Build and Use a Homebrewed IEEE 802.11 Wireless Network Sniffer

Mingzhe Li, Mark Claypool, Robert Kinicki
{lmz,claypool,rek}@cs.wpi.edu
CS Department at Worcester Polytechnic Institute
Worcester, MA, 01609, USA

November 29, 2005

1 Introduction

IEEE 802.11 Wireless Sniffers have been widely used in research and network management communities because of their capabilities to do network monitoring and MAC layer frame sniffing. However, most commercial wireless sniffers are costly and are complex to use. This document describes how to build and use a simple IEEE 802.11 wireless sniffer from open source software and off-the-shelf wireless network interface cards. This homebrewed wireless sniffer can provide the basic functionality for monitoring wireless 802.11 networks and MAC layer frame capture at reduced cost. All of the tools and packets included in this document were gathered from Internet sources. You can refer to the links embedded in this document to get detailed information.

The document is organized as follows. Section 2 introduces related wireless measurement approaches and other related works. Section 3 described how to build the wireless sniffer. Section 4 discusses what can be done by using this homebrewed sniffer. Finally, have fun and use the sniffer at your own risk.

2 Background and Related Work

Wireless measurements can be performed either on the Access Point (AP), mobile host, or by a special designed network monitoring/sniffing system. For instance, the research in [1] characterizes user behavior and wireless network performance in the public IEEE 802.11 network of a conference by collecting Simple Network Management Protocol (SNMP) traces from the APs. Similarly, research in [2, 3, 4] analyzes either metropolitan area or campus wide wireless network by collecting AP system log and SNMP information. Additionally, Ho *et al.* [5] present VISUM, a scalable framework for wireless network monitoring based on similar methodology. VISUM relies on a distributed set of agents within the network to monitor network devices and therefore supports a much larger scale of networks.

Wireless measurement can be applied to the mobile host. Wireless Research API (WRAPI) [6] is a software library that allows applications running in user-space on mobile hosts (and APs) to query/set information in the IEEE 802.11 network. WRAPI provides an interface for applications to monitor the WLAN in real time by interacting with Network Driver Interface Specification (NDIS) stack of Windows XP. Since WRAPI does not make direct contact with the hardware driver, it is hardware independent and supports all 802.11b and 802.11g compliant hardware in Windows XP systems. However, WRAPI cannot provide detailed information, such as packet level statistical information and does not work in promiscuous mode, which limits its capability as a network monitor tools. Recent research [7] uses WRAPI to capture the WLAN performance information, including wireless layer Received Signal Strength Indicator (RSSI), MAC layer retry counts, multiple retry counts, ACK failure counts and duplicate frame counts.

To get MAC level frame information for a wireless network, a wireless sniffer system is usually used. A wireless sniffer can be installed on a measured host, but in most cases, it is installed on an independent device, such as a mobile computer or an PDA system. Therefore the sniffer can monitor the wireless network in promiscuous mode without interfering with the stations under measurement. Wireless sniffers can capture not only the data frames, but also management frames, such as beacon frames, and RTS/CTS/ACK frames. However, the wireless sniffer requires special hardware and driver support. The most popular wireless sniffer and analyzer software includes Ethereal [1], Kismet [2] and some commercially available wireless sniffers such as Sniffer Wireless (Used to be Network Associates Sniffer) [3], AiroPeek NX [4], etc. Wireless sniffers have been widely used in wireless performance research, such as the independent sniffer used in the measurement of streaming media over wireless research [8, 9], and the on host software sniffer used in the link level measurement research for a wireless roof network [10]. Moreover, in the network monitor research in [11], a complete wireless sniffer system is implemented and used to characterize a typical computer science department WLAN traffic.

3 Build the Sniffer

The wireless sniffers used in our research group are built on computers with Linux operating systems and prism GT based wireless interface cards. The operating systems we tested are SuSe (Novell) Linux release 9.0/9.1/9.2/10.0 and Linux Fedora Core 3 where the kernel version can be either 2.4.x or 2.6.x. The wireless network interface cards tested are Netgear WG 511 version 1 PCMCIA card and Allnet ALL0271 54Mbit Wireless PCI adapter. Both of these cards are built on prism GT chip set. The following steps describe how to build a wireless sniffer with SuSe Linux operating system.

1. Install the SuSe Linux system with your prism GT PCI/PCMCIA card plugged in the system. The prism GT card will be automatically detected during the installation. However, the firmware is not installed by default. You need to run `yast2` manually to perform online update and select to download the firmware. To make the next step easier, you also need to install Linux kernel source when you install the Linux operating system. You can refer to SuSe Linux [5] for a detailed installation guide.

2. Update the driver (prism54 kernel module) to the latest version. If the Linux version is SuSe 9.2 or later, you do not need to update the driver version, unless you want to rebuild the driver to modify the functionality.

   Before you update the driver (kernel module prism54), you should make sure that your wireless interface card works well under the default prism54 module provide by SuSe release. The general Linux commands for testing the card status include `iwconfig, iwlist and iwpriv`. Assume the wireless interface card is binded to interface name `eth1`. The commands `iwconfig eth1`, `iwlist eth1 scan` will show the configuration of the wireless interface and the currently available wireless networks, respectively. For more information about wireless tools under Linux, please refer to the wireless tools for Linux [6]. However, it is not necessary to install these tools because they are already included in the SuSe Linux release.

[1] http://www.ethereal.com
[2] http://www.kismetwireless.net/index.shtml
[3] http://www.sniffer.com
[4] http://www.wildpackets.com/elements/AiroPeek_NX.pdf
[5] http://www.novell.com/linux/suse/
[6] http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

The latest stable release of the prism54 kernel module is version `rel-1-2`, which can be downloaded from the Prism54 Project webpage [7]. Detailed installation steps of the prism54 is included in the README file included in the `rel-1-2` tarball.

3. Create an interface configuration file. After the prism54 module is installed, you can modify the interface configure file that will be used to bring the interface up. For SuSe Linux, you can use either the configure tool `yast2`, or manually modify the configuration file under the default location `/etc/sysconfig/network/ifcfg-eth1`. There are few lines you need to change:

```
ONBOOT='no'
WIRELESS='yes'
WIRELESS_MODE='Monitor'
```

   Applying `sudo ifup eth1; sudo iwpriv eth1 set_prismhdr 1` will bring the interface up in the monitor mode, with `AVS` [8] header dump (which dumps the extra PHY/MAC layer information into a emulated header of the wireless frames) option enabled. The AVS header is discussed in Section 4 in detail.

4. Use network sniffing tools to capture frames. After bringing the network interface up, you may use popular network sniffing tools to capture and analyze the frames, such as `tcpdump` [9], Ethereal/tethereal or Kismet. To select the right channel to sniff, you can use `sudo iwconfig eth1 channel x` to setup the channel you want to monitor, where `x` is the integer number from 1 to 11 (for USA) of the channel.

4 Use the sniffer

As discussed in Section 2, network sniffing is one of the methods used to monitor wireless networks. It comes with advantages and disadvantages. For example, sniffing from an independent sniffer will not cause any interference with the experiments in wireless networks. Plus, sniffing can provide frame level information and wireless network conditions, such as RSSI and the sending rate. Wireless sniffers can also capture wireless management frames, such as RTS/CTS, Authentication/Deauthentication, and Association/Disassociation. Thus, sniffers can also be used as wireless network diagnostic tools. However, one of the challenges is that the sniffer cannot record all the frames that are trasnmitted over the network [12]. This is because the sniffer is only capturing the frames at its own location. Therfore, the packets lost due to a hidden terminal and bit errors are not captured. Additionally, RSSI (Received Signal Strength Indicator) is measured relatively from the location where the sniffer is installed, but not at the AP or the clients. The location of the sniffer is an important issue related to the purpose of the sniffing. For example, a location very close to an AP is helpful when studying the AP behavior. However, it may miss some traffic sent from a distant client due to signal attenuation.

Beside the basic frame capturing functionality of a sniffer, this sections briefly reviews some additional features of this homebrewed sniffer.

First, the wireless sniffer can provide extra PHY/MAC layer information by an extra emulated header, the AVS header. The AVS header includes the RSSI, rate, channel, PHY and other important information for each frame. The commend to enable wireless AVS header capture is `sudo iwpriv set_prismhdr 1`. The following example of AVS header is captured from an IEEE 802.11g network using Ethereal.

[7] http://prism54.org/
[8] http://www.ethereal.com/docs/dfref/w/wlancap.html
[9] http://www.tcpdump.org/

```
AVS WLAN Monitoring Header
    Header revision: 1
    Header length: 64
    MAC timestamp: 4046472034
    Host timestamp: 1040663
    PHY type: OFDM 802.11g (6)
    Channel: 11
    Data Rate: 54000 Kb/s
    Antenna: 0
    Priority: 0
    SSI Type: Raw RSSI (3)
    SSI Signal: 99
    SSI Noise: 97
    Preamble: Unknown (0)
    Encoding Type: Unknown (0)
```

The data shows that the type of PHY layer is OFDM 802.11 and current channel is 11. It also indicates that the sending rate of this frame is 54Mbps, the Raw RSSI at the sniffing location is 99 and the noise level is 97. However, we can not compute the SNR (signal-noise-ratio) by $99 - 97$ in this case, because the noise in AVS usually is captured as the background noise level when no packet is transmitted. Thus, the `SSI Noise` can be interpreted as link quality but cannot be used to compute the SNR.

Second, each captured frame has an IEEE 802.11 header section, which provides frame information, such as MAC layer retry, power management and WEP. For example, the following header section is captured from an IEEE 802.11g network by Ethereal. It shows that this frame is a retry of the previous frame, and the network has WEP enabled.

```
IEEE 802.11
    Type/Subtype: Data (32)
    Frame Control: 0x4908 (Normal)
      Version: 0
      Type: Data frame (2)
      Subtype: 0
      Flags: 0x49
          DS status: Frame is entering DS (To DS: 1  From DS: 0) (0x01)
          .... .0.. = More Fragments: This is the last fragment
          .... 1... = Retry: Frame is being retransmitted
          ...0 .... = PWR MGT: STA will stay up
          ..0. .... = More Data: No data buffered
          .1.. .... = WEP flag: WEP is enabled
          0... .... = Order flag: Not strictly ordered
    Duration: 213
    BSS Id: SSS_77:88:99 (00:0b:85:77:88:99)
    Source address: XXX_11:22:33 (00:90:4b:11:22:33)
    Destination address: YYY_22:33:44 (00:00:5e:22:33:44)
    Fragment number: 0
    Sequence number: 1127
    TKIP/CCMP parameters
      TKIP Ext. Initialization Vector: 0x000000000013
      Key: 0
```

We can also trace the retry behavior at the frame level. For example, the following trace shows that the data frame had been sent three times (i.e, two retries) before the receiver successfully receives the frame. An Acknowledgment frame sent from the receiver indicates the data frame is received.

```
No.     Time       Source         Destination          Protocol Info
2458 55.951347                  XXX_1a:97:ab (RA)   IEEE 802.11 Clear-to-send
2459 55.951553  XXX_1a:97:ab  YYY_11:30:a8        IEEE 802.11 Data
2460 55.951831                  XXX_1a:97:ab (RA)   IEEE 802.11 Clear-to-send
2461 55.952174  XXX_1a:97:ab  YYY_11:30:a8        IEEE 802.11 Data
2462 55.952847                  XXX_1a:97:ab (RA)   IEEE 802.11 Clear-to-send
2463 55.953895  XXX_1a:97:ab  YYY_11:30:a8        IEEE 802.11 Data
2464 55.954070                  XXX_1a:97:ab (RA)   IEEE 802.11 Acknowledgement
```

Finally, the sniffer can also be used for security purposes. For example, it can be used to detect intrusions and spoof attacks. However, since our focus is on performance studies, we do not consider this functionality in this document.

References

[1] Anand Balachandran, Geoffrey M. Voelker, Paramvir Bahl, and P. Venkat Rangan, "Characterizing user behavior and network performance in a public wireless lan," in *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2002, pp. 195–205. 1

[2] Diane Tang and Mary Baker, "Analysis of a metropolitan-area wireless network," in *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, 1999, pp. 13–23. 1

[3] Diane Tang and Mary Baker, "Analysis of a local-area wireless network," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, 2000, pp. 1–10. 1

[4] David Kotz and Kobby Essien, "Analysis of a campus-wide wireless network," in *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, 2002, pp. 107–118. 1

[5] Camden C. Ho, Krishna N. Ramachandran, Kevin C. Almeroth, and Elizabeth M. Belding-Royer, "A scalable framework for wireless network monitoring," in *WMASH '04: Proceedings of the 2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, 2004, pp. 93–101. 1

[6] A. Balachandran and G. Voelker, "WRAPI – Real-time Monitoring and Control of an 802.11 Wireless LAN," Tech. Rep., CS at UCSD, 2004. 1

[7] Feng Li, Jae Chung, Mingzhe Li, Huahui Wu, Mark Claypool, and Robert Kinicki, "Application, Network and Link Layer Measurements of Streaming Video over a Wireless Campus Network," in *Proceedings of the 6th Passive and Active Measurement Workshop (PAM)*, Boston, Massachusetts, USA, Apr. 2005. 1

[8] Tianbo Kuang and Carey Williamson, "RealMedia Streaming Performance on an IEEE 802.11b Wireless LAN," in *Proceedings of IASTED Wireless and Optical Communications (WOC)*, July 2002, pp. 306–311. 2

[9] Guangwei Bai and Carey Williamson, "The Effects of Mobility on Wireless Media Streaming Performance," in *Proceedings of Wireless Networks and Emerging Technologies (WNET)*, July 2004, pp. 596–601. 2

[10] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris, "Link-level measurements from an 802.11b mesh network," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, Portland, Oregon, USA, 2004, pp. 121–132. 2

[11] Jihwang Yeo, Moustafa Youssef, and Ashok Agrawala, "A framework for wireless lan monitoring and its applications," in *ACM Workshop on Wireless Security (WiSe 2004) in conjunction with ACM MobiCom 2004*, Philadelphia, PA, USA, Oct. 2004. 2

[12] Mark Claypool, "On the 802.11 turbulence of nintendo ds and sony psp hand-held network games," in *Proceedings of the 4th ACM Network and System Support for Games (NetGames)*, Hawthorne, NY, USA, Oct. 2005. 3