

ReTiMon – A Real Time Network Monitor

Sudipto Das, Vinod Kone
University of California, Santa Barbara
sudipto, vinod@cs.ucsb.edu

ABSTRACT

*Wireless Networks are characterized by frequent failures, and analyzing the performance and diagnosing problems in these networks is a challenging task. To improve network performance and user experience, network administrators should be able to diagnose and troubleshoot the faults in real time. In this paper, we present a tool, **ReTiMon**, that does real time network monitoring and is aimed to help the network administrators analyze the network's performance. **ReTiMon** provides statistics for the entire network in general – and each node in specific – and helps find the “pain points” in the network and ascertain various causes for such dismal performance of the network. We explain in detail the architecture and various features of this tool and the tests performed to validate the correctness of the tool.*

1. INTRODUCTION

Wireless Networks especially 802.11 based wireless LANs [1] have become overwhelmingly popular over the last decade or so [17]. Variants of such networks in popular use include Infrastructure based or Access Point (AP) based networks, Mobile Ad hoc Networks (MANETs) [21], and Wireless Mesh Networks (WMNs) [8,9,13–15,22–26,28]. But the unreliable nature of the wireless medium has somewhat dampened the popularity and growth of such networks. Studies show that users generally complain about various problems related to wireless networks [2]. The diagnosis and identification of faults in these networks is essential before we can take steps to correct these discrepancies. For efficient network management and reliable network performance, not only do we need to be able to analyze the behavior, but also be able to perform this analysis in “real-time”.

Most of the existing work, that analyzes the performance of these networks, does “offline” analysis of the data gathered by sniffers deployed in the network. But it is of very little help to the network administrators / users when fault diagnosis and troubleshooting the network is of primary importance. As a result, people have felt the need for having analysis tools that would help monitor as well as analyze the networks in “real-time”, diagnose the faults, and pin-point the faulty locations. The earlier the faults are detected and corrected, the better is the user experience.

In this paper, we describe a tool that does “real-time” monitoring of a wireless network. We call this tool *ReTiMon* and it generates “real-time” graphs of vital network statistics to aid the user / administrator of the network analyze the network as well as diagnose faults, if any. This is critical for the efficient deployment and reliable performance of the network. As the networks grow in size, the need for such tools become critically important, since manually troubleshooting such huge networks and pin-pointing the “pain points” becomes an intractable problem. Our goal is to design a system

that would have a sniffer (Section 4) that passively monitors the system, sniffs the traffic around it, and then calculates some network wide as well as node-specific statistics that can be displayed graphically by a GUI. The statistics calculated include Signal Strength, Utilization, Data and Control Packets Sent/Received, Throughput, and Goodput. The GUI also generates a layout of the nodes seen by the sniffer (Section 5).

The remainder of this paper is organized as follows: Immediately after this section is Section 2 where we describe work done so far in monitoring the wireless networks and analyzing their performance thereby re-asserting the requirement for such a tool. This is followed by Section 3 which describes the architecture of the tool and provides a very high level view of the different important parts of the tool. We then move on to Section 4 where we describe in detail the functioning of the Data Gathering System, the part of the system that obtains the data to be visualized and Section 5 where we describe the GUI which provides a graphical representation of the network as well as the various metrics calculated. Section 6 provides a general description of how we tested the tool for correctness, while Section 7 concludes this paper.

2. RELATED WORK

There has been considerable research in the area of wireless network monitoring [7, 11, 17, 20] in the past few years. Network monitoring can be broadly classified into Active monitoring and Passive monitoring. In Active monitoring, APs and/or clients are instrumented to collect the necessary information. But, with the abundance of enterprise and home wireless LANs, instrumentation technique is becoming increasingly prohibitive. The other monitoring technique is called Passive monitoring, where one or more sniffers placed at certain location(s) in the network capture the traffic that they can hear. This method has the advantage of not needing to change the software running at clients or APs and hence easily deployable. Coupled with additional mechanisms like inference [11, 17], merging [20] etc. Passive monitoring is known to be quite powerful in analyzing the characteristics of a wireless network.

On a complementary plane, network monitoring can be classified into Offline analysis and Real-time analysis. The former involves collecting the traffic trace of a wireless LAN in real networks like enterprise [2, 7], conference [17, 20] or from a test-bed implementation [11] and then analyzing the network semantics offline. Though this method provides good insight into the network characteristics, the delay for trouble-shooting often undermines its effectiveness. For example, network administrators of a conference need to identify the pain points in the wireless LAN in real-time in order to be able to rectify the problem and aid in the smooth running of the proceedings. In such a case, doing an of-

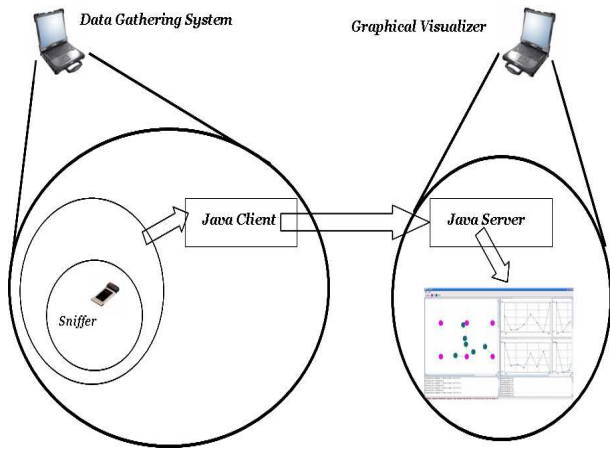


Figure 1: Architecture of the Real Time Network Monitor. Consists of two main parts, the Data Gathering System and the Graphical Visualizer

fine analysis wouldn't help the conference attendees in a meaningful way. On the other hand, Real-time analysis (and monitoring) helps in significantly reducing the troubleshooting delays where time is critical [10]. Though there are some tools [3–6] available for providing real-time network statistics, they are either for commercial use or provide insufficient information to the users regarding the local network.

The primary motivation of our work is to provide an open source real-time monitoring tool to the general users and network administrators, which

1. is easy to use and understand, and
2. provide enough MAC layer information for understanding the network behavior

The first goal is achieved by providing an intuitive java based GUI, with a network layout complemented with graphs for various metrics. The second goal is achieved with the help of using open source tools like Madwifi [19] and tshark [27] supported by perl scripts for parsing. The details of our system are explained in the later sections.

3. ARCHITECTURE

In this paper, we describe a tool, called **ReTiMon**, that is designed to perform network monitoring in “real time”. This section explains the high-level architecture of **ReTiMon**. Figure 1 provides a graphical view of framework of **ReTiMon**. This tool consists of two major parts:

Data Gathering System The main function of this system is to passively monitor the network and dump all the packets that it can hear in its neighborhood. The sniffer – which actually sniffs the packets and is explained in further detail in Section 4 – is coupled with scripts that convert the dump into a format that can be read easily by the client. The packets seen during the last time period – one second in our case – is analyzed and various network metrics are calculated and written in a file which the Java based client can read. The client then sends the data to the Server which is part of the Graphical Visualizer.

Graphical Visualizer This part consists of a Client-Server based GUI built using Java and JFC Swing. **ReTiMon** is designed in a way that the client works in conjunction with the sniffer / Data Gathering system and reports the sniffed packets to the server. The server in turn works with the Graphical Visualizer and feeds information to it. The Graphical Visualizer processes the data and populates its internal data structures that store the node-specific as well as network wide information. It then displays a variety of graphs and a layout of the network showing APs and the clients associated with the Access Points (APs). The features of the Visualizer is described in further detail in Section 5.

This client server based design is somewhat elegant as it decouples the two sub-systems, viz. the Graphical Visualizer and the Data Gathering System, and lends a cleaner design to the system. The design does not mandate the visualizer and the sniffer to be co-located. This freedom lends greater flexibility to the system.

4. DATA GATHERING SYSTEM

This is the back-end component of our system and resides on the client side of our architecture. It is responsible for collecting, analyzing and communicating formatted data to the server. We describe the details of each function below:

Collection For collecting the data, we use an open source tool called `tshark` (formerly `ethereal`). The client machine runs Desktop version of Ubuntu 7.04. Atheros based wireless cards were used to sniff the air traffic. We set the cards in Monitor mode, with the help of Madwifi driver, to capture all types of traffic including management and control frames. `tshark` dumps the packet data in a `pdml` format which is fed into the analyzing unit in real-time.

Analysis A set of perl scripts parse the `tshark` output and store the data in a hash table. In particular, the hash table is indexed by the Mac address of the nodes (clients and APs) in the network. Each entry of the hash table stores information such as average signal strength, total number of different types of frames sent and received by this node, uplink and downlink throughput etc. In addition, the associated client nodes have the corresponding AP's Mac address in their hash table entries. Similarly, the AP's have the total number of associated users. For identifying associations, we only consider an association between a client and AP if we can hear control or data frame exchanges. For estimating the goodput of a node or network as a whole, we leverage the retransmission bit in the frames. Specifically, all the frames with retransmission bit set are excluded from the goodput calculation though they are included in the throughput calculation.

Two main features of our system are estimating channel utilization and inference of lost packets. For calculating channel utilization, the total time is divided into one second intervals and the air-time taken by the packets seen in the last one second is estimated. We use the values reported in [18]. The percentage of total air time of packets in the last one second gives an estimate of the channel utilization. The other important

feature is the inference of lost packets. Note that, sniffers have limited capabilities and it is not possible for them to capture all the packets in the air due to finite processing power and their location. We leverage the atomicity of 802.11 protocol to infer lost DATA, RTS and CTS frames. For example, if an ACK frame is captured and there is no corresponding DATA frame in our trace then we infer that the DATA frame was indeed transmitted but not captured by the sniffer. Analogous are the cases for RTS-CTS atomicity (to infer lost RTS frames) and RTS-CTS-DATA atomicity (to infer lost CTS frames). We inflate our metrics (throughput, utilization etc) accordingly to take this into account. The output (hash tables) of the perl scripts are fed into the communication component to send it to the server.

Communication The last component of the data gathering system involves communicating the perl output to the java server running on a different machine. A java program runs on the client machine which takes the output of the perl scripts and encapsulates them in an object format that can be understood by the java server. To be more precise, the perl scripts generate the hash tables for the last one second and write them to a plain text file. The java client then reads this file and communicates the data to the server every second. Synchronization is achieved so that the plain text file doesn't get overwritten by the perl scripts before the java client reads and communicates it to the server.

5. GRAPHICAL VISUALIZER

The Graphical Visualizer sub-system obtains the data from the Data Gathering sub-system and converts it to a form that can be displayed graphically. The Visualizing System is built entirely in Java, JFC Swing and consists of two major sub-parts: the Server thread and the Graphical User Interface. We explain these sub-parts in the following two sub-sections.

5.1 Server Thread

The Server listens for connections from the client that is part of the Data Gathering System. The client periodically connects to the server and sends the statistics of the packets that were collected in the last time period. The Server operates as a separate thread and uses blocking I/O, i.e., the server will block when it is processing requests from the client. The server thread is spawned when the Visualizer system starts. Once the server receives a connection from the client, it obtains the statistics, which is transferred as a Java object, and calls a method in the visualizer that will process the object and update the display accordingly.

5.2 ReTiMon GUI

The main function of the GUI is to display the data obtained in a way which the user can use to analyze the network and diagnose faults, if any. As soon as a connection is received by the server, and it receives the object that contains all the statistics, it passes the object to the GUI that converts the data to an internal representation and updates the display. The GUI maintains and displays network wide statistics, node specific statistics, draws a layout of the nodes seen in the network, and also performs some diagnostics of the network. Figure 2 provides a screenshot of the GUI. The canvas on the left displays a layout of the nodes that are sensed in

the neighborhood, the text area at the bottom left corner displays some of the status messages, the panel in the top right half displays the graphs corresponding to the network wide statistics, while the list at the bottom right corner of the screen provides a mapping between integer node IDs and their MAC addresses. The graphs displayed are Signal Strength in dBm, Number of Data Packets in the network, Number of Control Packets, Number of Management Packets, Throughput in Kbps, Goodput in Kbps, Percentage Network Utilization, Percentage of Packets missed by the sniffer, and Number of Clients per AP. The GUI is refreshed only on receipt of a new statistics object at the server. We would now describe some of the most important parts of this sub-system.

5.2.1 Displaying the Graphs

The graphs are one of the most important parts of the GUI. Most of the information is displayed either as line or point graphs (`GraphCanvas` object) while some information is displayed as bar graphs (`BarGraphCanvas` objects). These Graphs are built using the Model view Container (MVC) architecture of JFC Swing. Each graph is a Java object that contains a Model – a java object that stores the points to be displayed (a `Graph` object) – and a logic to display the contents of the Model. This logic is known as the view logic, and is responsible for rendering the graphs in the Container. So whether it is the graph showing the signal strength values or the utilization or the throughput, it boils down to a set of $\langle x, y \rangle$ values. All the line / point graphs are in time scale, so the x value is the time at which this value was received and the y value corresponds to the metric that is being plotted. For example, if we are plotting the utilization, the the y value represents the percentage utilization that is seen at the time corresponding to the x value. A feature of the graph is that it modifies its scales according to the points that are currently being displayed, i.e., it calculates the minimum and maximum values, and sets the lower and upper bounds of the graph according to that. The user can choose between line and points according to his / her convenience.

The user can also change the time duration for which the data points are being displayed. To implement this feature, the data model performs some kind of aggregation on the data values that are being recorded. By default, the graphs display points for the last minute. All the points that are passed to the model are in granularity of seconds. But when the user asks for the graph over an extended interval such as an hour or a day, they generally want to have look at the trend of the values over this extended duration. This has prompted us to perform some aggregation on the data in order to save space, which becomes critical owing to the huge amount of information we are storing, when it comes to a network of reasonable size, say 40-50 nodes. The model will store in the granularity of seconds only for those points that are less than 3 minutes old compared to the last point that has been seen. Beyond that, the points are aggregated to granularity of a minute. All the points that are worth a minute's data are taken and averaged and the average value is stored. This is again done for points for the last 3 hours and beyond that they are similarly aggregated to hourly points. As a result, when the user has chosen a display interval of 3 minutes or less, the each of the point received from the data gathering system is being displayed. Any interval beyond that shows averaged data and is representative of the trend.

The Bar Graphs are used to display histograms and does not

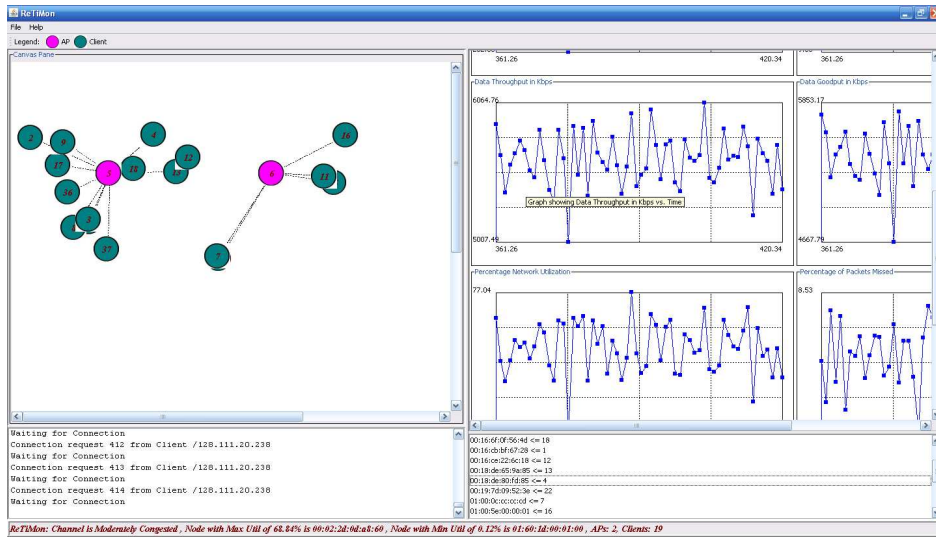


Figure 2: A Screenshot of the GUI of ReTiMon

display time-scale data. Again, the bar graph also calculates the maximum y value being displayed and scales the other values according to the maximum value displayed. The width of the bars depend on the number of bars being displayed. The number of bars to be displayed can be set by the user, but there is a limit of the number of bars that can be displayed. This limit is placed for maintaining the sanity of the display, allowing too many graphs would make the graph very hard to comprehend.

These graphs, both the line / point graphs and the bar graphs, are initially displayed with some default preset size, but if the user wants to have a larger display of a specific graph, double-clicking on a graph opens it in a new window. Both the Line and the Bar graphs also allow users to change the rendering colors as per their convenience. The user can also select between line and point graphs.

5.2.2 Laying out the Nodes

In addition to displaying the graphs corresponding to the various metrics, the GUI also generates a layout of the nodes that has been sniffed in the neighborhood of the sniffer. The Clients and the APs are given different color codes. The APs are placed in a grid layout, where the canvas is divided into grids of some fixed size, and the APs are placed at the center of this grid. The nodes that are associated to an AP, are placed around it in its grid and the association is displayed by a broken line between the client and the AP to which it is associated. The location of the nodes within the grid is random. So there might be instances when the nodes overlap. We do not implement a logic that places the nodes in a non-overlapping manner and we rely on the user to drag and place them to a new location, when needed. The nodes that are not associated to any AP are placed randomly in the canvas.

This layout is very dynamic in the sense that the placement of the nodes change if their association changes at any instant of time. The canvas displays only those nodes that have been heard atleast once in the last 60 seconds. If a node wasn't heard for the last 60 seconds, then it would be removed from the canvas. The nodes are assigned a unique integer ID which is displayed over the node when they are placed in the canvas.

There exists a one-to-one mapping between the node IDs and their MAC addresses that is displayed in the list seen at the bottom right corner of the main window in Figure 2.

5.2.3 Node Specific Information

The main window of the GUI displays the graphs for the entire network. But the system maintains individual node specific information as well, that is displayed *on-demand*. Each node is indexed by its MAC address. The statistics object received from the client through the server contains a Hashtable of the individual information that is also indexed by the MAC addresses of the nodes. This information in the hashtable is them converted to a form that can be stored and displayed and is stored in the internal structure indexed by the MAC address. The building block for these structures is a Graph object that also acts as the model for the GraphCanvas and BarGraphCanvas objects (Section 5.2.1). A mapping between the MAC addresses and the node IDs is provided as a list at the bottom right corner. Double-clicking on a node laid out in the canvas, or an item in the list mentioned, opens a new window that displays the graphs of the node specific metrics of the corresponding node. The window contains graphs of Signal Strength in dBm, Percentage Network Utilization, Data Packets Sent, Data Packets Received, Receive Data Throughput in Kbps, Send Data Throughput in Kbps, Send Data Goodput in Kbps, Receive Data Goodput in Kbps, Control Packets Received, and Management Packets Received.

The node specific information is saved only for those nodes that have been heard atleast once in the last 180 seconds. If a node was not heard in this period of time, then the information corresponding to this node is purged. It may be recalled that a node is removed from the canvas is it was not heard for the last 60 seconds (Section 5.2.2). Hence, if a node is not heard for a minute, it will be removed from the canvas, but its information is still being maintained. So, if the node re-appears within the remaining 2 minutes, it will be placed back in the canvas, and all its information will remain intact. If a node re-appears after 3 minutes, then it will be treated as a newly seen node. The intuition behind this logic of having

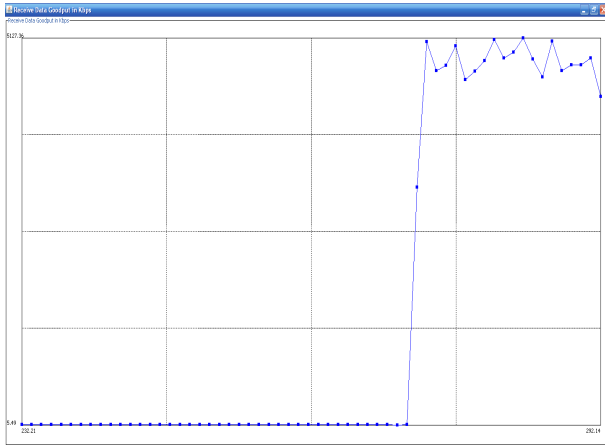


Figure 3: Received Goodput of the node acting as the Server. An iperf server receives data from the client

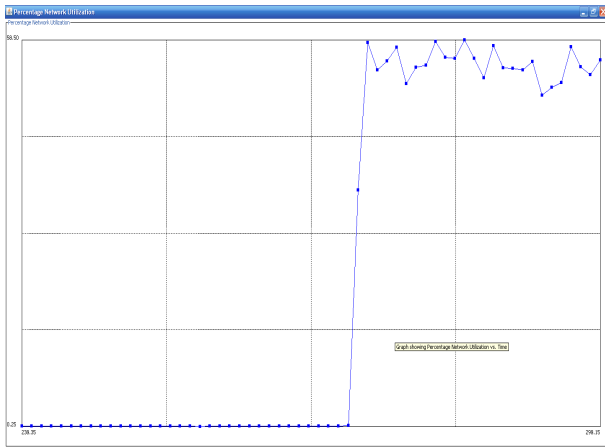


Figure 4: Network Utilization of the node acting as the iperf server

two different intervals is that we don't want to lose the node-specific information too early as well as we don't want to keep the idle nodes too long in the canvas. This prevents cluttering of nodes in the canvas.

5.2.4 Some Diagnostics

The Visualizer also performs some diagnostics on the network. These diagnostics are displayed as the status bar of the main window. Based on the present utilization of the network, it determines whether the network is Uncongested, Moderately Congested, or Congested. The threshold values used for this classification are taken from the paper by Jardosh et. al. [17]. It also determines the node with the maximum and minimum utilization. The utilization value is from an Exponentially Weighted Moving Average (EWMA) calculated within the `Graph` objects with 0.5 used as the weight factor. Also displayed is the number of clients and APs that are visible in the canvas.

6. DISCUSSION

In this paper we describe a tool, called *ReTiMon*, that monitors the network in real time. Having discussed about the features of this tool, we spend some time in this section

to prove the correctness and credibility of the tool, i.e., the view of the network provided by the *ReTiMon* is correct and the users can rely on it to analyze the network and diagnose fault, if any. Throughout this paper, we have expressed the system as two different parts. So we will also divide the testing of the system into testing the two parts separately and then testing them together.

6.1 Testing the Data Gathering System

We divide the Data Gathering System into two sub-systems as described in the previous sub-section and hence we define the testing specifications of the two sub-systems separately.

6.1.1 Testing the Sniffer

Any monitoring system should be validated to determine the extent to which it is sniffing the packets so that there is an estimate of the error, if any, introduced by the data missed. To be more specific, there should be an estimate of how the missed packets are affecting the inferences drawn by the system. We would like to know whether our inference mechanism is under-estimating or over-estimating the traffic in the network. To test this, we introduce a custom generated UDP traffic in the network and estimate the packets sent/received, throughput and goodput of the node sending/receiving traffic. We measure this with what the expected values to validate the correctness of the sniffer. We also infer the number of packets missed by the sniffer to have an idea of how well the sniffer is performing. The logic used for the inference is very similar to that explained in [17].

6.1.2 Testing the Parser

The parser consists of scripts that calculate the per node as well as network-wide statistics. To validate the correctness of these scripts we need to fall back on a static data-set (like [12]), for which the values to be calculated are known in advance. A close alignment of calculated metrics with the ones reported by authors in [17] helped us determine the correctness of these scripts.

6.2 Testing the Visualizer

Testing the Visualizer comprises of testing each of the individual features discussed Section 5. This can be achieved by having test cases that are prepared by hand and test all the mentioned features. To have a manageable network, we have simulated data from a small network of about 30-40 nodes and provide statically generated data. This will help validate the visualizer as we know in advance the data we are providing, and from the visualization produced we can categorically state whether the visualizer is displaying information as is given in the specification. To be more specific, we created a scenario where we know which nodes are APs and which clients are associated to which APs, and we also know the performance of the each of the individual nodes. Once we are sure that the visualizer correctly displays the static data, to evaluate the performance of the tool in a real scenario, we have tested it on a real data set [12] where we will emulate the data as being a stream, similar to what we will have in a real network scenario. The authors in [17] analyze this data set and provide graphs on different metrics. A similarity in trends of the two sets of graphs (the ones in the paper and the ones generated by the visualizer) would help establish the correctness of the tool for a real data set. Once the tool is determined to be correct on these static data sets, it

can be used to monitor the real-time networks in conjunction with the sniffer.

6.3 Testing the Entire System

Having tested the individual parts of the system, we now test the entire system so that we can validate the the individually correct parts glue together in a way that the entire system is correct. We do this in a way very similar to that explained in Section 6.1.1. We introduce custom traffic into the network using IPerf [16]. We use the IPerf client to send UDP traffic to an IPerf server. In IPerf, the client generates traffic at the specified rate and sends it to the server. Both the client and the server display statistics of the data transferred in the last second. In our experiment, we have the client in the wired network, and the server is one of the nodes in the network. We monitor the node-specific graphs of the node on which the IPerf server is running. We set the rate of sending to 5Mbps, and the Goodput and Utilization of the server, as displayed by ReTiMon can be seen in Figure 3, and Figure 4. The sudden increase in goodput and utilization corresponds to the point when the Iperf client started sending traffic into the network. The Goodput reported is around 5Mbps which is infact very close to that reported by the IPerf server. The difference is on account that IPerf report the application layer goodput while *ReTiMon* report the MAC layer throughput, and the packet sizes in the two layers are different. It is also to be noted that as the server in IPerf acts as the receiver, we are displaying the Received Goodput of that node. This proves the correctness of the entire system.

6.4 Additional Notes

These are some of the observations we had while testing the system for correctness.

- The GUI does a lot of calculation and is very CPU Intensive. In order that we have a smooth display of the graphs, we need to run the GUI on a CPU with good computing resources.
- The sniffer must not have any other CPU intensive process running while it is sniffing the network. The presence of a CPU Intensive process results in the loss of a large number of packets. We tried running IPerf from the same laptop where the sniffer was running, and this resulted in a high percentage of packet losses, which was as high as 30% whereas the normal trend being 5 – 10%
- Separating the Data Gathering System and the Graphical Visualizer using the client/server architecture was an initial design decision taken to lend a clean interface to the system, but later on, this decision became a requirement considering the two things stated above.

7. CONCLUSION

In this paper, we describe a tool, called *ReTiMon*, that performs network monitoring in *real-time*. This tool consists of two major parts, the *Data Gathering System* and the *Graphical Visualizer*. The *Data Gathering System* passively monitors the network, gathers the packet headers and performs calculation of the various metrics, which is then communicated to the visualizer. The *Graphical Visualizer* converts this data to a form that can be displayed graphically. It then generates graphs on various network wide metrics as well as graphs

for the individual nodes in this network. This paper explains in detail these two parts of the tool as well as various other features of the tool. This tool is designed to help the network administrators to help diagnose the network faults in real time so that they can troubleshoot them and improve the network performance as well as user experience. This paper also describes the tests that were used to validate the correctness of the different parts of the tool as well as the entire tool itself.

Acknowledgment

We would like to thank MOMENT Lab at the Dept. of Computer Science, UCSB for providing us with the equipments (wireless cards and laptops) to carry out our work, as well as for their feedback that was very useful during the course of the work.

REFERENCES

- [1] IEEE Standard 802.11, 1999.
- [2] ADYA, A., BAHL, P., CHANDRA, R., AND L. QIU. Architecture and Techniques for Diagonising faults in IEEE 802.11 Infrastructure Networks. In *Proc. of MobiCom* (Philadelphia, PA, 2004).
- [3] AiroPeek. <http://www.wildpackets.com/products/airopeek>.
- [4] AirTight Networks. <http://www.airtightnetworks.com>.
- [5] Airwave - Wireless Network Management Software. <http://www.airwave.com>.
- [6] Aruba Networks. <http://www.arubanetworks.com>.
- [7] BAHL, P., CHANDRA, R., PADHYE, J., RAVINDRANATH, L., SINGH, M., WOLMAN, A., AND ZILL, B. Enhancing the Security of Corporate Wi-Fi Networks Using DAIR. In *Proc. of MobiSys* (Uppsala, Sweden, 2006).
- [8] Bay Area Wireless User Group. <http://www.bawug.org/>.
- [9] BICKET, J., AGUAYO, D., BISWAS, S., AND MORRIS, R. Architecture and evaluation of an unplanned 802.11b mesh network. In *Proc. of MobiCom* (New York, NY, 2005).
- [10] CHANDRA, R., PADMANABHAN, V. N., AND ZHANG, M. WiFiProfiler: Cooperative Diagnosis in Wireless Monitoring. In *Proc. of MobiSys* (Uppsala, Sweden, 2006).
- [11] CHENG, Y.-C., BELLARDO, J., BENKO, P., SNOEREN, A. C., VOELKER, G. M., AND SAVAGE, S. Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis. In *Proc. of SIGCOMM* (Pisa, Italy, 2006).
- [12] CONAN: Congestion Analysis of Wireless Networks. <http://moment.cs.ucsb.edu/conan/>.
- [13] Champaign-Urbana Community Wireless Network. <http://www.cuwireless.net/>.
- [14] Wireless LAN Infrastructure Mesh Networks: Capabilities and Benefits. www.firetide.com, July 2004.
- [15] An Introduction to Wireless Mesh Networking. <http://whitepapers.techrepublic.com.com>, March 2005.
- [16] NLANR/DAST : Iperf - The TCP/UDP Bandwidth Measurement Tool.

- <http://dast.nlanr.net/Projects/Iperf/>.
- [17] JARDOSH, A. P., RAMACHANDRAN, K. N., ALMEROOTH, K. C., AND BELDING-ROYER, E. M. Understanding Congestion in IEEE 802.11b Wireless Networks. In *Proc. of Internet Measurement Conference* (2005).
 - [18] JUN, J., PEDDABACHAGARI, P., AND SICHITIU, M. Theoretical Maximum Throughput of IEEE 802.11 and its Applications. In *Proc. of IEEE International Symposium on Network Computing and Applications* (Cambridge, MA, 2003).
 - [19] MADWiFi - Multiband Atheros Driver for WiFi.
<http://www.madwifi.org>.
 - [20] MAHAJAN, R., RODRIG, M., WETHERALL, D., AND ZAHORJAN, J. Analyzing the MAClevel Behavior of Wireless Networks in the Wild. In *Proc. of SIGCOMM* (Pisa, Italy, 2006).
 - [21] IETF MANET Working Group Charter.
<http://www.ietf.org/html.charters/manet-charter.html>.
 - [22] Mesh Dynamics Structured Mesh Networking for Mobile Data, Video and Voice.
http://meshdynamics.com/documents/MD4000_BROCHURE.pdf.
 - [23] Self-Organizing Neighborhood Wireless Mesh Networks.
<http://research.microsoft.com/mesh/>.
 - [24] RAMAN, B., AND CHEBROLU, K. Experiences in using WiFi for rural internet in India. *IEEE Communications Magazine* 45, 1 (Jan 2007), 104–110.
 - [25] SFLan - Community Wireless.
<http://www.sflan.org>.
 - [26] Tropos MetroMesh Architecture Overview.
http://www.tropos.com/pdf/metromesh_datasheet.pdf, 2006.
 - [27] tshark - Network Protocol Analyzer.
<http://www.wireshark.org>.
 - [28] Wireless Leiden - Netherlands.
<http://www.wirelessleiden.nl>.