Real Time Wireless Packet Monitoring with Raspberry Pi Sniffer

Yusuf Turk, Onur Demir, Sezer Gören

Abstract This paper proposes a real time wireless packet monitoring system using a Raspberry Pi. The system is a low cost alternative to commercial packet capture devices and analysis software. In our solution, captured packets from sniffer are sent to main server to gather statistics. Packets are analyzed and only the relevant data are stored in database. A notification server developed in Node.js provides communication between database and user interface developed with Django web framework. The performance of the proposed solution is successfully evaluated in an environment with multiple wireless networks and results are presented.

1 Introduction

While the usage of Wireless Local Area Networks (WLAN) has been tremendously increasing, the cost of wireless networking devices has decreased. This enables to extend network in a more easier way. With the upcoming 802.11ac protocol supporting speeds competing with Gigabit Ethernet and increased usage of mobile devices have boosted the demand of WLANs. Access points are replacing the wired networks in homes and offices.

Considering the improvements in the networking technology, more and more devices are expected to have wireless networking capabilities. During the development of upcoming networked devices their networking capabilities and performance must be tested in an effective manner. Companies developing products with WLAN

Yusuf Turk
Dept. of Computer Eng. Yeditepe University, Istanbul-Turkey e-mail: yturk@cse.yeditepe.edu.tr

Onur Demir
Dept. of Computer Eng. Yeditepe University, Istanbul-Turkey e-mail: odemir@cse.yeditepe.edu.tr

Sezer Gören
Dept. of Computer Eng. Yeditepe University, Istanbul-Turkey e-mail: sgoren@cse.yeditepe.edu.tr

capabilities are in need of stable testing environments. The test environment may contain one or more sniffers to ensure the quality of transmission of 802.11 packets. Packet sniffers are used for comparing the expected network traffic with the captured traffic or analyzing the real time traffic. Because of the physical magnitude of WLANs, multiple packet capturing devices are distributed over the coverage area. Besides testing purposes, sniffers can also be used for security applications to detect intrusions by analyzing the traffic. Aside from that, both wired and wireless network monitoring, bandwidth utilization, and statistics gathering can be done with sniffers. Sniffers are useful because they allow several mechanisms to analyze the traffic thoroughly.

A study by Anh and Shorey [1] reviewed the current network sniffing tools that are used to obtain data by using the network interface card (NIC) of a PC. According to this study, not all the network sniffing tools support monitoring wireless traffic. Jipping and Holland [2] demonstrated that a network sniffer and packet classifier can be developed ina high level language. Another study done by Shum et al. [3] showed that the location of the source can be estimated if the physical location of the access point is known. In this work, a custom software on a router is used to gather network traffic. Location estimation is done by calculating the signal strength. Following study [4] showed that Received Signal Strength Indicator (RSSI) can be given as an input to estimate the location. Shum and Ng [5] showed that there is a correlation between RSSI and inter-device distance. Henderson et al. [6] logged terabytes of data with an access point sniffer. Then the collected data are used to gather statistics about the network usage in a campus. Although this study provides information about network usage patterns, security related problems are not covered. Boughaci et al. [7] used a sniffer module to detect intrusions. But using many agents in the system will slow down the system and timing is very important for intrusion detection. The attacks can also be from an internal source according to Henders et al. [8]. Data are read by a software tool running on NIC and written to a MySQL database.

There are numerous open-source or commercial capture software alternatives in the market. Tcpdump [9] and Wireshark [10] are the most popular open-source tools. Both of them offer detailed filtering options, but lack the visual analysis support. An expensive commercial alternative, OmniPeek [11], offer detailed visual analysis. Packet capture also done using OpenWRT firmware running on the supported hardware [12]. But many access points have lower CPU power and RAM size compared to Raspberry Pi. Access points with similar CPU values are more expensive than the Raspberry Pi. In a study by Polli et al. [13], Raspberry Pi with a USB wireless adapter is used to capture wireless packets. Although it is not used as a monitor mode sniffer, the study showed that the device is capable of capturing packets in a WLAN.

In this paper, we propose a low cost wireless packet sniffer using Raspberry Pi. Our solution is easily customizable and uses inexpensive off the shelf components. In addition, the monitoring software provides real time statistics of the current wireless network traffic and offers simpler user interface for analysis using a lightweight

packet database. It also offers traffic frequency information about nearby access points.

The outline of the paper is as follows: Sec.2 presents the components of the proposed system, Sec. 3 gives the performance results, and finally Sec.4 concludes the paper including the future work.

2 Proposed Solution

The proposed solution consists of three functions: capturing, parsing, and monitoring. Fig. 1 shows the components of the system. Capture operation is handled by a Raspberry Pi [14] with one USB Wireless NIC (WNIC) running a libpcap (packet capture library) [15] application. The output of this application is sent to the main server periodically. In this main server, a parser application extracts statistics and information from the packets. Parser application stores the summary of the packets in a database. For the monitoring part of the solution, a notification server is implemented to support communication between the database and the user interface.

The general overview and the details of the solution will be explained in the following subsections. First, packet capturing and wireless networking modes will be discussed. In Sec. 2.1, the details of Raspberry Pi is given and how the sniffing packets is achieved is explained. Real time parsing of the capture file .pcap using a Python program is described in Sec. 2.2. The section also covers retrieval and storage of the packet information in a database. Sec. 2.3 covers information about how real time monitoring interface is designed and implemented.

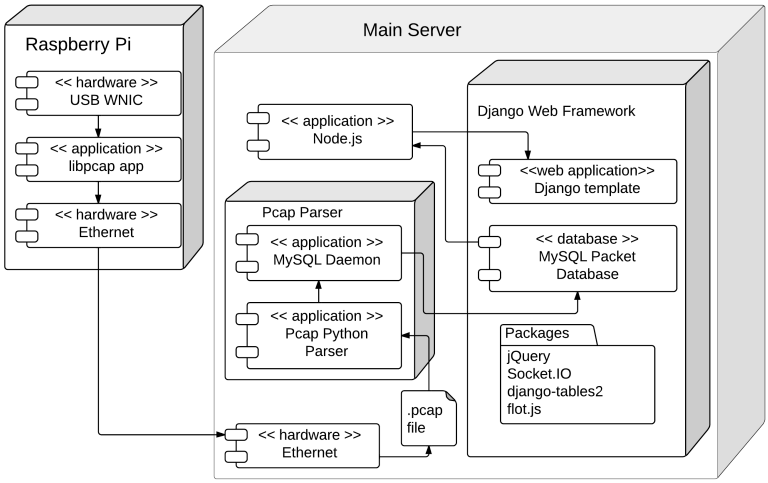

Fig. 1: Component Diagram of the Solution

2.1 Sniffing on Raspberry Pi

Packet capturing is the process of grabbing a copy of a packet off the wired or wireless network before it is processed by the operating system. If the packet capture interface is connected to a network, all frames are unencrypted and can be seen in the packet analyzing software. Otherwise, if the packets are captured in promiscuous or monitor mode, packets are still captured, but they are encrypted.

There are six operating modes of a 802.11 WNIC. These are master, managed, ad-hoc, mesh, repeater, and monitor modes [16]. Monitor mode is similar to promiscuous mode, but it is only applicable for wireless networks. Unlike promiscuous mode, devices do not have to be in a network. Monitor mode allows capturing of all the packets that can be seen by the WNIC. Monitor mode is dependent on the wireless adapter driver, firmware, and the chip-set features. Considering the limitations, not all the adapters support monitor mode. Sniffing of 802.11 packets can be done using a USB WNIC connected to a PC or using an access point. We have demonstrated the usage of both methods in our earlier work [17].

Raspberry Pi (RPi) is a low cost, small sized computer that can run many Linux distributions such as Debian, Fedora, and Arch Linux. With its Broadcom BCM2835 SoC including 700MHz ARM processor and 512MB RAM, RPi is more powerful and cheaper than most of the low cost off-the-shelf access points. The device does not have wireless interface onboard, but a USB WNIC can be easily attached to capture 802.11b\g\n packets. The mode of the WNIC is set to the monitor mode. A libpcap application in C is developed to capture packets. By adding the startup scripts, mode of the interface is set to monitor mode and the device starts to capture packets immediately. This makes the RPi a wireless packet sniffer. The channel that the device is listening on can be specified in the startup script or a random channel can be selected. RPi also supports multiple USB inputs, and a powered USB hub can be connected to the device. Multiple USB WNICs can be connected to this USB hub and each card may listen to another channel. The proposed work uses one USB WNIC connected to a RPi. Captured packets are forwarded to the specified Ethernet port of the device. Main server also listens to the same port and stores the raw data as soon as they are received. Port listening is done by using the netcat utility. Netcat is used to establish TCP and UDP connections and listen to a specified port.

2.2 Parsing Pcap Output

Packets received by the main server are appended to a file with .pcap extension. Pcap parser is developed in Python to parse the file in real time. Python is chosen because it is faster in reading files and easier to develop a parser with its standard library functions such as dictionaries. Main objective of the pcap parser program is to detect headers of each packet. Since all packets have the radiotap header by default, header detection is sufficient to identify the packets. Each time a header is

detected in parser program, the subtype finding state starts and followed by parsing the access point information.

In the initialization state of the program log files, database tables, and variables are initialized. Since the objective is getting the real time monitoring, previous log files are truncated. MySQL database connection and a cursor is created at this state. Capture file with the .pcap extension is read one byte at a time in the reading state. Each header block is sent to the header detection function and if a header is detected, state of the parser changes to header detected state. Detection of a header needs to be completed by finding the subtype of the frame. Subtype finding state can lead to SSID finding state if the frame is beacon, probe request, probe response, association, or reassociation frame. In SSID finding state, MAC addresses are retrieved. Subtype, SSID, and other information are logged in the logging state. Logging state is followed by the reading state. Program stays in the reading state until a header is detected.

Pcap parser writes statistical information directly to the database and information regarding the SSID and MAC addresses to a text file. This text file is simultaneously read by the MySQL daemon Python program. This is done to increase the parsing speed and avoid the large MySQL database queries in the main thread. It is intended to run MySQL queries at the background. MySQL queries can take long time when the data are significantly large. Therefore, only the data such as SSID and MAC address information of the frame are inserted into the database.

2.3 Real Time Monitoring

Node.js is a platform built on Google Chrome's Javascript runtime for building fast network applications [18]. Node.js is an appropriate solution for real-time applications. Since non-blocking models are used, thread does not have to wait for I/O. Although the platform can function as a web server, it can also be used as a server-side application. In this work, a Node.js application is developed to implement long polling functionality from Django web framework templates to MySQL database tables. Long polling is checking a data source for new data and instead of sending an immediate response, server waits until new data are available and then send the response. Long polling is an emulated version of push technology [19].

Node.js server and Django web framework do not have a default communication system. When new data pulled from the database server, it must be sent to the web application immediately to have real time monitoring. Socket.IO is used to provide communication between the two servers. Socket.IO is a Websocket framework that enables communication between WebSockets and real time monitoring. Websocket is a full duplex communication on TCP connection. Socket.IO can be used with Node.js, HTML, JQuery. The Node.js application creates a socket.IO interface on a specific port. Django application also listens from the same port. Node.js applications are event-driven. All listeners start at the same time and poll from the database

tables at every given interval time. Server responds only when a new information is available. Socket.IO allows sending notification messages to one server to another.

Django is a Python web framework uses MVC (model - view - controller) pattern to create database powered web applications. Database tables are defined as models and implemented as Python classes. Web pages such as files with .html extension are called templates. Templates are redirected from the index page by using URL configuration file with regular expression support.

Monitoring of the captured data is supported with a user interface. First component displays the packet counters and current state of the parser. Second component displays the information regarding management frames such as SSID, dBm signal rate, data rate, and MAC addresses. Third component displays the averaged dBm signal of each SSID.

3 Performance and Results

Several test platforms are prepared for testing the system. First, a network with an access point with two Linux based computers run the iperf application to transfer UDP packets. While running the iperf client and server commands, Raspberry Pi sniffer captured the transferred packets without connecting to the network. An another access point, a computer and a mobile phone are also connected to a different network in the same test environment. In the second network, a video stream setup is prepared to provide constant streaming of videos from computer to mobile phone. Sniffer captured packets without connecting to any network. Fig. 2 shows the initial test setup.

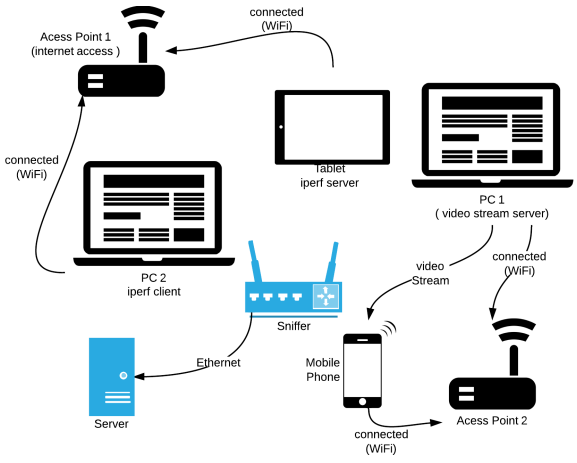

Fig. 2: Test setup with two independent networks

Main objective of the parser application is to generate statistics from the captured packets. Packets are appended to a file and it is read by the parser simultaneously. Program waits until new packets are available and runs in an infinite loop until an interrupt occurs. In this test case, a previously captured file with .pcap extension will be parsed by the program in order to document the speed of reading and parsing. Test system is running Ubuntu 12.04, has two CPU cores at 1.7GHz, and the memory is 1GB. Input .pcap file contains more than two million packets captured over several hours. The intention of this test is to see whether the parser is fast enough to handle the high traffic loads or not. Memory consumption is one of the concerns of real time data monitoring. Applications running constantly have to optimize the memory usage. The final version of the pcap parser consumes low memory and does not store data on the memory. By doing so, memory usage is always stable and memory usage for the system only running the parser rarely passed the thirty percent. According to the log, 500,000 packets are parsed in execution time of 1481.1 seconds that is approximately 24.7 minutes. This means that parser application can provide real time output for the traffic load of one million packets per hour without any delay. While analyzing the capture files, we have observed that among the 22 million packets, 80% of the packets had size varied between 20 and 320 bytes.

4 Conclusion and Future Work

Our solution offers a low cost, off the shelf, and customizable alternative to both commercial and open source packet capture systems. User interface provides statistics such as percentages of frame types and subtypes, and the number of frames transmitted by each access point. Since we do not store entire packet contents, our database only contains the practical information about the frames. Our work is expected to be beneficial for academic projects and startups.

For future work, sniffer packet sending performance can be improved. Also, components such as pcap parser and Node.js server can be modified to run on RPi. Another future work plan is to trim the packets in RPi and send the relevant information to the main server. By doing so, number of packets that can be monitored in real time will significantly increase.

References

1. Anh, N.T.; Shorey, R., "Network sniffing tools for WLANs: merits and limitations," Personal Wireless Communications, 2005. ICPWC 2005. 2005 IEEE International Conference on, 23-25 Jan. 2005 doi: 10.1109/ICPWC.2005.1431372
2. Michael J. Jipping, Andrew Kalafut, Nathan Kooistra, and Kathleen Ludewig. 2004. Investigating wired and wireless networks using a java-based programmable sniffer. SIGCSE Bull. 36, 3 (June 2004), 12-16. http://doi.acm.org/10.1145/1026487.1008003

3. Shum, K.C.Y.; Quan Jia Cheng; Ng, J.K.Y.; Ng, D., "A Signal Strength Based Location Estimation Algorithm within a Wireless Network," Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on, 22-25 March 2011 doi: 10.1109/AINA.2011.80
4. Shum, K.C.Y.; Ng, J.K.-Y.; Quan Jia Cheng, "The Design and Implementation of a Wireless Location Estimation System in a Wireless Local Area Network," Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on, 26-29 March 2012 doi: 10.1109/WAINA.2012.169
5. Shum, K.C.; Ng, J.K., "Detecting, Locating, and Tracking Hacker Activities within a WLAN Network," Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010 IEEE 16th International Conference on, 23-25 Aug. 2010 doi: 10.1109/RTCSA.2010.46
6. Tristan Henderson, David Kotz, and Ilya Abyzov. 2004. The changing usage of a mature campus-wide wireless network. In Proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom '04). ACM, New York, NY, USA, 187-201.http://doi.acm.org/10.1145/1023720.1023739
7. Dalila Boughaci, Kamel Ider, and Sofiane Yahiaoui. 2007. Design and implementation of a misused intrusion detection system using autonomous and mobile agents. In Proceedings of the 2007 Euro American conference on Telematics and information systems (EATIS '07). ACM, New York, NY, USA, , Article 12 , 8 pages. http://doi.acm.org/10.1145/1352694.1352707
8. Rich Henders and Bill Opdyke. 2005. Detecting intruders on a campus network: might the threat be coming from within?. In Proceedings of the 33rd annual ACM SIGUCCS fall conference(SIGUCCS '05). ACM, New York, NY, USA, 113-117. http://doi.acm.org/10.1145/1099435.1099461
9. http://www.tcpdump.org/ retrieved 15/12/2013
10. http://www.wireshark.org/ retrieved 15/12/2013
11. http://www.wildpackets.com/products/ retrieved 15/12/2013
12. http://wiki.openwrt.org/toh/start retrieved 04/06/2014
13. Anna Maria Polli, Matthias Korn, and Clemens Nylandsted Klokmose. 2013. Local area artworks: collaborative art interpretation on-site. In Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication (UbiComp '13 Adjunct). ACM, New York, NY, USA, 79-82. http://doi.acm.org/10.1145/2494091.2494114
14. http://www.raspberrypi.org/ retrieved 22/04/2014
15. http://www.tcpdump.org/pcap.html retrieved 22/04/2014
16. http://wireless.kernel.org/en/users/Documentation/modes retrieved 22/04/2014
17. Turk, Y. (2013). Wireless Sniffer System. Senior Project Report. Yeditepe University: Istanbul
18. http://nodejs.org/ retrieved 04/01/2014
19. http://en.wikipedia.org/wiki/Push_technology retrieved 22/04/2014