RFDump: An Architecture for Monitoring the Wireless Ether

Kaushik Lakshminarayanan, Samir Sapra, Srinivasan Seshan, Peter Steenkiste
Carnegie Mellon University
Pittsburgh, PA 15213
{kaushik, ssapra, srini, prs}@cs.cmu.edu

Abstract

Networking researchers have been using tools like *wireshark* and *tcpdump* to sniff packets on physical links that use different types of datalink protocols, e.g. Ethernet or 802.11, allowing them to monitor higher level protocols sharing these links. However, monitoring wireless links is more challenging, since the transmission medium is shared by flows using diverse datalink protocols (e.g. 802.11, Bluetooth) and physical layer schemes (e.g. QPSK and GFSK). To this end, we propose *RFDump*, a software architecture for monitoring packets on heterogeneous wireless networks. The key idea underlying our architecture is the use of a fast detection stage which can tentatively map signals to protocols very efficiently. As a result, RFDump can scale up to a modest number (5-10) of wireless technologies.

We implemented RFDump on the GNU Radio and USRP platforms. This is, to our knowledge, the first inexpensive software-based infrastructure for simultaneously analyzing multiple wireless protocols in real-time. Using traces from the real world and from a wireless emulator testbed, we show that our implementation is efficient and accurate. Further, we demonstrate that our system is extensible and scales with the addition of new protocols.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Measurement Techniques]; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless Communication*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*

General Terms

Measurement, Experimentation, Performance

Keywords

software defined radio, wireless networks, monitoring, Wi-Fi, Bluetooth, tcpdump

1. INTRODUCTION

Tcpdump, Wireshark/Ethereal and similar applications have become a critical part of the tool collections used by networking researchers, networking administrators and application developers. These tools expose the operation of a network in a detailed, cross-layer fashion. Based on this exposed information, users are able to monitor and analyze the interactions between different nodes, different protocols, different protocol layers and different applications in the network. This has enabled activities such as diagnosing network protocols, optimizing network performance and even teaching network protocol operation.

Unfortunately, applying these tools in wireless networks fails to provide the same level of insight into the operation of the network. There are two reasons for this problem. First, these tools operate at the link-layer and above. In wireless settings, the behavior of the physical layer is critical to the operation of the network. Second, these tools are limited to operation over a single network interface card (NIC), such as an 802.11 NIC. As a result, they can only report on the detailed operation of the associated network link technology. However, unlike wired networks, the physical medium over which the network operates is shared by many link technologies. For example, the 2.4 GHz unlicensed spectrum band is shared by 802.11, Bluetooth, ZigBee, cordless phones and a wide range of other link technologies. Making observations on a single link technology hides many of the node, protocol and application interactions that users are attempting to observe with such tools. In this paper, we describe the design of RFDump, a tool that extends the monitoring capabilities below the link layer and enables more effective monitoring of the wireless ether.

In order to be practical, a monitoring tool for wireless networks must meet two key requirements. First, we must be able to monitor packets that use a wide variety of protocols, so the tool must efficiently support multiple protocols and it must be easy to add new protocols in the future. Second, the tool must run in real-time so it can be used for run-time analysis and troubleshooting. Note that we do not expect our system to interact with the monitored links (i.e., it does not need to implement the link-layer protocol). As a result, our system can process transmissions after some delay (e.g., a second) but the processing must keep up with the rate of packet transmissions. In addition, while core functions, such as identifying packets and the technology they use, must occur in real time, more complex functions, such as full decoding of payloads or deep packet inspection, may only be feasible for a subset of the traffic in the ether.

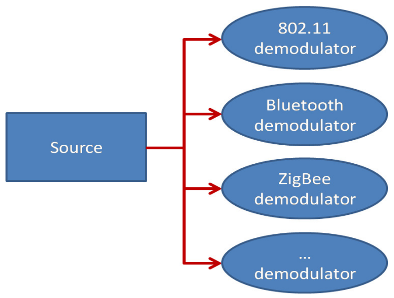

Figure 1: The naïve architecture

This may not seem like a significant challenge since tools such as Tcpdump are able to decode a wide range of protocols efficiently. The key to this efficiency is that each protocol layer specifies the protocol used by its contents. For example, the IP header contains a *protocol* field that identifies the transport protocol of the datagram contents. This allows Tcpdump to run just the code needed to decode the appropriate protocol. Unfortunately, the physical layer does not explicitly identify the protocol used by an active transmissions. Instead wireless networking cards use a combination of preambles, modulation and coding schemes, and header information to determine the protocol. As a result, the most obvious and naïve solution (Figure 1) to performing wireless monitoring would require that we monitor all link-layers in parallel (i.e. try to interpret every signal with every protocol). This solution is either expensive (for hardware) or slow (for software).

The core of our design is the decomposition of the problem into a detection stage followed by the demodulation stage. The detection stage can tentatively map signals to protocols very efficiently – essentially providing a protocol tag much like the ones that Tcpdump relies upon. We rely on some key observations to make these detection modules much more light-weight than complete demodulation. First, these detectors can operate with some delay, which enables the use of algorithms that are not appropriate for demodulation. Second, unlike demodulators, these detectors are allowed to have false positives. If that happens, the signal is passed to demodulation code to interpret the content of the transmission and the demodulator will then determine that the signal does not represent a valid packet for that protocol.

This paper makes three contributions. First, we present the RFDump architecture for monitoring diverse wireless links. The architecture introduces a light-weight detection stage before demodulation so that demodulation needs to be performed only on actual RF transmissions. Second, we introduce a specific set of fast early detectors for devices using 802.11b/g and Bluetooth, as well as other RF devices such as microwave ovens. Finally, we present a prototype implementation of the RFDump architecture on the GNU Radio [14] and USRP [17] software defined radio (SDR) platforms. Our implementation is an early prototype (limited in various ways by the underlying hardware platform we use) used to evaluate the potential benefits of the architecture. We compare its performance with a naïve solution and show that our architecture is much more efficient, while maintaining the same level of accuracy. We not only detect most of the packets detected by the naïve solution, but also packets

GNU Radio Block	CPU time / Real time
802.11 demodulation (1 Mbps)	0.6
Bluetooth demodulation	0.7
Peak/Energy detection	0.05

Table 1: Time taken by some blocks

which cannot be demodulated due to the limitations of the USRP interface. Although the individual detection modules themselves play a key role in achieving efficiency and scalability, the main contribution of the paper lies in how the architecture is designed for monitoring different types of wireless link technologies in an efficient manner.

The rest of the paper is structured as follows. The next section presents the RFDump architecture and Section 3 describes our early detection modules that can detect packets belonging to a number of protocols without demodulating and decoding. Section 4 describes the implementation of the architecture on the GNU Radio and USRP framework. Section 5 evaluates our implementation for efficiency and accuracy by comparing it with straightforward but naïve alternatives. Section 6 compares our architecture with alternate approaches and we summarize our work in Section 7.

2. ARCHITECTURE

2.1 Motivation and Requirements

Monitoring wireless networks is difficult because activity on the transmission medium (the ether) is difficult to observe and decode. The problem is that unlicensed spectrum is open to anybody with only minimal limitations, and as a result, a wide variety of physical and datalink layers are in use. Nevertheless, it is important to get a full picture of the activity in the shared spectrum. For example, when diagnosing Wi-Fi problems, a full picture is critical because non-Wi-Fi users can reduce the (Wi-Fi) network capacity by reducing transmission opportunities or, even worse, cause high packet error rates if the technologies cannot coexist.

This leads to the following requirements for a wireless monitoring tool.

- **Multi-protocol:** it must support simultaneous monitoring of at least a small (e.g. 5-10) number of protocols and RF sources.
- **Real-time:** it must perform core functions, e.g. identifying packets and the technology they use, in real time.
- **Protocol Extensible:** it must be relatively simple to add support for new protocols, e.g. 802.11n.
- **Functionality Extensible:** it should be possible to add additional modules that further analyze traffic, e.g. demodulator, diagnostic modules, deep packet inspection. Unless otherwise specified, we refer to the process of demodulation, decoding and analysis together as demodulation for the rest of the paper.

Given these high-level requirements, let us consider the suitability of the naïve architecture shown in Figure 1. Here, the entire input stream is sent to demodulators for all technologies that may be in use. Implementing this architecture using separate hardware for each demodulator is both