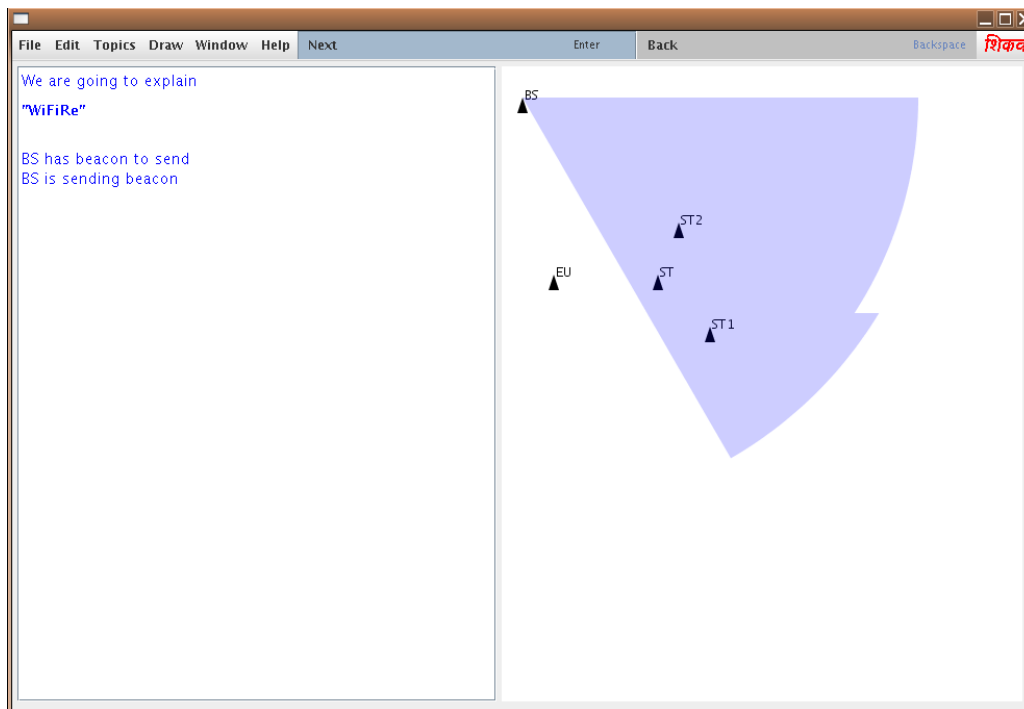


Figure A.3: Screenshot of animation showing the working of WiFiRe protocol

# Bibliography

- [1] *GVU center - Polka*. <http://www.gvu.gatech.edu/gvu/softviz/parviz/polka.html>.
- [2] *The Samba homepage*. <http://www.cc.gatech.edu/gvu/softviz/algoanim/samba.html>.
- [3] Sridhar Iyer, Krishna Paul, Anurag Kumar, and Bhaskar Ramamurthi. Broadband wireless for rural areas—*WiFiRe: medium access control (mac) and physical layer (phy) specifications*. <http://www.cewit.org>, August 2006.
- [4] Ankit Jain. Adding simulation capabilities to shikav. Technical report, Department of Computer Science and Engg., Indian Institute of Technology Bombay, India, 2007.
- [5] Aditya Kelkar. *Shikav - A FrameWork for Interactive Electronic Lessons*. Master’s thesis, Department of Computer Science and Engg., Indian Institute of Technology Bombay, India, 2006.
- [6] John A. Miller, Andrew F. Seila, and Xuewei Xiang. *JSIM Web-based Simulation. Future Generation Computer Systems (FGCS), Special Issue on Web-Based Modeling and Simulation*, 17(2):119–133, 2000.
- [7] Jon Postel, editor. *Transmission Control Protocol-DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION*. RFC793, September 1981.
- [8] Abhiram Ranade, Aditya Kelkar, and Prasad Naik. Electronic teaching using shikav. Technical report, Department of Computer Science and Engg., Indian Institute of Technology Bombay, India, 2006.

- [9] Susan Rodger. *Java and Web based Algorithm Animation*. <http://www.cs.duke.edu/csed/jawaa2/>.
- [10] W. Stevens. *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*. RFC2001, January 1997.