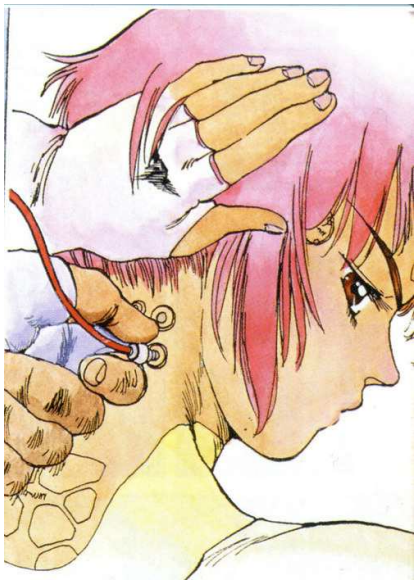
FQA 6 – Networking



6.1 – Before we go any further

Plan 9's approach to networking is unusual: most operations comprise reading and writing ("composing") byte streams ("files"). For the bulk of this document, it helps if you have read and at least partially understood *FQA 0.1 – What is Plan 9?*

Next, read: *The Organization of Networks in Plan 9*

If you are working with applications such as web servers, FTP servers, and mail servers, you may benefit greatly by reading the RFCs.

Note: A script for downloading all the RFCs is located in `/lib/rfc/grabrfc`. It copies the files into `/lib/rfc/`, and it may take hours for the script to run to completion.

6.2 – Network configuration

Basic networking is initially configured by the installation process. However, more complex settings or services may be desired. In Plan 9, network configuration is organized in `ndb`, the network database.

From `ndb(6)`:

> The network database consists of files describing machines known to the local installation and machines known publicly. The files comprise multi-line tuples made up of attribute/value pairs of the form attr=value or sometimes just attr. Each line starting without white space starts a new tuple. Lines starting with # are comments.
>
> The file /lib/ndb/local is the root of the database. Other files are included in

the database if a tuple with an attribute-value pair of attribute database and no value exists in /lib/ndb/local. Within the database tuple, each pair with attribute file identifies a file to be included in the database. The files are searched in the order they appear. For example:

```
database=
        file=/lib/ndb/common
        file=/lib/ndb/local
        file=/lib/ndb/global
```

declares the database to be composed of the three files `/lib/ndb/common`, `/lib/ndb/local`, and `lib/ndb/global`. By default, /lib/ndb/local is searched before the others. How- ever, `/lib/ndb/local` may be included in the database to redefine its ordering.

Within tuples, pairs on the same line bind tighter than pairs on different lines.

As mentioned, the installer adds basic information about the machine to the file `/lib/ndb/local`, based on the questions asked during the installation. This file may be edited to modify or expand the definition of the local network.

6.2.1 - Host name

Each machine on the network receives a corresponding section in `ndb`. The host name (hereafter referred to as `sysname`) is assigned by setting the `sys=` tuple:

```
sys=x301
```

The resulting `sysname` is used by the `/rc/bin/termrc` and `/rc/bin/cpurc` startup scripts, which in turn call upon any additional configuration that may exist in `/cfg/$sysname/`. (Look at the scripts to see how they deal with `/cfg`.)

6.2.2 - Identifying and setting up your network interfaces

Network interfaces are recognized by their MAC addresses, which are identified to `ndb` using the `ether=` tuple:

```
sys=x301 ether=00226811f7dd
```

Additional tuples in the same grouping will be used to configure the interface in question.

6.2.2.1 - WiFi

The following sections provide information pertaining to specific chipsets.

Read: `plan9.ini(8)`, *FQA Section 3.2 - Known Working Hardware*

6.2.2.1.1 – Interfaces

6.2.2.1.1.1 – wavelan

Lucent Wavelan (Orinoco) IEEE 802.11b and compatible PCMCIA cards. Compatible cards include the Dell TrueMobile 1150 and the Linksys Instant Wireless Network PC Card. Port and IRQ defaults are 0x180 and 3 respectively.

These cards take a number of unique options to aid in identifying the card correctly on the 802.11b network. The network may be ad hoc or managed (i.e. use an access point): `mode=[adhoc, managed]` and defaults to managed. The 802.11b network to attach to (managed mode) or identify as (ad hoc mode), is specified by `essid=string` and defaults to a null string. The card station name is given by `station=string` and defaults to Plan 9 STA. The channel to use is given by `channel=number` where number lies in the range 1 to 16 inclusive; the channel is normally negotiated automatically.

If the card is capable of encryption, the following options may be used: `crypt=[off, on]` and defaults to on. `keyN=string` sets the encryption key N (where N is in the range 1 to 4 inclusive) to string; this will also set the transmit key to N (see below). There are two formats for string which depend on the length of the string. If it is exactly 5 or 13 characters long it is assumed to be an alphanumeric key; if it is exactly 10 or 26 characters long the key is assumed to be in hex format (without a leading 0x). The lengths are checked, as is the format of a hex key. `txkey=number` sets the transmit key to use to be number in the range 1 to 4 inclusive. If it is desired to exclude or include unencrypted packets `clear=[off, on]` configures reception and defaults to inclusion.

The defaults are intended to match the common case of a managed network with encryption and a typical entry would only require, for example `essid=left-armpit key1=afish key2=calledraawaru` if the port and IRQ defaults are used. These options may be set after boot by writing to the device's ctl file using a space as the separator between option and value, e.g. `echo 'key2 1d8f65c9a52d83c8e4b43f94af' >/net/ether0/0/ctl` Card-specific power management may be enabled/disabled by `pm=[on, off]`

6.2.2.1.1.2 – wavelanpci

PCI Ethernet adapters that use the same Wavelan programming interface. Currently the only tested cards are those based on the Intersil Prism 2.5 chipset.

6.2.2.1.1.3 – iwl

Intel Wireless WiFi Link mini PCI-Express adapters require firmware from http://firmware.openbsd.org/firmware/iwn-firmware*.tgz to be present on attach in `/lib/firmware` or `/boot`. To select the access point, the `essid=` and `bssid=` parameters can be specified at boot or set during runtime like:

```
echo essid left-armpit >/net/ether1/clone
```

If both `essid=` and `bssid=` are specified, both must match. Scan results appear in the `ifstats` file and can be read out like:

```
cat /net/ether1/ifstats
```

Ad-hoc mode or WEP encryption is currently not supported. To enable WPA/WPA2 encryption, see `wpa(8)` for details.

6.2.2.1.1.4 – rt2860

Ralink Technology PCI/PCI-Express wireless adapters require firmware from http://firmware.openbsd.org/firmware/ral-firmware*.tgz to be present on attach in `/lib/firmware` or `/boot`. See the iwl section above for configuration details.

6.2.2.1.1.5 – wpi

Intel PRO Wireless 3945abg PCI/PCI-Express wireless adapters require firmware from http://firmware.openbsd.org/firmware/*/wpi-firmware*.tgz to be present on attach in `/lib/firmware` or `/boot`. See the iwl section above for configuration details.

6.2.2.1.2 – WPA

WPA1/TKIP and WPA2/CCMP are supported with the use of the `wpa(8)` command.

Read: `wpa(8)`

6.2.2.1.3 – WiFi Roaming

A script can be used to dynamically re-associate with available wifi access points:

http://plan9.stanleylieber.com/rc/wifiroam

Example usage:

```
@{wifiroam attwifi | aux/statusmsg -k wifiroam} &
```

6.2.2.1.4 – WiFi Debug

For cards that use the wifi layer, debug prints (**note:** will appear on the console) may be enabled with:

```
echo debug >´#l0/ether0/clone´
# change this to suit if wifi interface is not #l0
```

or by adding `debug=1` to the interface definition in `plan9.ini`.

Read: `plan9.ini(8)`

6.2.3 – IP address The ip= tuple is used to associate an IP address with the machine:

```
sys=x301 ether=00226811f7dd ip=192.168.0.31
```

If no ip= tuple is present, the boot scripts will attempt to bring up the interface using DHCP (see below). If multiple ip= tuples are present, an ipnet entry is required for each.

6.2.4 – Default gateway

The default gateway is configured using the ipgw= tuple, usually under an ipnet= section that defines default settings for an entire subnet:

```
ipnet=9front ip=192.168.0.0 ipmask=255.255.255.0 ipgw=192.168.0.1
```

but it may also be specified on a per-machine basis:

```
sys=x301 ether=00226811f7dd ip=192.168.0.31 ipgw=192.168.0.1
```

Note: Tuples included in the definition of a machine supercede those defined for the network to which the machine belongs.

6.2.5 – DNS Resolution

DNS resolvers may be specified using the dns= tuple, and may be configured for an entire network:

```
ipnet=9front ip=192.168.0.0 ipmask=255.255.255.0 ipgw=192.168.0.1
        dns=192.168.0.1
```

or on a per-machine basis:

```
sys=x301 ether=00226811f7dd ip=192.168.0.31 dns=192.168.0.1
```

These changes will take effect after a reboot. To configure a DNS resolver on the fly, it is possible to manually edit /net/ndb:

```
ip=192.168.0.31 ipmask=255.255.255.0 ipgw=192.168.0.1
        sys=x301
        dom=x301.9front
        dns=192.168.0.1
        # add or modify dns= lines to associate the DNS
        # server 192.168.0.1 with the running system
```

Note: /net/ndb is a synthetic file that represents the current operating state. It does not persist across reboots, and is only pre-populated when system networking was configured via DHCP. Changes to a blank /net/ndb file will match on the ip= tuple.

Read: ip(3)

Finally, to turn on debug in dns:

```
echo -n debug >/net/dns
```

6.2.5.1 – Caching DNS server

To run a caching DNS server, modify `/cfg/$sysname/termrc` or `/cfg/$sysname/cpurc` (whichever is appropriate) to include the following:

```
ndb/dns -rs
```

The caching DNS server will be started at boot time.

Next, modify `/lib/ndb/local` such that the desired machines will use the IP address of the new caching DNS server as their DNS server, either by changing the `dns=` tuple under the `ipnet` of the corresponding network or by adding a `dns=` tuple to the line of each desired machine.

Read: `ndb(6)` and `ndb(8)`

6.2.5.2 – DNS authoritative name server

An authoritative domain name record, with associated reverse-lookup and sub-domains, looks like this:

```
dom=bell-labs.co soa=
        refresh=10800 ttl=10800
        # serial is automatically maintained if omitted
        serial=2012110732
        ns=ns5.he.net
        ns=ns4.he.net
        ns=ns3.he.net
        ns=ns2.he.net
        ns=nm.iawtp.com
        ns=pp.iawtp.com
        ns=mars2.iawtp.com
        dnsslave=slave.dns.he.net
        mb=sl@stanleylieber.com
        mx=pp.inri.net pref=5
        mx=nm.inri.net pref=10
        mx=mars2.inri.net pref=15
        txt="v=spf1 mx -all"

dom=125.191.107.in-addr.arpa soa=
        refresh=3600 ttl=3600
        ns=nm.iawtp.com

dom=bell-labs.co ip=107.191.125.208

dom=www.bell-labs.co cname=bell-labs.co
```

An FQDN may be assigned to an existing machine by adding the `dom=` tuple to its definition:

```
sys=x301 dom=x301.bell-labs.co ether=00226811f7dd ip=192.168.0.31
```

Note: The dnsslave entries specify slave DNS servers that should be notified when the domain changes. The notification service also requires the `-n` flag:

```
ndb/dns -nrs
```

Read: ndb(8)

6.2.5.2.1 – Troubleshooting DNS authoritative name server

An online tool that evaluates the DNS configuration of a given domain name is available at: https://intodns.com

6.2.6 – Network-wide configuration

Settings for an entire network subnet may be defined under an ipnet= tuple:

```
ipnet=9front ip=192.168.0.0 ipmask=255.255.255.0
        ipgw=192.168.0.1
        auth=192.168.0.2
        authdom=9front
        fs=192.168.0.3
        cpu=192.168.0.4
        dns=192.168.0.1
        dnsdomain=9front
        smtp=192.168.0.4

# ethernet/wifi router
sys=onoff dom=onoff.9front ip=192.168.0.1

# auth server
sys=auth dom=auth.9front ether=00d059b6dac8 ip=192.168.0.2
        bootf=/386/9bootpxe

# cpu server
sys=cpu dom=cpu.9front ether=001125149137 ip=192.168.0.4
        bootf=/386/9bootpxe

# file server
sys=fs dom=fs.9front ether=001641360117 ip=192.168.0.3

# terminal
sys=x301 dom=x301.9front ether=00226811f7dd ip=192.168.0.31
        bootf=/386/9bootpxe
```

Note: Each ipnet entry matches on a single IP prefix. Attributes missing from an ipnet entry can be inhereted from more inclusive entries with shorter prefixes.

6.2.7 – Activating the changes

6.2.7.1 – NIC

Network interfaces are automatically initialized at boot time. To make a manual change without rebooting, use the ipconfig(8) command:

```
ip/ipconfig -g 192.168.0.1 ether /net/ether0 \
        192.168.0.31 255.255.255.0
```

6.2.7.2 – cs

To refresh the network database **NOW** after changing `/lib/ndb/local`:

```
echo -n refresh > /net/cs
```

6.2.7.3 – dns

```
echo -n refresh > /net/dns
```

6.2.8 – Verifying network settings

```
% cat /net/ndb
ip=192.168.0.31 ipmask=255.255.255.0 ipgw=192.168.0.1
        sys=x301
        dom=x301.9front
        auth=192.168.0.2
        dns=192.168.0.1
```

6.2.8.1 – Checking routes

```
% cat /net/iproute
0.0.0.0         /96  192.168.0.1     4    none   -
192.168.0.0     /120 192.168.0.0     4i   ifc    0
192.168.0.0     /128 192.168.0.0     4b   ifc    -
192.168.0.31    /128 192.168.0.31    4u   ifc    0
192.168.0.255   /128 192.168.0.255   4b   ifc    -
255.255.255.255 /128 255.255.255.255 4b   ifc    0
```

6.2.8.1.1 – Adding static routes

Route requests for `192.168.1.0/24` through the gateway `192.168.0.99` (which itself must already be accessible via the existing network configuration):

```
echo ´add 192.168.1.0 255.255.255.0 192.168.0.99´ >/net/iproute
```

Note: Manual configurations such as this may be added to optional boot scripts created in `/cfg/$sysname/`.

Read: `ip(3)`

6.2.9 – Setting up your 9front box as a forwarding gateway

Read: `ip(3)`

6.2.10 – Setting up aliases on an interface

Read: `ip(3)`

6.3 – How do I filter and firewall with 9front?

No.

6.4 – Dynamic Host Configuration Protocol (DHCP)

6.4.1 – DHCP client

In `/lib/ndb/local`, if no `ip=` tuple is present in the machine's definition, the boot scripts will attempt to obtain an IP address via DHCP.

To obtain a DHCP lease manually:

```
ip/ipconfig
```

Read: `ipconfig(8)`

6.4.2 – DHCP server

From `dhcpd(8)`:

[Continued on next page]

Dhcpd runs the BOOTP and DHCP protocols. Clients use these protocols to obtain configuration information. This information comes from attribute/value pairs in the network database (see ndb(6) and ndb(8)). DHCP requests are honored both for static addresses found in the NDB and for dynamic addresses listed in the command line. DHCP requests are honored if either:
- there exists an NDB entry containing both the ethernet address of the requester and an IP address on the originating network or subnetwork.
- a free dynamic address exists on the originating network or subnetwork.

A BOOTP request is honored if all of the following are true:
- there exists an NDB entry containing both the ethernet address of the requester and an IP address on the originating network or subnetwork.
- the entry contains a bootf= attribute
- the file in the bootf= attribute is readable.

Dynamic addresses are specified on the command line as a list of addresses and number pairs. For example,

```
     ip/dhcpd 10.1.1.12 10 10.2.1.70 12
```

directs dhcpd to return dynamic addresses 10.1.1.12 through 10.1.1.21 inclusive and 10.2.1.70 through 10.2.1.81 inclusive.

Dhcpd maintains a record of all dynamic addresses in the directory /lib/ndb/dhcp, one file per address. If multiple servers have access to this common directory, they will correctly coordinate their actions.

Attributes come from either the NDB entry for the system, the entry for its subnet, or the entry for its network. The system entry has precedence, then the subnet, then the network. The NDB attributes used are:

ip	the IP address
ipmask	the IP mask
ipgw	the default IP gateway
dom	the domain name of the system
fs	the default Plan 9 file server
auth	the default Plan 9 authentication server
dns	a domain name server
ntp	a network time protocol server
time	a time server
wins	a NETBIOS name server
www	a World Wide Web proxy
pop3	a POP3 mail server
smtp	an SMTP mail server
bootf	the default boot file; see ndb(6)

Dhcpd will answer BOOTP requests only if it has been specifically targeted or if it has read access to the boot file for the requester. That means that the requester must specify a boot file in the request or one has to exist in NDB for dhcpd to answer. Dhcpd will answer all DHCP requests for which it can associate an IP address with the requester.

To configure a DHCP server on your system:

```
mkdir /lib/ndb/dhcp
```

and then modify `/cfg/$sysname/cpurc` or `/cfg/$sysname/termrc` (whichever is appropriate) to start `dhcpd` and `tftpd` at boot time:

```
ip/dhcpd
```

Read: `dhcpd(8)`

6.5 – PPP

Read: Dailup modem config at the Bell Labs Plan 9 wiki.

6.6 – Setting up a network bridge in 9front

Read: `bridge(3)` and `ip(3)`

6.7 – How do I boot from the network?

First, read *FQA 7.3.3 – Setting up a listener for network connections.* The file server should already be running a listener, and an auth server should already be configured and running on the network.

6.7.1 – How do I tcp boot?

It is possible to boot from local media and then mount the root file system over the network. At the `bootargs` prompt, type `tls` (the old `tcp` boot option will still work but is not recommended because the connection will not be encrypted). At this point, `ip/ipconfig` will determine network parameters using DHCP. When file (`fs`) or authentication (`auth`) server IP addresses could not be determined over DHCP then the boot process will prompt for those. When prompted for a `user`, enter a valid username and password that has already been configured on the auth server. The machine should then proceed to mount its root file system from the file server.

Note: Values for `fs` and `auth` may be added to `plan9.ini`.

Read: `plan9.ini(8)`

6.7.1.1 – Passing arguments to ipconfig at the bootargs prompt

When a DHCP server is not available, you may still tcp boot by configuring networking manually at the `bootargs` prompt. Everything after `tcp!` is passed as arguments to the `ipconfig` command.

At the prompt:

```
bootargs is (tcp, tls, il, local!device) [tcp]
```

enter something like the following:

```
tls!-g 192.168.0.1 ether /net/ether0 192.168.0.23 255.255.255.0
```

where `192.168.0.1` is the gateway, `192.168.0.23` is the static IP address and `255.255.255.0` the subnet mask.

Read: `ipconfig(8)`

6.7.2 – How do I boot using PXE?

It is also possible to PXE boot a system.

On the file server, add the following lines to `/cfg/$sysname/cpurc` to start `dhcpd` and `tftpd` at boot time:

```
ip/dhcpd
ip/tftpd
```

Add an entry for `tftp` under the appropriate `ipnet=` tuple in `/lib/ndb/local`:

```
ipnet=9front ip=192.168.0.0 ipmask=255.255.255.0 ipgw=192.168.0.1
        auth=192.168.0.2
        authdom=9front
        cpu=192.168.0.4
        dns=192.168.0.1
        dnsdomain=9front
        smtp=192.168.0.4
        tftp=192.168.0.3
```

Reboot the file server:

```
fshalt -r
```

To configure machines that will PXE boot from the file server, edit `/lib/ndb/local` on the file server and add a `bootf=` (boot file) tuple to the line representing each machine:

```
sys=x301 dom=x301.9front ether=00226811f7dd ip=192.168.0.31
  bootf=/386/9bootpxe
```

For the system `x301` we would then create a file `/cfg/pxe/00226811f7dd` on the file server to serve as its `plan9.ini`:

```
bootfile=/386/9pc
bootargs=tls
nobootprompt=tls
auth=192.168.0.2
fs=192.168.0.3
mouseport=ps2intellimouse
monitor=vesa
vgasize=1440x900x32
*acpi=1
user=sl
```

Note: The `user=` parameter refers to a single username that has been added both to the file server (for file permissions) as well as to the auth server (for network authentication).

If a file matching the remote system's MAC address is not found under `/cfg/pxe/`, the file `/cfg/pxe/default` (if it exists) will be used instead.

Finally, boot the desired remote systems via PXE. When prompted for a `user`, enter a valid username and password that has already been configured on the auth server. The remote system should now proceed to boot from the file server.