CHAPTER 1

INTRODUCTION

Traffic congestion is a challenging problem today because of increasing number of vehicles day by day. To overcome this problem, an economical solution is to administer the traffic flow. Traffic Modeling is a technique that is used to manage and control the flow of traffic.

A distributed camera network based traffic model has been designed from analysis presented in [1]. The work presented in [2] focuses on the development of a traffic model that extracts video based features and processes the same to classify the road condition as open, slight congestion, heavy congestion or traffic jam. The work successfully monitors traffic conditions across a road; however it does not include a communication module to transfer traffic statistics to the server.

The traffic system that has been designed based on [2] involves extracting the features (SIP and STIP) from the video frames generated by the camera. The number of SIP is indicative of number of vehicles on a road and ratio of STIP to number of SIP is suggestive of percentage of moving vehicles. To enable sharing of traffic statistics among nodes, we need to establish a wireless network. Also, we intend to upload traffic status of roads to the server over the internet, so that any user can access the same to know about present traffic state at a particular road. The work presented in this report focuses on the same.

This project intends to implement this wireless network using 3G network. We first present architecture to interface a single system with 3G network. Thereafter we extend the architecture to involve several traffic monitoring systems that are interfaced to the 3G network, which in turn facilitates sharing of traffic statistics among nodes. The same 3G network can be used to upload the data over the server, then the server may process the received data from various local traffic monitoring systems using suitable algorithms and based on the results it can issue traffic statistics, which can be utilized to administer the traffic flow. The work presented here utilizes a 3G modem to interface the traffic monitoring system with the 3G network.

1.1 Basic Traffic Monitoring System

The basic traffic monitoring system that we are going to consider for interfacing with 3G network has features as follows.

It employs a distributed camera network to monitor traffic. These cameras record the activities on each road and then process the recorded data to identify a pattern. Based on the analysis of the traffic videos recorded by cameras, traffic status is determined for a given road link. Now, GMM-EM based classification and HMM based prediction is used to determine the optimum path by assigning proportional weights to the predicted states of the associated roads. This method is quite flexible as compared to others and performs efficient analysis with relatively light computations. Unlike other methods available to perform such analysis this method does not draw

analogies of traffic moving as particles, also it does not impose any restrictions on the conditions of the road under consideration or road tributaries and distributaries.

The block diagram of the traffic prediction model utilized to implement the said traffic monitoring system is as shown below.

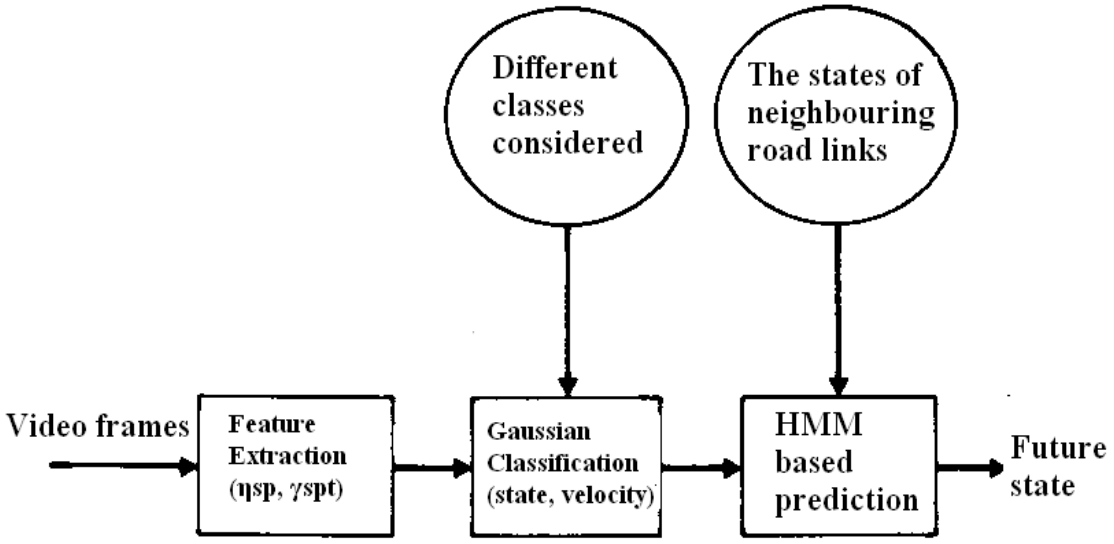

Fig. 1: Traffic prediction model

Here we have a distributed camera network based traffic monitoring system. This system records the traffic situation across a given road and then process the recorded video to classify the traffic status of the given road link as open, slight congestion, heavy congestion or traffic jam. This traffic status can be used to administer the traffic intelligently. One such approach to manage traffic intelligently has been presented in [3], wherein various road links are assigned weights based on their traffic status' (using Optimum Weight Calculator and Fair Weight Calculator algorithms) and then duration of traffic signals (Red, Yellow and Green etc.) can be dynamically adjusted to achieve minimum possible congestion.

Nodes are installed across the road network along with required accessories (camera, 3G modem etc) to process the data and determine traffic status thereof and communicate the same to nearby nodes. The features (SIP and STIP) are derived from the video recorded by the camera.
Spatial interest points are points (SIP) in spatial domain with significant variation in local intensities where as spatio-temporal interest points (STIP) are points in space time domain with significant variation in local intensities.

SIP is generated by the vehicles on a road and STIP is generated by moving vehicles. Hence, SIP represents number of vehicles present on a road and ratio of STIP to SIP gives percentage of moving vehicles. These points are classified using GMM and then traffic state prediction is carried out using HMM. Once the traffic status of each road is available, optimal path can be obtained by assigning weights (in accordance with the predicted state) to each road.

1.2 Related Work

In road traffic management, several kinds of approaches can be considered for monitoring vehicle movement keeping in mind the requirements of a particular place. There are different technologies which can be employed for traffic monitoring [4]. Magnetic sensor networks can be used for real time traffic monitoring [5]. Another approach could be of performing traffic surveillance using wireless magnetic sensors [6]. Similarly, other approaches like radar based systems [7], laser sensors based systems [8], infrared detector based systems [9] etc. can also be used for observing the overly crowded roads. Information gathered from the sensors is processed to obtain the required traffic statistics which are needed to take care of the congestion on the roads. In this paper the traffic data used is in the form of Spatial Interest Points (SIP) or Spatio-Temporal Interest Points (STIP) [1]. Different kinds of node processing are also proposed in some of the researches. One such research presented in [10] employs multiple sensors like magnetometer, infrared, accelerometer, acoustic microphone etc. at every node. However, this makes it quite expensive. Design of a data transmission system based on 3G and embedded technology has been presented in [11] wherein a 3G module (AD3812-V2) is employed to provide connectivity with 3G networks.

1.3 Outline

The organization of the rest of the report is as follows. We first present the details of the available traffic monitoring system and the associated hardware. Thereafter we present the system architecture to interface a given node with 3G network. Similarly, other architectures such as single camera architecture, multi-camera architecture and architecture to facilitate traffic data sharing will be discussed.

Also, the approach (protocol) adopted to facilitate 3G based wireless communication will be discussed with results thereof. The report also proposes an algorithm to determine overall traffic status of a node with multi-camera architecture.

CHAPTER 2

DETAILS OF THE AVAILABLE TRAFFIC MONITORING SYSTEM HARDWARE

This chapter presents implementation of traffic monitoring system, based on ARM 11 processor. Traffic monitoring system extracts features (spatial interest points (SIP) and spatio-temporal interest points (STIP)) from traffic video captured by CMOS camera. ARM 11 processor is used to classify (using Gaussian Mixture Model-Expectation Maximization) traffic states of the road network. The system employs BCM2835 board (ARM11) as a processing unit interfaced with a CMOS camera (OV7670) to build an image acquisition system whose output is interfaced with a 16X2 LCD screen, and consists of all necessary communication interfaces to complete the design of hardware platform. Then drivers were composed and ARM-Linux operating system was transplanted. The system has been tested and works well in different light and weather conditions.

2.1 Overview

Traffic congestion is a challenging problem today because of increasing number of vehicles day by day. To overcome this problem, an economical solution is to administer the traffic flow. Traffic Modeling is a technique that is used to manage and control the flow of traffic. The technique that is used is based on modeling of road traffic by the use of distributed camera network [1]. In this chapter the hardware implementation of this technique is presented. In conjunction with the implementation the CMOS camera module is used as sensor module for getting the input from a particular branch or a link of a road network which is fed to processing unit which extracts the features from the video and classifies the road conditions at a particular time instance. The road conditions are classified into four states depending on the state of traffic which is obtained by the sensor module and processing unit. The combined unit forms the image acquisition system which is then interfaced with a display module.

The image acquisition system is implemented using a CMOS camera module Omnivision OV7670 and BCM2835 board (ARM 11) processor interfaced with a LCD module for displaying traffic state which can be further replaced by wireless module to transfer the traffic state to remote location.

2.2 Problem Formulation

A sample road network is shown in figure 2. R1 – R7 are different road links and C1 – C7 are cameras for recording the activities on the road links respectively. The video features of each road link will be extracted, processed, predicted and communicated to the nearby nodes periodically by each node. Each node will be sending the predicted states of the road and neighboring road links along with the ID of the respective road link for updating the current status. If the number of vehicles in a road link is η and the no. of moving vehicle is Υ, we can classify the road state $\eta\Upsilon$ domain. The major classification considered is: Stopped (S), Heavy Traffic (HT), Slight Traffic (ST) and open (O).

Fig. 2: Road Links

2.2.1 Interest Points

Spatial interest points (SIP) are points in spatial domain with significant variation in local intensities whereas spatio-temporal interest points (STIP) are points in space time domain with significant variation in local intensities. It is observed that images of vehicles on a road generate SIP and the images of moving vehicles generate STIP.

Fig. 3: Red Points Indicate SIP and Blue Points Indicate STIP

The number of SIP is indicative of number of vehicles on a road and ratio of STIP to number of SIP is suggestive percentage of moving vehicles. The spatial interest points are found by recording the intensity values along each dimension in a local neighborhood and then found the Eigen values of recorded 2 dimensional data. This concept is then further extended to temporal domain. This way we obtained spatial and corresponding spatio-temporal interest points with detection of minor changes in intensity over temporal domain. These points are classified using GMM.

2.2.2 Feature Extraction and Classification

Traffic on any road can be completely defined by the number of moving vehicles and their average velocity. But these two features depend on each other. Therefore we classify the road states by comparing the no. of vehicles with the number of moving vehicles. If the no. of vehicles in a road link is η and the no. of moving vehicle is γ, we can classify the road state ηγ domain. The major classification considered is: Stopped (S), Heavy congestion (HC), Slight Traffic (ST), and Open (O) .We can extract SIP and STIP from the video frames recorded by the camera. The classification of the traffic state can be done based on the ratio of STIP to SIP as the ratio will give the indication of the state of the road. This is possible only when we get a dense feature set of correlated SIP and STIP, whereas the existing operators are providing sparse feature set of uncorrelated SIP and STIP. Hence a novel spatial interest point detector is made which provides dense feature set of correlated points by modifying the Harris corner detector.

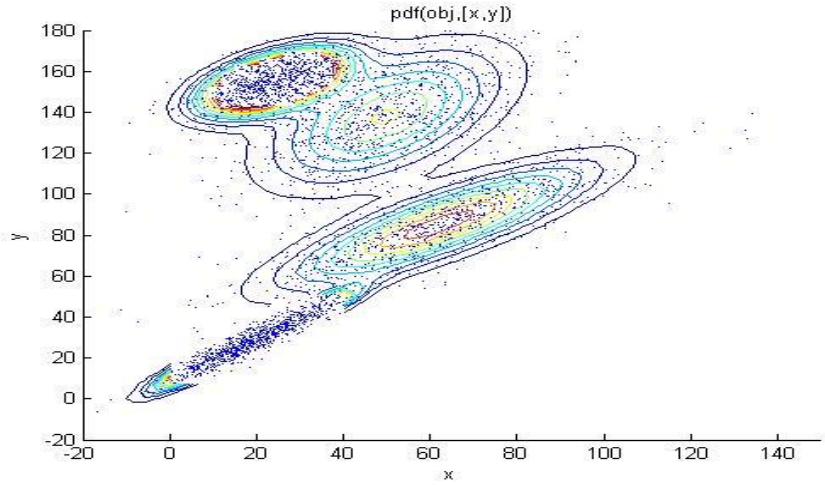

Fig. 4: Classification using GMM as Classifier

The parametric model of the saliency features distribution is learned by fitting a Gaussian Mixture Model (GMM) using the Expectation-Maximization (EM) algorithm on a hand labeled training data set. This approach is used because it is suitable for fast data processing with only three features, weight, mean and covariance. Using the spatial (η_{sp}) and spatio-temporal interest (γ_{pt}) points we can classify the road state by using Gaussian mixture model .Let the feature vector be

S = (η_{sp}, γ_{pt}) , no. of classes be k and no. of Gaussian mixtures be n. The conditional probability that belongs to K_{th} class is given by:

$$p\left(S/_{M^k}\right) = \sum_{i=0}^{n^k} \frac{w_i^k}{2\pi^{d/2} |\Sigma_i|^{1/2}} exp^{(S-\mu_t^k)\Sigma_t^k (S-\mu_t^k)}$$

After simulating this formula and maximizing the expectation, we got 4 Clusters which tells us about the 4 classes that we have specified with different mean and variance values.

2.3 System Architecture

2.3.1 Image Acquisition System

This consists of CMOS camera module OV7670 which has single chip 1280X1024 SXGA camera and has high sensitivity for low light operation and is useful for low operating voltages and portable applications. This camera provides a images in different formats such as full-frame, sub-sampled and 8-bit/10-bit windowed formats and this is controlled via Serial Camera Control Bus Interface (SCCB). In addition to this the image processing functionalities such as noise cancellation and colour saturation is also controlled via SCCB interface.

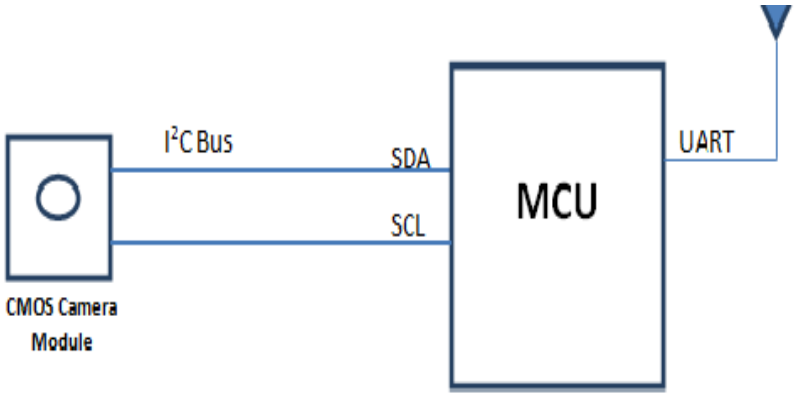

Fig. 5: Single Camera Architecture

Block diagram for interfacing of camera to RaspberryPi module is shown above. I2C bus is used for interfacing camera to RaspberryPi module. It involves two signals:

- SDA – Serial Data Line
- SCL – Serial Clock Line

Both SCL and SDA lines are of open-drain design, thus, pull-up resistors are needed. Pulling the line to ground is considered a logical zero while letting the line float is a logical one. High speed systems (and some others) also add a current source pull up, at least on SCL; this accommodates higher bus capacitance and enables faster rise times.

Block diagram for multiple camera architecture is shown below.

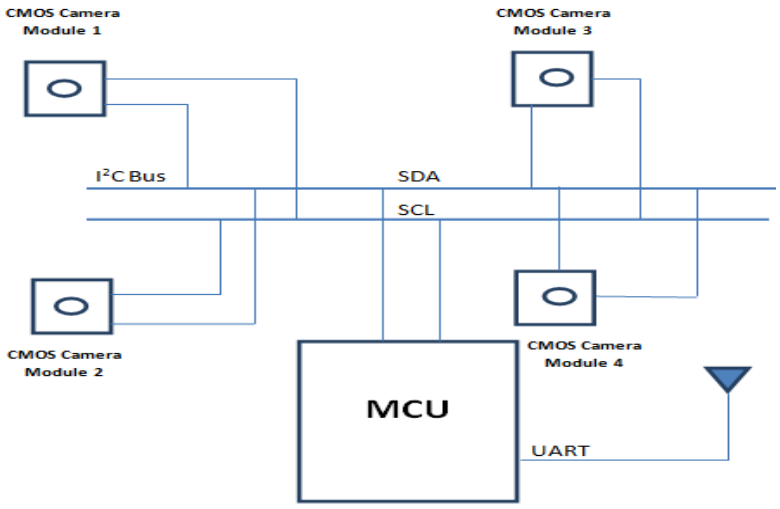

Fig. 6: Multiple Camera Architecture

2.3.2 Output Module

The output module consists of 16X2 LCD which displays the current state of the traffic and the concerned node. This can further be replaced by a wireless transmission module over UART protocol.

2.3.3 Storage Module

This is used to store and compare the images obtained from the previous and the present state in order to obtain the Spatial interest points (SIP) and spatio-temporal interest points (STIP).

2.3.4 Control Module

This consists of ARM 11 Processor (BCM2835 board) which consists of advanced image sensor pipeline and can be used for advanced multimedia applications. The image acquisition module is interfaced with the processor using

Serial Peripheral Interface SPI protocol or I^2C protocol (Inter-Integrated Circuit) Protocol. This module is also referred to as raspberry pi.

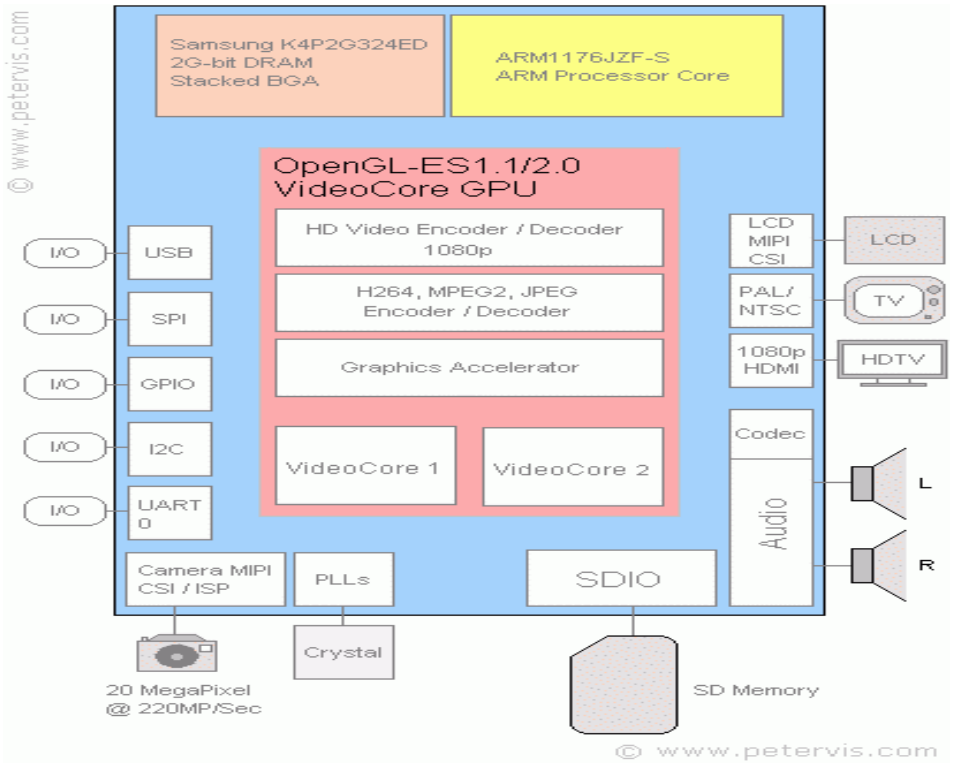

Fig. 7: BCM2835 Block Diagram

2.3.5 Open Computer Vision Library

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. OpenCV has a modular structure, which means that the package includes several shared or static libraries.

Note that the code written using OpenCV library to determine SIP and STIP has been given in Appendix B.

To configure OpenCV on your windows PC you can go through following steps:

1. Extract OpenCV into ‘C:\opencv’.
2. Install latest MinGW in ‘C:\Mingw’.
3. Install CMake 2.8.7 (doesnt matter if it is 32-bit).
4. Add MinGW to User Variables in Environment Variables ‘C:\MinGW\bin’ by following the path ‘MyComputer > Properties >

Advanced System Settings > Environment Variables > User Variables For xxx > Path > C:\MinGW\bin'.

5. Now open the CMake and give the source directory as 'C:\opencv' and destination directory as 'C:\opencv_MinGW'. The later directory is newly created directory.

6. Now Configure the Files using Configure and Generate button.

7. Run the command line mode (cmd.exe) and go to the destination directory 'C:\opencv_MinGW'.

8. Type "mingw32-make" (without quotation marks). You will see a progress of building binaries.

9. In Windows system PATH (My Computer > Right button click > Properties > Advanced > Environment Variables > Path) add the destination's bin directory i.e. 'C:\opencv_MinGW\bin'.

10. Restart computer.

11. Go to the Eclipse CDT IDE; create a C++ program using the sample OpenCV code.

12. Go to 'Project > Properties > C/C++ Build > Settings > GCC C++ Compiler > Includes' and add the source OpenCV folder "C:\opencv\build\include" (including quotation marks).

13. Go to 'Project > Properties > C/C++ Build > Settings > MinGW C++ Linker > Libraries' and add to the Libraries (-l) one by one (this could vary from project to project, you can add all of them if you like or just the ones required for your project).

14. Add the built OpenCV library folder (including quotation marks), "C:\OpenCV2_MinGW\lib" to Library search path (-L).

15. When we compile and try to run the code it may not run if the application file is present in some other directory, so either you can copy the files from the 'opencv_MinGW\bin' folder to the Release folder in the Eclipse Project or try to keep your workspace in the bin folder of 'opencv_MinGW'.

2.4 Implementation Details

In order to implement the algorithm into a hardware module the input is taken from a camera module which captures the images from a live video stream every 5 seconds. The captured image frame is given as an input to the processor which carries out the feature extraction. Based on the number of spatial interest points and spatio-temporal interest points it calculates the state of the traffic and transmits it over to the serial interface. The processor module is implemented using ARM 11 core which is interfaced to the output module over SPI/I2C/USART protocol. ARM 11 provides an additional advantage of having an enhanced support for multimedia applications as compared to its earlier counterparts. The board is provided with an additional storage of 8 GB for different device drivers. In this the cross compilation tool used is ARM-Linux-gcc in order to port the code onto to ARM-Linux board. The flowchart for Camera Based Traffic Monitoring is shown in figure 8.

2.4.1 Image Extraction

The image frame is extracted from the camera module and sent to the processor. The processor performs an initial task of conversion of image to gray scale and extracts the object of interest which in this case is a road network and erasing all the other objects captured by the camera module.

2.4.2 Image Processing

The resultant image is convolved with a rotationally symmetric Gaussian Low Pass Filter for image blurring. The resultant image is convolved with two derivative mask which represent horizontal and vertical edges of the image. The gradient is calculated by convolving the squared image derivative with Gaussian kernel.

2.4.3 Detection of SIP (Spatial Interest Points)

A non maximal suppression is performed on the resultant smoothed image squared derivatives to scan the image in the direction of gradient and to suppress all the information from the image that is not a part of local maxima. Thus the spatial interest points are detected.

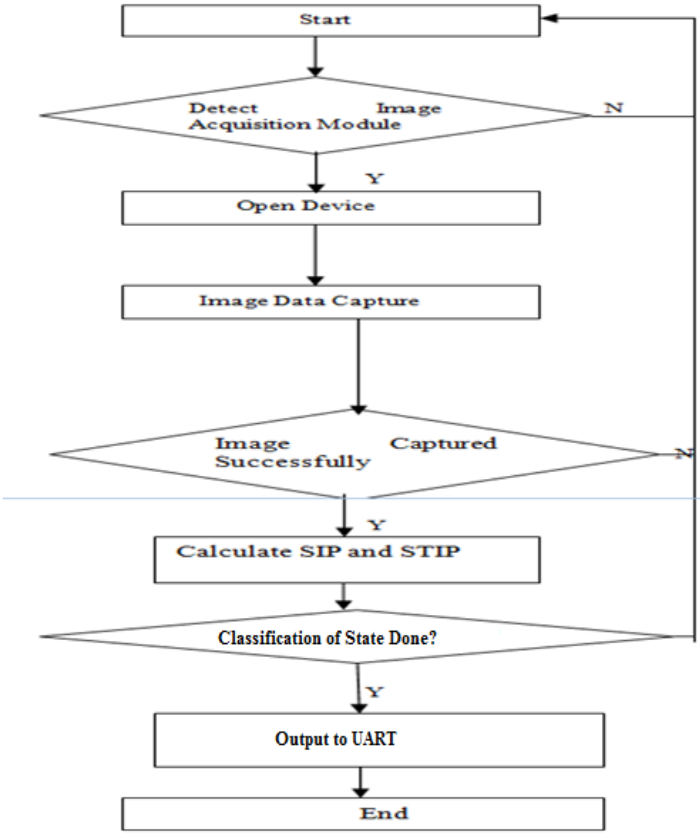

Fig. 8: Flowchart for Camera Based Traffic Monitoring System

2.4.4 Detection of STIP (Spatio-Temporal Interest Points)

In order to detect the Spatio-temporal interest points the two image frame are captured and the matrices of the two images obtained while deriving the SIP are taken into account and are incremented in a row-wise taking the column as a constant field. The matrix elements are compared and if they are not equal the variable STIP is incremented.

2.4.5 Classification

In order to classify the traffic states the normalized feature states was obtained depending on the number of vehicles which are in a unit of area. This feature set is obtained on the basis of the number of SIP and STIP per unit area. This is compared with the threshold set and the state of the traffic is classified.

2.5 Output

The implementation of the above system was done using a USB webcam and ARM11 board. The total simulation time in MATLAB was found to be 2.38sec. The total memory size was found to be 42Kb.

The output consists of 8 bits in which the state of the road consists of 2 bits, Node or Branch of 4 bits and 2 bits have been kept for future use as indicated in the table. The indication of different traffic states are as shown in figure 9.

Bit Status	**Traffic State**
00	Stopped
01	Slight Traffic
10	Heavy Traffic
11	Open

Table 1: Status for traffic monitoring

In hardware shown below, the traffic state is obtained from one of the cameras. This idea can be extended, by using I2C protocol, for getting image data from more than 1 camera and using the DBN model, as suggested in the work, to classify the future traffic state of the main road by capturing data from main road as well as linking roads.

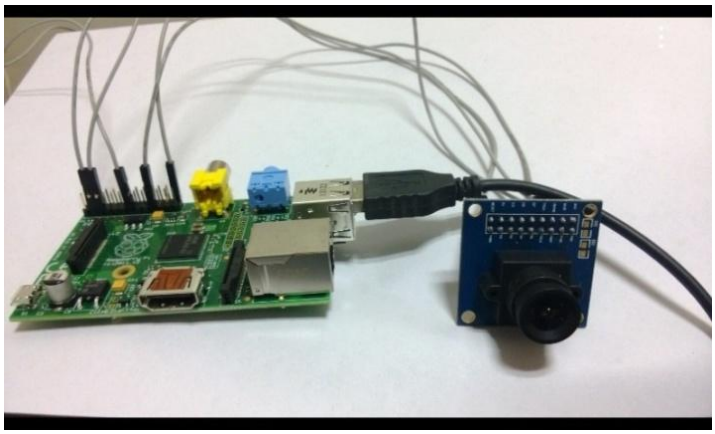

Fig. 9: CMOS Camera Module interfaced to BCM2835 board

Fig. 10: Indication of traffic states obtained in different conditions using MATLAB and its implementation on hardware implementation using OpenCV(Indicated in last two images)

CHAPTER 3

PROPOSED SYSTEM ARCHITECTURES

Proposed architectures for interfacing the available traffic monitoring system with 3G networks are discussed below. We present the design of a data transmission system which is based on 3G and embedded technology. The program uses ARM 11 Processor (BCM2835 board) as a controller and 3G modem.

The basic traffic monitoring system provides us the status of traffic at any given point. This status can be transmitted to the 3G network by interfacing output of the ARM 11 processor with 3G modules which facilitate transmission of the same to the 3G network.

Though the approach discussed in this section is very basic, it is possible to extend the same for more complex systems whereby input data from multiple CMOS camera modules is taken and processed by ARM processor section. The ARM processor section is equipped with suitable algorithm to decide the overall status of the traffic. This overall status is then transferred to 3G networks via 3G module.

3.1 Single Camera Architecture

The basic block diagram of the single camera architecture interfaced with 3G network is shown below.

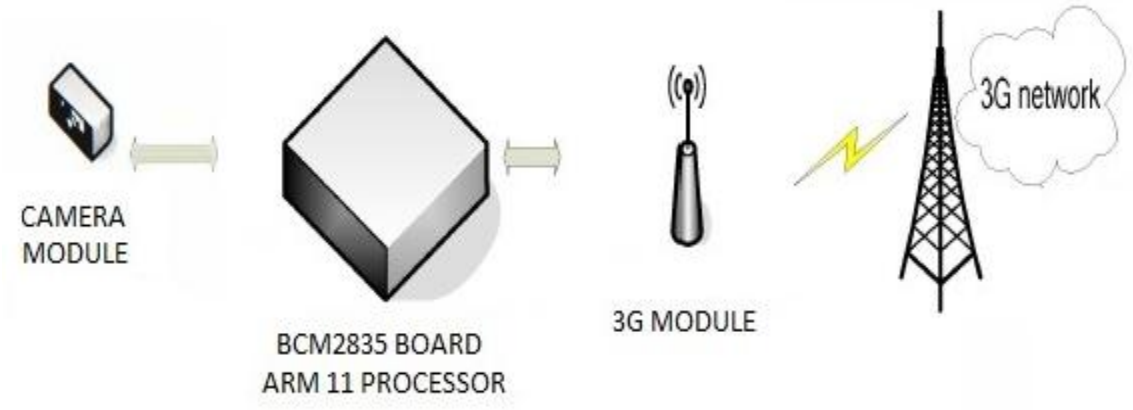

Fig. 11: Interfacing Single camera architecture with 3G module

Working of the above system can be summarized in following points:

1. The CMOS camera module captures images of present traffic and passes them to the ARM 11 processor.
2. The BCM2835 module processes the images received from camera module to determine the traffic status.
3. The BCM2835 module sends the traffic status to the 3G module.
4. 3G module sends the traffic status to the 3G network.

Above architecture shows only single camera module, there may as well be multiple cameras connected to BCM2835 module in an advanced traffic monitoring system. The architecture of the same is discussed in next section.

3.2 Advanced Traffic Monitoring System

An extension of the system discussed in the previous section is as shown below.

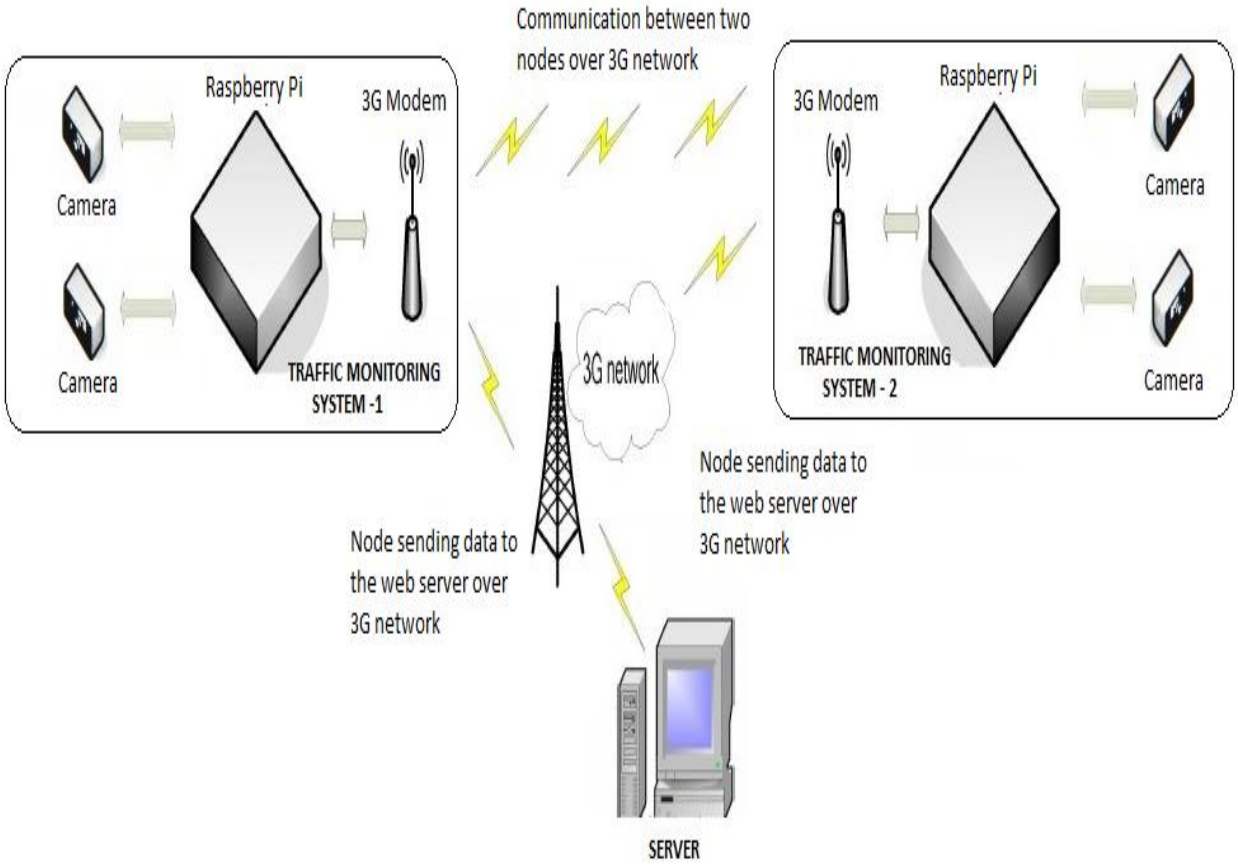

Fig. 12: Advanced traffic monitoring system

As shown in the figure above an advanced traffic monitoring system may involve several such systems interfaced to the 3G network. Individual nodes evaluate traffic status of the road link they are installed across, they also upload the same traffic status to the web server over 3G network.

A server can also be used to analyze the information sent by all of the traffic monitoring systems together. As shown in the above figure the web server receives traffic statistics from two traffic monitoring systems. The web server can process the received information and hence provide basis to administer traffic flow intelligently.

The proposed prototype uses only two interconnected traffic monitoring systems. When numerous such traffic monitoring systems are installed across the city, with each such system uploading corresponding traffic status to the web server, the web server can process inputs from all such nodes and hence can provide good analysis of the traffic situation across the city. Such analysis can be used to determine the least congested path

between given source and destination and also to avoid traffic congestion by intelligently administering the traffic flow.

3.3 Prototype to show communication between a node with multi-camera architecture and its neighbor

Consider diagram shown below. This diagram shows a node N1. This node has a multi-camera architecture. It has three cameras C1, C2 and C3 installed across road links R1, R2 and R3 respectively. Node N1 receives traffic video inputs from each of the cameras C1, C2 and C3 and then uses algorithm (described later) to determine overall traffic status. This overall traffic status is stored in 'index.xml' (or other suitable format) file of node N1. Another node N2, which is neighbor of N1, can copy the XML file stored at N1 to determine traffic status at that node.

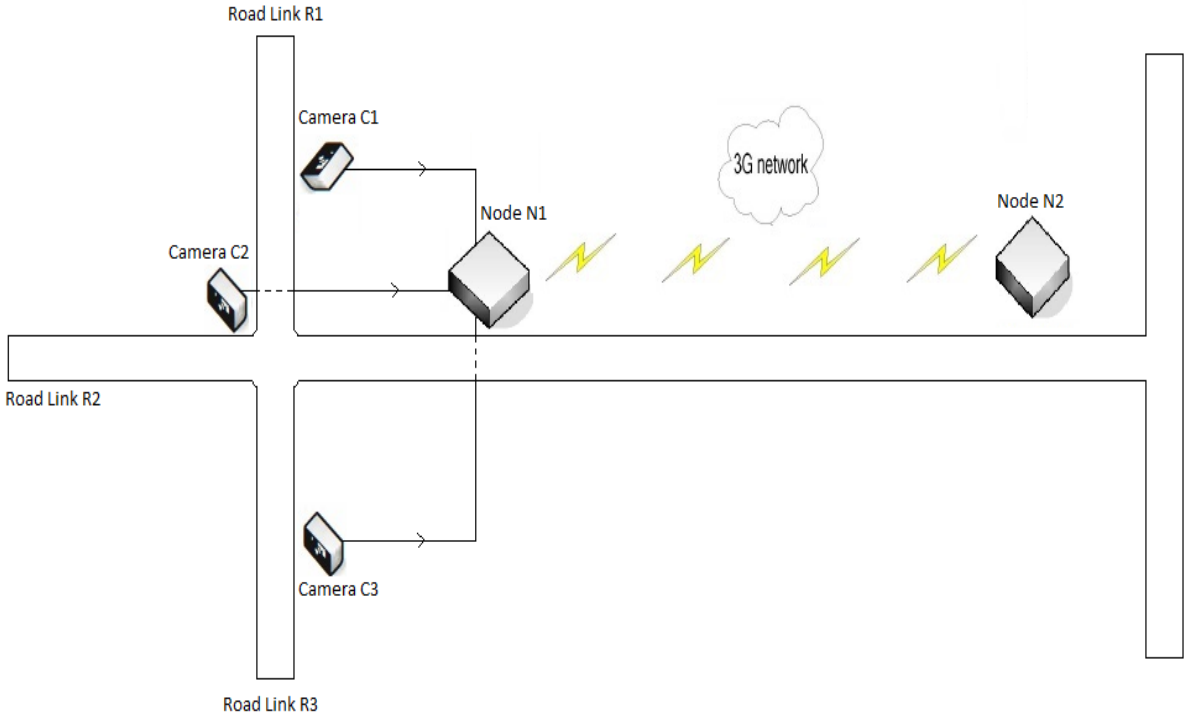

Fig. 13: Prototype to show communication between a node with multi-camera architecture and its neighbor

Node N1 receives traffic recording from C1, C2 and C3 and processes the same to determine traffic state at road links R1, R2 and R3 respectively. Now, to determine overall traffic status, proposed algorithm is given below:

1. Traffic situation is recorded by cameras C1, C2 and C3 across R1, R2 and R3 respectively and sent for processing.

2. Let,

   S1 = traffic status across R1
   S2 = traffic status across R2
   S3 = traffic status across R3

   The different traffic status values are mentioned in Table-1. S1, S2, S3 may assume values as given in the Table-1.
3. The processing algorithm decides the overall status based on majority as explained below,
   - If S1 = S2 = S3 then overall status is chosen as S1; since all the statuses are same so majority requires the overall status to be the same as that of S1.
   - If S1 = S2 ≠ S3, then overall status is chosen as S1; since both S1 and S2 are same and hence represent the majority.
   - If S1 = S3 ≠ S2, then overall status is chosen as S1; since S1 and S3 are same and hence represent the majority.
   - If S2 = S3 ≠ S1, then overall status is chosen as S2.
   - If S1 ≠ S2 ≠ S3, then we can decide the overall traffic status based on particular design decisions. Such cases can be considered while finalizing design specifications and suitable solution for the overall traffic status can be chosen. One such approach is as mentioned below.

     The statuses S1, S2, S3 may represent any combination of statuses 'Stopped', 'Slight Traffic', 'Heavy Traffic' and 'Open' and permutations thereof. Based on the following table we can determine the overall traffic status.

     Here overall traffic status is chosen such that it offers the best compromise between the input traffic status values. For instance, if the three statuses are 'Stopped', 'Slight Traffic' and 'Heavy Traffic' then the overall status can be chosen as 'Heavy Traffic' because it is close to two of the statuses that are 'Stopped' and 'Heavy Traffic'. Thus, overall traffic status is chosen such that it agrees with at least two of the input statuses closely.

Input Traffic Status Values (S1, S2, S3)	**Overall Traffic Status**
Stopped, Slight Traffic, Heavy Traffic	Heavy Traffic
Stopped, Slight Traffic, Open	Slight Traffic
Stopped, Heavy Traffic, Open	Heavy Traffic
Slight Traffic, Heavy Traffic, Open	Slight Traffic

Table 2: Design decisions for overall traffic status when S1 ≠ S2 ≠ S3

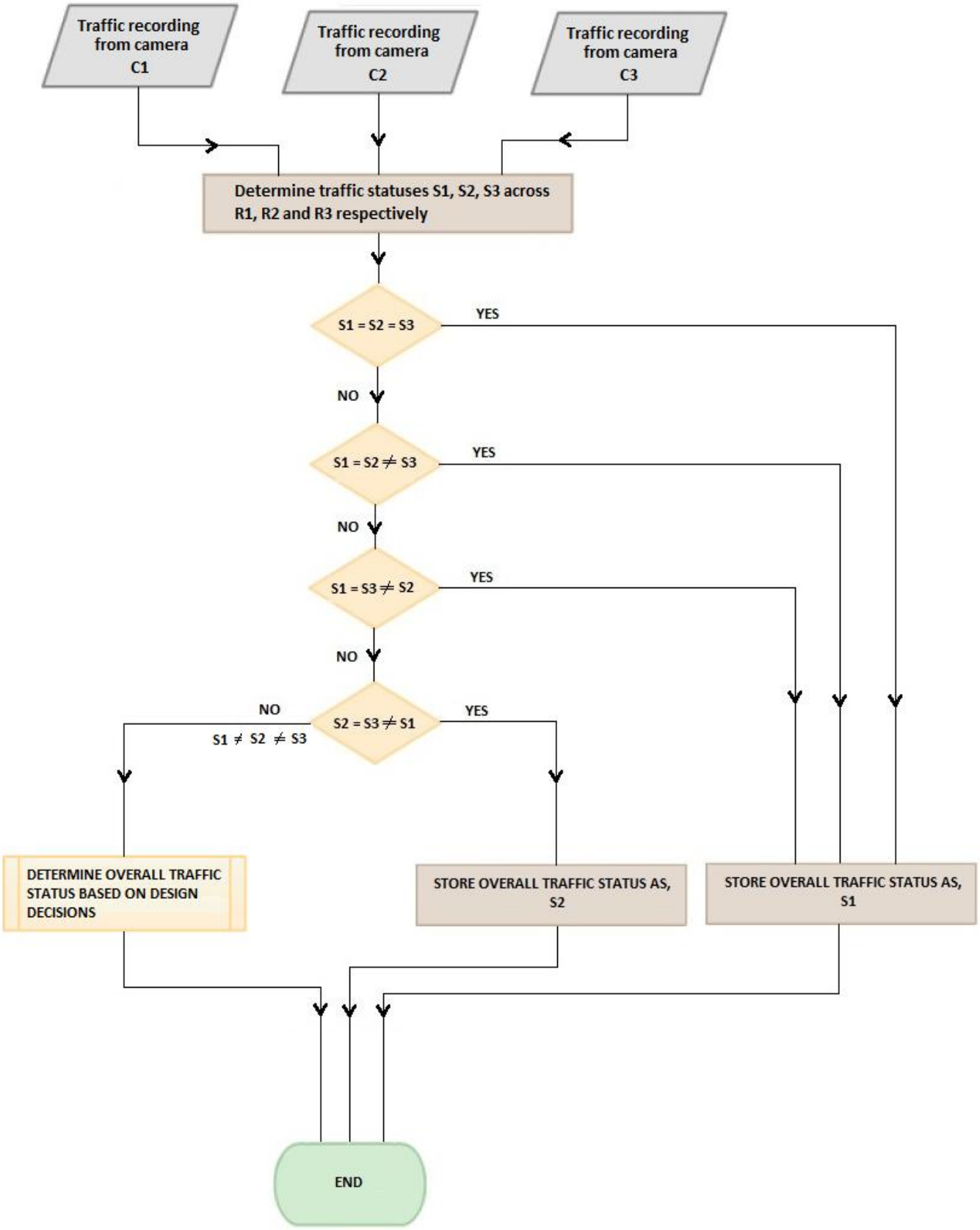

Fig. 14: Algorithm employed at node N1 to determine overall traffic status

Note that, in Table-2 different permutations of a particular combination of S1, S2 and S3 values are not represented to avoid unnecessary complexity. For each such permutation the overall traffic status would be the same as that of the corresponding combination in the Table-2.

All the traffic statuses of road links R1, R2 and R3 and overall traffic status may be stored in 'index.xml' file (or other suitable format). This file is copied periodically by node N2 to determine information recorded at node N1.

3.4 Grouping of nodes in the network

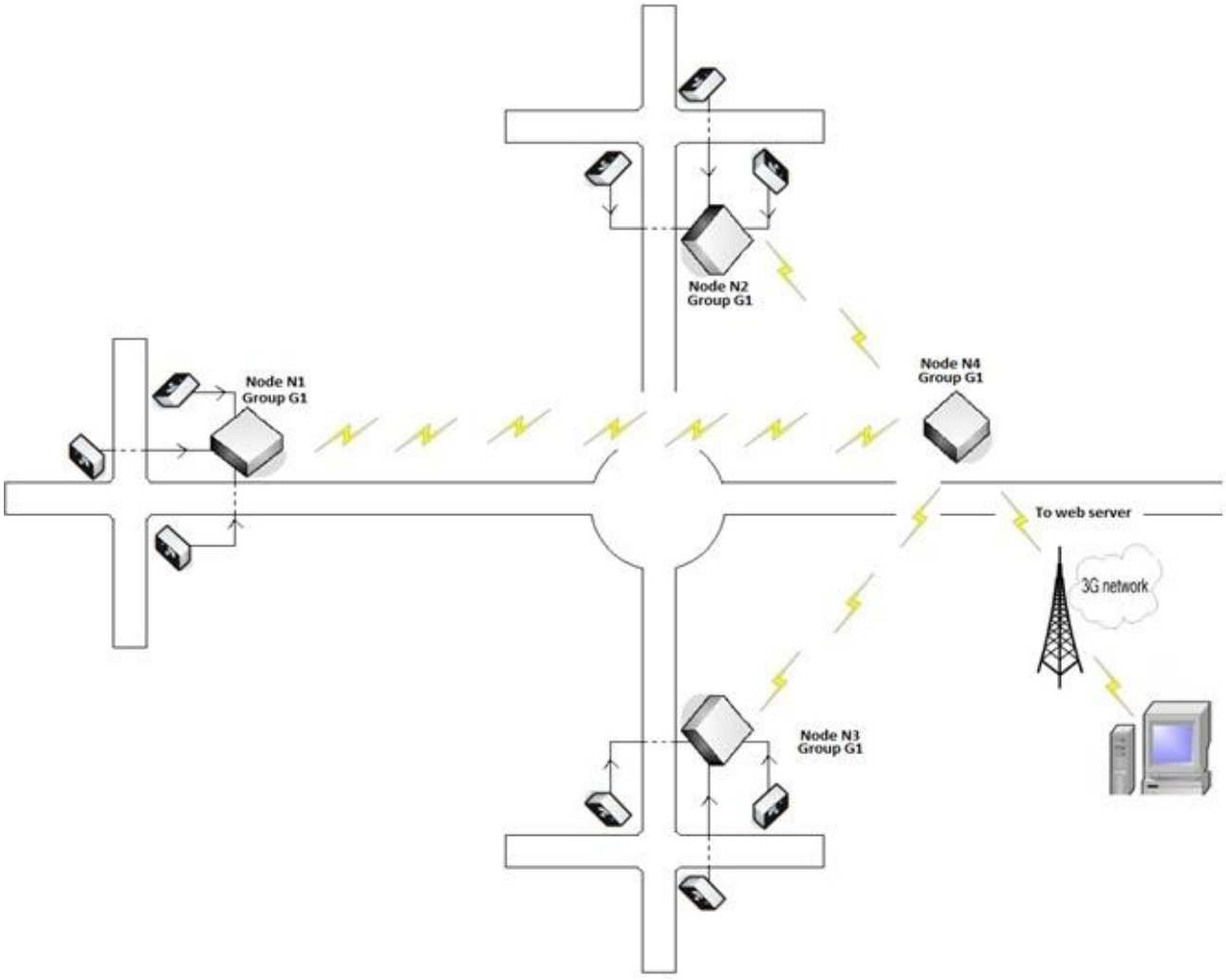

Fig. 15: Nodes working as a group

Nodes may be divided into groups to represent traffic status of a cluster of road links. Such an arrangement is especially suitable when we need to update the overall traffic status of the cluster of road links to the web server. When we update this status to the web server a user can log onto the website to check traffic status of the area as a whole, instead of having to check each road link separately.

Also, practically we do not need traffic status of each and every road link to be stored at the website. Instead it will be good to store overall traffic state of important areas and road junctions, so that a typical user does not need to check individual road links and determine overall traffic status of the area. Dividing the nodes in groups facilitates this idea.

Such grouping is demonstrated by the diagram shown above.

CHAPTER 4

PROTOCOL EMPLOYED - SSH

This chapter discusses SSH based approach to implement a wireless communication network based on 3G internet. Also, key challenges and limitations faced in this approach will be discussed subsequently.

SSH stands for "Secure Shell". To establish communication among the nodes SSH protocol may be employed. Once each of the nodes is connected to 3G network using 3G modem, then each node is installed with SSH client and SSH server to facilitate communication with other nodes in the network. For this communication to take place IP addresses allocated to the nodes, by 3G service provider, are used.

Each of the nodes contains a configuration file stored in it. This file contains basic configuration details required to establish communication with other nodes. The file may be kept in any standard format like XML, text file or comma-separated values etc. Suppose we have chosen XML format then each node in the network contains corresponding XML file, containing configuration details, stored with the name 'index.xml'. Selection of XML format would be advantageous from the point of view of ease of processing the stored information thereafter; such processing may be required to display the traffic status, at different nodes, on the website or other types of processing of traffic statistics as may be required to administer the traffic flow.

To communicate information among the nodes this XML file (or other suitable format) is used. For example, if one node wants to check traffic status at the other node, it can simply copy the 'index.xml' file of the node concerned (via SSH) and then process the file to extract desired information. Also, this node maintains its own 'index.xml' file and keeps the same updated with latest data, so that other nodes can access this file as and when required.

4.1 Introduction to SSH

SSH ("Secure SHell") is a protocol for securely accessing one computer from another. Despite the name, SSH allows you to run command line and graphical programs, transfer files, and even create secure virtual private networks over the Internet.

To use SSH, you will need to install an SSH client on the computer you connect from, and an SSH server on the computer you connect to. To install the OpenSSH server, install the following package: opensshserver.

4.2 OpenSSH

OpenSSH (OpenBSD Secure Shell) is a set of computer programs providing encrypted communication sessions over a computer network using the ssh protocol. It was created as an open source alternative to the proprietary Secure Shell software suite offered by SSH Communications Security.

4.3 How SSH aids information sharing?

For establishing communication over 3G network we use IP addresses allocated to the nodes by 3G service provider. Each of the nodes in the network contains SSH-server and SSH-client installed in it.

SSH-client installed at this node allows this node to communicate with other nodes via sending SSH-request to the SSH-server installed at those nodes. Each of the nodes stores the traffic status and other details in some standard file format (say XML format). This file is stored by each node with the name 'index.xml'. Using this SSH request we can copy 'index.xml' file stored in other nodes and hence determine the traffic status recorded there.

SSH-server installed on a node allows it to serve SSH requests received from other nodes in the network. These requests are invoked by other nodes when they want to know the traffic status recorded at this node. Other nodes send SSH request to this node to copy its 'index.xml' file, so that they can know the traffic status at this node.

Using SSH protocol nodes can request for copying 'index.xml' file of other nodes. This facilitates communication among the nodes because any information that needs to be communicated to other nodes may be stored in 'index.xml' file with appropriate tag. Once this xml file is copied by other nodes they will have all the information that this node wanted to communicate to them.

4.4 Block Diagram

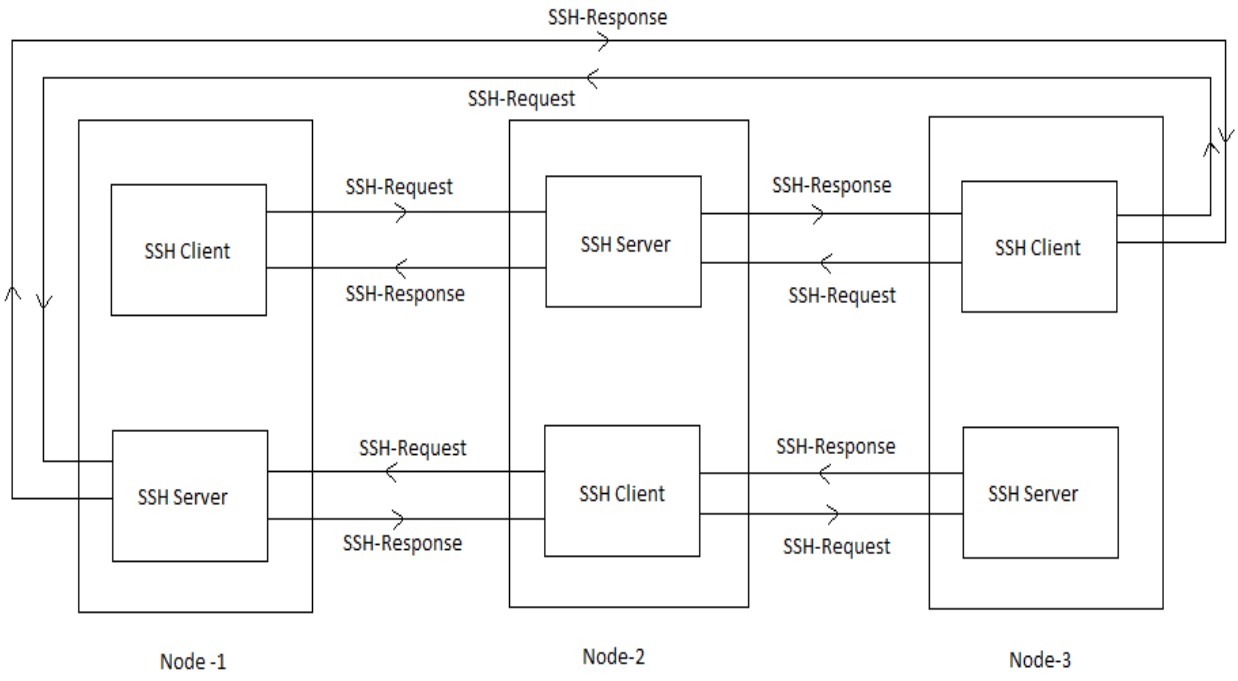

Fig. 16: Request-Response model for communication among nodes

4.5 How to configure SSH in any node?

OpenSSH server can be installed using following command:

sudo apt-get install openssh-server

To configure it we need to edit sshd_config file in the /etc/ssh directory. But it is advised that before editing this file you take a backup of this file in home directory, or you could as well make it read only in /etc/ssh by following command:

sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.factory-defaults
sudo chmod a-w /etc/ssh/sshd_config.factory-defaults

Taking backup helps you restore original configuration if it is required. Now, sshd_config file can be edited using text editor in following way:

sudo gedit /etc/ssh/sshd_config

Above command runs the most recent standard text editor in Ubuntu. To get older versions of text editor replace "sudo" by "gksudo" in above command. After opening text editor you can make required changes therein but to apply these changes you need to run following command.

sudo restart ssh

Configuring OpenSSH helps in maintaining a balance between security and comfort in using the same.

4.6 Disable Password Authentication

Usually SSH requests made to SSH-server require SSH-clients to provide username and password for authentication before any such requests can be served. But, in the scenario where we are going to employ SSH, we cannot expect the nodes to provide username and password each time they want to know the traffic status at other nodes.

To disable password authentication check sshd_config file for following line:
#PasswordAuthentication yes

Replace above line with the following:
#PasswordAuthentication no

4.7 How can the nodes communicate?

Once all the hardware is in place, nodes communicate using SSH protocol described in section 4.3 above. The sequence of events during such communication is described below:

a) When a node is switched on, it determines IP address assigned to it by the network provider. The same IP address is communicated to neighboring nodes using one of the techniques mentioned in section 4.10.2 of this report. Right now, we have chosen manual configuring option.
b) Each node determines traffic status corresponding to road links it is installed across and saves the same in some standard format with additional required parameters. Standard format may be either XML or text file or comma-separated values. Considering XML format, this file will be given the name 'index.xml'. Some of the parameters in this file are NodeID, NodeIP, NodeGroup, TrafficStatus, NeighborNodes.
c) TrafficStatus node of this file contains child nodes in which latest and previously recorded traffic statuses are stored, with respective times of recording. On demand latest traffic status can be determined based on time of recording.
d) NeighborNodes node in this file contains IP addresses of a node's neighbors as its child. These IP addresses are used to read traffic status recorded at the neighboring nodes. This is done via copying 'index.xml' file from other nodes using secure copy (scp) facilitated by SSH protocol. Once we have copied 'index.xml' file from other nodes, we can read traffic status of a node from corresponding 'index.xml' file.
e) Now each node in the network periodically records traffic status and updates the 'index.xml' file with this new status and its time of recording.
f) Also, each node periodically copies 'index.xml' files of its neighbors (these neighbors are mentioned in NeighborNodes node of the 'index.xml' file of the node under consideration), using 'index.xml' file of its neighbors a node can know the traffic status recorded at neighbor nodes.
g) Using traffic status, recorded by a node on its own, and those received from the neighbor nodes a node can determine overall traffic status of a group of nodes. This traffic status represents traffic state of the corresponding road links placed in a group.
h) This overall traffic status of a group can be uploaded on the web server, so that a user can access the same over the internet to determine the present traffic status at particular road links.
i) Sample 'index.xml' file is given in appendix-A for your reference.

4.8 Copying files over SSH

Files can be copied using Secure copy command as follows:
scp [options] original_file destination_file

Whenever we need to address a remote file, we can do it as shown below:
user@server:path/to/file

Server may be a URL or an IP address, then the path to the file or folder in question is specified as shown below:
scp –P 40050 Desktop/url.txt yatri@192.168.1.50:~/Desktop/url.txt

[-P] flag in above command helps us to specify a port number instead of default 22. Similarly we can copy files from remote server as shown below:

scp –P 40050 yatri@192.168.1.50:~/Desktop/url.txt Desktop/url.txt

Here, we have copied a file from the remote computer's "~/Desktop/" folder to our computer's "Desktop" folder. To copy whole directories, we need to use the [-r] flag.

scp -r –P 40050 Desktop/ yatri@192.168.1.50:~/Desktop

We can also combine flags. Instead of 'scp –P –r …' we can just do 'scp –Pr …'

4.9 SSH and SCP with Public/Private Keys

Secure copy can be put in scripts and used to do backups to remote computers. The problem is that it asks to enter the password. We can avoid this need to enter the password, for this we can have the computer generate two key files:

- Public - that belongs on the remote server
- Private - which is on our computer

These keys help in doing away with passwords. Following command will generate the two keys and put them in '~/.ssh/' with the names "id_rsa" for your private key, and "id_rsa.pub" for your public key.

ssh-keygen –t rsa

Next, you'll be asked to enter a passphrase. Hit Enter to leave this blank, then do it again when it asks for confirmation. Now to copy the public key file to your remote computer you can use scp:

scp –P 40050 .ssh/id_rsa.pub yatri@192.168.1.50:~/.ssh/authorized_keys2

The destination for your public key is on the remote server, in the following file:

~/.ssh/authorized_keys2

Subsequent public keys can be appended to this file, much like the '~/.ssh/known_hosts' file.

4.10 Key Challenges

The key challenges in implementing above design are as follows:

4.10.1 Obtaining IP address assigned to a node after initial boot up

Whenever RaspberryPi module boots up, it is likely to get different IP address assigned to it by the 3G service provider. In such a network it is not possible to assign any static IP address to individual nodes, because

assignment of IP addresses is done by network provider dynamically, based on IP addresses available that time.

In order to communicate with this node, we need to know its IP address first. Fortunately, we can use some scripts to determine the current IP address. One such script is mentioned below. This is just an illustration and it must be tested before use.

```
//=====================================================
// function: GetLocalIP
// description: retrieve current network ip and mask
//=====================================================
int GetIP(char *ifname, in_addr_t *ip, in_addr_t *mask)
{
struct ifreq ifr;
int sock = socket(AF_INET,SOCK_DGRAM,0);
// Get the interface IP address
strcpy( ifr.ifr_name, ifname );
ifr.ifr_addr.sa_family = AF_INET;
if (ioctl( sock, SIOCGIFADDR, &ifr ) < 0)
perror( "SIOCGIFADDR" );
*ip = ((struct sockaddr_in *)&ifr.ifr_addr)->sin_addr.s_addr;
shutdown(sock,SHUT_RDWR);
return 0;
}
```

4.10.2 Sharing of IP addresses among nodes to facilitate information sharing

Once the IP address of a particular node has been obtained, now we need to communicate the same to other nodes to facilitate information sharing. If IP addresses of other nodes are known beforehand, then no problem arises.

But, what if other nodes have also undergone booting at the same time? In such a case all of these nodes will be assigned different IP addresses by the network provider. The real problem lies here, because without knowing IP addresses of neighboring nodes, it is not possible to use IP address for communication. We need to be able to share IP addresses assigned to particular nodes with their neighbors to allow information sharing. For this problem possible solution are as follows:

- First, we can assume that, after initial boot-up the nodes remain on continuously and hence after initial boot-up IP addresses of neighbors can be fed manually. This will allow particular node to have IP address of its neighbors. But here the problem is that, if

any time later, should the node shutdown, we need to reconfigure it manually.

- Second, we can consider broadcasting of a node's IP address. During dynamic IP assignment, complete IP address does not change, in fact only a part of it changes, so keeping this part static and varying all the remaining part for all possible values, we can obtain possible IP addresses that might have been assigned to neighboring nodes. In this way we can get possible IP addresses of the neighbors and hence broadcast a particular node's IP address to these addresses to establish communication.
- Third, after every boot-up node retrieves IP address assigned to it and communicates the same to a web server. Other nodes in the network will also do the same. Also, the node can access web server to determine IP addresses of its neighbors. In this way each node can register itself at the server, and get IP addresses of its neighbors to establish information sharing.

4.11 Use of Amazon Web Service (AWS) to SSH into the Remote Server

Trying to SSH with IP addresses is somewhat troublesome because if we do not have access to the server (or to router configuration) it makes it difficult to map IP addresses to corresponding nodes because we do not have control over server (or router) configuration in that case. Hence we propose to use Amazon Web Service (AWS) for SSH purpose. Using AWS allows us to upload traffic data (stored in configuration file in suitable format like 'index.xml') from the node to the server using bash scripts directly, and we need not worry about IP addresses because such a mapping is taken care of by the AWS. Similarly, we can download traffic data, uploaded to server by other nodes, using bash script.

CHAPTER 5

IMPLEMENTATION DETAILS

As proposed in last chapter, we will use Amazon Web Service (AWS) to facilitate sharing of traffic data (based on SSH) among nodes of traffic monitoring system.

5.1 What is Amazon Web Service (AWS)?

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.

Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment.

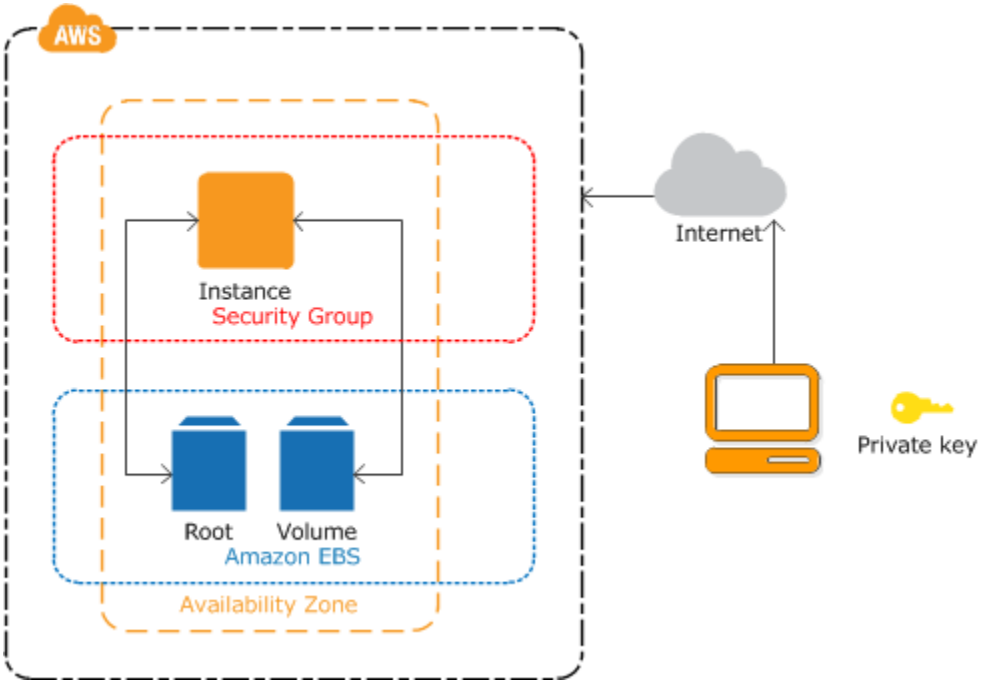

Fig. 17: Amazon EC2 Linux Instance

Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios. Features of Amazon EC2 are given below.

a) It provides virtual computing environments, referred to as instances.
b) It provides preconfigured templates for our instances. These templates are known as Amazon Machine Images (AMIs). These templates package the bits required for our server (including the operating system and software).
c) It provides various configurations of CPU, memory, storage and networking capacity for our instances, these are known as instance types.

d) Provides secure login information for our instances using key pairs (Public key is stored by AWS and we store the private key in a secure place).
e) It provides storage volumes for temporary data that is deleted when we stop or terminate our instance, these are known as instance store volumes.
f) It provides persistent storage volumes for our data using Amazon Elastic Book Store (Amazon EBS), these are known as Amazon EBS Volumes.
g) It allows multiple physical locations for our resources, such as instances and Amazon EBS Volumes, known as Regions and Availability Zones.
h) It provides a firewall that enables us to specify the protocols, ports and source IP ranges that can reach our instances using security groups.
i) It provides Static IP addresses for dynamic cloud computing, these are referred to as Elastic IP Addresses.
j) It provides Metadata, known as tags that we can create and assign to our Amazon EC2 resources.
k) Using AWS we can create virtual networks that are logically isolated from the rest of the AWS cloud. We can optionally connect to our own network, known as Virtual Private Clouds (VPCs).

5.2 Enabling 3G internet on Raspberry Pi

First of all we cannot connect our 3G dongle directly to raspberry pi in order to get internet connectivity. This is because 3G dongle takes more power than the Pi's USB sockets can supply. Hence 3G dongle is connected to Raspberry Pi via a powered USB hub (Transcend 4-port USB 3.0 hub). This ensures that sufficient power is available to fulfill the requirement of 3G dongle.

Now, after connecting the 3G dongle properly (via powered USB hub) we have used Sakis3g script to create an internet connection using 3G modem. This script works well for most of the USB and Bluetooth modems. This script is frequently used with modules such as Raspberry Pi and Beagle-Bone Black. This script can be used either through command lines or via a graphical user interface.

To use Sakis3g script through command lines following steps may be followed:

a) As a first step you need to install ppp package, command for the same is given below.
sudo apt-get install ppp

b) Now we can download the Sakis3g package using following command.
sudo wget "http://www.sakis3g.com/downloads/sakis3g.tar.gz" -O sakis3g.tar.gz

c) Now, unzip the above package.
sudo tar -xzvf sakis3g.tar.gz

d) Now, to make the file executable use following command.
sudo chmod +x sakis3g

e) Finally to launch the package use following command.
./sakis3g –interactive

After fourth step, shown above, Sakis3g will start in an interactive mode. Now we need to select the right options for our modem. We can choose either a predefined APN or enter a custom APN. If one of the custom APN fields is empty (no user ID and/or no password required), then enter 0.
On successful connection a confirmation message is received. If the modem is identified, but it fails to connect, the APN settings may be wrong. Sometimes modem may require additional configuration.

We need to use scripts such as Sakis3g because whenever a 3G modem is connected to raspberry pi then raspberry pi identifies this 3G modem as a USB by default. The 3G modem also contains some memory therefore it is recognized as a USB by raspberry pi module. To recognize the 3G modem properly a USB-modeswitch is required. Scripts such as Sakis3g facilitate such mode-switch therefore we have used the same. The GUI for Sakis3g is shown below.

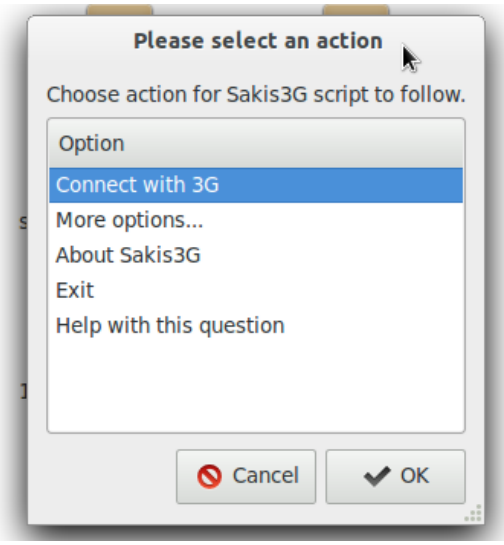

Fig. 18: Sakis3g GUI

5.3 Setting up with Amazon EC2

Follow steps given below to use Amazon EC2.

a) Sign Up for AWS – As a first step we need to sign up for Amazon Web Services (AWS). Once we have created AWS account it is automatically registered for all services in AWS (including Amazon EC2).

b) Create an IAM User – Amazon EC2 requires us to provide credentials in order to access the AWS services. This is required to verify whether you have the permission to access AWS services. We need to provide password into the

console in order to access AWS services. We can also create access keys for our AWS account to access the command line interface or API.

c) **Create a Key Pair –** AWS employs cryptography of public-key to ensure that the login information for our instance is secure. Since Linux instance has no password, we have to use key-pair to log into our instance securely.

d) **Create a Virtual Private Cloud (VPC) –** Amazon VPC allows us to launch AWS resources into a virtual network (defined by us). If we have a default VPC, we can skip this step.

e) **Create a Security Group –** Security groups act as a firewall for associated instances. These groups control both incoming and outgoing traffic at an instance. We can define rules that enable us to connect to our instance from our IP address using SSH.

Further details about steps (described above) are available at 'http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html'.

5.4 Problem Formulation

Let N1 and N2 represent two nodes in the system. Each of these nodes is basically a raspberry pi computer. Each node contains associated camera (USB camera), input devices such as keyboard and mouse, LCD display, a powered USB hub and a 3G modem. Note that the input devices (keyboard, mouse etc.) and LCD display are required only during design phase and can be removed after installation. Block diagram of a node is shown below.

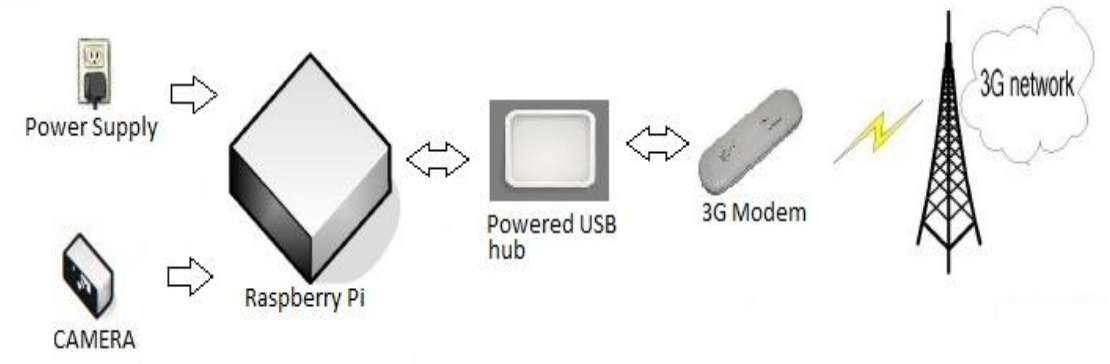

Fig. 19: Block diagram of a node

Each of the nodes records video of traffic through the camera and uses the same to extract traffic features. We have used traffic features same as those presented in [2]

to represent various traffic conditions. Now these traffic features must be shared among the nodes to facilitate intelligent management of traffic. Also, this traffic status must be uploaded over internet, so that one can see traffic status at these two nodes using one's mobile or using a PC connected to internet.

Thus, node N1 must be able to see traffic status recorded by N2 and must share its own traffic status with N2. Similarly, N2 must be able to see traffic status at N1 and must share its own traffic status with N1. Both N1 and N2 should upload their traffic statuses to central server, to allow access to same via websites. Power supply at any node represents battery of appropriate power.

5.5 Implementation

5.5.1 System layout for a system with 2 nodes

System architecture for a system, consisting of 2 nodes, is shown in figure 20.

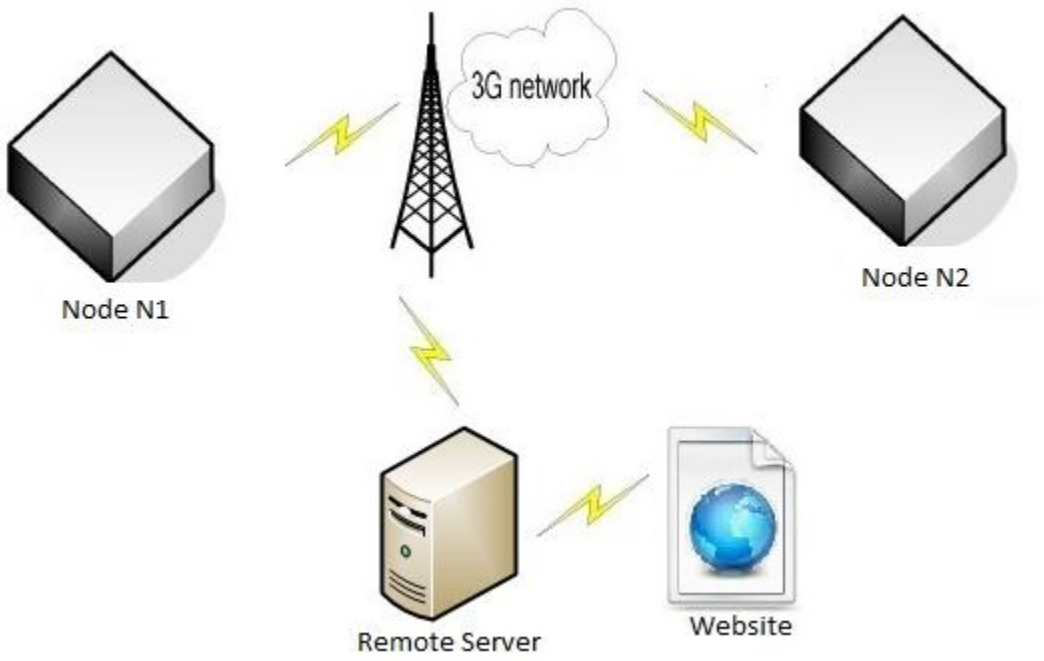

Fig. 20: System layout for a system with two nodes

As shown in above figure, Node N1 and Node N2 represent two nodes of the system. These two nodes are connected to 3G internet network, via 3G modems installed at each of them. Each of the nodes uploads its traffic status to a remote server using AWS. This traffic status can be downloaded by other nodes in the system using AWS. In this way traffic data sharing is facilitated. Since traffic data, corresponding to all the nodes

employed in the system, is available at remote server, this can be rendered in a webpage that one can access over the internet.

5.5.2 Hardware

The processing unit at each node is a Raspberry Pi B+ board which is a low power, small visiting card-sized computer based on Debian OS and has a processor of 700MHz. A USB camera is used to monitor the real time traffic. The real time images, acquired by the camera, are processed by Raspberry Pi to determine the traffic status across the corresponding road link. This traffic status needs to be transmitted to other nodes as well. The Pi board is powered using a micro-USB power adaptor.

The traffic status is put into a text file. This text file is transmitted over 3G network to allow sharing of traffic data. We have used Micromax MMX353G USB modem to connect the Pi to the internet. The steps required to enable 3G internet connection have already been described in section 5.2 above. The traffic data is sent to a remote server from one node and is downloaded at another node if required. Individual nodes download the traffic data files of nearby nodes only, because a node is concerned with traffic status at nearby nodes only.

As described in section 5.2 above, USB socket of Raspberry Pi does not provide enough power to meet the power requirements of a 3G dongle. A separate powered USB hub is required, to supply the required power to the modem. We have used Transcend 4-port USB 3.0 hub, which is powered externally by 18W DC adaptor, to serve the purpose. It is easily connected to the Pi board using the USB 3.0 cable. The USB modem is then connected to the hub. A 3G data enabled sim is inserted in the modem to provide 3G internet access to the Pi board.

5.5.3 Software

The traffic status, determined from the processing of the frames captured by the USB camera, is put into a text file. The text file so obtained is to be shared with other nodes for efficient traffic management. This sharing is done using Amazon Web Services. The server being used is a t1.micro EC2 instance, a cloud computing service provided by AWS. Amazon Web Services are used because of their high reliability and low maintenance cost. It has following features:

Operating System: Ubuntu
Memory: 0.594 GB
Architecture: 64 bit
Cost per hour: $0.020

5.5.4 Working

The nodes run a bash script to append the contents of text file (in which traffic data is stored) to a file on the remote server. This bash script contains login information to the server (the Private Key and the server IP address), which makes the system more reliable. The key must not be publicly visible during SSH login or transfers, so its permissions are changed. The bash script uses this information to send the data file to the server by running an infinite loop. A node can access data from other nodes simultaneously using appropriate filename.

The flowchart of the bash script, which communicates the data file to the server, is shown in figure 21.

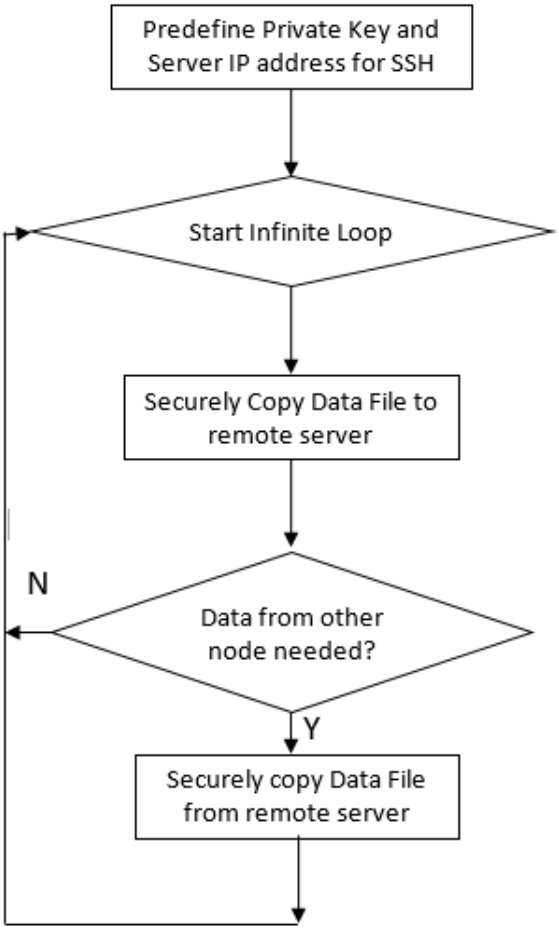

Fig. 21: Flowchart of the bash script used to upload traffic data

All the nodes upload the traffic data files on the server. On the other hand, each node periodically downloads the data file of only those nodes which are adjacent to it. Here, we have performed the process by considering two nodes. Both the nodes upload the data file on the server. Since both of

them are adjacent to each other (as per our assumption), we have showed that each of them downloads the file uploaded by the other. This concept can be extended to a multi-node system. For example, if we have 10 nodes and each one of them is uploading data to the server. Among these 10 nodes, each node will download the data files of its adjacent nodes; e.g. node 4 will download the files from nodes 3 & 5, similarly node 6 will download the files from nodes 5 & 7 and so on.

Another key job of the server is to show the traffic data (received from various nodes) into a website, so that anyone can know about traffic status at any node. This is done by using a web application hosting service called Heroku. Heroku is used because it allows us to develop our website and store traffic statuses (corresponding to system nodes) easily using git command line utility. Git is used as an intermediary for storing our website, before hosting it on Heroku platform. The flowchart to render traffic data into a website is shown in figure 22.

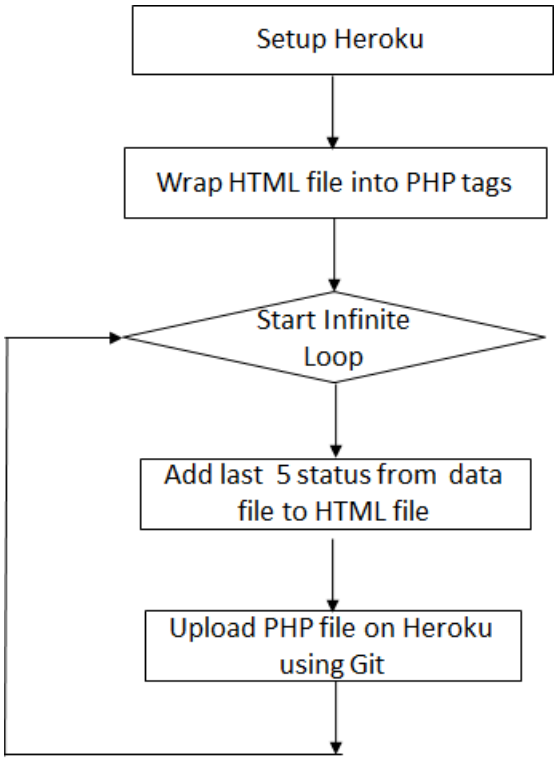

Fig. 22: Flowchart to render traffic data into a website

The traffic statuses received by the server from various nodes are updated to an HTML web page. Heroku currently is not directly compatible with HTML but since it supports PHP and PHP is highly compatible with HTML, we wrap the HTML file into PHP. This file is then uploaded on

the heroku web app server in an infinite loop, continuously updating it with relevant traffic information.

5.5.5 Bash script required at a node

The bash script used at node N1 is given below.

```
#!/bin/bash
chmod 400 key1.pem
for((;;))
do
for (( i=1;i<=4;i++ ))
do
scp -i key1.pem Node_N1.txt ubuntu@52.25.59.102:Node_N1.txt
echo "Traffic Status File of Node N1 Uploaded"
done
echo  "Downloading Neighbouring Node Status"
scp -i key1.pem ubuntu@52.25.59.102:Node_N2.txt .
echo "Traffic Status File of Node N2 Downloaded"
done
```

Similar bash script is used at node N2 as well, to upload the traffic data file of node N2 and download the traffic data uploaded by node N1. This bash script used at node N2 is given below.

```
#!/bin/bash
chmod 400 key2.pem
for((;;))
do
for (( i=1;i<=4;i++ ))
do
scp -i key2.pem Node_N2.txt ubuntu@52.25.59.102:Node_N2.txt
echo "Traffic Status File of Node N2 Uploaded"
done
echo  "Downloading Neighbouring Node Status"
scp -i key2.pem ubuntu@52.25.59.102:Node_N1.txt .
echo "Traffic Status File of Node N1 Downloaded"
done
```

Note that key1.pem and key2.pem, used in above scripts, represent authentication keys used at node N1 and N2 respectively

5.5.6 Bash script required to generate HTML file which displays traffic statuses recorded at nodes of the system

The bash script used to update the traffic statuses, received by the server from various nodes, to an HTML web page is given below.

```
#!/bin/bash

for ((i=0;i<1;i++))
do
rm home.html
echo "<HTML><TITLE>Traffic Monitoring
System</TITLE>">>/home/ubuntu/phpheroku/home.html
echo "<HEAD>Find traffic status update at Node N1 and Node N2
below</HEAD><br/><br/>">>/home/ubuntu/phpheroku/home.html
echo "<BODY>">>/home/ubuntu/phpheroku/home.html
echo "<TABLE><TR><TD><b><p>Traffic Status at Node
N1:</p></b><br/>">>/home/ubuntu/phpheroku/home.html
tail -20
/home/ubuntu/Node_N1.txt>>/home/ubuntu/phpheroku/home.html
echo "</TD><TD><b><p>Traffic Status at Node
N2:</p></b><br/>">>/home/ubuntu/phpheroku/home.html
tail -20
/home/ubuntu/Node_N2.txt>>/home/ubuntu/phpheroku/home.html
echo
"</TD></TR></TABLE></BODY></HTML>">>/home/ubuntu/phphero
ku/home.html

git add .
git commit -m "commited"
git push heroku master

done
```

5.5.7 HTML file generated by bash script

The HTML file generated by bash script (given above) is shown on the next page.

HTML file generated by bash script

```
<HTML>
<TITLE>Traffic Monitoring System</TITLE>
<HEAD>Find traffic status update at Node N1 and Node N2
below</HEAD><br/><br/>
<BODY>
        <TABLE>
        <TR>
                <TD>
                        <b><p>Traffic Status at Node  N1:</p></b><br/>
                        Date/Time: 18052015/ 15:22:30 <br/>
                        Traffic Code: 11 <br/>
                        Traffic Status: "Open" <br/><br/>
                        Date/Time: 18052015/ 15:22:55 <br/>
                        Traffic Code: 01 <br/>
                        Traffic Status: "Slight Traffic" <br/><br/>
                        Date/Time: 18052015/ 15:23:10 <br/>
                        Traffic Code: 10 <br/>
                        Traffic Status: "Heavy Traffic" <br/><br/>
                        Date/Time: 18052015/ 15:23:35 <br/>
                        Traffic Code: 10 <br/>
                        Traffic Status: "Heavy Traffic" <br/><br/>
                </TD>
                <TD>
                        <b><p>Traffic Status at Node N2:</p></b><br/>
                        Date/Time: 18052015/ 15:22:30 <br/>
                        Traffic Code: 11 <br/>
                        Traffic Status: "Open" <br/><br/>
                        Date/Time: 18052015/ 15:22:55 <br/>
                        Traffic Code: 01 <br/>
                        Traffic Status: "Slight Traffic" <br/><br/>
                        Date/Time: 18052015/ 15:23:10 <br/>
                        Traffic Code: 01 <br/>
                        Traffic Status: "Slight Traffic" <br/><br/>
                        Date/Time: 18052015/ 15:23:35 <br/>
                        Traffic Code: 11 <br/>
                        Traffic Status: "Open" <br/><br/>
                </TD>
        </TR>
        </TABLE>
</BODY>
</HTML>
```

5.6 Results

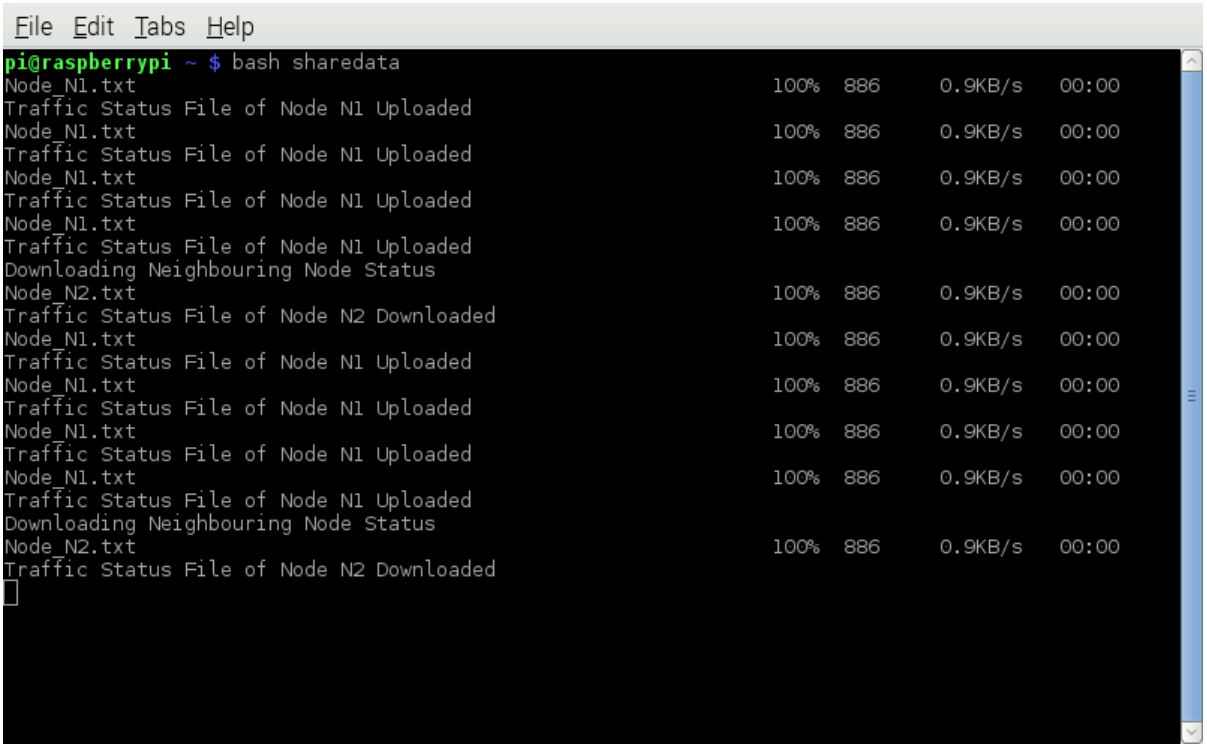

Fig. 23: Snapshot of traffic data upload/download at node N1

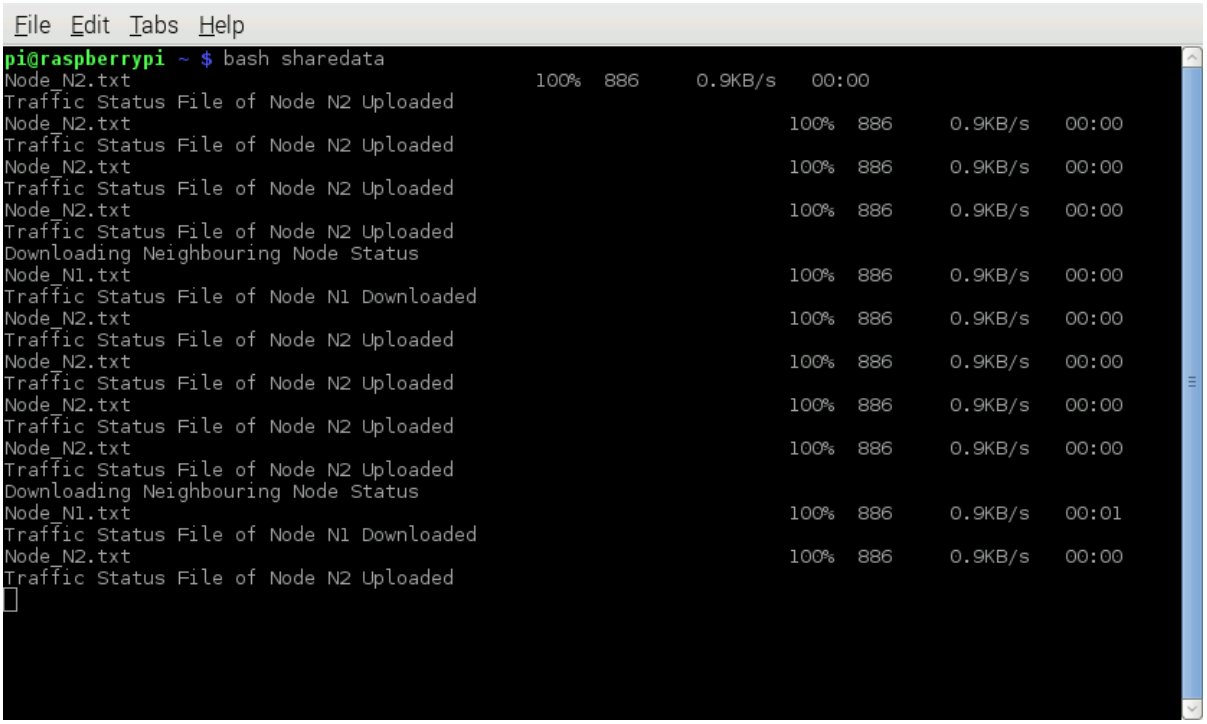

Fig. 24: Snapshot of traffic data upload/download at node N2

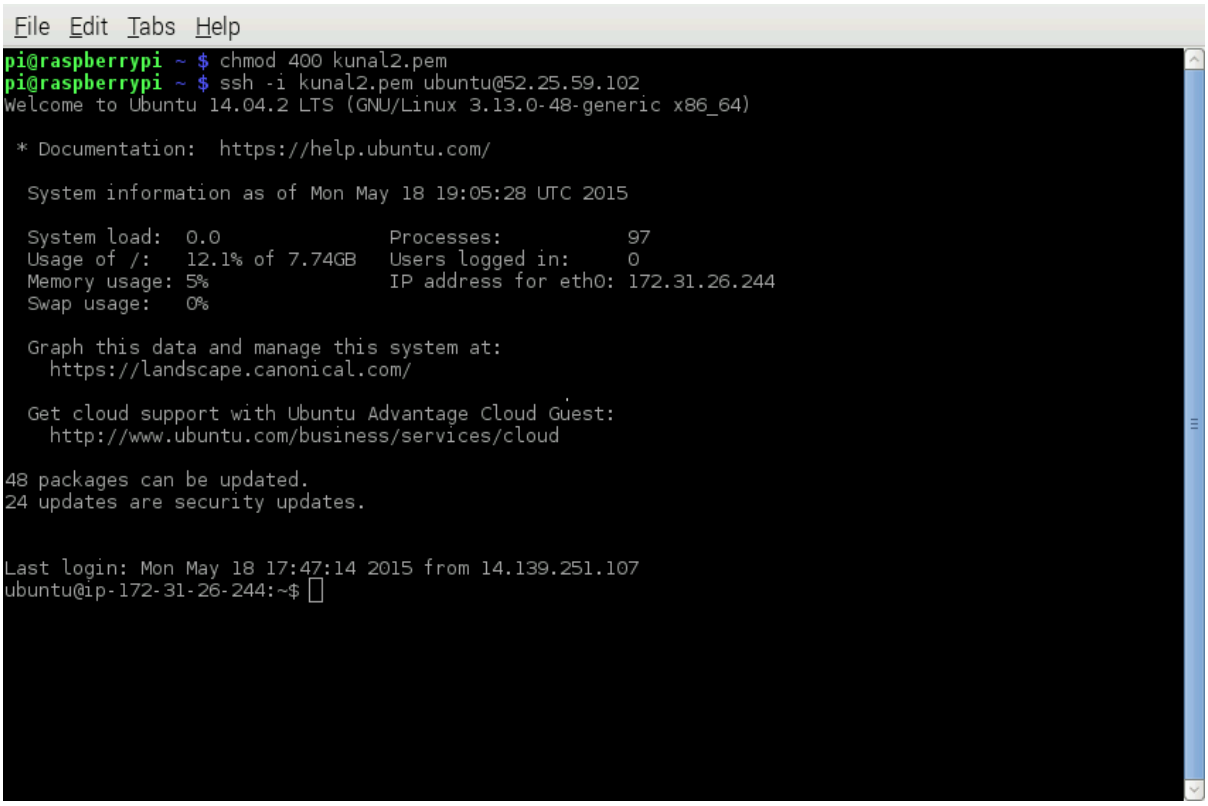

Fig. 25: Snapshot of logging into the remote server

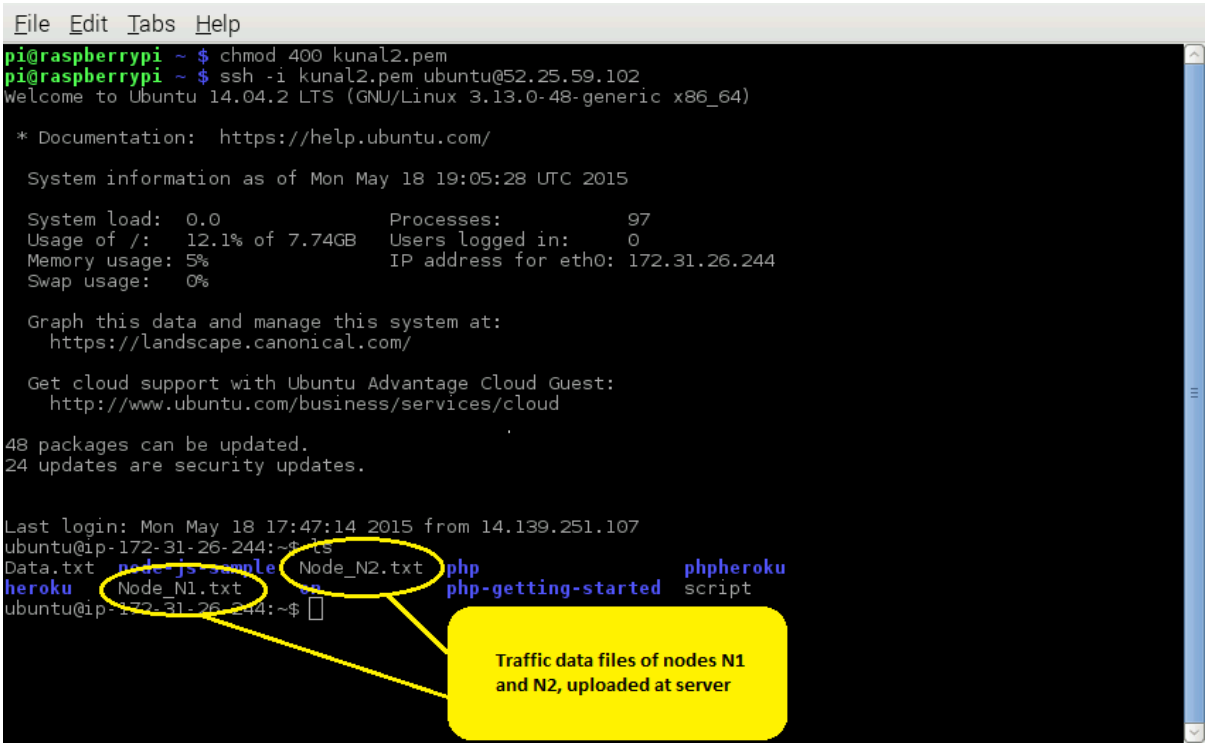

Fig. 26: Snapshot showing traffic data files uploaded on the server

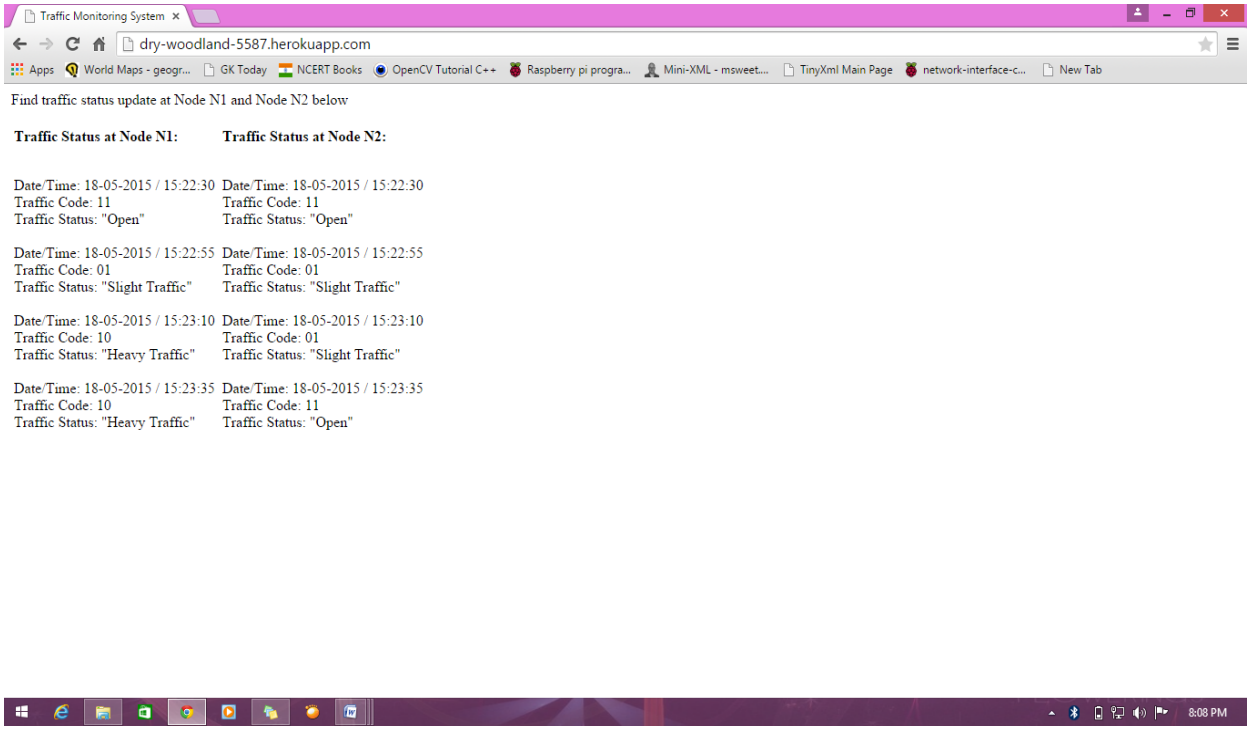

Fig. 27: Snapshot of website showing traffic status at nodes N1, N2

Fig. 28: Snapshot of traffic data accessed via smart phone

CHAPTER 6

CONCLUSION & FUTURE SCOPE

This report explained implementation of wireless communication network among nodes of a traffic monitoring system using 3G network. The approach adopted using SSH in conjunction with AWS has been explained with results thereof. The system thus implemented has following advantages, limitations and future scope.

6.1 Advantages

The wireless communication network implemented in this project offers following advantages:

1. A wireless 3G network facilitates implementation of ad-hoc network based on distributed processing.
2. Since the network is wireless it improves the system's scalability.
3. The data received at the web server can be used to monitor traffic status across the city.
4. The same server can help determine the least congested path between given source and destination. Also, it can help in avoiding the traffic congestion by facilitating smooth administration of traffic.
5. By developing suitable mobile applications, it can be possible to get the traffic status dynamically, provided the mobile has access to 3G network. In this way people can be more informed about the traffic status across the city and plan their route accordingly.
6. Traffic data recorded at various nodes, once sent to the server, can be used to formulate traffic statistics during different times of the day.
7. Using suitable processing algorithms the traffic data can be processed and analysis thus performed can be used to adjust the duration of traffic lights dynamically, this helps in efficient regulation of traffic. The analysis presented in [3] provides two such algorithms, namely Optimal Weight Calculator (OWC) and Fair Weight Calculator (FWC), which can be used to reduce waiting period of vehicles at road junctions without detecting or tracking vehicles.

6.2 Limitations

The limitations of the system implemented herein are as below:

1. Once configured, a node will be able to provide service only as long as the power supplied by associated battery is sufficient as per system requirements.
2. If the node shuts down due to power shortage or some other failure then it needs to be configured again after initial boot-up. This reconfiguration might involve running of bash scripts and ensuring connectivity to 3G network.
3. The efficiency of data transmission is limited by the performance of the 3G network employed.
4. The features available with free version of AWS are limited and to get access to additional features a paid subscription of this web service will have to be obtained.
5. The availability of free version of AWS is subject to discretion of the service provider, and may cease to be available if the provider so desires.

6.3 Future Scope

In future the same system can be extended to include multi-camera architecture, in which a single node may have multiple cameras associated with it, images from each camera are processed and then traffic statuses corresponding to data recorded by individual cameras are used to determine overall traffic status of that node.

Traffic statuses received from all the nodes in the network can be used to dynamically adjust traffic flow, such that the roads with maximum traffic are given green light for relatively longer duration as compared to those with moderate traffic. An analysis given in [3] presents algorithm towards the same. It is possible to determine traffic statistics from the data and analyze the same to identify the traffic patterns.

One of the important extensions can be to remove dependency on AWS by finding appropriate substitute for the same. In the implementation presented in this report we have assumed that traffic data is recorded in plain text file (.txt file) at individual nodes, if the traffic data is recorded in more sophisticated file formats (e.g. XML) then suitable code must be developed to process such files.

REFERENCES

[1] S. Indu, Sankalp Arora, Santanu Chaudhury, Asok Bhattacharyya. Road Traffic Model using distributed Camera Network. ICVGIP ’10, December 12-15, 2010, Chennai, India.

[2] S. Indu, Rajat Agarwal, Akshay Khalatkar, Vikash Maurya. Implementation Of Camera-based Traffic Monitoring System. WIT Transactions on Information and Communication Technologies, Paper DOI-10.2495/ICTE130541, 2014. Southampton, UK.

[3] S. Indu, Varun Nair, Shashwat Jain, Santanu Chaudhury. Video based Adaptive Road Traffic Signaling. Distributed Smart Cameras (ICDSC), 2013 Seventh International Conference, October 29, 2013-November 1, 2013. Palm Springs, CA.

[4] Prof. Purushottam Kulkarni, Ashish Dhar. Traffic and Road Condition Monitoring System. M.Tech. Stage 1 Report, Department of Computer Science and Engineering Indian Institute of Technology, Bombay, November-2008. Mumbai, India.

[5] Lei Zhang, Rui Wang, Li Cui. Real-time Traffic Monitoring with Magnetic Sensor Networks. Academia Sinica Institute of Information Science, July 29, 2010. Taipei, Taiwan.

[6] Sing Yiu Cheung, Sinem Coleri Ergen, Pravin Varaiya. Traffic Surveillance with Wireless Magnetic Sensors. ITS World Congress, November 2005. Sariyer, Istanbul.

[7] P. Samczynski, K. Kulpa, M. Malanowski, P. Krysik, Ł. Maślikowski. A Concept of GSM-based Passive Radar for Vehicle Traffic Monitoring. Microwaves, Radar and Remote Sensing Symposium (MRRS), 2011. IEEE, August 25-27, 2011. Kiev, Ukraine.

[8] Nieves Gallego, Antonio Mochol´I, Miguel Menendez, Raymundo Barrales. Traffic Monitoring: improving Road Safety using a Laser Scanner Sensor. Electronics, Robotics and Automotive Mechanics Conference, 2009. CERMA '09. IEEE, September 22-25, 2009. Cuernavaca, Morelos.

[9] Tarik M. Hussain, Tarek N. Saadawi, Samir A. Ahmed. Overhead Infrared Sensor for Monitoring Vehicular Traffic. IEEE Transactions on Vehicular Technology, Vol. 42, No. 4, November, 1993.

[10] Eng-Han Ng, Su-Lim Tan, Jesus Garcia Guzman. Road traffic monitoring using a wireless vehicle sensor network. International Symposium on Intelligent Signal Processing and Communication Systems, 2008 (ISPACS 2008) SwissOtel Le Concorde, February 8-11, 2009. Bangkok, Thailand.

[11] Cheng Qiong, Liu Li, Sun Min. The Design of a Data Transmission System Based on 3G and Embedded Technology. The 8th International Conference on Computer Science & Education (ICCSE 2013), April 26-28, 2013. Colombo, Sri Lanka.

APPENDICES

APPENDIX A
SAMPLE 'INDEX.XML' FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<Node>
        <NodeID>N1</NodeID>
        <GroupID>G1</GroupID>
        <NumOfNeighbors>1</NumOfNeighbors>
        <NumOfChildRoadLinks>3</NumOfChildRoadLinks>
        <NodeIP>192.168.1.100</NodeIP>
        <TrafficStatus>
                <RoadLink>
                        <RoadLinkID>R1</RoadLinkID>
                        <CameraID>C1</CameraID>
                        <TrafficState>
                                <BinaryCode>00</BinaryCode>
                                <State>Stopped</State>
                                <Date>30/11/2014</Date>
                                <Time>7:10:22 PM</Time>
                        </TrafficState>
                                                <TrafficState>
                        <BinaryCode>01</BinaryCode>
                        <State>Slight Traffic</State>
                        <Date>30/11/2014</Date>
                        <Time>6:15:56 PM</Time>
                        </TrafficState>
                </RoadLink>
                <RoadLink>
                        <RoadLinkID>R2</RoadLinkID>
                        <CameraID>C2</CameraID>
                        <TrafficState>
                                <BinaryCode>10</BinaryCode>
                                <State>Heavy Traffic</State>
                                <Date>30/11/2014</Date>
                                <Time>7:10:22 PM</Time>
                        </TrafficState>
                </RoadLink>
                <RoadLink>
                        <RoadLinkID>R3</RoadLinkID>
                        <CameraID>C3</CameraID>
                        <TrafficState>
                                <BinaryCode>01</BinaryCode>
                                <State>Slight Traffic</State>
                                <Date>30/11/2014</Date>
                                <Time>7:10:22 PM</Time>
                        </TrafficState>
```

```
            </RoadLink>
        </TrafficStatus>
        <OverallTrafficStatus>
            <RoadLinksCovered>
                <RoadLinkID>R1</RoadLinkID>
                <RoadLinkID>R2</RoadLinkID>
                <RoadLinkID>R3</RoadLinkID>
            </RoadLinksCovered>
            <TrafficState>
                <BinaryCode>10</BinaryCode>
                <State>Heavy Traffic</State>
                <Date>30/11/2014</Date>
                <Time>7:12:22 PM</Time>
            </TrafficState>
            <TrafficState>
                <BinaryCode>11</BinaryCode>
                <State>Open</State>
                <Date>30/11/2014</Date>
                <Time>5:02:35 PM</Time>
            </TrafficState>
        </OverallTrafficStatus>
        <NeighborNodes>
            <NeibhborNode>
                <NeighborID>N2</NeighborID>
                <NeighborIP>192.168.55.103</NeighborIP>
            </NeighborNode>
        </NeighborNodes>
</Node>
```

APPENDIX B

C++ code (Developed using OpenCV Platform) to find traffic status from input given by the camera

```
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <time.h>
#include <opencv/cxcore.h>
#include <iostream>
#include <fstream>
#include <cstdio>
#include <sys/timeb.h>

int GetMilliCount()
{
	// Something like GetTickCount but portable
	// It rolls over every ~ 12.1 days (0x100000/24/60/60)
	// Use GetMilliSpan to correct for rollover
	timeb tb;
	ftime( &tb );
	int nCount = tb.millitm + (tb.time & 0xfffff) * 1000;
	return nCount;
}

int GetMilliSpan( int nTimeStart )
{
	int nSpan = GetMilliCount() - nTimeStart;
	if ( nSpan < 0 )
	nSpan += 0x100000 * 1000;
	return nSpan;
}

using namespace std;
using namespace cv;

int countSIP=0;
int countSTIP=0;
int
array1[320][240]={0},array2[320][240]={0},array3[320][240]={0},a1s[80][60],a2s[80][60],a3s[
80][60];

int thresh=90;
struct resFinal{
	Mat destImag;
	int count;
};
void out(Mat , Mat);
resFinal cornerHarrisFunc(Mat src_gray);
   ofstream fp;
```

```
int a;
const int frs=2;
int main()
{

	CvCapture* capture = cvCaptureFromCAM(0);
	Mat frame1,frame2;
	fp.open("Points.txt");
	cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, 320);
	cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, 240);

	time_t start, end;
	double fps;
	int counter = 0;
	double sec;

	IplImage* curFrame[frs];
	Mat fr[frs];
	Mat f1,f2;
	if( capture)
		{
			//sleep(1);
			for (int x=0;x<frs;x++)
				curFrame[x]=cvCloneImage(cvQueryFrame(capture));

			int k=0,b=frs-1;
			time(&start);
			cout<<GetMilliCount();
			//  cout<<coun;
			for(;;)
				{
					if(k==-1) k=frs-1;
					if(b==-1) b=frs-1;

					curFrame[k] = cvCloneImage(cvQueryFrame(capture));
					frame1 = curFrame[k];
					if(frame1.empty())
				{
					cerr << "ERROR: Unable to capture frame" << endl;
					break;
				}
				frame2= curFrame[b];
	    cvNamedWindow("Frame 1");
	    cvNamedWindow("Frame 2");

	    cvMoveWindow("Frame 1",0,0);
	    cvMoveWindow("Frame 2",700,0);
	    imshow( "Frame 1", frame1 );
```

```
            imshow( "Frame 2", frame2 );
                                        k--;
                                        b--;
                                        if (!frame1.data)
                                        cout << "Unable to load source image1";
                                        if (!frame2.data)
                                        cout << "Unable to load source image2";

                                        time(&end);
                                        ++counter;
                                        sec = difftime (end, start);
                                        fps = counter / sec;
                                        cout<<"FPS = "<< fps<<endl;
                                        f1=frame1;
                                        f2=frame2;
                                         //  cout<<"before out"<<GetMilliCount()<<endl;
                                         a=GetMilliCount();
                                        out(f1,f2);
                                        if( waitKey( 10 ) >= 0 )
                                        goto releaseLabel;
                                }

                        waitKey(0);
                    releaseLabel:
                cvReleaseCapture( &capture );
                }
        fp.close();

        cvDestroyWindow("Frame 1");
        cvDestroyWindow("Frame 2");
        return 0;
}



void out(Mat edges1, Mat edges2)
{

        Mat test;
        test=edges1;
        countSTIP=0;
        resFinal res1,res2,res11,res22;
        cout<<"ch1"<<GetMilliCount()<<endl;
        res1=cornerHarrisFunc(edges1);
        for(int i=0;i<320;i++)
                for(int j=0;j<240;j++)
                        array2[i][j]=array1[i][j];
        //cvMoveWindow("SIP",0,350);
        //imshow("SIP",res1.destImag);
        int aa= GetMilliSpan(a);
```

```
	cout<<"Check 1  "<<aa<<endl;
	 // cout<<"ch2"<<GetMilliCount()<<endl;
	int  b=GetMilliCount();
	res11=cornerHarrisFunc(edges2);
	 int bb= GetMilliSpan(b);
	 cout<<"Check 2 "<<bb<<endl;
	 int  c=GetMilliCount();
	for(int i=0;i<320;i++)
		for(int j=0;j<240;j++)
			array3[i][j]=array1[i][j];
	cout<<"No. of SIPs::"<<res1.count<<endl;

	for (int j = 0; j < 320; j++)
		{
			for (int i = 0; i <240; i++)
				{
					if ((array2[i][j]==1)&&(array2[i][j]!= array3[i][j]))
					{
						countSTIP++;
					}
				}
		}
	 int cc= GetMilliSpan(c);
	cout<<"Check 3  "<<cc<<endl;
	//cout<<"after stip"<<GetMilliCount()<<endl;
	 /*  for (int j = 0; j < 320; j++)
		{
			for (int i = 0; i <240; i++)
				{
					if ((array2[i][j]==1)&&(array2[i][j]!= array3[i][j]))
					{
					    if (countSTIP<600 )
					    circle(test, Point(j, i), 5, Scalar(0,0,255), 0, 8, 0);
					}
				}
		}
	fp<<"SIP 1\t"<<res1.count<<"\nSIP 2\t"<<res11.count<<"\nSTIP\t"<<countSTIP<<endl;

	 cvMoveWindow("STIP",0,0);
	 imshow("STIP",test);*/
	cout<<"No of STIPs= "<<countSTIP<<"\n"<<endl;

}
resFinal cornerHarrisFunc(Mat src_gray)
{
	Mat dst, dst_norm, dst_norm_scaled;
	Mat finalDst;
	dst=src_gray;
	if (src_gray.empty())
	cout << "Unable to load source image";
	finalDst = src_gray;
```

```
	for(int i=0;i<320;i++)
		for(int j=0;j<240;j++)
			array1[i][j]=0;
	int blockSize = 7;
	int apertureSize = 7;
	double k = 0.04;
	Mat framex,framey;
	cvtColor(src_gray, framex, CV_BGR2GRAY);
	framex.convertTo(framey, CV_32F);
	if (!framey.data)
	cout << "Unable to load source imagex";
	int  d=GetMilliCount();
	cornerHarris(framey, dst, blockSize, apertureSize, k, BORDER_DEFAULT);
	int dd= GetMilliSpan(d);
	cout<<"Check 4 "<<dd<<endl;
	if (dst.empty())
	cout<< "Unable to load destination image";
	normalize(dst, dst_norm, 0, 255, NORM_MINMAX, CV_32FC1, Mat());
	convertScaleAbs(dst_norm, dst_norm_scaled);
	countSIP = 0;
	cout<< "Thresh = "<<thresh<<endl;
	for (int j = 0; j < dst_norm.rows; j++)
		{
			for (int i = 0; i < dst_norm.cols; i++)
				{
					if ((int) dst_norm.at<float> (j, i) > thresh)
					{
					array1[j][i]=1;
						countSIP++;
					}
				}
		}
/*for (int j = 0; j < dst_norm.rows; j++)
{
	for (int i = 0; i < dst_norm.cols; i++)
		{
			if ((int) dst_norm.at<float> (j, i) > thresh)
			{
			if (countSIP<500)
			{
			  circle(dst_norm_scaled, Point(i, j), 5, Scalar(255,255,0), 0, 8, 0);
			  circle(finalDst, Point(i, j), 5, Scalar(255,255,0), 0, 8, 0);
			} } }
	}
	//cvMoveWindow("cornerHarris",700,0);
	imshow("cornerHarris",dst_norm_scaled );*/
	resFinal res;
	res.destImag=finalDst;
	res.count=countSIP;
	return res;
}
```

APPENDIX C
IMAGES OF HARDWARE USED

Fig. 29: Hardware for Raspberry Pi to 3G modem interfacing

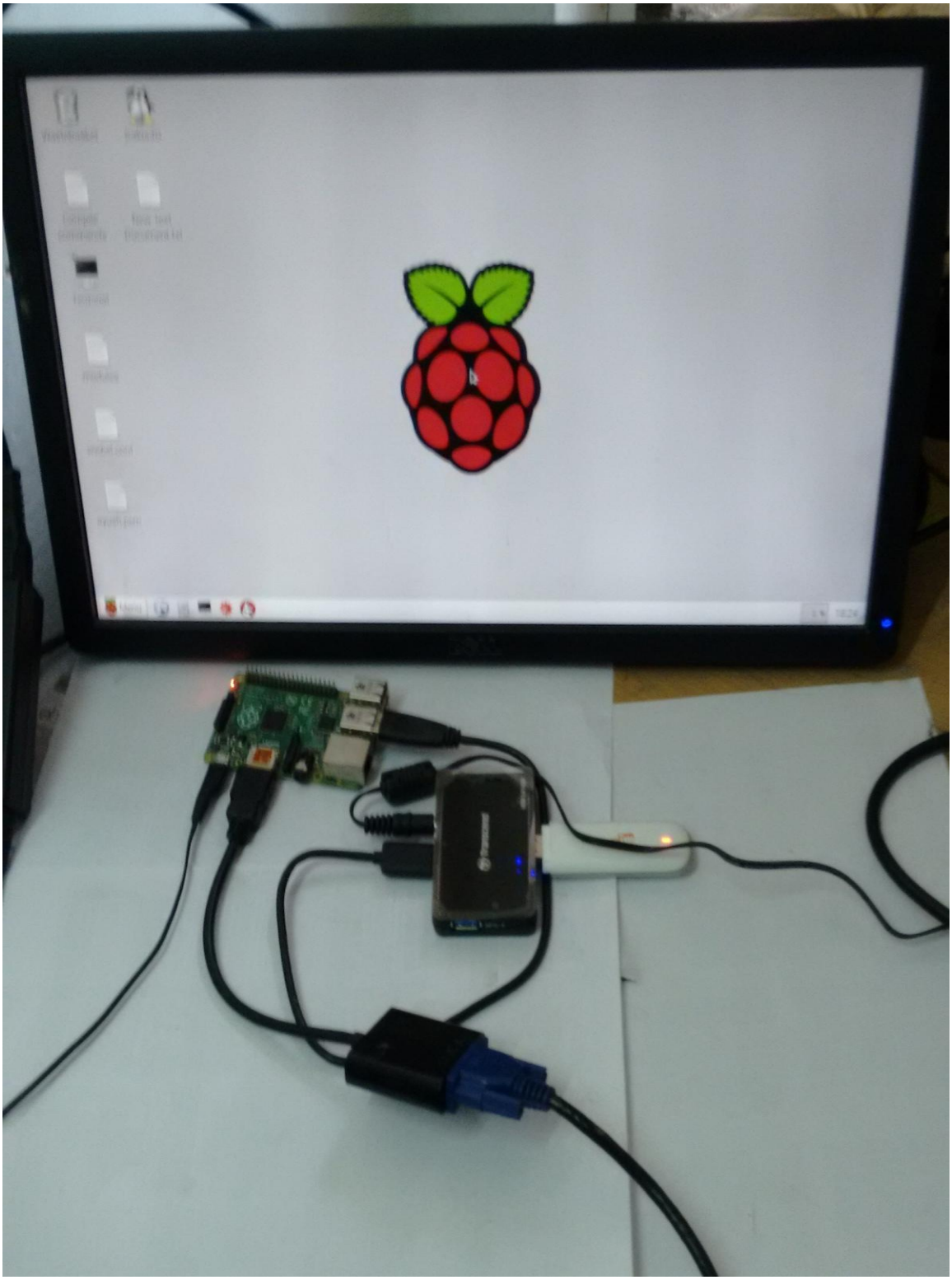

Fig. 30: Hardware required at a node

APPENDIX D
SAMPLE TRAFFIC DATA FILES

Sample traffic data file used at Node N1 ('Node_N1.txt')

Date/Time: 18-05-2015 / 15:20:10
Traffic Code: 00
Traffic Status: "Stopped"

Date/Time: 18-05-2015 / 15:20:30
Traffic Code: 01
Traffic Status: "Slight Traffic"

Date/Time: 18-05-2015 / 15:20:50
Traffic Code: 01
Traffic Status: "Slight Traffic"

Date/Time: 18-05-2015 / 15:21:00
Traffic Code: 10
Traffic Status: "Heavy Traffic"

Date/Time: 18-05-2015 / 15:21:20
Traffic Code: 10
Traffic Status: "Heavy Traffic"

Date/Time: 18-05-2015 / 15:21:40
Traffic Code: 01
Traffic Status: "Slight Traffic"

Date/Time: 18-05-2015 / 15:22:10
Traffic Code: 11
Traffic Status: "Open"

Date/Time: 18-05-2015 / 15:22:30
Traffic Code: 11
Traffic Status: "Open"

Date/Time: 18-05-2015 / 15:22:55
Traffic Code: 01
Traffic Status: "Slight Traffic"

Date/Time: 18-05-2015 / 15:23:10
Traffic Code: 10
Traffic Status: "Heavy Traffic"

Date/Time: 18-05-2015 / 15:23:35
Traffic Code: 10
Traffic Status: "Heavy Traffic"

Sample traffic data file used at Node N2 (‘Node_N2.txt’)

Date/Time: 18-05-2015 / 15:20:10
Traffic Code: 11
Traffic Status: "Open"

Date/Time: 18-05-2015 / 15:20:30
Traffic Code: 01
Traffic Status: "Slight Traffic"

Date/Time: 18-05-2015 / 15:20:50
Traffic Code: 10
Traffic Status: "Heavy Traffic"

Date/Time: 18-05-2015 / 15:21:00
Traffic Code: 00
Traffic Status: "Stopped"

Date/Time: 18-05-2015 / 15:21:20
Traffic Code: 10
Traffic Status: "Heavy Traffic"

Date/Time: 18-05-2015 / 15:21:40
Traffic Code: 00
Traffic Status: "Stopped"

Date/Time: 18-05-2015 / 15:22:10
Traffic Code: 10
Traffic Status: "Heavy Traffic"

Date/Time: 18-05-2015 / 15:22:30
Traffic Code: 11
Traffic Status: "Open"

Date/Time: 18-05-2015 / 15:22:55
Traffic Code: 01
Traffic Status: "Slight Traffic"

Date/Time: 18-05-2015 / 15:23:10
Traffic Code: 01
Traffic Status: "Slight Traffic"

Date/Time: 18-05-2015 / 15:23:35
Traffic Code: 11
Traffic Status: "Open"

APPENDIX E
LIST OF ABBREVIATIONS

- ✓ AWS — Amazon Web Service
- ✓ GMM-EM — Gaussian Mixture Model using the Expectation-Maximization
- ✓ HMM — Hidden Markov Model
- ✓ I2C — Inter-Integrated Circuit
- ✓ OpenCV — Open Source Computer Vision Library
- ✓ OpenSSH — OpenBSD Secure Shell
- ✓ SCL — Serial Clock Line
- ✓ SDA — Serial Data Line
- ✓ SIP — Spatial Interest Points
- ✓ SSH — Secure Shell
- ✓ STIP — Spatio-Temporal Interest Points
- ✓ XML — Extensible Markup Language