Demo : Linux Goes Apple Picking: Cross-Platform Ad hoc Communication with Apple Wireless Direct Link

Stute, Milan; Kreitschmann, David; Hollick, Matthias

(2018)

Demo: Linux Goes Apple Picking: Cross-Platform Ad hoc Communication with Apple Wireless Direct Link

Milan Stute
Secure Mobile Networking Lab
TU Darmstadt, Germany
mstute@seemoo.de

David Kreitschmann
Secure Mobile Networking Lab
TU Darmstadt, Germany
dkreitschmann@seemoo.de

Matthias Hollick
Secure Mobile Networking Lab
TU Darmstadt, Germany
mhollick@seemoo.de

ABSTRACT

Apple Wireless Direct Link (AWDL) is a proprietary and undocumented wireless ad hoc protocol that Apple introduced around 2014 and which is the base for applications such as AirDrop and AirPlay. We have reverse engineered the protocol and explain its frame format and operation in our MobiCom '18 paper "One Billion Apples' Secret Sauce: Recipe of the Apple Wireless Direct Link Ad hoc Protocol." AWDL builds on the IEEE 802.11 standard and implements election, synchronization, and channel hopping mechanisms on top of it. Furthermore, AWDL features an IPv6-based data path which enables direct communication.

To validate our own work, we implement a working prototype of AWDL on Linux-based systems. Our implementation is written in C, runs in userspace, and makes use of Linux's *Netlink* API for interactions with the system's networking stack and the *pcap* library for frame injection and reception. In our demonstrator, we show how our Linux system synchronizes to an existing AWDL cluster or takes over the master role itself. Furthermore, it can receive data frames from and send them to a MacBook or iPhone via AWDL. We demonstrate the data exchange via ICMPv6 echo request and replies as well as sending and receiving data over a TCP connection.

CCS CONCEPTS

• **Networks** → **Ad hoc networks**; **Link-layer protocols**;

KEYWORDS

AWDL, IEEE 802.11, Apple, macOS, iOS, Linux, Netlink

MobiCom '18, October 29–November 2, 2018, New Delhi, India

ACM Reference Format:
Milan Stute, David Kreitschmann, and Matthias Hollick. 2018. Demo: Linux Goes Apple Picking: Cross-Platform Ad hoc Communication with Apple Wireless Direct Link. In *The 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18), October 29–November 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3241539.3267716

1 INTRODUCTION AND BACKGROUND

New types of proximity-based services such as contactless payment (e. g., NFC), location-aware advertisements (e. g., Bluetooth LE), user-aware security measures (e. g., Apple Auto Unlock), peer-to-peer file transfers (e. g. Apple AirDrop), and media streaming (e. g., Apple AirPlay) have reignited the interest in wireless ad hoc communications. One key enabling technologies for such services is AWDL which provides Wi-Fi speed data transfers between neighboring devices but is—unfortunately—only available in Apple devices. In our MobiCom'18 paper [7], we investigated the workings of this protocol, released an open source Wireshark dissector [3], and conducted a performance evaluation with Apple's implementations. In this paper, we draw on these findings and present a working prototype of AWDL as a Linux userspace daemon which can take part in the AWDL election and synchronization process, can be discovered by others, and receives and transmits data frames from and to other AWDL devices. Our implementation integrates itself in the Linux networking stack by providing a virtual network interface such that existing IPv6-capable programs can use AWDL without modification. Our setup consists of a Linux-based machine and macOS/iOS devices, and we can show that arbitrary programs (e. g., `ping` and `netcat`) successfully run over our AWDL implementation. Our work proves that cross-platform ad hoc communication is feasible and we provide a base for future cross-platform ad hoc applications.

2 IMPLEMENTATION

We implement our prototype in plain C for performance reasons and to facilitate porting the code to other platforms.

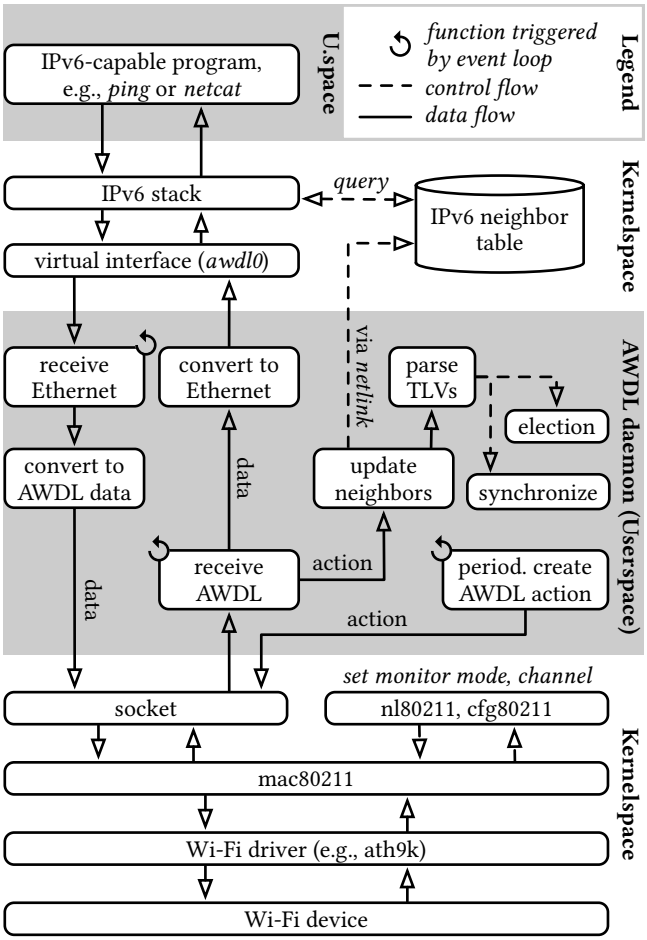

Figure 1: Architecture of our AWDL prototype and integration in the Linux networking stack.

2.1 Overview

We depict the architecture and integration of our AWDL daemon in Fig. 1. At its core, the daemon uses an event loop (*libuv*[1]) that (1) listens on the Wi-Fi interface for new IEEE 802.11 frames using *libpcap*,[2] (2) listens on a virtual Ethernet interface for new traffic from the host system, and (3) periodically schedules the transmission of AWDL action frames that carry information used for peer discovery, synchronization, and election procedures.

When a new Wi-Fi frame is available on the monitoring interface, we check whether the frame is an AWDL action or data frame. Other frames such as regular IEEE 802.11 frames are dropped by a BPF filter that only forwards frames with the AWDL-specific BSSID `00:25:00:ff:94:73`. If we receive an action frame, we derive [2] a link-local IPv6 address from the source Ethernet address and add both to the system's neighbor table. Based on the included Type-Length-Value (TLV) fields, we run the election and synchronization mechanisms as described in [7]. If we receive a data frame, we strip the AWDL data header, replace it with a regular Ethernet header, and forward the frame to the virtual `awdl0` interface. We do the inverse for Ethernet frames that we receive from `awdl0` and add the AWDL sequence number from an internal counter. In addition, the daemon periodically emits AWDL action frames that it builds from its internal synchronization and election state. We document the complete frame format in our Wireshark dissector [3].

2.2 Portability and Future Work

Since our prototype is written in C, it should be possible to port the code to different operating systems. However, we have the following dependencies that each target platform needs to provide: (1) a Wi-Fi card supporting active monitor mode with frame injection to be able to receive and send IEEE 802.11 frames, (2) a means to change the Wi-Fi channel such as *nl80211*, (3) access to the system's IPv6 neighbor table, and (4) a facility to create virtual network interfaces such as TUN/TAP. In principle, this should allow implementations on Android smartphones where monitor mode and frame injection can be enabled using the Nexmon framework [5]. Our prototype currently lacks a channel switching mechanism that would be required to follow nodes to a different channel. However, since AWDL devices usually meet on one social channel (6, 44, or 149), our prototype still works by continually listening on a fixed channel.

2.3 Enabling AWDL in macOS and iOS Third-party Applications

On Linux, every program using sockets can use our `awdl0` interface. On macOS, programs must set an XNU-specific `SO_RECV_ANYIF`[3] socket option.[4] Using this option, any software on macOS using sockets can support AWDL, thus, enabling cross-platform applications with minor modifications to the code. As an alternative, programs on macOS and iOS can use the higher-level `NetService` API [1] which activates mDNS/DNS-SD and establishes TCP connections via the `awdl0` interface. We provide an example application implementing a TCP–AWDL proxy [6]. For cross-platform communication, the Linux system must support mDNS/DNS-SD, e. g., via avahi.[5]

[1] http://libuv.org
[2] https://github.com/the-tcpdump-group/libpcap
[3] https://opensource.apple.com/source/xnu/xnu-4570.41.2/bsd/sys/socket.h
[4] The socket option is the "default packet filter" that we discuss in [7].
[5] https://www.avahi.org

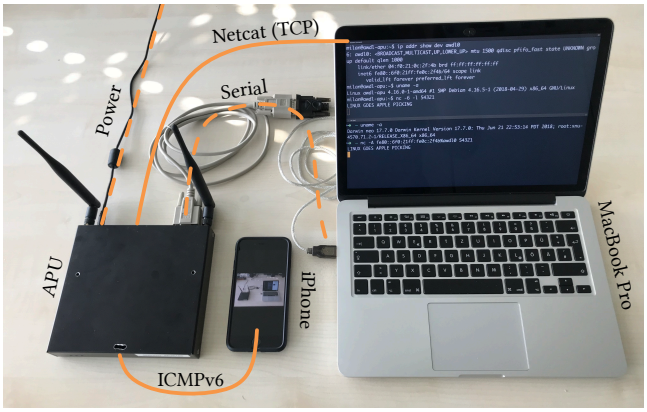

Figure 2: Demonstrator setup consisting of a Linux-based APU board, an iPhone 8, and a MacBook Pro. The terminal on the MacBook's screen shows a working TCP-over-AWDL connection between the APU board and the MacBook.

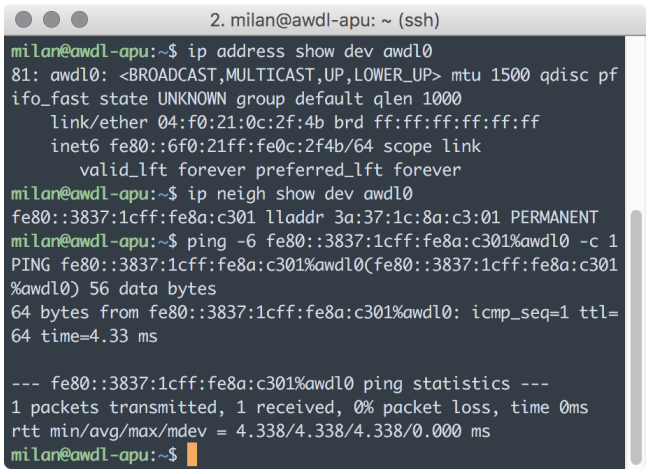

Figure 3: Terminal showing our AWDL implementation running on a Linux host.

3 DEMONSTRATOR

We briefly summarize our demonstrator devices, the activities that the attendee can see, and list our requirements for our on-location setup. We show our setup in Fig. 2.

3.1 Devices

Our AWDL implementation runs on an APU board [4] with a Qualcomm Atheros AR928X Wi-Fi card which implements the IEEE 802.11n standard and uses the ath9k driver which provides frame injection in monitor mode. We use various Apple devices such as iPhone and MacBook to demonstrate cross-platform communication.

3.2 Activities by Attendee

We can offer to show different aspects of our cross-platform communication depending on the attendees' demands.

- We can show the neighbor tables in both Apple and Linux implementations which contains the AWDL peers if they emit action frames.
- We can conduct a Wireshark live capture of AWDL frames and dissect their content. We can then also analyze these capture files w.r.t. election behavior and synchronization accuracy similar to [7].
- We can send and receive ICMPv6 echo requests and replies by using `ping`.
- We can establish a TCP connection between two AWDL nodes and send messages by using `netcat`.
- We invite attendees to do all of the above with their own macOS or iOS devices as well.

We depict some of the above in Fig. 3 and might offer more activities subject to the results of our ongoing research.

3.3 Requirements at Location

For our demonstrator, we require (1) a table, (2) three power outlets with Europlugs next to the table, and (3) about one hour for the setup.

ACKNOWLEDGMENTS

This work is funded by the LOEWE initiative (Hesse, Germany) within the NICER project and by the German Federal Ministry of Education and Research (BMBF) and the State of Hesse within CRISP-DA.

REFERENCES

[1] Apple Inc. 2018. NSNetService Class Documentation. Retrieved June 28, 2018 from https://developer.apple.com/documentation/foundation/nsnetservice

[2] Robert M. Hinden and Stephen E. Deering. 2006. IP Version 6 Addressing Architecture. *RFC* 4291 (Feb. 2006). https://doi.org/10.17487/RFC4291

[3] David Kreitschmann and Milan Stute. 2018. AWDL and CoreCapture Wireshark dissector. https://seemoo.de/wireshark-awdl

[4] PC Engines. 2018. APU Platform. Retrieved June 28, 2018 from https://www.pcengines.ch/apu.htm

[5] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. 2017. Nexmon: The C-based Firmware Patching Framework. https://nexmon.org

[6] Milan Stute. 2018. proxAWDL: simple AWDL–TCP proxy. https://seemoo.de/proxawdl

[7] Milan Stute, David Kreitschmann, and Matthias Hollick. 2018. One Billion Apples' Secret Sauce: Recipe for the Apple Wireless Direct Link Ad hoc Protocol. In *The 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18).* https://doi.org/10.1145/3241539.3241566