Client based auto-configuration in heterogeneous networks

Tobias Roth, Torsten Braun, Matthias Scheidegger

Institute of Computer Science and Applied Mathematics, University of Bern
E-mail: {roth, braun, mscheid}@iam.unibe.ch

August 24, 2005

Abstract

This paper describes an extension to the FreeBSD start-up system called profiles*. It is primarily targeted at mobile nodes and allows them to auto-detect in what network they are located and to configure themselves accordingly. In case no known network is detected, a fallback configuration will be used. Each network to be used needs an initial, manual configuration. The network then gets auto-detected whenever the node is powered up or brought back from a power-saving state.*

1 Introduction

Service discovery is a generic term for methods that allow two or more independent devices to talk to each other and to use the services the others provide. This has to happen without external configuration, in a purely ad-hoc manner, and is therefore primarily targeted at mobile devices such as laptops or personal digital assistants. However, often when a laptop is being used, it is located in a well-known network, for example at home or at the workplace. It is thus desirable to have the laptop probe for these known networks at start-up, and to configure itself automatically if it finds one. This helps to minimise start-up times by only deploying service discovery when the laptop is started up in an unknown location, i.e. to use service discovery only as a fallback solution.

Related work can be divided into two categories. On one side, there are complete service discovery solutions that handle acquisition of IP connectivity as well as service discovery and service promotion. On the other side, there are many small tools that allow laptops to automatically discover when they are started up in a known network and acquire IP connectivity. Most of these tools deal with configuration of network parameters only, and do not do service configuration or discovery. We can therefore define the task carried out by the latter category as *laptop multihoming*, while the former category does *service discovery*. The presented work combines both realms.

Since GNU/Linux[1] distributions differ highly in the way they manage the boot process, the task of laptop multihoming cannot be solved in a generic way for GNU/Linux systems. In the BSD family of operating systems, FreeBSD[2] and NetBSD[3] provide a similar and advanced way of managing system start-up. This was the reason to use FreeBSD as the reference platform for the implementation in this work, while trying to be as portable as possible, so that adaptation to other Unix derivatives and different GNU/Linux distributions can be achieved with a minimum of work.

The requirements of an auto-configuration system for laptops were identified as follows. After an initial one-time configuration, where the characteristics of different networks are defined, the system must be able to automatically recognise in which of these pre-configured networks it is started up. Changes within the visited networks are not required, which means the configuration is completely client-side. If a known network is detected, the laptop may use services offered by the network, and it may offer services itself. For example, the laptop may use a proxy server provided by the network, and may export a portion of its own filesystem via NFS. Service discovery may take place when no known network is detected, i.e. when the laptop is started in an unknown environment. Other requirements are:

- The functionality of booting from a read-only medium must not be disrupted. Therefore, no files may be written to the boot device during booting.
- Start-up has to be fully automatic. A sort of boot-menu, where the user decides what the active profile shall be, is not desired.
- Low redundancy: Only overrides of the default configuration need to be specified for each network.
- Hooks must be present allowing third-party applications to configure themselves based on the current location.
- All available communication media must be supported. If the laptop has both an ethernet and a wireless network interface, both interfaces must be considered during network probes.

The set of auto-configured parameters can have a broad spectrum. A minimal configuration could define nothing but basic network connectivity. Extensions such as proxy settings, NFS mounts or exports, the printer environment, firewall settings and many others are imaginable.

2 Related work

As already mentioned, related work can be grouped into two categories: Solutions for service discovery and solutions for laptop multihoming. This section gives an overview over representatives of both categories.

The IETF zeroconf[4] protocol attempts to standardise service discovery. The most widely used implementation of this protocol is Bonjour[5] from Apple. It implements both acquisition of a routable IP address and the actual discovery and promotion of available services. It could be combined with the presented *profiles* system by allowing a system to fall back to Bonjour in case no known network can be detected.

The following solutions all solve the task of laptop multihoming, i.e. they take care of the acquisition of a routable IP address, but do not perform discovery of offered services. Two systems for FreeBSD and two systems for Debian Linux[6] are presented.

Setnetparm[7] for FreeBSD offers a boot-menu to select the current location. It supports adjusting of the basic networking parameters, everything else location-specific must be handled through external shell scripts. While automatic detection of the environment may be added relatively painless, the limiting factor of this approach is that it was meant to adjust networking parameters only.

The location.sh[8] script for FreeBSD can automatically detect in what environment the laptop is located. When it finds a known location, it creates symlinks for all relevant files under */etc* that will point to location-specific files. The drawbacks of location.sh are that

the script has to write to /etc at every boot, that the original files have to be restored when the script is to be removed from the system, and that changes to /etc may require changes to location.sh. Also, there is no simple way to use files from /etc as default for some networks and override them for others.

Whereami[9] for Debian Linux is extremely flexible and can do everything *profiles* can do. This flexibility, however, is achieved through complexity. The actions that are taken when a certain location is detected are all defined by separate scripts. This approach, though, makes whereami relatively stable against distribution changes.

Guessnet[10] for Debian Linux consists mainly of code used to auto-detect known networks. It runs with a very simple configuration file, much like that of the *profiles* script. It can be integrated with Debians *ifupdown* facility to reach the same flexibility as whereami. System integration is better than with whereami, yet the complexity is almost the same.

3 System Environment

3.1 The FreeBSD 5 boot process

To understand the design of the presented work, it is necessary to understand the operation of the FreeBSD boot process. This section gives a short description of the *rcng* start-up mechanism, as it was introduced in FreeBSD 5.0. The mechanism was adopted from NetBSD and is described in [11] in more detail.

1. *init(8)* is the last stage of the boot process. It calls *rc(8)* and if that call succeeds, *init(8)* starts multi-user operation.
2. rc(8) invokes *rcorder(8)* to find out in what order to execute the scripts located in */etc/rc.d/*.
3. rcorder(8) computes this order by looking at specific tags at the beginning of each start-up script.

Every service and operation has its own script in /etc/rc.d/. There is a script for filesystem checks, one for configuring interfaces, one for starting *sshd(8)* and so on. The script for sshd(8) for example can contain tags that indicate that sshd(8) should not be run before the network interfaces are configured. This allows to define an order in which the rc.d scripts are being executed.

3.2 Union mounts

Next to this ordering of start-up scripts, *union mounting* is the second requirement for the presented approach. When a filesystem is union-mounted, the new name-space at the mount point is the union between the mounted filesystem and the original mount point. This has the effect of a two-layer filesystem. Reads are first conducted in the upper layer (the mounted filesystem). The lower layer is only accessed when the upper layer cannot satisfy a read request for a specific file. All writes are made to the upper layer (though this is of no importance for the implementation of *profiles*).

4 Design and Operation

When an rc.d script is inserted into the boot sequence, this script has control over the actions of all rc.d scripts executed later in the boot process. Insertion at a specific place in the rcorder chain can be achieved by putting the right tags at the beginning of the script. The script is run after the filesystem checks are done, but before the root filesystem is remounted in read/write mode. This is the earliest possible moment for executing the *profile* script without risking filesystem corruption. While placing it before the filesystem

check script would be possible (and was done in early versions of *profiles*), it was decided that executing the filesystem checks independently from the chosen profile will avoid possible problems.

The operation of the script is as follows: It reads from */etc/rc.conf* which profiles have to be probed for and in what order. Each profile consists of a file that contains a file system inside. This filesystem can be mounted, and it contains a set of files that are specific for this profile. When probing for the profile, the filesystem is union-mounted over the standard */etc* directory, allowing the profile-specific set of files override those inside the standard /etc. For example, if one profile wants to override the name-servers used, it will contain a resolv.conf file, which will override the resolv.conf file from the standard /etc directory.

An rc.conf file is mandatory for every profile, because it contains instructions on how to recognise the network environment. These instructions contain which interface to use, interface specific configuration parameters (for example the media type or the ssid for wireless networks) and the test method. The test method can be DHCP, ping or arp. With DHCP, the *profile* script tries to acquire a valid DHCP lease. With ping, it tries to ping the IP address of a known host (which is also defined in profile-specific rc.conf file). With arp, the arp cache is inspected to see if the known host is present on the network but not sending back echo replies. The test methods can also be chained. For instance, to distinguish between two different DHCP based networks, a ping test can be conducted after the DHCP test was successful. Arp tests can be used to distinguish between two networks with the same IP space and the same known host IP address. This flexibility allows to handle complex setups, such as IPsec protected wireless ad-hoc networks.

If the network probes for one profile are unsuccessful, the profile-specific union-mount over /etc is unmounted and the next profile is tried. If a profile is successfully probed for, the union-mount is not unmounted and the boot process continues as usual, allowing the profile-specific files to override their counterparts in /etc and affect the remainder of the boot process. This also has the benefit that for each profile, only the files that should be overridden need to be present, everything else is taken from the standard /etc directory. If no profile matches the current environment, booting continues without a union-mount in place, allowing the standard /etc directory to act as a fallback or default profile.

The presented approach is not only suited to manage system start-up, but also suspend/resume cycles commonly used with laptops. A system can be suspended, disconnected from the network, carried into a different environment, reconnected and resumed. Upon suspension, the *profile* script uses rcorder to compute a list of rc.d start-up scripts. Since each start-up script also supports a stop argument, the *profile* script can go through the computed list in reverse order and call each script with the stop argument. When the laptop is resumed, the *profile* script acts the same as when the system is started, i.e. it checks in what network it is located and lets the system execute the remaining rc.d scripts. All scripts that are not dependent on the current location are tagged with a special keyword that prevents stopping and restarting during the suspend and resume cycle. With this approach, location changes during laptop suspension can be detected when the system resumes.

5 Porting

In this section, the possibility of porting *profiles* to other free unix-like operating systems, namely GNU/Linux, NetBSD and

OpenBSD[12] is discussed.

A port to another operating system has two requirements: The target system must have an ordered sequence in its start-up scripts and it must support a form of union mounts. So far, only NetBSD satisfies both dependencies natively. OpenBSD offers union mounts but lacks the sophisticated start-up scheme of NetBSD and FreeBSD. A port to NetBSD could thus be done with only slight modifications, while for a port to OpenBSD, more work would be necessary to compensate for the deficiencies in its start-up mechanism.

For GNU/Linux, the situation is a bit different. When the *profiles* project was started, the kernels available for GNU (Linux 2.6 and GNU Hurd[13]) did not support union mounts natively. All projects that add union mount support to Linux were in alpha stage or were said to be nothing more that a proof of concept. Today, there is still no native support for union mounts in the Linux kernel, however the status of the available add-ons has improved quite a bit. The following two projects that enhance the Linux kernel with union mounts could be considered for use in a port of *profiles* to GNU/Linux: Unionfs[14] and the Cowloop driver[15]. Still in proof of concept state is the Translucency loadable kernel module[16].

GNU distributions all differ in their start-up scheme. Some use a SystemV style start-up mechanism (Debian, Fedora[17]), some use OpenBSD-like rc scripts (Slackware[18]), and some go towards a mix of SystemV and NetBSD rcng (Gentoo[19]). Configuration of network services also differs between each distribution. This diversity makes it necessary to look at each distribution separately when attempting to port *profiles* to GNU/Linux.

6 Conclusion

The presented approach provides an improvement over pure service discovery by making use of information about frequently used network environments. Since users tend to use the same small set of networks most of the time, and since the basic network properties of most networks don't change very often, it is beneficial to configure this information statically, in order to speed up and automate network configuration. Service discovery is then used only in unknown networks.

An aspect completely ignored in the proposed solution is security. It is relatively easy for a hostile network to intercept and analyse network probes, and then to act as a different network. This may lead to a situation where the mobile node is lowering his defences and offering private services to the wrong network. If all friendly networks would authenticate themselves to the mobile node, this weakness could be avoided. The mobile node would then maintain a high security profile in all unauthenticated networks. However, this means that a change to the network is necessary, which would break one of the original requirements of this work.

Work is currently ongoing in integrating *profiles* into the FreeBSD base system.

References

[1] Rémy Card, Èric Dumas, Frank Mével, "The Linux Kernel book", *Wiley & Sons*, Jun. 1998.

[2] Marshall Kirk McKusick, George V. Neville-Neil, "The Design and Implementation of the FreeBSD Operating System", *Addison-Wesley*, Aug. 2004.

[3] Luke Mewburn, "NetBSD: Platform for the future", *BSDcon 2000*, Oct. 2000.

[4] Erik Guttman, "Autoconfiguration for IP Networking: Enabling Local Communica-

tion", *IEEE Internet Computing Magazine*, Jun. 2001.

[5] P.J. Connolly, "Say 'howdy' to Bonjour", *InfoWorld Media Group Magazine, Issue 23*, Jun. 2005.

[6] J.H.M. Dassen, Chuck Stickelman et al., "The Debian GNU/Linux FAQ", *Free Software Foundation*, 1996.

[7] Hellmuth Michaelis, "The setnetparm utility for FreeBSD", [Online, visited May 2005], Available: *http://www.kts.org/hm/download/setnetparm*

[8] Lars Eggert, "The location.sh script", [Online, visited May 2005], Available: *http://www.isi.edu/larse/etc.html*

[9] Andrew McMillan, "Whereami network autoconfiguration utility", [Online, visited May 2005], Available: *http://debiana.net/whereami*

[10] Enrico Zini, "Guessnet Resources", [Online, visited May 2005], Available: *http://guessnet.alioth.debian.org/*

[11] Luke Mewburn, "The design and implementation of the NetBSD rc.d system", in *Proceedings of the USENIX 2001 Annual Technical Conference*, Jun. 2001.

[12] Michael W. Lucas, "Absolute OpenBSD: UNIX for the Practical Paranoid", *No Starch Press*, Jun. 2003

[13] Marcus Brinkmann, "The GNU Hurd", *UKUUG Linux Developers' Conference*, Jul. 2002.

[14] C. P. Wright and E. Zadok, "Unionfs: Bringing File Systems Together", *Linux Journal, Issue 128*, Dec. 2004.

[15] Gerlof Langeveld, "Copy-on-write loop driver", [Online, visited May 2005], Available: *http://directory.fsf.org/network/security/cowloop.html*

[16] Dirk von Suchodoletz, "Jump Starting the Network", *Linux Magazine, Issue 27*, Feb. 2003.

[17] Paul Hudson, Andrew Hudson, Bill Ball, Hoyt Duff, "Red Hat Fedora 4 Unleashed" , *Sams*, Jun. 2005.

[18] David Cantrell, Chris Lomens, Michele Membrila, "Slackware Linux Essentials", *Walnut Creek*, Jul. 2000.

[19] Christopher Negus, "Linux Bible", *Wiley & Sons*, Jan. 2005.