Low-Cost IP Camera for Traffic Monitoring

Ali Hasan Rizvi, Adnan Jamil, Muhammad Tayyab Sadiq and Maria Samad
Department of Computer and Information Systems Engineering
NED University of Engineering and Technology
University Road, 75270, Karachi, Pakistan

Shabana Rizvi and Zeeshan Abbas
Department of Physics
University of Karachi
University Road, 75270, Karachi, Pakistan

ABSTRACT
This paper presents the research conducted to provide low cost traffic monitoring solution to the ever increasing traffic congestion problems in Pakistan. The embedded system created is a low-power system that can replace the expensive and complicated systems that add up to the high installation and controlling costs. The feasibility is demonstrated with the ability to determine traffic congestion at real time. Such a system could be used to operate traffic signals for the better flow on roads and to aid law enforcement agencies. The IP camera is constructed using an Arduino board, a camera module and Wi-Fi (802.11) implementation as an embedded system. The video frames are wirelessly streamed (in HEX format) to the cloud-based TCP web-sockets server. The HEX stream is then converted by cloud application to JPEG images, which are displayed to user on web interface, providing live feed for traffic monitoring purposes.

General Terms
Telemetry, Internet of things, Embedded Systems.

Keywords
UART, TCP, WebSockets, 802.11, Camera Module, IaaS, Cloud, real-time.

1. INTRODUCTION
The term IP Camera is used in reference to digital video surveillance systems. It, basically, streams video frames in digital format using LAN or WLAN methods and protocols. A very capable webcam may also offer, more or less the same functionality but it is not considered a discrete independent system as is expected from video surveillance equipment.

Nowadays two kinds of IP Cameras are prevalent in markets. Decentralized IP Cameras have built-in storage and processing capability which makes it suitable in applications which require standalone units. On the other hand centralized IP Cameras require a central system which handles output of camera and stores or processes it in the way necessary. Both these types have their pros and cons. Traffic monitoring demands a large scale implementation of suggested solution even if the area under consideration is a single junction. Also, implementing standalone cameras will increase the cost exponentially. Thus it was decided to implement centralized version of IP Camera and thus its cost was further decreased.

In a mega polis like Karachi, considering the state of roads and transportation infrastructure, a monitoring system will create improvement in the flow of traffic. It may also help departments like traffic police and the city's development authorities to pinpoint junctions which critically need management or development. It has been proven by research that such roadside cameras help keep rowdy citizens in check and encourage safe diving. It may also become an asset for law enforcement to track down criminals or capture pieces of critical evidence in case of occurrence of crime.

IP Camera available offer both, wired and wireless connectivity. In an application like this, wired cameras will prove to be a hassle in setup and maintenance as it is meant to be used in public places. Wireless IP Cameras, on the other hand, will obviously be convenient and their range and positioning will not be a factor as long as they are inside the radius of a router [1].

Wireless versions of IP Cameras, though are a lot more expensive. This solution offers wireless IP Cameras connected to cloud-based application [2] which are not only cheaper than even the wired alternatives, but also offset its implementation cost because of lengthy cables required [3].

2. HARDWARE COMPONENETS
A number of major hardware components were used in the implementation of our version of IP

2.1 Camera Module
This is essentially a CMOS sensor coupled with a small PCB to transfer image. It has a Photo-diode (a P-N junction diode which converts light photons into electrical signal of proportional magnitude) and a CMOS sensor that converts the electrical signal immediately. The processor on PCB receives this signal and in turns transmits it further. This processor can also be instructed to use the camera in a number of different modes.

The most commonly available CMOS camera module offer the image transmission either through UART or I^2C interface. UART is an older and simpler protocol for digital data transmission and is easy to get up and running. Most current microcontrollers (including the one selected) have built-in UART/USART ports. I^2C, on the other hand, is a newer and more advanced protocol which requires more lines of code to set up and is not generally built-in.

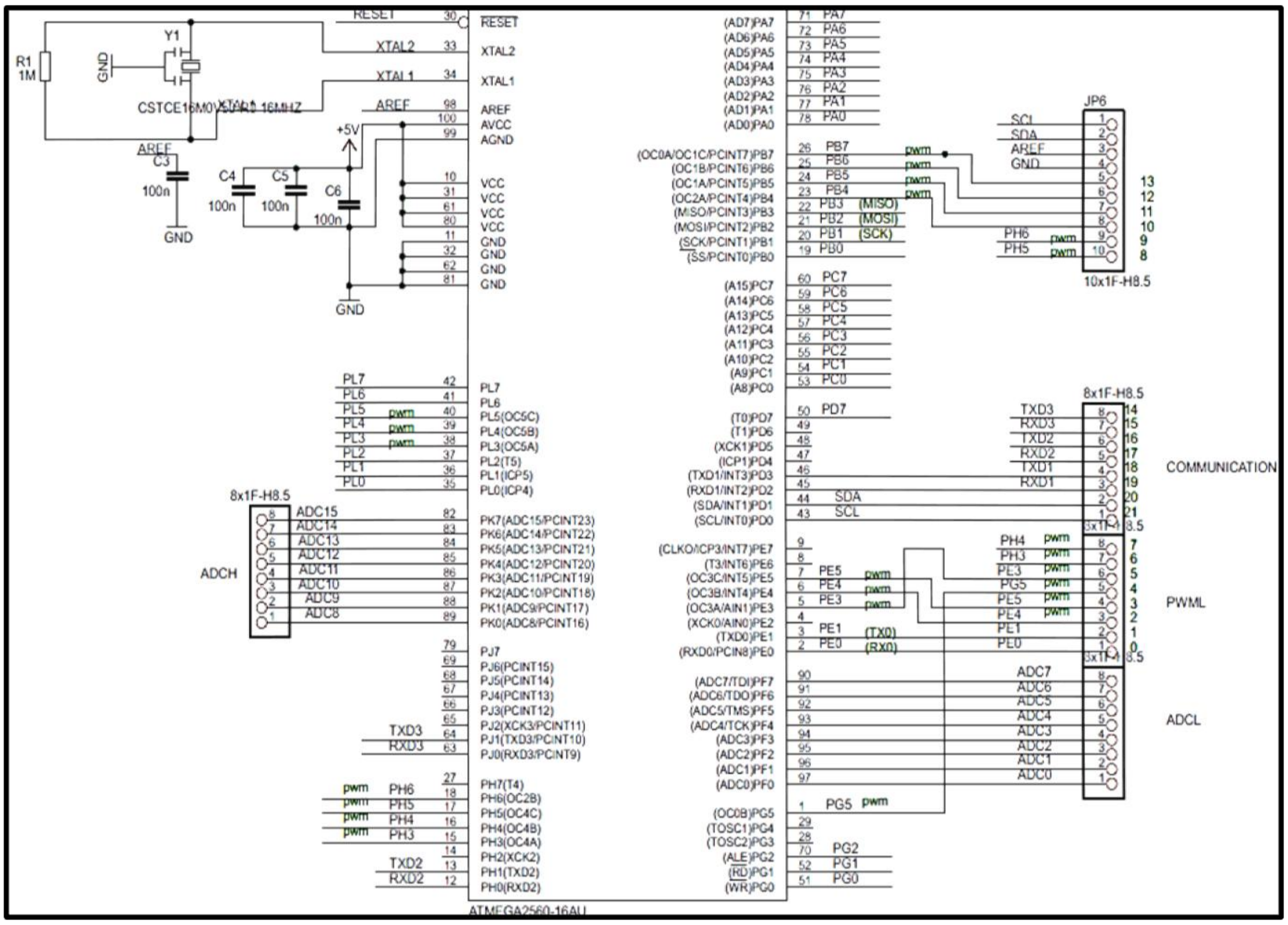

Fig 1: Arduino Mega 2560 pin mapping and pin out

It was a priority to find a UART interface camera, as selected microcontroller has limited memory and it cannot store most of the data streamed. Also, the MCU has 2 built-in USART ports. The LinkSprite UART camera was available in the local market as well as on Pakistan based online stores. As it checks almost all boxes, this particular make and model was selected.

2.2 Microcontroller Board

The microcontroller board acts as the heart of IP camera. It acts as an interface between camera and WI-FI modules. Each of the other two modules were selected after their compatibility was ascertained with the MCU board.

Initially, it was decided that an 8-bit Atmel AVR should be used. These are a series of low cost, single chip microcontrollers which have 8-bit modified Harvard architecture developed by Atmel. Thus a particular model in appropriate packaging had to be selected. After a little more research, it was realized that the data (Video frames) from the camera (in lowest resolution) could not be stored on the memory of the AVR due to small memory size, and relatively low clock rate. Thus it was decided not to store the data in the memory but directly pass it through. Even for that, an 8-bit AVR model with the largest flash (program) memory, highest clock rate, factoring in the market availability, was selected.

Arduino Mega 2560 was selected which fits stated requirements in all aspects. Figure 1 shows ATMEGA 2560 to Arduino pin mapping and pin out of this board.

2.3 Wi-Fi Module

Wi-Fi Module is a single board Wi-Fi solution which transmits data through IEEE 802.11 WLAN. This module was needed to transmit video frames coming from the camera module, through the microcontroller, over WLAN to a remote server. The module accomplishes this task by connecting to a pre-configured Wi-Fi network. The said module is a very powerful device with a processor of its own, manages a TCP/IP stack, real time clock, I/O pins etc.

After surveying the internet, it became apparent that two devices from different manufacturers and compatible with AVR are available, which fulfil these requirements:

1) WiFly Module (RN-XV) 2) XBee Wi-Fi Module

The AVR board will communicate (send video data stream) through UART and the Wi-Fi device needs to send this data to the cloud over UDP/TCP (connectionless or connection oriented). Thus a module with UART RX, and having capability to forward data over UDP and/or TCP, was required. Any of the two mentioned above are capable of both these functionalities.

Both of these module have almost the same configuration, size, features and even cost [4]. The local markets were browsed, and eventually WiFly (RN-XV) was found. So due to availability WiFly was preferred over XBee module.

The WiFly module used in the project is RN-XV 802.11. The module is based on Roving Networks' robust RN-171 Wi-Fi module and incorporates an 802.11 b/g radio, a 32-bit SPARC processor, a TCP/IP stack, a real-time clock, a crypto accelerator, power management unit, and an analog sensor interface. The module is pre-loaded with firmware to simplify integration and minimize applications development time. In the simplest configuration, the hardware only requires four connections (PWR, TX, RX, and GND) to create a wireless data connection. Figure 2 shows WiFly module operation model.

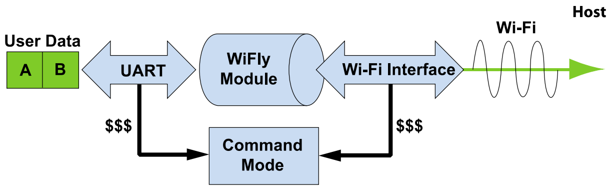

Fig 2: Model-based depiction of the operation of WiFly.

3. IMPLEMENTATION

This work was divided into four distinct stages of implementation, which are detailed below.

3.1 Camera to Arduino Connection

The LinkSprite camera used takes JPEG images and outputs them in HEX format. The camera also requires commands to be sent over UART. The commands are also HEX strings. In order to send the necessary commands to the camera, a library was written which provide functions which send commands to the camera. A routine was also written which, after instructing the camera to start sending the picture, parses through the sent HEX stream and determines where an image starts and where it ends. Thus identifying each image as it's received.

The camera used transmits image as a string of hex digits. A simple utility software (PC) provided by the manufacturer could be used to send necessary commands to the camera. It also receives HEX data stream sent by the camera and compiles it into an image. This utility can only be used if the camera is directly connect to PC. In addition, the utility is not open-source and the manufacturer has not provided any documentation to determine how to convert the hex stream into image

3.2 Arduino to WiFly

The operation of WiFly module was tested by connecting it directly to PC through serial port. The same hardware was used as the camera connection. In addition, another level converter had to be used because WiFly uses logic level of 0 to +3.3 volts. The team was able to connect to the Wi-Fi network WiFly broadcasted with its own SSID, using a range of devices.

After this, different protocols and technologies the WiFly device works with were researched to find the one which suited the requirements best. Two ways were soon discovered to implement the desired functionality. One was to configure WiFly as a TCP/HTTP server and, by using the Arduino to send out data as if sent by a webserver. This approach is easier to implement but would have required a lot of Arduino program memory as the HTTP headers would have to be hard coded and sent along with the actual data. Thus this was not feasible as frame rate was already a cause of concern.

A second option was to configure WiFly as a TCP/HTTP host. This option was almost fully implemented and was thoroughly tested but did not work as was expected. The reason was that packet end symbol encoded in WiFly was different from what is generally accepted by most HTTP/Webservers. Thus when information was sent by WiFly this way, most of the hosts were not able to identify the end of packet and this resulted in error. Additionally, using HTTP and related protocols for real-time communication over the internet is generally frowned upon. This is because HTTP was designed to transmit and deliver webpage which usually are not updated in real-time.

Short latency, hard and soft real-time communication is done with protocols specially designed for this purpose. Examples of these include, VoIP for conferencing, RTP & RTCP for audio, video and simulation. Protocols used over the internet for applications other than these cater to very specific purposes.

Finally, WiFly was configured as a TCP client and a customized webserver was written tailored according to the special requirements. WebRTC is an inter-browser protocol which helps access media files and P2P file sharing without the use of plugins [5]. Server-Sent Events (SSE) is a way for server to initiate data transfer after a client has already made contact. It is used to send distinct though continuous messages, with low latency. Wave federation protocol is and extension on XMPP that is used for inter-server communication. Internet Relay Chat (IRC) is used in real-time internet messaging, file transfer etcetera. Though, it cannot be used to continuously stream data, efficiently [6]. Lastly, WebSockets is a protocol/technology that provides full-duplex, bidirectional data communication over a single TCP connection. The connection need only be opened once and can remain open for as long as data transfer continues. Although it's still not widely accepted and fully implemented, it is compatible with most of the popular web browsers used nowadays. A request from a WebSocket is interpreted as an upgrade request by HTTP servers, which helps in the new technology's implementation using existing internet infrastructure [7].

For the implementation of IP Camera, WebSockets was determined as the most appropriate real-time continuous communication protocol. WiFly device requires at least 1 second to open/close a TCP/UDP connection. Most of the other protocols mentioned require multiple connections to transmit data. Some others communicate in form of separate messages, few hundred bytes long. Both these types did not go well with the hardware specifications and other requirements. The amount of data sent by camera per image is approximately 22 KB. Most other protocols would divide this data and send it over multiple connections, each of which would increase time per frame for IP camera.

WebSockets server is a connect-and-forget deal for the system. Multiple (or all) frames could be sent over a single TCP connection. WebSocket is designed for real-time data transfer thus it not only facilitates continues data streaming but also allows for BLOB transfer of binary or other basic data types.

Procedures were integrated into the already written Arduino code, which initialized WiFly and instructed it to receive image data stream and transmit through Wi-Fi. HEX data was received onto PC this way.

3.3 Testing and Simulation of Webserver

Once WebSockets protocol was chosen the next step was to select which Transport layer protocol is to be used. WebSockets technology support both TCP and UDP protocols. While UDP offers the advantage of lesser overhead, i.e. smaller packet size and negates the time needed to open/close a connection, it also affords the risk of packet loss and unordered reception. Test results showed gain of approximately 3 seconds per frame received, while there was significant pixel loss (about 20 pixels per frame) and about one in hundred frames were lost or arrived late. TCP connected Socket provided jitter free frames and almost no noticeable loss of packets at the cost of decreased frame rate. Thus according test conducted, the advantage of latency could not offset the assurance of complete and timely packet reception.

There were a number of options available when deciding what type of server to configure and which platform to choose for doing so. Research was done about Node.js and its socket.io library for web-socket server [8]. Although it is a lightweight and attractive solution, it is a high level library which, on some instances, is a disadvantage when increased control of socket setup is required. The solution sought was the provision of creating arbitrary low-level sockets. Another reason was its inability to readily interface with already written code [9].

Python and its socket library not only allowed more low level control with its ability to allow the opening of raw TCP/UDP sockets, but also facilitated the creation of a TCP/UDP server and separate classes so that multiple server instances could be formed [10][11].

In IP Camera setup, the WiFly module is configured to act as a client to WebSockets server. Though for testing purposes, it was programmed as a local client which acts exactly as expected of the IP camera hardware. That is, it opens a TCP socket and sends the HEX stream sequentially to the server. This was achieved by writing a python routine which uses the socket library to open a raw TCP socket on the IP address and port of the server. Then it opens a file which contains the output of camera (in HEX), previously taken. Finally it sends the contents to the server through the opened socket.

A TCP WebSockets server was written, again in python using socket library, which waits for requests of clients for connection on a certain port and IP address of the server machine. When requested, server establishes a connection and waits for the client to start sending the HEX stream. Thereafter, it stores the received HEX string in a newly created file while simultaneously receiving newer data being sent. Once it detects that a complete image has been received, (by continuously checking for a newline character) it converts this file into binary and stores it as a JPEG image.

When tested in this manner, it was discovered that the JPEG image files created were unreadable, as the binary file created could not be read as an image. Upon closer inspection, it was discovered that the HEX stream being received contains unexpected white space characters, which when converted to binary made the image file non-JPEG type. This was rectified by first running a function which parses on the HEX stream saved, and removes any white space characters encountered.

Figure 3 shows data (picture frames) flow in the system. Components are shown as blocks.

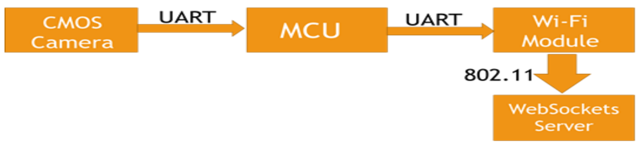

Fig 3: Block Diagram showing flow of data.

3.4 WebSockets Server and Cloud Application

It was an option to select from a variety of different protocols and method. It was thought of implementing a HTTP server and receive the HEX stream appended to a GET or POST request but this was not selected due to the reasons mentioned earlier along with the fact that we wanted to send at least one frame per request so that we could achieve better frame rate. There are some method such as long polling etcetera which are used for such tasks, but these are inefficient and put load on devices. Thus, as previously mentioned, we resorted to using TCP WebSockets server written in Python.

After thorough testing and simulation, a similar server was written (as mentioned above), but in accordance with the requirements of WiFly module. It was made sure that Arduino, and in turn WiFly module will only send a single newline character and that too after end of file characters were detected. The baud rate of both camera to Arduino and Arduino to WiFly module UART connections was set to 38400, so that no buffering was required in intermediate devices. The Arduino code was edited to offer a 1 second delay in sending subsequent images. Then WiFly module was setup to wait for 1 second of non-activity on the UART as a signal to send the packet. This allowed us to receive one image per packet which prevented uncalled for complication and delays in reception. WiFly module was setup with IP address and port number on which the server was running. It was also configured to automatically connect to this server upon start-up.

Figure 5 shows the resulting hardware setup of IP camera components. A solid green LED on WiFly would indicate successful connection to the server. A simple web/cloud application was written, which is meant to be hosted on the same machine/cloud that runs the WebSockets server, which would fetch and display the images created and saved by the server on to a web browser. Many have written involved web app architectures for specific applications in Node.js [12] and other platforms.

A simple Website/Web app was also implemented which was hosted by the same remote server receiving data from WiFly. It may also be ran on any app engine or a second server. This server though will be an HTTP server as the function of this server is communicate with browsers and send them IP Camera images. This server was also written in Python.

While the website/app is hosted on HTTP server, the app itself is written in a number of languages. This method uses languages such as HTML for page layout, JavaScript for logic implementation and image extraction, CSS for page effects [13][14]. It first asks for authentication credentials before providing access. When a user has successfully logged in, they can view the stream of camera images being received. Website also displays processed images [15].

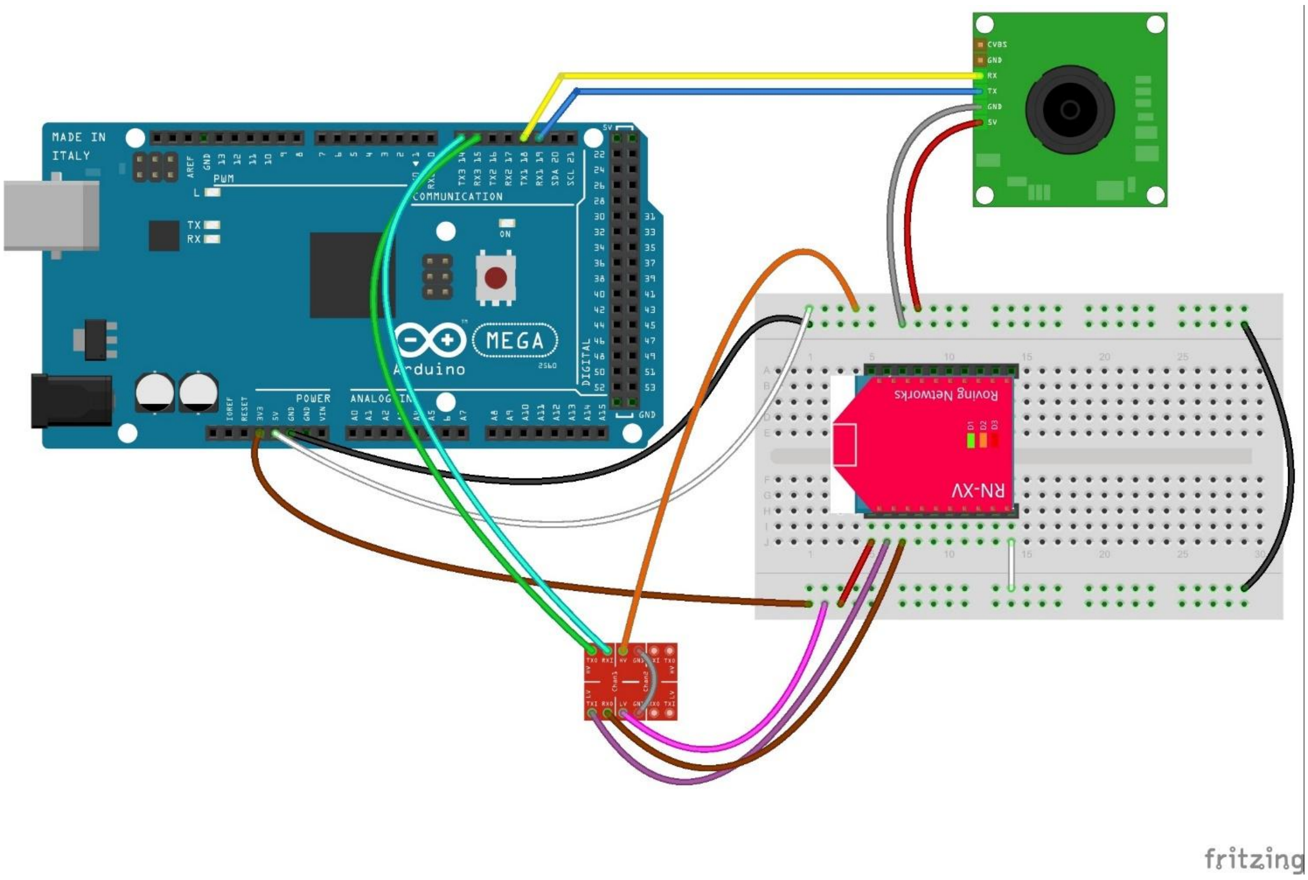

Fig 5: Hardware Setup and Connections of IP Camera

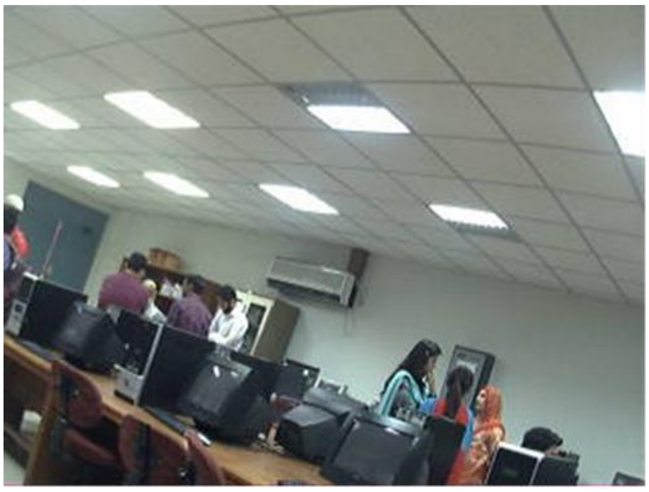

Fig 6: An image frame from IP Camera

4. RESULTS AND CONCLUSIONS

The image displayed in Fig. 6 (above) shows an image taken by IP Camera. It takes about 20 to 30 seconds for an image to be received by the server. Thus the frame rate can be approximated at almost 2 frames per 60 seconds. The camera works standalone and as of now problem of dropped connection it yet to be experienced. Once the TCP connection is made images start to be received. The distance the camera could be installed from the Wi-Fi router is approximately the same as the tolerable range mentioned in router specifications. The image quality is crisp and dimensions are 320x240 pixels. The size of a single image is about 11 to 12 Kilobytes.

A cloud based solution was designed that provides a live feed of the desired location through Wi-Fi in real-time. The WebSockets server can sustain multiple clients simultaneously. This solution could be a tremendous leap for traffic management and monitoring in a developing country like Pakistan. The low-cost and the widespread internet and Wi-Fi availability attracts attention and makes it feasible to set it up in the most remote and far off regions.

5. ACKNOWLEDGMENTS

The authors would like to acknowledge the continued help and support of all faculty and staff at the Department of Computer & Information Systems Engineering, NED University throughout the course of this work.

6. REFERENCES

[1] Arshiya Jabeen, 2013, Security using an IP Camera in Cloud Computing, Munich, GRIN Publishing GmbH, http://www.grin.com/en/e-book/230910/security-using-an-ip-camera-in-cloud-computing

[2] F. Prieta, M. Navarro, J. A. García, R. González, S. Rodríguez in Progress in Artificial Intelligence (2013), Multi-agent System for Controlling a Cloud Computing Environment, ser. Lecture Notes in Computer Science Volume 8154, 2013, pp 13-20

[3] Tiffany Chua, Mark Bachman in Engineering Psychology and Cognitive Ergonomics (2013), Web-Based Architecture for At-Home Health Systems, ser. Lecture Notes in Computer Science Volume 8020, 2013, pp 315-322

[4] M. Saleiro, B. Carmo, J. M. F. Rodrigues, J. M. H. du Buf in Social Robotics (2013), A Low-Cost Classroom-

Oriented Educational Robotics System ser. Lecture Notes in Computer Science Volume 8239, 2013, pp 74-83

[5] WebRTC 1.0: Real-time Communication Between Browsers, http://dev.w3.org/2011/webrtc/editor/webrtc.html

[6] C. W. Dai, S. H. Yang and R. Knott, “Data transfer over the internet for real time applications”, International Journal of Automation and Computing, vol. 3, Iss. 4, pp. 414-424, Oct 2006

[7] WebSockets (2013), http://www.websocket.org/

[8] Node.js (2013), http://nodejs.org/

[9] https://github.com/kanaka/websockify

[10] Python (2013), http://www.python.org/

[11] Autobahn Python, http://autobahn.ws/python

[12] Ioannis K. Chaniotis, Kyriakos-Ioannis D. Kyriakou in Mobile Web Information Systems (2013), Proximity: A Real-Time, Location Aware Social Web Application Built with Node.js and AngularJS, ser. Lecture Notes in Computer Science Volume 8093, 2013, pp 292-295

[13] Marco Casario, Peter Elst, Charles Brown, Nathalie Wormser, Cyril Hanquez in HTML5 Solutions: Essential Techniques for HTML5 Developers (2011)

[14] LaGrone, B.: HTML5 and CSS3 Responsive Web Design Cookbook. Packt Publishing (2013)

[15] J. Rodrigues, P. J. S. Cardoso… in Universal Access in Human-Computer Interactions (2014), A Computer Vision Based Web Application for Tracking Soccer Players, ser. Lecture Notes in Computer Science Volume 8513, 2014, pp 450-462