ADAM: Administration and Deployment of Adhoc Mesh networks

Thomas Staub*, Simon Morgenthaler*, Daniel Balsiger*, Paul Kim Goode* and Torsten Braun*
*Institute of Computer Science and Applied Mathematics
University of Bern
Neubrückstrasse 10
CH-3012 Bern, Switzerland
Email: {staub|morgenthaler|balsiger|goode|braun}@iam.unibe.ch

***Abstract*—Costly on-site node repairs in wireless mesh networks (WMNs) can be required due to misconfiguration, corrupt software updates, or unavailability during updates. We propose ADAM as a novel management framework that guarantees accessibility of individual nodes in these situations. ADAM uses a decentralised distribution mechanism and self-healing mechanisms for safe configuration and software updates. In order to implement the ADAM management and self-healing mechanisms, an easy-to-learn and extendable build system for a small footprint embedded Linux distribution for WMNs has been developed. The paper presents the ADAM concept, the build system for the Linux distribution and the management architecture.**

***Keywords*-wireless mesh networks; software distribution; network configuration; network management**

I. Introduction

Wireless Mesh Networks (WMNs) are one of the key technologies to provide ubiquitous network access to end users and sensing equipment. They offer a cost-efficient last-mile access network to cover wide areas with broadband network services. Existing deployments are either related to research such as MIT Roofnet [1], Berlin RoofNet [2], Heraklion MESH [3], WiLDNet [4] and QuRiNet [5] or provide public network services in a metropolitan area such as community networks. Most of these networks cover large geographical areas and include node locations that are difficult to reach, e.g., roof tops. In addition, they may be deployed in hostile environments such as deserts, mountain or arctic regions. Physical access to certain node sites may be very restricted or even impossible at all due to administrative or technical reasons. In general, all on-site repairs are time-consuming and costly. Therefore, their number should be minimised.

During the network lifetime, reconfiguration and software updates are necessary in any WMN. Unfortunately, faulty reconfigurations and software updates may disrupt the nodes' network connectivity and then manual on-site repairs are required. The three main reasons for on-site repairs are modified network parameters, corrupt software updates, and nodes that become unavailable during the processing of updates. First, modifications of network parameters, especially the radio parameters such as reducing transmission power or changing the wireless channel, may drastically impact the network topology or even cause disconnection of some nodes from the network. Second, a corrupt software update may prevent a node from working correctly. Third, some nodes may be temporary unavailable during the reconfiguration or software update distribution. Afterwards, they may not able to integrate themselves into the network due to modifications missed during their disconnection from the network. Examples are solar-powered nodes with drained batteries or transmission difficulties due to special weather conditions. Without any self-healing mechanism providing an automatic recovery, costly physical access is required to repair these disconnected nodes.

Our contribution is a novel management framework for WMNs called Administration and Deployment of Adhoc Networks (ADAM) that avoids the described on-site node repairs caused by configuration errors, faulty software updates, missed configuration updates and environmental changes. ADAM improves accessibility of individual nodes in any circumstance. Its management architecture is based on decentralised distribution of software images and network configurations as well as on self-healing mechanisms. Besides the main contribution, which is a flexible and extensible framework to set up and maintain a heterogeneous WMN by safe reconfigurations and software updates, the ADAM framework provides a simple, intuitive build system for an embedded Linux distribution. This build system automates all steps required to compile and generate a Linux distribution optimised for WMNs from source archives.

The remainder of the paper is organised as follows. In Section II, different build systems for embedded devices and management architectures for WMNs are discussed. Section III presents the core concepts of the ADAM build and management architecture. Section IV describes the ADAM build system. Section V presents the implementation of management architecture. We conclude with Section VI.

II. Related Work

Linux is often used as the operating system for embedded systems such as wireless mesh nodes. Several different distributions and build systems tailored for embedded systems exist. In order to be suitable for our ADAM framework, the build system has to fulfil the following requirements:

978-1-4577-0351-5/11/$26.00

- Seamless integration of management functionalities incl. splitting binaries and configuration
- Support for multiple CPU architectures
- System with a small memory footprint
- Similar software on all supported platforms
- Easily extendable for additional software.

In the following, three existing cross compilation build systems are discussed. The OpenWrt [6] Linux distribution is tailored for embedded devices. It uses the small C library replacement μClibc to reduce the footprint of the compiled software, which is also used by ADAM. OpenWrt uses a package manager based approach for software installation on the nodes. This provides a high flexibility for customisation of individual nodes with existing packages, but requires a read/write file system on the secondary storage. Often, more RAM memory than secondary storage is available on inexpensive wireless mesh nodes (e.g., OpenMesh OM1P with 32 MB RAM and 8 MB flash storage). In this case, a compressed read-only software image can hold more software and, therefore, provide more functionality. In contrast to the targeted decentralised distribution approach, packages can only be retrieved from a central instance, to which each node has to connect during the update. There is no support to get cached updates from neighbours. Another drawback of the package manager based approach is that binaries and configuration data are combined in one single package. It would, therefore, require a major effort to adapt OpenWrt to split software and configuration images for ADAM.

OpenEmbedded (OE) [7] is a collection of recipes for the BitBake [8] tool to automatically compile and install packages for an embedded Linux system. In OE, the customisation of the compilation and installation process is highly flexible. A separation of binaries and configuration data could therefore be implemented. Unfortunately, OE is difficult to understand due to its complexity. Another drawback is the poor support of μClibc for build compact software images. Ångström [9] is a user-friendly OE distribution. OE also shares the drawbacks of package manager based software installation of OpenWrt.

A different approach for building a Linux system is the manual from Linux From Scratch (LFS) [10]. It provides step-by-step instructions to manually build a Linux system from the available sources. Cross compilation aspects are handled by the subproject Cross Linux From Scratch (CLFS). Being a collection of documentation, its major disadvantage is the missing automated build process. Nevertheless, instructions found in CFLS helped in the development of the ADAM build system.

The main aspect of ADAM is management of heterogeneous WMNs. Besides Simple Network Management Protocol (SNMP), CAPWAP (RFC 5414, RFC 5416) and Broadband Forum's TR-069, existing management approaches tailored for WMNs include MAYA [11], JANUS [12], DAMON [13], ATMA [14], MeshMan [15] and Abaré [16]. MAYA is based on OpenWrt and AODV routing. It provides mechanisms to configure multiple selected nodes either over remote secure shell or by sending an encrypted UDP packet. It relies on a working routing protocol and cannot handle nodes off line during configuration time, which is possible with ADAM. ATMA is a management framework for wireless test beds. It deploys a parallel multi-hop WMN to provide out-of-band centralised management of the actual testbed. Although, this might be a reasonable approach for test bed management, it is not suitable for productive networks. JANUS is a fully distributed agent based monitoring architecture using a p2p overlay network for communication. Besides missing management capabilities, the JAVA-based implementation cannot be run on resource-restricted devices. Meshman is a management architecture providing an SNMP replacement that considers network dynamics in WMNs. By combining source routing and a hierarchical address scheme, it is independent of the routing scheme used. The current implementation only provides information retrieval, but no configuration. Abaré provides a software-assisted process for installation and management of a WMN. It is based on a central database to co-ordinate the management. Firmware is delivered individually to each node. Compared to ADAM, it cannot cope with misconfigured nodes.

In ADAM, we have solved the drawbacks of the predecessor architecture "Secure Remote Management and Software Distribution for Wireless Mesh Networks" SRM [17]. SRM requires some fixed network parameters, e.g., ESSID, and externally synchronised clocks. Moreover, only a small set of network parameters can be configured. The build system of SRM is limited to x86 compatible nodes and does not support cross compilation. In contrast, ADAM provides a complete cross-compilation build system and increased flexibility in management with a more modular approach including full IPv6 support and configuration of network services.

III. ADAM: Concept and Architecture

The three main concepts of ADAM are:

- Decentralised distribution of software images and network configurations
- Self-healing mechanisms
- Separation of node specific configuration and binary software images that are specific for a node type

The first main concept of ADAM is a decentralised mechanism for distributing software and configuration updates. Each node periodically (every 2 min) pulls new software or configuration updates from its one-hop neighbours. Updates are therefore propagated from one node to the other throughout the network. This update mechanism works independent of the routing protocol used. If a node is not reachable during the reconfiguration, it fetches the updates when it is up again. If the gathered updates target the node, they are automatically applied. The successful application and

network connectivity are supported by self-healing mechanisms, which are the second main concept of ADAM.

The self-healing abilities of ADAM are manifold. They include monitoring of the network topology during updates, detection of isolated nodes, and automatic rollback to the latest running software if a software update fails to boot properly. Monitoring of the network topology and appropriate reaction is the first self-healing mechanism. If the network parameters that may disrupt network connectivity, such as a lower transmission power, are modified, self-healing mechanisms recover the network connectivity if necessary. For example, if the self-healing mechanism discovers a reduced number of neighbours after lowering the transmission power, it step-wisely increases the transmission in order to reach at least predefined network connectivity. The detection of isolated nodes is the second self-healing mechanism. It supports that temporarily unavailable nodes can be reintegrated into the network, even if the network configuration has completely changed during their absence. Isolated nodes may discover their state and follow an automatic lost node procedure for re-joining the network. The final self-healing mechanism takes care of faulty software updates. Although ADAM uses checksums to detect data corruption and to guarantee error-free transmission of updates, the configured software updates may contain errors that prevent the nodes from properly booting after the update. If such errors occur, an automatic rollback process is started. The node then automatically reboots and loads the latest known working software.

ADAM separates software and configuration data on a node to exploit similarities between the nodes and reduce the amount of transferred data in a network. It is not efficient to just distribute a software image for each individual node. Most software such as the operating system kernel and binaries for tools and applications are the same for similar types of nodes. Therefore, each node in an ADAM network contains two image files. One image file holds the operating system kernel and the binaries. This image is the same for all nodes of a similar type. The other image just holds all the node specific parts. These are mainly configuration files, which can vary for individual nodes. ADAM even splits up this configuration image into the normal configuration files and a special network configuration file. This network configuration file holds all dynamic network parameters, from which the normal configuration files are automatically generated. Therefore, ADAM must usually only distribute this network configuration file with a size of 10 KB for each node and the software image per node type (<6 MB). This drastically reduces the total amount of transferred data for an update.

IV. ADAM: Build System

No existing build system for an embedded Linux distribution (e.g., OpenWrt, OE or CFLS) properly supports all requirements for ADAM, e.g., splitting binaries and configuration. As none was suitable for the implementation of ADAM management approach, we decided to develop an own easy to understand build system based on the documentation of CFLS and some patches from OpenWrt.

The ADAM build system is especially tailored for WMNs and supports several target platforms. To prove heterogeneity support, we currently use nodes from three vendors, namely PCEngines ALIX and WRAP embedded boards, Open-Mesh Mini and OM1P, and Meraki Mini. The nodes differ in their processor architecture (x86 compatible, MIPS), their amount of RAM (32 - 256 MB) and secondary storage (8 MB - 4 GB). Despite these significant differences, all nodes provide similar functionality of installed utilities and software to the user. ADAM provides a build system that produces software and configuration images for different node types. The operating system is a fully customised embedded Linux. It offers all key functionalities for a WMN node within a small memory footprint ($<$ 6 MB), which is a key factor for the deployment on embedded systems. In order to achieve this small footprint, the μClibc that requires only 400 KB of storage replaces the standard C library and BusyBox replaces the standard UNIX tools (e.g., sh, cp, mv, grep, sed, and awk), saving more than 4 MB of storage. In contrast to OpenWrt, the Linux kernel and the binaries are stored in a read-only compressed image on secondary storage, which is decompressed to the RAM during run-time. This results in up to 6 MB additional software packed on the OM1P (8 MB secondary storage) compared to running OpenWrt with a file system on the secondary storage.

The goal of the ADAM build system is to simplify all necessary steps for image creation. It avoids a steep learning curve for new users by focusing on functionalities used in WMNs. It provides a simple and intuitive command line interface. It is easily extendable with additional software packages as well as to other hardware platforms by integrating new build profiles. Moreover, the software requirements of ADAM are moderate. A standard desktop machine with a current Linux distribution (Fedora, Ubuntu, Debian, Gentoo) providing the ordinary development and build tools such as the GNU compiler collection (gcc) and its standard development tools is sufficient for using ADAM.

The ADAM build system consists of two tools, namely the *build-tool* to compile the software and the *image-tool* to pack the software correctly into the images. Figure 1 illustrates the necessary steps to build a Linux distribution for an ADAM mesh node. In step 1, after installation of the ADAM build system, the set-up procedure is started by the *build-tool*. It creates a build environment for the target platform by adding a user on the local machine. The command shell environment of the new user, e.g., alix-builder, is set up with all necessary parameters for the cross compilation process, such as library and compiler paths. The parameters are defined in the build profile of the selected target platform.

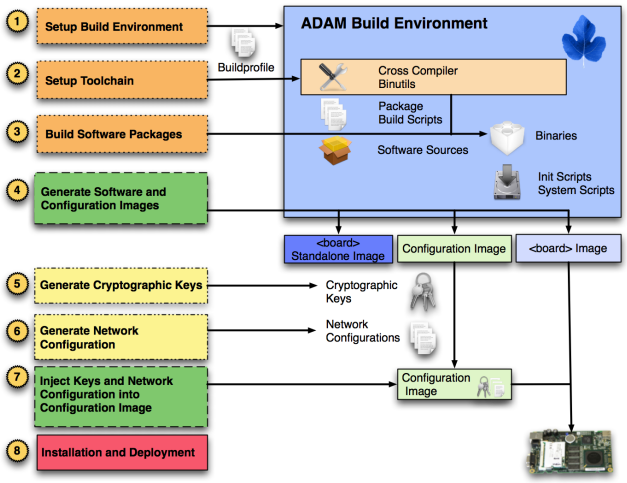

Figure 1. Steps of the build and setup process for a node.

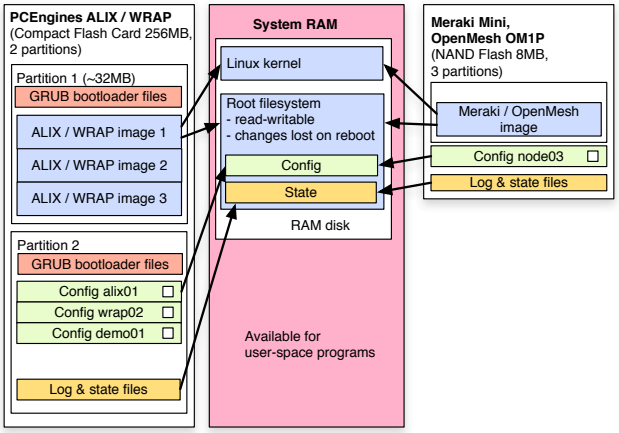

Figure 2. Run time layout of system RAM and the secondary storage for PCEngines ALIX/WRAP, Meraki Mini and OpenMesh OM1P nodes.

In step 2, the tool-chain for the cross compilation is set up and installed for the user. The operating system headers are installed. Then, machine-specific Executable and Linkable Format (ELF) binary tools, the intermediate cross compiler, the C library μClibc for the target platform and the final cross compiler are compiled and installed one after the other. The final cross compiler is used to compile all software packages for the target platform in step 3. An individual software package in ADAM is defined as a recipe for compilation and installation. It is implemented as simple shell script. This package script downloads the particular package source archive, decompresses it, applies necessary patches, configures it for cross compiling and installs the binaries and configuration files to the correct directories after successful compilation. In step 4, the *image-tool* is used to generate the software image for the target platform and individual configuration images for each node. In the steps 5 and 6, cryptographic key pairs for the distribution engine and the network configuration for each node are generated. The node-specific keys and the network configurations are then injected into the configuration image of the corresponding node in step 7. In the final step, the generated Linux system images are loaded on the secondary storage of the new nodes or distributed using the ADAM distribution engine.

Besides software and configuration image, the *image-tool* creates another image type - the stand-alone image. This specific image is fully self-contained and does not require any configuration image. In ADAM, it is only used for test purposes and the installation of normal software images on the secondary storage of Meraki and OpenMesh nodes. The boot loader of these nodes does not support writing files larger than 5 MB to the secondary storage. Therefore, a stand-alone image is booted over the network to write the software images under the temporary Linux system.

Figure 2 shows the run time memory layout of ADAM nodes. Depending on the platform, a node can store multiple software and configuration images. During run time, a software image and a configuration image are mapped to a root file system. As the state, such as random seeds and log files, should not be lost, when software or configuration images are exchanged, it is stored in a special permanent storage on the node, which is also mapped to the root file system in the RAM.

Figure 3 illustrates the boot process of an ADAM node. After being switched on, the boot loader reads the boot configuration and then loads the Linux kernel and the initial RAM based file system from the software image. During OS initialisation, the kernel then loads the root file system. After initialisation, the content of configuration image is mapped on top of the root file system, the permanent storage with the node's state and log files is mounted in the system.

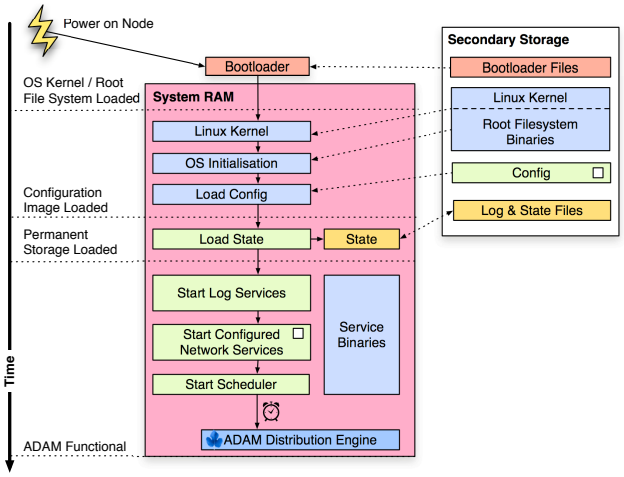

Figure 3. Detailed boot process.

The Linux system applies the network configuration, starts the configured system services including the time-based job scheduler. The system is now fully functional and the job scheduler periodically starts the ADAM distribution engine.

V. ADAM: Management Operation

After initial installation, each node holds a software image and a configuration image with the initial network configuration. The node is physically deployed at the final location (e.g., on a rooftop). Henceforth, physical access to the node may be costly or difficult. The node should therefore be completely managed from remote by the ADAM configuration framework. The configuration framework is a complete redesign using some basic ideas from SRM [17].

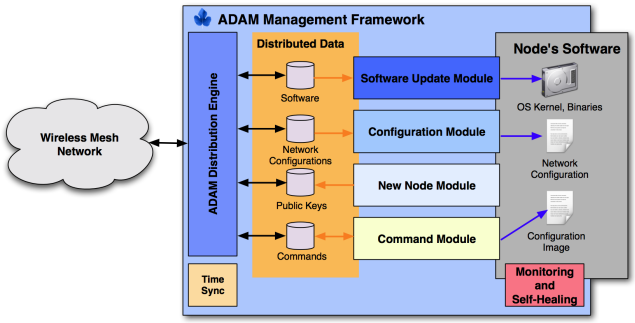

Figure 4. General ADAM management architecture.

Figure 4 shows the general ADAM management architecture. It consists of the ADAM distribution engine and modules for network configuration, integration of new nodes, software update and a generic command module. The modules are described in the following subsections.

A. ADAM Distribution Engine

The ADAM distribution engine for configuration and software updates is based on *cfengine* [18], which is a powerful utility for organising and distributing system administration tasks. The communication is encrypted using separate public/private key pairs for each node. This guarantees that only authorised nodes participate in the network, but requires time synchronisation. The decentralised distribution approach of ADAM can cope with nodes that are unreachable during configuration. Each node periodically starts reachable peers detection, synchronises its clock, connects to all its detected one-hop neighbours and checks the availability of newer network configurations or software images. If there are updates available, they are pulled by the node. This epidemic distribution works without a configured routing protocol. In order to be independent of other network configuration settings, the ADAM distribution engine communicates over a dedicated IPv6 network. The dedicated IPv6 management network is always present and cannot be switched off.

Upon reception of new network configurations or new software, the node automatically applies them using the configuration and the software update module, if it is the target. Otherwise, the files just remain in the exchange storage.

B. Configuration Module

The configuration module is started by the ADAM distribution engine if a new network configuration (*<hostname>.conf*) has been received. The *<hostname>.conf* file contains key value pairs for most configuration parameters, e.g., *eth0_IP="130.92.66.40"*. These dynamic parameters are used to generate and update most of the other configuration files. If the file name matches the host name of the node, the configuration module automatically applies the new network configuration to the node, restarts the network interfaces, and reloads all affected system services.

C. New Node Module

The ADAM distribution engine only accepts communications from known nodes. In order to guarantee encrypted communication, it has to know the public keys of all its communication peers. The new node module handles the integration of new nodes, of which public keys and network configurations are unknown within the network.

A newly set-up node includes already all configurations and keys of the other network nodes. In contrast, the already deployed nodes are not aware of the new node. The network administrator provides the public key and network configuration of the new node to the new node module, which then distributes them using the ADAM distribution engine. If the ADAM web configuration tool is used to generate the new configuration, keys and network configurations are automatically handed over to the new node module.

D. Software Update Module

The software update is responsible for applying a new software image to the node. Software images are distributed together with an update file that contains detailed information about the specific update action. It contains the file name of the software image, the node type, the update version and a checksum of the software image. If the node retrieves a new software image by the ADAM distribution engine, the software update module checks if node type, version and check-sum match. If positive, the module calls the update procedure specific for the node type. A safe update procedure, as described in SRM [17], is started on nodes with sufficient secondary storage capacity. This uses an additional update partition on the secondary storage and the Grub boot loader. The update partition holds a secondary set of boot loader configuration that boots the current image. In case of an update, the software update module copies the update image to the first partition and adjusts the boot entry on the first partition to boot the update image. The node is rebooted. During start-up, the boot loader rewrites the

Master Boot Record to point to the safe entries on the second partition. If the update image can be successfully booted, the update is made permanent by replacing the standard image and readjusting the boot loader files. Otherwise, a boot flag of Linux enforces a reboot, if a kernel panic occurs or the root file system could be reloaded. In this case, the node automatically reboots and loads the standard image using the correct boot loader configuration on the second partition. In this way, a safe update of the software image can be guaranteed in any circumstances.

Platforms without sufficient secondary storage do not support this safe update procedure. Consequently, a failed software update requires physical access in case of such platforms, e.g., Meraki Mini and the OpenMesh OM1P.

E. Command Module

The node-specific configuration images are not directly distributed as a whole image over the ADAM distribution engine due to the huge transmission overhead. There are usually only potentially small changes in the configuration image. Adding, removing and modifying configuration files in the configuration image is performed by the generic command module. This module can execute user-defined commands on user-defined groups of nodes. The commands are then only executed on the specified nodes. The commands are written in a command file, which is then propagated together with data files within the network using the ADAM distribution engine. For example, if a bug fix for the file *hotplug2.rules* should be applied to some nodes, a command file with instructions for the file replacement is copied together with the new file to the exchange directory of a node. ADAM then distributes the files within the network and the bug fix is applied on all specified nodes.

F. Lost Node Detection

There are scenarios, in which a node can totally loose connectivity to all other network peers due to misconfiguration that is not properly handled by sanity checks during the updates. After a predefined time-out without any network connection (2 h recommended due to decentralised distribution mechanism and update time on low cost nodes), the node therefore resets its transmission power to the maximum value and then searches on all wireless channels for an ad-hoc network with the service set identifier that matches an IPv6 prefix. If such a permanent IPv6 management network is found, the node connects to it and then fetches a new network configuration.

VI. Conclusions

ADAM provides a secure and safe management architecture for WMNs. It improves connectivity between the network nodes and avoids costly onsite repairs by a decentralised distribution mechanism, safe configuration and software updates, and self-healing mechanisms. ADAM can cope with unavailable nodes and automatically repairs configuration and software update errors. It does not require a co-located backbone network for management. ADAM provides a user-friendly, easy to understand and extendable build system for a customised embedded Linux operating system for WMNs. The ADAM framework is released under GPLv2 license [19]. It has been used in several projects, e.g., CTI-Mesh [20], WISEBED [21] and A4-Mesh [22].

References

[1] J. C. Bicket, D. Aguayo, S. Biswas, and R. Morris, "Architecture and Evaluation of an Unplanned 802.11b Mesh Network," in *MobiCom '05*, Cologne, Aug. 28 - Sept. 2 2005.

[2] R. Sombrutzki, A. Zubow, M. Kurth, and J.-P. Redlich, "Self-Organization in Community Mesh Networks - The Berlin RoofNet," in *OpComm*, Berlin, Germany, September 2006.

[3] V. Angelakis, M. Genetzakis, N. Kossifidis, K. Mathioudakis et al., "Heraklion MESH: An Experimental Metropolitan Multi-Radio Mesh Network," in *WinTECH '07*, Montreal, Canada, Sept. 2007.

[4] R. Patra, S. Nedevschi, S. Surana, A. Sheth, L. Subramanian, and E. Brewer, "WiLDNet: Design and Implementation of High Performance WiFi Based Long Distance Networks," in *4th USENIX NSDI'07*, Cambridge, MA, USA, April 2007.

[5] D. Wu and P. Mohapatra, "QuRiNet: A Wide-Area Wireless Mesh Testbed for Research and Experimental Evaluations," in *COMSNETS*, Jan. 2010.

[6] "OpenWrt," http://openwrt.org.

[7] "OpenEmbedded," http://www.openembedded.org.

[8] "BitBake," http://developer.berlios.de/projects/bitbake.

[9] "Ångström," http://www.angstrom-distribution.org.

[10] "Linux From Scratch," http://www.linuxfromscratch.org.

[11] D. Manzano, J.-C. Cano, C. Calafate, and P. Manzoni, "MAYA: A Tool For Wireless Mesh Networks Management," *MASS*, Oct. 2007.

[12] R. Riggio, N. Scalabrino, D. Miorandi, and I. Chlamtac, "JANUS: A Framework for Distributed Management of Wireless Mesh Networks," in *TridentCom 2007*, Orlando, Florida, USA, May 21-23 2007.

[13] K. N. Ramachandran, E. M. Belding-Royer, and K. C. AImeroth, "DAMON: A Distributed Architecture for Monitoring Multi-Hop Mobile Networks," in *IEEE SECON*, Santa Clara, CA, USA, Oct. 4 - 7 2004.

[14] K. N. Ramachandran, K. C.Almeroth, and E. M. Belding-Royer, "A Framework for the Management of Large-Scale Wireless Network Testbeds," in *WiNMee*, Riva del Garda, Trentino, Italy, April 3 2005.

[15] V. Aseeja and R. Zheng, "MeshMan: A Management Framework for Wireless Mesh Networks," in *IM '09*, June 2009.

[16] B. Pinheiro, V. Nascimento, E. Cerqueira, W. Moreira, and A. Abelém, "Abaré: A Coordinated and Autonomous Framework for Deployment and Management of wireless mesh networks," in *LNCS*, vol. 6157. Springer, 2010.

[17] T. Staub, D. Balsiger, M. Lustenberger, and T. Braun, "Secure Remote Management and Software Distribution for Wireless Mesh Networks," in *ASWN*, Santander, Spain, May 2007.

[18] M. Burgess, "A Tiny Overview of Cfengine: Convergent Maintenance Agent," in *MARS/ICINCO*, Barcelona, Spain, Sept. 2005.

[19] "ADAM," http://rvs.unibe.ch/research/software.html.

[20] "CTI-Mesh," http://cti-mesh.ch.

[21] "WISEBED," http://wisebed.eu.

[22] "A4-Mesh," http://a4-mesh.unibe.ch.