Preparing FLOSS for Future Network Paradigms: A Survey on Linux Network Management

Alfredo Matos, John Thomson, Paulo Trezentos

Caixa Mágica Software
Edificio Espanha - Rua Soeiro Pereira Gomes
Lote 1 - 8 F, 1600-196 Lisboa
{alfredo.matos,john.thomson,paulo.trezentos}|@caixamagica.pt

Abstract. Operating system tools must fulfil the requirements generated by the advances in networking paradigms. To understand the current state of the Free, Libre and Open Source Software (FLOSS) ecosystem, we present a survey on the main tools used to manage and interact with the network, and how they are organized in Linux-based operating systems. Based on the survey results, we present a reference Linux network stack that can serve as the basis for future heterogeneous network environments, contributing towards a standardized approach in Linux. Using this stack, and focusing on dynamic and spontaneous network interactions, we present an evolution path for network related technologies, contributing to Linux as a network research operating system and to FLOSS as a whole.

1 Introduction

Free, Libre and Open Source Software (FLOSS) is often characterized by a distributed and even fractured development model. This can lead to different applications with similar purposes, where the consequence is often effort duplication. While this can simultaneously be characterized as an advantage or handicap of the FLOSS world, these characteristics are also observable in the networking aspects of Linux. The Linux Kernel is very rich in terms of networking functionality, with a modern stack that makes it a reliable network Operating System (OS). However, most network management[1] operations are usually executed in user-space, by distribution specific tools, which can vary across distributions.
This dichotomy is further exemplified by the networking paradigms: on one hand, they require supporting multiple heterogeneous networks as specified by Next Generation Networks (NGN), relating to different concurrent technologies on the device, such as WiFi, 3G, WiMax

[1] When we consider management, we are in fact referring to bringing up devices, selection of network attachment points, performing dynamic configurations and the associated integration that cannot be configured statically or hard coded into applications.

or even Bluetooth; while on the other hand, dynamic and spontaneous configuration require supporting peer-to-peer interactions that operate without infrastructure support such as Zeroconf [?] technologies and even Ad-Hoc or Mesh Networks. Therefore, as we move towards NGN environments, where wireless connectivity is the norm rather than the exception, powered by WiFi or 3G connections, it becomes increasingly necessary to handle all the different connectivity scenarios and technologies, without compromising the user experience. This creates added complexity for the OS and network stack that must allow a seamless user experience with competing requirements.
To handle these concurrent vectors, we need a consistent FLOSS network stack that aligns the different tools and approaches, to match the needs of the evolving network environments. To achieve this, we present a survey that looks at the current Linux network model and existing technologies. Based on the survey of both FLOSS tools and Linux distributions, we open the door to an aligned network view by proposing a reference network stack that promotes standardized approaches. This reference stack, which takes into account both heterogeneous networks and spontaneous dynamic environments, can contribute to the evolution of FLOSS, and especially Linux, by helping to prepare for new and forthcoming NGN network scenarios that are being promoted in different venues, such as the ULOOP [?] IST Project.
By promoting a common vision based on current research scenarios, it is possible to promote Linux as a leading research platform, and simultaneously contribute to less effort-duplication and avoiding fractures in the development model, thus contributing to FLOSS as a whole, as is discussed in Sect. **??**. This can be achieved using the reference stack presented in Sect. **??** as a starting point. The remainder of the paper is organized as follows: Section **??** highlights the importance and structure of the survey, while the tools are presented in Sect. **??** and the choices of network strategy for Linux-based distributions are presented in Sect. **??**. We conclude the paper in Sect. **??** focusing on future work.

2 Tools for Evolving Paradigms

The first step towards tackling the complexity of FLOSS user-centric network management tools is conducting a survey that reflects the current state of Linux network management. We focus on two different aspects: heterogeneous network support and dynamic configurations.
From the Linux operating system perspective, it is important to see how the different technologies are handled, like WiFi, 3G or WiMax. But, heterogeneous networking goes beyond the support of multiple technologies, and implies a seamless experience, where the different technologies are integrated from the user's perspective. It is becoming increasingly important that these technologies work together, and managing how network selection is performed. Therefore, before implementing complex network solutions [?], it is necessary to determine the current state of the art, especially in FLOSS.

As a complementary aspect to network selection, we also focus on spontaneous and dynamic configurations. It is important to analyse how dynamic configurations occur, along with the benefits that they provide, especially considering wireless environments and user-centric technologies [**?**], regardless of whether infrastructure support exists. In this domain, we highlight two complementary approaches: IPv6 and Zeroconf. IPv6 has built-in mechanisms that allow automatic address configuration and peer discovery on the local link. Stateless Address Auto Configuration (SLAAC) [**?**] allows a node to generate an address for local communication in the fe80::/10 range through an EUI64 expansion of its MAC address. Peer discovery can be performed using special IPv6 multicast groups (e.g. *all-nodes*).
In IPv4, peer discovery can be achieved through Zeroconf [**?**], which is a protocol suite that aims to provide a fully functional IPv4 stack without the need for special configuration servers. It focuses on network address configuration and local name resolution, without resorting to DHCP or DNS servers. Address configuration is done through IPv4 Link-Local Addresses [**?**], which is a mechanism that enables the configuration of local addresses in a special address range (169.254.0.0/16), similar to IPv6 local link addresses. For local name resolution, Zeroconf defines the usage of Multicast DNS (mDNS) [**?**]. mDNS requires that each host stores its own DNS records (A,MX,SRV) locally, answering queries sent to a specific Multicast address. Whoever knows the answer, i.e holds the record, should respond to queries (resolve the address). This establishes a simple protocol for DNS supported communication without a central server. Using mDNS it is possible to provide service discovery on the local link through DNS Service Discover (DNS-SD) [**?**], also part of Zeroconf. Using DNS-SD, a node can join the proper mDNS multicast group and query for well known DNS records (SRV, TXT and PTR) that have service instances names, according to a *dns-sd.org* list [**?**].
In the lower layers, Ad-Hoc (802.11 Independent Basic Service Set mode, IBSS) and Mesh networking (802.11s) can provide access without centralized infrastructure, but have limited support for dynamic configurations, which usually depends on higher layer technologies. WiFi Direct [**?**] is a WiFi Alliance proposed certification that extends the Ad-Hoc support in 802.11 with better security and simultaneous WiFi network connections. It provides the means for establishing dynamic connections between 802.11-enabled peers and also, according to preliminary findings, supports peer discovery on the link layer.
By focusing on heterogeneous networking and dynamic configuration technologies it is possible to evaluate the FLOSS tools, and how they are integrated in the different distributions, which is presented over the next sections.

3 Linux Tools

The main objective of this survey is to catalogue and analyse the most important network tools in Linux-based operating systems. To provide a thorough survey that covers the different FLOSS tools and technological aspects, we must look at the configuration and management tools currently available in Linux distributions. We focus on those that gather information from user input (through configuration interfaces) and translate it into the necessary commands and operations that are understandable by the lower level daemons and applications that interact with the network, and with the Linux Kernel.

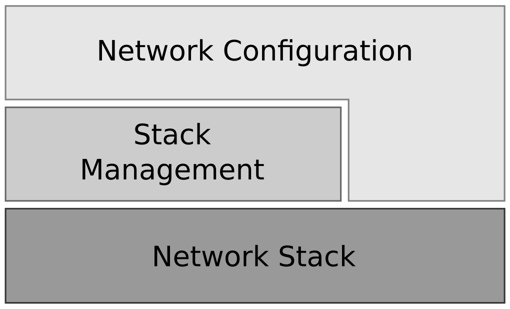

Fig. 1. Three-part network management stack.

Therefore we follow a top-down approach, as conceptually reflected in Fig. **??**, focusing on the tools and processes existing at each level. We start with the connection managers in current FLOSS systems, which define how a user configures and interacts with the *Network Configuration*. We then explore the tools required to translate the configuration towards the system, providing *Stack Management*. Finally we focus on the *Network Stack*, defining how tools interact with the network, mostly within the Linux Kernel.

3.1 Network Manager

In recent years, Network Manager [**?**] (NM), a GPLv2 project by Red Hat and Novell, has emerged as the primary network configuration application for the Linux desktop. Its main purpose is to provide a hassle free networking experience, without compromising usability. This means that the focus is on reducing the amount of manual configuration exposed to the end-user, aiming at connectivity that "just-works".
By integrating network configuration and management, it creates a central control point across the entire desktop that is tightly integrated with the operating system and applications. Its modular design, shown in Fig. **??**, includes several supported technologies, managing both wired and wireless connections. NM is split into two components: a system daemon that controls the networking infrastructure and a management application (usually graphical, e.g. network-manager-applet [**?**]) that handles user interactions. In fact, the daemon is controlled through a D-Bus [**?**] interface, a FreeDesktop [**?**] standard, allowing a flexible integration with different clients. NM architecture supports both IPv4 and

IPv6, along with several access technologies, such as WiFi, WiMax, GSM/CDMA, Mesh and even Bluetooth, as shown by its architecture. This is achieved through different sub-systems that communicate with the main daemon through various interfaces, such as D-Bus, Netlink Sockets, Unix sockets or system call wrappers. The WiFi interactions are handled through the Supplicant Manager, a D-Bus interface module for WPA Supplicant (Sect. **??**), complemented by the Linux Wireless Extensions (WEXT, **??**). 3G support is provided by Modem Manager [**?**], which supports most modern 3G devices.

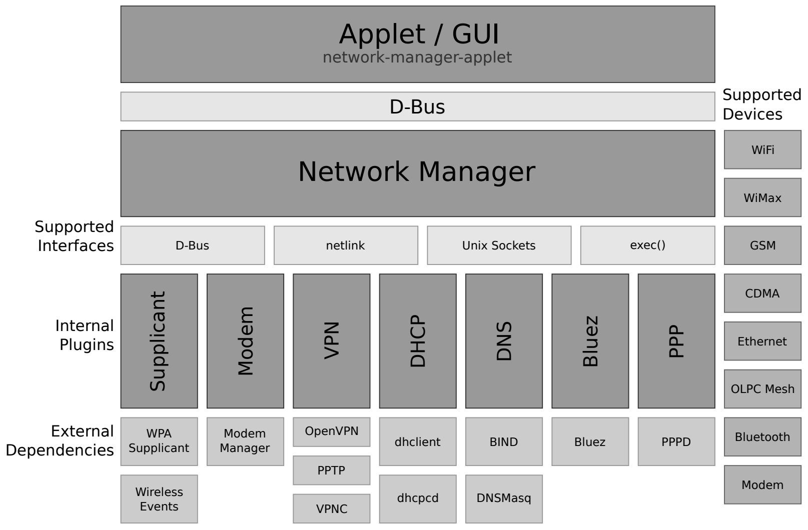

Fig. 2. Network Manager internal architecture.

Concerning network selection in Network Manager, it has a static preference list based on device types, enabling first Ethernet, WiFi, GSM, CDMA, Bluetooth, Mesh and finally WiMax. The connections, however, are timestamped and network manager will always prefer the last known active connection, when two connections of the same type exist (e.g. two wireless networks). For the actual selection of an Access Point (AP) within an Extended Service Set Identifier (ESSID), network manager relies on wpa_supplicant.

3.2 Connman

Connection Manager (ConnMan) [**?**] is a small and lightweight daemon designed for managing network connections on Linux embedded devices. It has a plug-in based design in order to build with as few components as possible, thus supporting customized configurations. Its main target is the inclusion in the MeeGo project [**?**], where it is the default network connection manager, unlike NM, which is general purpose for

every Linux-based OS. It supports most technologies through plug-ins, namely, Ethernet, Bluetooth, WiFi, UMTS and even WiMax. It also supports network protocols through plug-ins, such as DNS, DHCP an VPN connections. The WiFi subsystem is composed of the main daemon and the WiFi plug-in, which connects to wpa_supplicant through a D-Bus interface (the preferred interface type). 3G support is achieved through oFono [**?**].
For managing connection preferences it uses a connection list, with both dynamic and static preferences. Previously used networks have a favourite status that takes precedence over new connection points. However, all things being equal it prefers Ethernet, Bluetooth, GSM, UMTS, WiMax and WiFi. These two combined mechanisms can be seen as a semi-static list that guides network selection.

3.3 wpa_supplicant and hostapd

wpa_supplicant [**?**] is a GPLv2 licensed WPA supplicant that supports WPA/WPA2. It is available on most Linux-based platforms and distributions, either directly or indirectly through NM. It implements the client component (the supplicant) of WPA, negotiating the encryption keys towards the WPA Authenticator (the server counterpart in WPA), supporting the 802.11i standard and also EAP/802.1X. It also supports several wireless extensions, such as 802.11r, Fast Base Station Transition (smoother roaming process between access points), 802.11w (management frame security) and even WiFi Protected Setup (WPS), a WiFI Alliance certification that simplifies WiFi setup.
wpa_supplicant interacts with NM (and similar clients) through a D-Bus interface, which is becoming the default interface. It also features a control (Unix) socket, which is still used by several clients (e.g. Android). In Linux, wpa_supplicant supports all drivers that use the recent mac80211 [**?**] stack (Sect. **??**, drivers which support WEXT (v19+) and several older drivers/chipsets.
Besides the security functions, it can also control roaming between Base-stations with the same BSSID, given the requirements for wpa_supplicant to perform scanning and associating procedures. When active, roaming decisions follow a specific priority list: WPA/WPA2 support, privacy capability support (a beacon bit that mandates encryption), transmission rate (if signal level is similar) and finally signal level.
While wpa_supplicant implements the supplicant in WPA, hostapd [**?**] (which shares the same author and codebase, featuring similar functionality) implements the authenticator, as well as being the most common software for running an 802.11 AP in Linux.

3.4 Wireless Communication Linux Kernel

The Linux Kernel supports all of the previous tools through device drivers and protocol implementations. The focus on wireless technologies dictates that we look at the Linux Kernel Wireless subsystem [**?**], composed of several building blocks. Currently, the most important component of this wireless stack is the mac80211 [**?**] framework. It provides a SoftMAC driver approach, i.e. most 802.11 protocol implementation (frame management) is done in software (inside the Kernel) rather than on every driver or card individually. While there are several advantages to this approach, the most important is that drivers share a common 802.11 implementation, only implementing device-specific callbacks, resulting in much simpler drivers. The main features of mac80211 include support for 802.11a/b/g/n, 802.11d, 802.11s (Mesh) and 802.11r. Interestingly, roaming is outsourced to user-land applications, like wpa_supplicant.

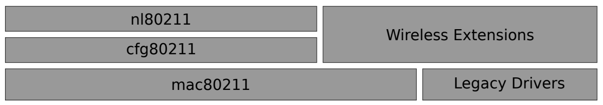

Fig. 3. Linux mac80211-based Wireless stack.

As shown in Fig. **??**, based on [**?**], mac80211 is composed of three subsystems: the mac80211 main block implements the 802.11 protocol, while cfg80211 implements 802.11 configuration and nl80211 implements the user-land communication through netlink sockets. However, as highlighted in Fig. **??**, the mac80211 system also supports Wireless Extensions (WEXT) [**?**], a legacy configuration interface that either interacts with cfg80211 or directly with the mac80211 core. As mentioned, WEXT is a legacy wireless configuration interface (only maintained, not being developed) running over IOCTL (Input/Ouput Control) calls. IOCTL have been steadily removed in favour of other transport mechanisms, such as netlink, for user-space/kernel-space communication. However, it is still used in different places (e.g. Android).
Recently, the Linux Kernel picked up initial WiFi Direct (or WiFi P2P) support (also supported in wpa_supplicant). A key issue that has surfaced in the process of proposing the P2P extensions is the need for a standardized API between connection managers and wpa_supplicant, which in turn interacts with mac80211 through nl80211.

3.5 Avahi

As discussed in Sect. **??**, one of the most important protocols in the context of local networking configuration is the Zeroconf suite. In Linux, Zeroconf is implemented by Avahi [**?**], which is a daemon that provides service discovery on the network through mDNS/DNS-SD and IPv4 address auto configuration through IPv4LL. IPv4 address configuration is done on demand, in most cases requested by NM, through a D-Bus interface. It is integrated into most Linux distributions, including embedded efforts such as OpenWRT, as presented in the next section.

4 Linux Network Stack

To understand how the tools are organized inside Linux, we must evaluate different Linux-based platforms. By examining the major distributions, it is possible to establish how most tools are organized in the Linux network stack and to determine the major trends concerning network management. The identified trends can provide insight into the best available tools, given that distributions spend a considerable integration effort and expertise towards building the appropriate network management stack and also consequently brings us closer to the goal of defining a reference Linux architecture.

4.1 Linux Distributions

Looking at the Linux distribution spectrum immediately suggests that there are several approaches towards network management. Distributions use different management tools, either scripts or applications, resulting in a uneven landscape. Here, we evaluate a select set of distributions, based on perceived importance [?]: we focus on those with most derivatives, from where tools are reused in each derivative distribution. We also focus on those with most user adoption, which helps determine the main ways in which users interact with Linux-based systems.

Fedora Fedora [?] is a user oriented distribution, a development effort sponsored by Red Hat [?]. We analysed the latest release - Fedora 14. It uses a custom tool for the most static and standard network configurations, *system-network-config*, which is part of the control panel options and provides scripts and (python) tools for static system configurations. However, NM (v0.8.1) is also included, superseding most of the functionality provided by *system-network-config*. As expected, NM is accompanied by the required wpa_supplicant for wireless management and security. Also, avahi is used and takes over all the Zeroconf aspects.

CentOS For a Red Hat Enterprise Linux (RHEL) [?] based distribution, which is a popular yet paid-for Linux distribution, we analysed CentOS [?]. CentOS is a free RHEL-based distribution, presenting an internal organization similar to Fedora, except that NM is disabled by default even though it is installed. However, NM is recommended for laptop usage [?]. In both cases, wpa_supplicant is used and avahi is running, controlling Zeroconf protocols.

Debian/Ubuntu Debian [?] is one of the major available distributions, generating many derivatives. For static configurations it uses ifupdown, a tool that implements scripts and configuration files to easily manage network interfaces. The remaining setup is similar to Fedora, where the main tool is NM (v0.6.6 in Debian Lenny and v0.8.1 in Squeeze), complemented by wpa_supplicant for wireless support. This setup is seen both in Debian and Ubuntu [?] (Maverick 10.10), and in all versions, avahi runs by default, handling Zeroconf functionality.

OpenSUSE OpenSUSE [?] and earlier SUSE systems, have historically relied on YAST for all system configurations. YAST handles most network configurations, using scripts to manage the different interfaces. In the initial interface configuration it is possible to activate NM, consequently becoming similar to the previous approaches, relying on NM and wpa_supplicant for most of the wireless interactions, and on avahi for Zeroconf.

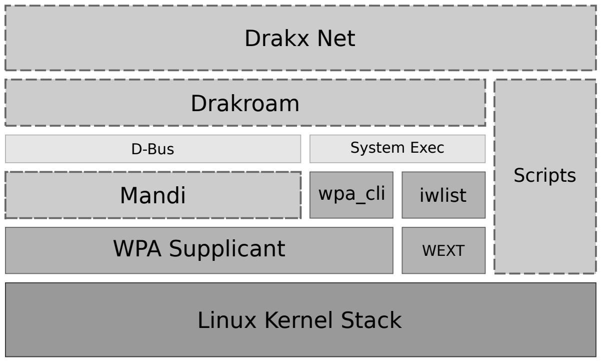

Fig. 4. Madriva and Caixa Mágica Network Management stack.

Mandriva and Caixa Mágica Mandriva [?] is the Linux distribution upon which Caixa Mágica [?] is built. We reviewed Caixa Mágica 15, as well as Mandriva 10.1 and 10.2, which share the same base. Mandriva, and consequently Caixa Mágica, do not follow the same pattern as other distributions using mostly custom tools, as seen in Fig. **??** where the Mandriva specific tools are highlighted via dashed stroke. The main networking configuration tool is Drakx-net, which is part of the Drak configuration toolset, a custom Mandriva system configuration tool. Drakx-net provides a configuration manager for networking settings, covering network interfaces and VPN.
Looking at the roaming/wireless subsystem, it is handled by Drakroam. This Mandriva developed tool is an application composed of scripts that interact with the OS, a graphical configuration interface, and an applet that provides a shortcut for network configuration with special emphasis on wireless. Drakroam uses mandi, a custom built D-Bus daemon that provides support for network configurations. It features a plug-in system, where the wireless part is an interface to the wpa_supplicant control interface. As a fall-back, Drakroam can support wpa_cli, a command line interface application provided by wpa_supplicant, and alternatively it falls back to iwlist (using WEXT). Beyond this, Caixa Mgica and Mandriva deploy Zeroconf mechanisms through the avahi daemon.

Other Distributions and Platforms While we mostly explored desktop-like distributions, it is worth considering other platforms, espe-

cially embedded devices. We analyse Android [?], which targets mobile devices, and OpenWRT, which targets embedded routers.
Android, aimed at mobile phones and embedded devices, has an approach to network management that is different from the previously discussed distributions. As shown in Fig. **??**, it uses Connectivity Manager [?], a Java connection manager, for controlling network interfaces and providing an API for applications interacting with the Android network management infrastructure. Similarly, WiFi is controlled through WiFi Manager [?], which has limited capabilities constrained by the Java exposed interface.

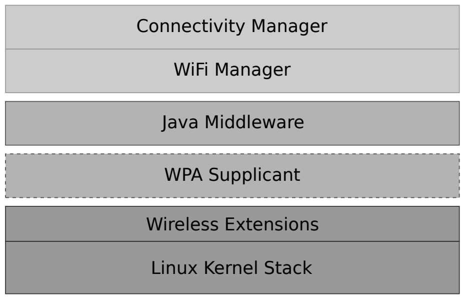

Fig. 5. Android network management stack.

WiFi functionality is supported by a modified version of wpa_supplicant that supports additional control commands specific to Android mobile devices. The middleware interactions with wpa_supplicant are done through the socket interface, given that there is no D-Bus support. However, because Android devices do not support the mac80211 stack, wpa_supplicant is limited to WEXT. Furthermore, the Android approach does not support Ad-Hoc networks [?], and lacks a fully compliant Zeroconf tool (only a Java library for mDNS [?] exists).
OpenWRT [?] uses slimmed down versions of the applications used by most distributions. Given that the main purpose of the distribution is acting as an router and AP, it supports hostapd (Sect. **??**, assuming the role of AP and WPA Authenticator with WPA/WPA2/802.11i capabilities. It supports several deployments, depending mostly on the hardware drivers to determine functionality, using the mac80211 stack as well as legacy drivers. Zeroconf can also be supported, by installing the provided packages for the avahi daemon, which can run on OpenWRT.
The static nature of the target deployment, implies that most network configurations are achieved statically through (BASH) scripts, using a flat database, the Unified Configuration Interface (UCI) module within OpenWRT, for storage.

4.2 Reference Architecture

After analysing all the different tools and distributions, an obvious pattern emerges, as shown in Table **??**. The Linux network stack, especially considering the wireless subset, is centered mostly around Network Manager, both for the graphical interface, as well as the system daemon. While there are some notable alternatives in the form of ConnMan, and some distribution specific efforts, there is a convergence within Linux towards the widespread use of Network Manager on the desktop and laptop platforms. The only noteworthy exception is Android, which uses a custom connectivity manager. Network Manager is tightly integrated in the Linux operating system, providing not only means to configure network settings, but also means for applications to determine whether an active network connection exists, as highlighted in the top-most part of Fig. **??**, which shows the reference Linux network architecture. However, as we follow down the proposed consolidated network stack, we observe a much clearer convergence across all platforms: WPA Supplicant. The WPA Supplicant daemon has become an expected presence on all Linux based operating systems, such as desktop, laptop and even handheld devices, also making an appearance on Android phones.
WPA supplicant started out by handling the security aspects of WiFi network connections, but also covers roaming between access points.

Distribution	**Network Management**	**Wireless Management**	**Wireless Stack**	**Zeroconf Support**
Fedora	system-network-config and Network Manager	wpa_supplicant	mac80211 WEXT	avahi
CentOS	system-network-config[1]	wpa_supplicant	mac80211 WEXT	avahi
Debian Ubuntu	Network Manager	wpa_supplicant	mac80211 WEXT	avahi
OpenSUSE	YAST and Network Manager[2]	wpa_supplicant	mac80211 WEXT	avahi
Mandriva Caixa Mágica	Drakx-net and Drakroam[3]	wpa_supplicant	mac80211 WEXT	avahi
Android	Connectivity Mgr. and WiFI Mgr.	wpa_supplicant	WEXT	-
OpenWRT	UCI/Scripts	wpa_supplicant	mac80211 WEXT	avahi[4]

Table 1. Tools summary per distribution.

[1] Network Manager is recommended but not installed.
[2] Network Manager is installed by default but is optional.
[3] Requires more helper applications such as mandi, wpa_cli and iwlist.
[4] Not installed by default, but available for the platform.

It integrates with all the network connection managers, such as NM, ConnMan, and even Android's Connectivity Manager.
When stepping into the actual network protocol implementations, we venture into the Linux Kernel, as depicted by the bottom-most part of Fig. **??**. The main presented focus is on WiFi, which is handled through the new mac80211 wireless subsystem. This provides support for most modern wireless cards, but also supports the legacy WEXT interfaces, kept for legacy support (of both user land tools and Kernel drivers).

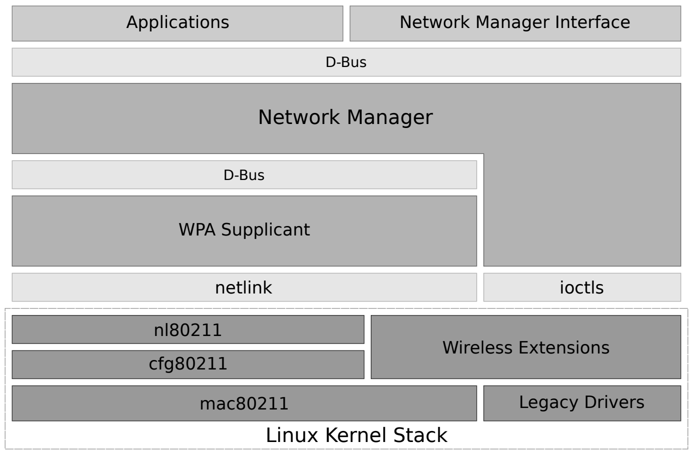

Fig. 6. Linux Network Management Reference Architecture.

Beyond the presented blocks, every distribution and system complements the network stack tools with auxiliary scripts that handle the static aspects of network configurations. While this where most differences exist, it does not represent a major divergence given that most scripts tend to be distribution specific. We also omit from the illustration the Zeroconf tools but those which are shown in Table **??**, given that the only real alternative in Linux is avahi, which already enjoys widespread deployment. When coupled with adequate Link Layer tools, it can provide a interesting effort in the self-managed network environments.

5 Overview and Future Directions

By looking at the previous sections, which are mostly summarized by Table **??**, obvious trends appeared, which led to the reference architecture presented in Fig. **??**. Using this information allows us to shift the focus to the key aspects presented in Sect. **??**, concerning heterogeneous networking and autonomous configurations as an evolution path for current network paradigms.

From the gathered results, it is possible to deduce that Linux-based systems can cope with the heterogeneous networking requirements that involve multiple technologies and dynamic environments. From a Linux perspective, most wireless technologies are already supported, with adequate tools to handle different aspects of networking (e.g. NM and wpa_supplicant).

However, there is an important gap concerning the control over network selection. As observed in the discussed tools, most network selection mechanisms imply static technology-based preference lists, simple pattern repetition (connect to last successfully used network) and simple network information (e.g. signal strength). When available networks and technologies become abundant, most of which might even be new to the user, these selection and configuration mechanisms become insufficient and must be improved. Therefore, it is important to increase the flexibility of existing control structure for network attachment. This can be achieved using two complementary approaches: 1) provide a flexible interactive API, exposed by the modules that directly control the network selection (e.g. wpa_supplicant or NM); and 2) introduce a component that collects the options coming from the different technologies, and provides consistent and reliable network selection decisions, which could be a part of NM, or even an on-demand external dependency that depends on the deployment scenario.

Concerning the autonomous configuration mechanisms and technologies that can work without infrastructure support, we observed that Linux already has a strong Zeroconf support, along with IPv6 and even WiFi Direct. This places FLOSS as a front-runner when considering these types of technologies. Avahi is already distributed with most Linux-based systems, and the Linux Kernel already support most modern technologies. Therefore, we can conclude that most tools are in place, leaving FLOSS in a good position to increase the integration of these technologies in the OS. What is missing now is the widespread adoption of these techniques along with extensions that enable us to integrate them in different applications. This is where FLOSS has the upper hand: through open interfaces and a collaborative model, it is possible to develop and integrate adoptable interfaces. This provides applications with a potential agility to quickly take advantage of the discussed autonomous configuration mechanisms, in different scenarios.

Following the open source model, this allows for an uncomplicated API for information sharing across different applications. The consequence is that, instead of being bound by standards, FLOSS can use them as a launchpad towards innovative efforts, taking advantage of local loop technologies and promoting research through extensions/tools that benefit the end user.

Lastly, it is worth mentioning that as convergence on the network management architecture occurs, the disadvantages of FLOSS development model get diluted, which is what we have observed and potentially contributed to. Right now, most fragmentation occurs only in the static configuration scripts, which are a matter of preference, style and legacy for each distribution.

6 Conclusions

Throughout the presented survey, we attempted to explore FLOSS technologies in the light of current and future technologies, by undertaking the effort of investigating the current state of the art Linux tools and distributions. The result was the proposed reference network architecture, that defines the baseline for the Linux network stack, highlights the strengths and gaps of current approaches.
Using this reference stack it is possible to outline different approaches that enable FLOSS to tackle the new networking environments, and also understand what can be can expected from the current Linux networking landscape. More importantly, by relating the current state with emerging technologies we have uncovered an opportunity for proposing future developments, contributing to the usefulness of open source operating systems in light of new technologies.
The two most important conclusions revolved around the need to improve Linux network selection mechanisms, in order to tackle dynamic and mobile heterogeneous environments, and also identified that it is necessary to place a strong emphasis on providing new and innovative services that use autonomous technologies, which are already available in Linux.
Nevertheless, the main contribution of a reference architecture and future evolution path, is that it enables the reduction of divergence and repeated work in FLOSS. This can increase the traction of existing technologies, highlighting the potential advantages of the FLOSS model in light of future research and development activities.

Acknowledgements

The work described in this paper was done in the scope of IST FP7 ULOOP STREP Project. ULOOP receives funding from the European Community's Seventh Framework Programme, under contract agreement n°257418. The views expressed in this publication are those of the authors and do not necessarily reflect the project or the European Commission's view on the subject.