Extending `dladm` for WiFi: An Administrative Overview

Peter Memishian (meem)

`wifi-dladm@sun.com`

Sun Microsystems, Inc.

Revision 1.9.2
November 14, 2006

Contents

1 Introduction

1.1 WiFi and Solaris

WiFi is a popular marketing name for a range of Wireless LAN protocols that have been developed by the IEEE – specifically, 802.11a, 802.11b, and, most recently, 802.11g. The usefulness, convenience, affordability, and availability of WiFi devices in the consumer marketplace has made interest in the technology skyrocket. Further, widespread support for WiFi among other popular operating systems such as Linux, Mac OS X, and Windows, has made WiFi an increasingly requested feature on Solaris. WiFi is especially popular with system developers and deployers – two demographics critical to the ultimate success of Solaris.

Accordingly, Solaris WiFi development has been proceeding at a rapid pace, culminating in the recent integration of the `ath(7D)` driver – supporting the popular Atheros wireless chipset – into OpenSolaris. In addition, drivers for several Intel wireless chipsets can be downloaded through `opensolaris.org`[1], and many other popular wireless chipsets will soon be available. Work is also underway to add support for the WPA and WPA2 (802.11i) suite of wireless security protocols, 802.1x access control, and other technologies critical for convenient and secure wireless deployment.

1.2 WiFi Administration Challenges

WiFi administration is currently performed through the Solaris-specific `wificonfig(1M)` command. This utility has a collection of subcommands that allow a variety of WiFi-specific administrative tasks to be performed – most commonly, connecting to a particular WiFi network.

However, during WiFi development, a new command, `dladm(1M)`, was introduced to perform generic link layer administration of Solaris network devices. Specifically, `dladm(1M)` administers all network devices written to the new GLDv3[2] device driver framework, which was also introduced during WiFi development. While the GLDv3 framework is currently Ethernet-specific, it is being generalized to support arbitrary link layers, including WiFi. Having a single point of administration for all link-layers – regardless of their type – is core to simplifying the administrative model. For instance, the administrator can trivially see all of the network links available on the system through a single `dladm show-link` operation, or can issue the same `dladm` command to adjust the MTU of any link, regardless of its type.

For these reasons, there is widespread agreement that all existing WiFi drivers should be ported to GLDv3 once WiFi support is available, and that all new WiFi drivers should be written to GLDv3. However, there was concern that if `wificonfig` remained, the resulting administrative model would be confusing, because two utilities would exist with partially overlapping but independent functionality, and overlapping but independent administrative abstractions.

To complicate matters further, `wificonfig` also overlaps with the Network Auto-Magic[3] (NWAM) project, which also began during WiFi development. Specifically, NWAM aims to make it easy to transition between different network configurations on a running system. For instance, a laptop user could configure one NWAM profile for "work", and another for "home", and transition between them at the touch of a button. The `wificonfig` command *also* provides a notion of profiles, though the profiles are WiFi-specific. For instance, a laptop user can have one `wificonfig` profile for a home WiFi network, and another for a work WiFi network. Again, there was concern that two utilities with partially overlapping but independent functionality would be problematic.

[1] http://opensolaris.org/os/community/laptop/wireless/
[2] http://sac.sfbay/Archives/CaseLog/arc/PSARC/2004/571/
[3] http://opensolaris.org/os/project/nwam/

Technologies that span both wireless and wired networks present additional complications. For instance, 802.1x, which is a critical piece of the WPA2 wireless security standard, is also an increasingly requested feature on wired networks. Thus, it is highly desirable to allow 802.1x to be administered identically in both cases, which rules out a WiFi-specific administrative utility for managing 802.1x.

1.3 WiFi Administration Plan

After considerable discussion, a decision was reached to pursue a two-pronged approach to WiFi administration in Solaris:

1. To provide short-term relief to cutting-edge customers, the existing `wificonfig` utility would be made available in both Solaris Express and OpenSolaris. However, it would be provided with no stability guarantees.
2. To provide a long-term solution for WiFi administration, the existing `dladm` utility would be extended to subsume `wificonfig` for WiFi administration. Additionally, the Network Auto-Magic project would be extended to subsume `wificonfig` for WiFi configuration profiles. At that point, `wificonfig` could be removed from Solaris and OpenSolaris.

A decision was also reached that it would be inappropriate to ship `wificonfig` in a Solaris Update only to later remove it in a subsequent Solaris Update. As such, support for WiFi in Solaris 10 should be based on the long-term plan outlined above.

1.4 Document Scope and Structure

This short document details the proposed extensions to `dladm` to provide long-term WiFi administration on Solaris. Though the proposed extensions are designed with Network Auto-Magic in mind, this document does **not** cover the proposed extensions to Network Auto-Magic to support WiFi. This document also does **not** cover the design of the underlying implementation.

The discussion is broken into two sections:

- Section 2 covers the core administrative model, and provides the rationale and design goals behind it. It also discusses how `dladm` will interact with Network Auto-Magic and general link layer administration, how `dladm` functionally compares to `wificonfig`, and how other evolving wireless technologies can be incorporated into `dladm` in the future.
- Section 3 specifies the exact changes being proposed to `dladm`.

2 Core Administrative Model

2.1 The `dladm` Model: Present and Future

To provide context for the proposed WiFi extensions, one must first understand the core `dladm` administrative model. Currently, `dladm` is quite simple, consisting of just eight subcommands.

The subcommand names have a consistent, structured format. Specifically, as per recent user interface guidelines, all subcommands are of the form `verb-noun`, with the `noun` reflecting the type of object the `verb` applies to – e.g, `show-link`, `show-dev`, and `show-aggr`; objects are usually various classes of links, as shown below.

There are also established conventions that the `-t` option will cause a subcommand that affects system configuration not to apply the change persistently. Similarly, the `-P` option will cause a subcommand that displays output to show the persistent configuration (if it exists) instead of the current configuration, and the `-p` option will display output using a machine-parseable format[4] The `-p` option underscores another key aspect of `dladm`: it is designed with scripting in mind.

Minimalism is also central to the `dladm` administrative model. Specifically, the eight existing `dladm` subcommands provide stark contrast to `ifconfig`, which warehouses more than fifty, and has repeatedly proven itself to be a usability nightmare. To that end, new `dladm` subcommands should be introduced reluctantly, should be as general as possible, and require as few options as necessary to perform their tasks.

Currently, most of the subcommands are used for configuring or examining 802.3ad link aggregations[5]. The remaining subcommands, `show-dev` and `show-link` are used to show network devices and data links, respectively.

The separation between devices and links will be core to the future `dladm` model, as being evolved by Clearview[6]. Specifically, after Clearview, *devices* will correspond to the driver names associated with physical devices, such as `bge0` or `ath0`. In contrast, *links* will correspond to the link layer, with names that may be selected by the administrator. For backward compatibility, by default, every device will have an identical link name, but not all links will correspond directly with devices. For instance, IP tunnels, and link aggregations will be represented as links but will have no directly associated device.

In essence, the *link* abstraction allows the administrator to separate the selection of networking hardware from the system's network configuration. Additionally, the abstraction allows data links that do not correspond directly to devices to be represented seamlessly. For instance:

```
# dladm show-link
LINK              CLASS  MTU    STATE   OVER
bge0              phys   1500   up      --
bge1              phys   1500   up      --

# dladm create-aggr -l bge0 -l bge1 aggr1        [ create link aggr1 ]

# dladm show-link
LINK              CLASS  MTU    STATE   OVER
bge0              phys   1500   up      --
bge1              phys   1500   up      --
aggr1             aggr   1500   up      bge0 bge1
```

[4]This format was introduced with `dladm` but unfortunately is currently awkward and undocumented.
[5]Similar families of subcommands are under development for VLANs and IP Tunnels.
[6]http:/opensolaris.org/os/project/clearview/

For these reasons, the link represents the fundamental abstraction that will be manipulated by future `dladm` subcommands. As shown in the previous example, the link itself may be one of several link classes – e.g., `phys` for a physical device link, or `aggr` for a 802.3ad link aggregation.

While Clearview's evolved `dladm` administrative model is still under development, it is important that any new subcommands be designed in a manner that will not require incompatible administrative changes once Clearview integrates. For instance, any new `dladm` subcommands should be designed in terms of *links*, rather than *devices*. Since devices and links are currently interchangeable, this is mostly a conceptual distinction that can be reflected in the documentation.

2.2 The `dladm` Model for WiFi

In keeping with the minimalism of `dladm`, only four new WiFi-specific subcommands are proposed:

- `scan-wifi`: scan WiFi networks on a link
- `connect-wifi`: connect a link to a specific WiFi network
- `show-wifi`: display status for all WiFi links
- `disconnect-wifi`: disconnect a link from a specific WiFi network

As a simple example:

```
# dladm show-link
LINK            CLASS  MTU    STATE   OVER
bge0            phys   1500   up      --
ath0            phys   1500   up      --

# dladm scan-wifi
LINK         ESSID            BSSID/IBSSID       SEC    STRENGTH   MODE   SPEED
ath0         wlan             08:00:20:AD:4A:F4  none   excellent  b,g    54Mb
ath0         wep-wlan         00:03:BA:00:DC:85  wep    very good  b      11Mb

# dladm connect-wifi -e wlan ath0
# ifconfig ath0 dhcp start
#  [ use ath0 ]
# ifconfig ath0 dhcp release
# dladm disconnect-wifi ath0
```

As shown above, some subcommands take command-line options to further modify their behavior (e.g., the `-e` option to `connect-wifi` selects a particular ESSID to connect to). A full list of command-line options for each subcommand is provided in Section 3; how this differs from `wificonfig` is discussed in Section 2.12.

While `dladm` can be used directly, there is a strong expectation that most subcommands will be used as building blocks, either invoked through scripts, or accessed through a parallel set of C library routines that will be added to `libdladm`. In fact, because `dladm` does not provide profiles, scripts will prove necessary in order to easily switch between WiFi networks until Network AutoMagic is available; see Section 2.10.

In addition to the subcommands shown above, two new sets of general subcommands are proposed: one set for administering properties associated with a link (see Section 2.3), and another for administering *secure objects* (such as keys) which can then be applied to a link (see Section 2.4). While these subcommands will initially only function on WiFi links, they will subsequently be extended to support other link types.

2.3 Link Properties

Three additional subcommands are proposed to administer link properties:

- `set-linkprop`: set one or more link properties
- `reset-linkprop`: reset one or more link properties to their default values
- `show-linkprop`: show the value of one or more link properties

As previously mentioned, these subcommands will initially only operate on WiFi links, but subsequent enhancements will add long-overdue property support for other types of links. Unlike legacy utilities such as `ndd`, the semantics and allowed values of each property will be known to `dladm`, reducing the incidence of administrative mistakes:

```
# dladm set-linkprop -p powermode=xyzzy ath0
dladm: link property 'powermode' must be one of: fast,max,off
# dladm set-linkprop -p powermode=fast ath0
# dladm show-linkprop ath0
PROPERTY      VALUE           DEFAULT         POSSIBLE
channel       2               --              --
powermode     fast            off             fast,max,off
radio         on              on              off,on
 [ ... ]
# dladm reset-linkprop -p powermode ath0
# dladm reset-linkprop -p channel ath0
dladm: cannot reset link property 'channel' on 'ath0': read-only property
#
```

For each property, the values the administrator may use are shown under `POSSIBLE`; if none (`--`) is shown, then the property is read-only. As shown above, `set-linkprop` and `reset-linkprop` cannot be used on read-only properties.

As with other `dladm` subcommands that alter configuration, changes made through `set-linkprop` and `reset-linkprop` will apply both to the current and persistent configuration of the link. Also as with other subcommands, the `-R` option will select an alternate root directory to use for persistent configuration, and `-t` ("temporary") will enable just the current configuration to be affected.

To prevent link properties becoming another sewer like `ndd`, each newly proposed link property will need to be carefully justified. For instance, any proposal should cover why the link property mechanism is appropriate (e.g., why and how system administrators or applications would interact with the property), why self-tuning is impossible or inappropriate, and how the link property has been generalized to ensure that it can be useful for future links. This aims to not only simplify and unify the administrative model, but also to enable the development of layered tools that can make use of these properties. It also implies that the link property names and semantics are stable and will be preserved across upgrades.

However, there will be cases where a given link property is either not generally applicable, or where we need to gain more experience before committing to an administrative model. For these cases, we propose to introduce a special namespace, starting with `link_`[7] that is reserved for properties that are specific to a given link and subject to change at any time. Accordingly, no guarantee is made to preserve behavior across an upgrade. This distinction will be documented in `dladm(1M)`, and any link-specific tunables will be covered in the appropriate network driver manpage.

A full list of WiFi link properties and allowed values is provided in Section 3; a discussion of how property handling differs from `wificonfig` is given in Section 2.12.

[7] The actual driver name should be omitted – e.g., `link_xyzzy`, *not* `link_bge_xyzzy`.

2.4 Secure Objects

As previously mentioned, there is a strong expectation that most subcommands will be used as building blocks (e.g., by scripts). However, it is unsafe to pass secure material, such as keys, directly to subcommands such as `connect-wifi`. As such, three additional `dladm` subcommands are proposed to administer secure objects:

- `create-secobj`: create a secure object
- `delete-secobj`: delete one or more secure objects
- `show-secobj`: show one or more secure objects

Unlike link properties, secure objects are **not** associated with a specific link, and none exist by default. Accordingly, the administrator selects the name for each secure object, along with its value and *class*. The class allows `create-secobj` to handle multiple types of secure objects, and to be extended to support new secure objects. For instance, to create a WEP key named `mykey`:

```
# dladm create-secobj -c wep mykey
provide value for 'mykey':  [admin provides value]
confirm value for 'mykey':  [admin reenters value]
#
```

Alternatively, to keep `dladm` scriptable, the value may be loaded from a file with a specified format:

```
# dladm create-secobj -c wep -f /tmp/mykey mykey
```

In either case, a secure object named `mykey` will be created. This name will then be able to be passed to subcommands that accept a secure object of the specified class. For instance, to use `mykey` to connect using `wep` security to `wep-wlan`:

```
# dladm connect-wifi -e wep-wlan -k mykey ath0
```

Note that the security mode to use (`wep`) will be implied by the key's class[8]. For more on how keys will be passed to `connect-wifi`, see Section 3.2.

The `show-secobj` subcommand will show the configured secure object and their classes:

```
# dladm show-secobj
OBJECT       CLASS
mykey        wep
#
```

For security reasons, the object contents are not shown. This subcommand is chiefly useful for jogging an administrator's memory when using `dladm` subcommands that require secure objects, though layered technologies such as Network Auto-Magic may also find the facility useful.

Finally, one or more secure objects can be deleted using `dladm delete-secobj`:

```
# dladm delete-secobj mykey
```

As with other subcommands that modify configuration, `create-secobj` and `delete-secobj` will apply to both the current and persistent configuration. Also as with other subcommands, the `-t` option will cause only the current configuration to be changed. A file will be used for persistent storage (see Section 2.6), but future options could direct `dladm` to use alternate persistent storage, such as a keystore contained on the WiFi device itself. Full synopses are given in Section 3; how and why key handling differs from `wificonfig` is covered in Section 2.12.

[8] Future security modes such as WPA/Enterprise do not directly take a key (since they generate it). In those cases, the security mode will need to be explicitly provided via `-s`.

2.5 RBAC Model

The `dladm` utility is currently part of the `Network Management` execution profile, and users must be added to that profile (or granted access to a role with that profile) in order to successfully invoke its subcommands. However, there are longstanding plans to enhance the `show` subcommands to allow any user to successfully invoke them, regardless of their profile or role[9].

Conceptually, granting access to a WiFi link is no different from a granting access to a traditional wired link – e.g., a user allowed to configure a wired link on their laptop should also be allowed to configure a WiFi link. For this reason, users belonging to the `Network Management` profile (or granted access to a role with that profile) will also be able to successfully invoke the proposed WiFi subcommands, as well as `show-secobj`. However, the ability to configure link keying material through the `create-secobj` and `delete-secobj` subcommands will be restricted to a new `Network Link Security` profile. Thus, sites will be able to give users the ability to view key names and invoke `connect-wifi` to connect using an existing key, but not give them the ability to arbitrarily modify the keying material.

To enforce the restrictions on `create-secobj` and `delete-secobj`, `dladm` will check for a new authorization, `solaris.network.link.security`. This authorization will be part of the `Network Link Security` profile. For convenience, by default, the existing `Network Security` profile will contain the `Network Link Security` profile.

2.6 Security Considerations for Keys

The keys manipulated through the `create-secobj` and `delete-secobj` subcommands will be stored in a file that will only be readable and writable by user `dladm`. The `dladm` user will be a locked account that has only `basic` privileges, but users belonging to the `Network Management` profile will (through RBAC) cause `dladm` to run with an effective user of `dladm` so that it may access the key file. This will allow keys to be kept confidential without the risks that would be associated with an application being `setuid root`. (As with traditional setuid applications, all other file access performed by `dladm` will be done as the real user.)

The WiFi devices will allow any process with the `PRIV_NET_SYS_CONFIG` privilege to modify their active keys, such as when `dladm connect-wifi` is performed [10]. Note that this means that any user with `PRIV_NET_SYS_CONFIG` may subvert the aforementioned `solaris.network.link.security` authorization checks in `dladm`. However, this is a generally accepted limitation of the authorization facility and is usually only problematic if privileges are directly assigned via `/etc/user_attr` (which, for this very reason, is strongly discouraged).

2.7 Auditing Considerations

Because `dladm` uses RBAC (rather than `setuid`), each `dladm` invocation will generate an audit record. Two new audit events, `AUE_dladm_create_secobj` and `AUE_dladm_delete_secobj`, will audit whether authorization to create or destroy a given secure object was granted. So that `dladm` can generate the audit events, a contract for the internal Solaris auditing interfaces[11] will be obtained.

[9] See CR 6219893.

[10] Note that `dladm` already needs this privilege in order to perform other subcommands.

[11] http://sac.sfbay/Archives/CaseLog/arc/PSARC/2000/517/

2.8 Configuration Files

Two new private configuration files – `secobj.conf` and `linkprop.conf`[12] – will store persistent secure objects and link properties in an implementation-private manner. To allow `dladm` to modify its configuration files without needing `root` credentials, each file will be owned by user `dladm` and reside in a new `/etc/dladm` directory, which will also be owned by user `dladm`. The lone existing `dladm` configuration file, `aggregation.conf`, will be moved from `/etc` to `/etc/dladm` (and changed to user `dladm`) upon upgrade.

As before, all `dladm` configuration files remain an implementation detail of `dladm`, and directly accessing or modifying them is unsupported. Comments atop of each file will explicitly state that it is subject to format change or outright removal at any time. Indeed, changes underway as part of Clearview's Vanity Naming will likely consolidate all non-sensitive `dladm` configuration files; that proposal will cover the specifics of the transition plan, which will be transparent to the user.

2.9 Boot

During boot, persistent link properties and persistent secure objects will be configured into the system as part of the `net-physical` SMF milestone. This will be done using two new undocumented `dladm` subcommands: `init-linkprop` and `init-secobj`. This mirrors the existing handling for persistent link aggregations, which are also recreated in `net-physical` through an undocumented `dladm` subcommand. Because `net-physical` runs after the root filesystem is mounted, the configuration files containing the persistent information need not be in the boot archive.

WiFi interfaces will be able to be plumbed at boot through the traditional `/etc/hostname.`*if* mechanism. Since the policy engine needed to choose a specific ESSID will be provided by Network Auto-Magic, no attempt will be made to use `dladm` to connect to a specific network during boot. However, existing logic in `net-physical` will cause `wificonfig` to attempt to connect to an ESSID using its preference mechanism. That logic will ultimately be replaced by Network Auto-Magic when `wificonfig` is removed.

Diskless boot over WiFi will not be supported.

2.10 Interaction with Network Auto-Magic

As previously discussed, the Network Auto-Magic (NWAM) project aims to make it easy to select and transition between different network configurations on a running system. NWAM is inherently a "vertical" technology that slices through many layers of the networking stack: as part of a transition, a wide range of network configuration in addition to link layer configuration will be changed, such as name servers and web proxies.

NWAM will layer on top of the `dladm` functionality proposed here, both in order to discover relevant configuration such as available WiFi links, and also to configure a specific WiFi link according to an established profile. To provide convenient access, `libdladm` library routines will be provided for each proposed `dladm` subcommand (this convention is already followed for all current subcommands). Note that the widespread use of command-line options (implemented as arguments to the `libdladm` library routines) will make it easy for NWAM to establish a given configuration. Further, named secure objects enable profiles containing references to keys to be stored or passed between layers without security implications.

Unfortunately, until the NWAM project is available, the lack of named persistent storage (outside

[12] The two files are separate because `secobj.conf` contains confidential information such as private keys.

of `create-secobj`) means that there will be no first-class support for switching between wireless networks with different configurations. For instance, different keys will have to be manually specified when invoking `connect-wifi`, or, alternatively, scripts will be needed to bridge the gap. However, we anticipate that most will prefer to use front-ends such as `inetmenu`[13], which in turn will be updated to use `dladm`, so this should not be a major issue. Further, the long-term implications of the alternative – having two partially overlapping but distinct profile mechanisms – is seen as unacceptable.

2.11 WPA/WPA2 Considerations

As previously mentioned, a separate project is underway to add support for the WPA and WPA2 (802.11i) suite of wireless security protocols, including support for 802.1x access control. While specific support for WPA and WPA2 are not covered here, the proposed design does take their requirements into account. Specifically:

- WPA/WPA2 pre-shared keys can be supported by creating new secure object classes.
- WPA/WPA2 Enterprise mode can be enabled by omitting a key when invoking `connect-wifi`.
- WPA/WPA2 Enterprise mode behavior can be controlled by creating new link properties.

Additionally, given the large and varied configuration associated with Enterprise mode, NWAM will play a crucial role in its long-term administration. Specifically, NWAM will prove vital in allowing the administrator to easily set up and store per-network Enterprise-mode settings, and automatically activating those settings when an appropriate network becomes available. Again, in the meantime, it is anticipated that scripts such as `inetmenu` will bridge the gap by providing a primitive form of profiles.

2.12 Functional Differences vs. `wificonfig`

The most notable functional difference between `wificonfig` and `dladm` is the absence of profiles. Specifically, profiles not only represent a sizable set of the `wificonfig` command set (6 subcommands), but also are the building blocks for its preference mechanism (3 subcommands), and enable a collection of parameters to be conveniently passed to its `connect` subcommand. Moreover, configuration specified using profiles is also used in lieu of command-line arguments to all `wificonfig` subcommands. For instance, with `dladm`, the `bsstype` will be specified directly using the `-b` option to `connect-wifi`; with `wificonfig`, `connect` is passed a profile, which in turn contains a `bsstype` value to use. However, as previously discussed, network-wide profile support is under development via Network Auto-Magic, which will include functional equivalents for all `wificonfig` profile functionality.

As a result of profiles, `wificonfig` makes widespread use of parameters to control many aspects of its behavior. For instance, the `createibss` parameter does not represent an attribute of the WiFi link, but instead alters the behavior of the `connect` subcommand. With `dladm`, *link properties* will always represent attributes of the link itself, and parameters such as `create-ibss` will instead be implemented as command-line options to the appropriate subcommands. The `wificonfig` utility also uses parameters to report status information such as signal strength; `dladm` will have a dedicated `show-wifi` subcommand to report status.

The `dladm` command will not provide an equivalent for the `wificonfig history` subcommand. Instead, it is expected that the Network Auto-Magic Project [14] will provide layered support for

[13] http://opensolaris.org/os/community/laptop/inetmenu/
[14] http://opensolaris.org/os/project/nwam/

tracking recently-discovered networks on a system-wide basis and implementing a user-specified policy (such as ignoring or automatically connecting) if they are rediscovered.

While both `wificonfig` and `dladm` implement WEP keys as write-only properties, the design is quite different. Specifically, with `wificonfig`, the key names provide semantic meaning, and correspond to a fixed key index (`wepkey1` - `wepkey4`), and setting a key value immediately affects the underlying hardware. A special-purpose `setwepkey` subcommand is provided to allow key values to be safely set. Finally, a separate `wepkeyindex` parameter allows the active transmit key to be set on the underlying hardware.

In contrast, with `dladm`, key names will be chosen by the administrator, and key semantics will be established by options specified during its creation (e.g., `dladm create-secobj -c wep mykey`). Rather than having a special subcommand for interacting with WEP keys, a generalized mechanism for administering secure values will be provided. Thus, the `dladm` subcommands will have no specific knowledge of WEP, and new key types can be introduced without introducing new subcommands or command-line options. In addition, the secure objects facility can be used for other secure information, such as certificates, if need be. Finally, unlike `wificonfig`, the underlying hardware will remain unaffected until the key is passed to a subcommand such as `connect-wifi`.

The `wificonfig` RBAC model is also different. Specifically, it makes exclusive use of the `Network Wifi Management` and `Network Wifi Security` execution profiles (the latter profile provides authorizations needed to modify keying material). As discussed in Section 2.5, there is no reason to separate the management of WiFi links from traditional wired links. As such, `dladm` allows members of the `Network Management` execution profile to administer any type of link. Further, while `dladm` also requires a separate execution profile to modify keying material, the profile has been generalized to cover any type of link, via the new `Network Link Security` execution profile.

2.13 Interaction with Future Wireless Technologies

Although there are many evolving wireless technologies, only IEEE 802.16 (WiMAX) appears likely to gain traction in the non-embedded OS space[15]. Compared to WiFi, WiMAX provides enhancements that will likely impact the administrative model and audience – notably:

Mesh Networking: A "self-healing" network topology which allows a network to dynamically work around failed nodes. This will likely require additional administrative control over network capacity, operating range, privacy, and security.

MAC Scheduling and QoS: CSMA/CA is replaced with a time-slot based approach to accessing the network. Accordingly, WiMAX allows both the scheduling slots and overall QoS guarantees to be tuned on a per-node basis.

Increased Range and Performance: An operating range in miles rather than feet means that strong security becomes a requirement, along with administrative mechanisms to shorten the range and alter the power consumption, when necessary. When coupled with increased performance, deployment scenarios beyond laptops become realistic – e.g., "last-mile" connectivity between NSP's and their customers.

In addition, while the WiFi operations (scan, connect, and disconnect) remain in WiMAX, the details differ greatly. For instance, established WiFi terminology such as ESSID, BSSID, BSS, ESS, and DS have been replaced with analogous but not identical WiMAX concepts. Further, WiFi concepts such as IBSS mode do not exist in WiMAX.

[15] In particular, the advent of 802.16e, along with political coups within the IEEE itself, make it unlikely that 802.20 (MobileFi) will be widely adopted.

As a result, even if the WiFi subcommands were generalized, the options those subcommands take would be quite different depending on whether a WiFi or WiMAX link was being administered. Further, the behavior of some subcommands would need to be different – e.g., a `show-wireless` subcommand would necessarily display different information for WiFi and WiMAX links.

For all of these reasons, it seems clear that WiMAX is best administered through an additional set of WiMAX-specific `dladm` subcommands. Of course, the new `dladm` subcommands proposed to administer link properties and secure objects will be able to be reused by WiMAX.

3 Proposed dladm Extensions

This section summarizes the precise changes being proposed. The design principles behind these extensions can be found in the preceding section.

3.1 scan-wifi

Usage: `dladm scan-wifi [-p] [-o <field>,...] [<link>]`

The `scan-wifi` subcommand scans for all WiFi networks. If no `link` is specified, then all WiFi links are scanned. If `link` is specified, the scan is restricted to it. If `link` is not a WiFi link, an error is returned. For each WiFi network found, the following fields are available for display:

- `LINK`: The link the WiFi network was found on.
- `ESSID`: The ESSID (name) of the WiFi network.
- `BSSID/IBSSID`: Either the hardware address of the WiFi network's Access Point (for BSS networks), or the WiFi network's randomly generated unique token (for IBSS networks).
- `SEC`: Either `none` for a WiFi network that uses no security, or `wep` for a WiFi network that requires WEP security.
- `MODE`: The supported connection modes: one or more of `a`, `b`, or `g`.
- `STRENGTH`: The strength of the signal: `excellent`, `very good`, `good`, `weak`, or `very weak`.
- `SPEED`: The maximum speed of the WiFi network, in megabits per second.
- `BSSTYPE`: Either `bss` for BSS (infrastructure) networks, or `ibss` for IBSS (ad-hoc) networks.

To keep the output under 80 columns, by default `BSSTYPE` is not displayed. Further, the fields displayed by default are subject to change. However, the `-o` option can be used to specify an alternate set of (case-insensitive) fields to display, and `-o all` can be used to display all fields. However, all fields are output by default when machine-parseable output is requested using the `-p` option.

Example: scan `ath0` for WiFi networks:

```
# dladm scan-wifi
LINK         ESSID            BSSID/IBSSID       SEC    STRENGTH   MODE   SPEED
ath0         wlan             08:00:20:AD:4A:F4  none   excellent  b,g    54Mb
ath0         wep-wlan         00:03:BA:00:DC:85  wep    very good  b      11Mb
#
```

3.2 connect-wifi

Usage: `dladm connect-wifi [-e <essid>] [-i <bssid>] [-a open|shared] [-m a|b|g]`
`[-b bss|ibss] [-c] [-k <key>,...] [-s wep]`
`[-T <timeout>] [<link>]`

The `connect-wifi` subcommand attempts to connect to a WiFi network. The connection process consists of four steps: discovery, filtering, prioritization, and association. The discovery step finds all available WiFi networks on the specified `link`. If the link is already connected, an error will be returned. For administrative convenience, if there is only one WiFi link on the system, the `link` argument may be omitted.

To allow connections to non-broadcast networks, and to improve performance in the common case, if `-i` or `-e` is specified, then `connect-wifi` bypasses the first three steps and immediately attempts to associate to a `bssid` or `essid` that matches the rest of the provided parameters. Since the BSSID is unique, if `-i` was specified and the association fails, the operation fails. However, the ESSID is not unique[16] – therefore, if `-e` was specified and the association fails, the normal discovery process begins, so that association with other networks matching the specified criteria can be attempted.

Once discovery is complete, the list of networks is filtered according to the supplied options:

- If `-e` is specified, networks not matching the specified ESSID are removed.
- If `-b` is specified, networks not of the specified bsstype (either `bss` or `ibss`) are removed.
- If `-k` is specified, networks not appropriate for the specified keys are removed.
- If `-s` is specified, networks not appropriate for the specified security mode are removed.
- If `-m` is specified, networks not appropriate for the specified 802.11 mode are removed.

Finally, the remaining networks are prioritized, first by signal strength, and then by maximum speed. The `connect-wifi` subcommand then attempts to associate with each network in the list, in order, until one succeeds or the list is exhausted. By default, `connect-wifi` waits up to 10 seconds for each association to either succeed or explicitly fail.

As can be seen above, many command-line options filter the set of considered networks. In addition:

- If `-a` is specified, then the requested authentication mode (either `open` or `shared`) is used. By default, `open` and `shared` are tried in order.
- If `-c` is specified, and `-b ibss` is also specified, then if an ad-hoc network matching the provided ESSID cannot be found, then a new ad-hoc network is created. If no ESSID is specified, then `-c -b iss` will always trigger the creation of a new ad-hoc network.
- If `-T <timeout>` is specified, then the associate step will wait for up to the specified number of seconds. If a timeout of `forever` is specified, then the associate will wait indefinitely. Timeouts smaller than the default may not operate reliably.
- If `-k <key>` is specified, then in addition to the filtering previously described, the specified key will be used to secure the association. The security mode to use will be based on the key's class; if an explicit security mode was specified using `-s` that is not compatible, an error will be returned.

  For security modes that support multiple key slots, the slot to place the key will be specified by a colon followed by an index[17]. Thus, `-k mykey:3` places `mykey` in slot 3. By default, slot 1 is assumed. For security modes that support multiple keys, a comma-separated list can be specified, with the first key being the active key. All keys must be of the same class.

If no connection can be established within the specified timeout period, an error is returned.

Examples:

On a system with a single WiFi link, connect to the most optimal unsecured network (as per the rules above):

```
# dladm connect-wifi
```

On a system with a single WiFi link, connect to the most optimal unsecured network of type `bss`:

```
# dladm connect-wifi -b bss
```

[16] As a result, `-e` cannot guarantee association with a specific network. If this is required, `-i` must be used.

[17] Since WEP appears to be the only security mode to expose key slots into the administrative model, and WEP itself appears destined for irrelevance, this appears sufficient.

Connect to ESSID `wlan` on link `ath0`:

```
# dladm connect-wifi -e wlan ath0
```

Connect using WEP with key `mykey` to ESSID `wlan-wep` on link `ath0`:

```
# dladm connect-wifi -e wep-wlan -k mykey ath0
```

Create a new ad-hoc network named ESSID `mynet` on link `ath0`:

```
# dladm connect-wifi -e mynet -b ibss -c ath0
```

3.3 disconnect-wifi

Usage: `dladm disconnect-wifi [-a] [<link>]`

The `disconnect-wifi` subcommand disconnects from one or more WiFi networks. If a `link` is specified, then it is disconnected. If the specified link is not connected, an error is returned. For administrative convenience, if only one WiFi link is connected, the `link` argument may be omitted. To simplify scripting, the `-a` option can be used to disconnect all connected links.

Example: disconnect the existing connection on `ath0`:

```
# dladm disconnect-wifi ath0
```

3.4 show-wifi

Usage: `dladm show-wifi [-o <field>,...] [-p] [<link>]`

The `show-wifi` subcommand shows the status of one or more WiFi links. If a `link` is specified, then only it is shown; otherwise, all WiFi links are shown. If the specified link does not exist, an error is returned. If the `-p` option is specified, then the output is returned in the established machine-parseable format. For each WiFi link, the following fields are available for display:

- `LINK`: The link whose status is being displayed.
- `STATUS`: Either `connected` if the link is connected, or `disconnected` if it is not connected. If the link is disconnected, all of the remaining fields are indeterminate (represented by `--`).
- `ESSID`: The ESSID (name) of the WiFi network the link is connected to.
- `SEC`: Either `none` or `wep`.
- `STRENGTH`: The connection strength: `excellent`, `very good`, `good`, `weak`, or `very weak`.
- `MODE`: The current connection mode: either `a`, `b`, or `g`.
- `SPEED`: The current connection speed, in megabits per second.
- `AUTH`: Either `open` or `shared`.
- `BSSID/IBSSID`: Either the hardware address of the WiFi network's Access Point (for BSS networks), or the WiFi network's randomly generated unique token (for IBSS networks).
- `BSSTYPE`: Either `bss` or `ibss`.

To keep the output under 80 columns, by default, `AUTH`, `BSSID/IBSSID`, and `BSSTYPE` are not displayed. Further, the fields displayed by default are subject to change. However, the `-o` option

can be used to specify an alternate set of (case-insensitive) fields to display, and `-o all` can be used to display all fields. However, all fields are output by default when machine-parseable output is requested using the `-p` option. Note that link properties can be retrieved using `show-linkprop`.

Example: show all WiFi networks:

```
# dladm show-wifi
LINK         STATUS        ESSID            SEC    STRENGTH   MODE   SPEED
ath0         connected     wlan             none   excellent  b      11Mb
bcmndis0     disconnected  --               --     --         --     --
#
```

3.5 show-linkprop

Usage: `dladm show-linkprop [-cP] [-p <prop>,...] <link>`

The `show-linkprop` subcommand shows the values of one or more properties on the specified `link`. If the `-p` option is omitted, then all properties are shown; otherwise, just the specified properties are shown. If the `-P` option is used, then the persistent values are shown instead of the current values. If the `-c`[18] option is used, then output is returned using the established machine-parseable format. For each property, the following information is displayed:

- `PROPERTY`: The name of the property.
- `VALUE`: The current (or persistent if `-P` is specified) property value. The current (or persistent if `-P` is specified) value is shown as `--` if unset, or `?` if unknown. Persistent values that are not set (or have been reset) will use the `DEFAULT` value (if any).
- `DEFAULT`: The default value of the property. If the property has no default value, `--` is shown.
- `POSSIBLE`: A comma-separated list of the values the property may have. If the possible values span a numeric range, *min-max* may be used as shorthand. If the possible values are unknown or unbounded, `--` is shown.

Note that while the `show-linkprop` subcommand applies to all links, only WiFi links currently have link properties. For a list of WiFi link properties, see Section 3.11.

Example: show all current properties on `ath0`:

```
# dladm show-linkprop ath0
PROPERTY      VALUE           DEFAULT         POSSIBLE
channel       2               --              --
powermode     fast            off             fast,max,off
radio         on              on              off,on
 [ ... ]
```

3.6 set-linkprop

Usage: `dladm set-linkprop [-t] [-R <rootdir>] -p <prop>=<value>[,...] <link>`

The `set-linkprop` subcommand sets the values of one or more properties on the specified `link`. The properties and values to set them to are specified using the `-p` option. If the `-t` option is used, the changes apply only to the current configuration; otherwise, the changes apply to both

[18] For "canonical". While an unfortunate inconsistency, preserving `-p` across all `prop` subcommands is paramount.

the current and persistent configuration. For consistency with existing `dladm` subcommands that operate on persistent configuration, `-R` selects an alternate root directory[19].

The allowed values of each property are known to `dladm`. If an attempt is made to set an unknown property, to set a property to an invalid value, or to set a read-only property, an error is returned.

Example: set `powermode` to the value `fast`:

```
# dladm set-linkprop -p powermode=fast ath0
```

3.7 reset-linkprop

Usage: `dladm reset-linkprop [-t] [-R <rootdir>] [-p <prop>,...] <link>`

The `reset-linkprop` subcommand resets one or more properties to their default values on the specified `link`. If no properties are specified, all properties are reset. If the `-t` option is used, the reset applies only to the current configuration; otherwise, the changes apply to both the current and persistent configuration. For consistency with existing `dladm` subcommands that operate on persistent configuration, `-R` selects an alternate root directory.

If an attempt is made to reset an unknown or read-only property, an error is returned.

Example: reset `powermode` to its default value:

```
# dladm reset-linkprop -p powermode ath0
```

3.8 create-secobj

Usage: `dladm create-secobj [-t] [-R <rootdir>] [-f <file>] -c <class> <secobj>`

The `create-secobj` subcommand creates a secure object named `secobj` in the specified `class`. If the `-t` option is used, the secure object is only created temporarily; otherwise, the secure object is created persistently. For consistency with existing `dladm` subcommands that operate on persistent configuration, `-R` selects an alternate root directory.

The secure object value can either be input interactively or read from a file specified via the `-f` option. The sequence of interactive prompts and file format depends on the secure object class.

The only defined class is `wep`. The WEP key can be either 5 or 13 bytes long. It can be provided either as an ASCII or hexadecimal string; e.g., `12345` and `0x3132333435`[20] are equivalent 5-byte keys. A file containing a WEP key must consist of a single line using either WEP key format.

The `create-secobj` subcommand is only usable by users or roles that belong to the `Network Link Security` profile. If the named secure object already exists, an error is returned.

Example: non-interactively create the secure object `mykey` containing a WEP key with value `12345`:

```
# cat >/tmp/mykey.$$ <<-EOF
     12345
EOF
# dladm create-secobj -f /tmp/mykey.$$ -c wep mykey
# rm /tmp/mykey.$$
```

[19] Unfortunately, `-R` support will also complicate the implementation needed to change the underlying persistent configuration method, since the alternate root may be associated with a different release.

[20] The leading `0x` is optional to simplify cut-and-paste of keys – e.g., from a browser window.

3.9 delete-secobj

Usage: `dladm delete-secobj [-t] [-R <rootdir>] <secobj>[,...]`

The `delete-secobj` subcommand deletes one or more specified secure objects. If the `-t` option is used, the deletion applies only to the current configuration; otherwise, the deletion applies to both the current and persistent configuration. For consistency with existing `dladm` subcommands that operate on persistent configuration, `-R` selects an alternate root directory.

The `delete-secobj` subcommand is only usable by users or roles that belong to the `Network Link Security` profile. If an attempt is made to delete a nonexistent secure object, an error is returned.

Example: delete the secure object `mykey`:

```
# dladm delete-secobj mykey
```

3.10 show-secobj

Usage: `dladm show-secobj [-pP] [<secobj>,...]`

The `show-secobj` subcommand shows the name and class of one or more specified secure objects. If no object names are specified, all available[21] objects are shown. If the `-P` option is used, then the persistent secure objects are shown instead. If the `-p` option is used, then the output is returned in the established machine-parseable format. For each secure object, the following is displayed:

- `OBJECT`: The name of the secure object.
- `CLASS`: The class of the secure object.

Example: show all current secure objects:

```
# dladm show-secobj
OBJECT       CLASS
mykey        wep
#
```

3.11 WiFi Link Properties

The following WiFi link properties are currently defined:

- `channel`: Specifies the channel to use. This property can only be modified by certain WiFi links when in IBSS mode. The default value and allowed range of values varies by regulatory domain.
- `powermode`: Specifies the power management mode of the WiFi link. Possible values are `off` (disable power management), `max` (maximum power savings), and `fast` (performance-sensitive power management). Default is `off`.
- `radio`: Specifies whether the radio is `on` or `off`; default is `on`.
- `speed`: Specifies a fixed speed for the WiFi link, in megabits per second. The allowed set of values depends on the hardware and driver (but is shown by `show-linkprop`); common speeds include 1, 2, 11, and 54. By default, there is no fixed speed.

[21] Specifically: the set of persistently configured objects, minus any removed by `delete-secobj -t`, plus any added by `create-secobj -t`.