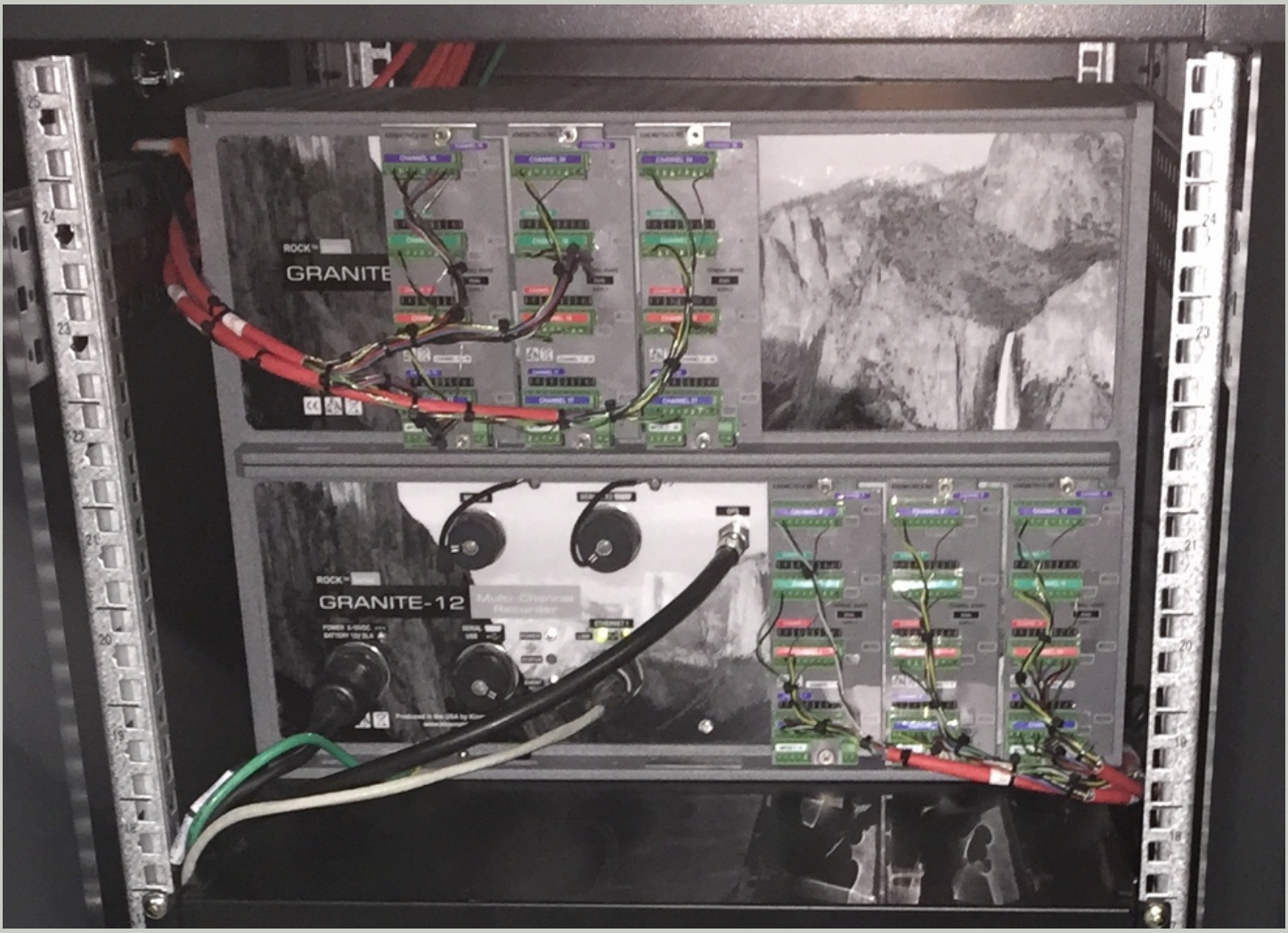
USGS
science for a changing world

Granite IP Network Default Route Disappearance—Diagnosis and Solution



Open-File Report 2018–1117

U.S. Department of the Interior
U.S. Geological Survey

Cover. Photograph of Kinemetrics, Inc., Granite strong-motion recorders operated by the U.S. Geological Survey National Strong Motion Project (NSMP). NSMP operates numerous strong-motion seismographs to monitor ground shaking and structural response caused by large, nearby earthquakes.

Granite IP Network Default Route Disappearance—Diagnosis and Solution

By Lawrence M. Baker

Open-File Report 2018–1117

U.S. Department of the Interior
U.S. Geological Survey

U.S. Department of the Interior
RYAN K. ZINKE, Secretary

U.S. Geological Survey
James F. Reilly II, Director

U.S. Geological Survey, Reston, Virginia: 2018

Suggested citation:
Baker, L.M., 2018, Granite IP network default route disappearance—Diagnosis and solution: U.S. Geological Survey Open-File Report 2018–1117, 35 p., https://doi.org/10.3133/ofr20181117.

ISSN 2331-1258 (online)

Preface

This report is highly technical and specialized. Readers are expected to understand Transmission Control Protocol/Internet Protocol (TCP/IP) computer-networking concepts and to have at least an intermediate level of competence administering a Linux system using command-line tools and shell scripts. Readers should be able to understand code fragments taken from shell scripts and C programs. Linux networking commands that might not be as familiar to the reader have links to explanatory Web pages in the appendix.

This report uses the following typographical conventions:

- When a technical term or phrase is first introduced, it is italicized:

  link down

- A typewriter-style font is used when the text refers to an item on a computer, such as the name of a program: `netwatcher`, a device name: `eth0`, a file name: `/etc/network/interfaces`, or a C program constant: `NETLINK_ROUTE`.

- A shell function name or C program function name is followed by parentheses:

  ```
  link_failure()
  ```

- Message text that would appear in a terminal window also uses a typewriter-style font:

  ```
  eth0: 10Base-T (RJ-45) has no cable
  ```

- Text captured from a terminal window is shown inside shaded boxes:

```
NETDEV WATCHDOG: eth0: transmit queue 0 timed out
```

  The text appears exactly as it does in the terminal window.

- Commands that are typed in the terminal window are shown in bold, usually followed by the output:

```
# lsmod | grep cs
cs89x0_0                6889  0
```

- Remarks to the reader are italicized.

Note that `eth0` *is configured as* `auto`*, not as* `allow-hotplug`*.*

- Especially important remarks to the reader are italicized and bold:

Unplug the Ethernet cable here.

Contents

Granite IP Network Default Route Disappearance—Diagnosis and Solution

By Lawrence M. Baker

1. Introduction

The U.S. Geological Survey (USGS) National Strong Motion Project (NSMP) operates numerous strong-motion seismographs to monitor ground shaking and structural response caused by large, nearby earthquakes. This report describes a problem NSMP scientists encountered communicating over the Internet with several Kinemetrics, Inc., Granite strong-motion recorders.

The Granite strong-motion recorders ("Granites") get into a state where they cannot be reached from the Internet and they cannot reach the Internet, yet they can reach and be reached from the local Ethernet subnet. The reason is that the Internet Protocol (IP) network default route has disappeared; only the local route is available. Diagnosis is complicated by the unpredictability of the circumstances leading to the failure. The failures have happened at several field sites but cannot be reproduced in the lab.

To recover from this situation, Kinemetrics wrote a Linux (the operating system that runs a Granite) `netwatcher` daemon, which `ping`'s a known external IP address. If the IP address becomes unreachable, the `netwatcher` daemon restarts the Linux `networking` service. Restarting networking is a brute-force solution. However, it does reestablish Internet connectivity.

The part of the `netwatcher` daemon that performs this operation is:

```
while true
do
    sleep 100
    ping -n -q -w 5 -c 2 $PING_HOST > /dev/null 2>&1
    sleep 20
    ping -n -q -w 5 -c 1 $PING_HOST > /dev/null 2>&1

    if [ $? == 0 ]; then
        FAIL_COUNT=0
    else
        if (( ++FAIL_COUNT >= FAIL_LIMIT )); then
           FAIL_COUNT=0
           Say "Restarting network..."
           /etc/init.d/networking restart > /dev/null
        fi
    fi
done &
```

When the `netwatcher` daemon restarts networking, it writes a "`Restarting network . . .`" message to the system log files. No further diagnostic information is provided.

The disappearance of the default route is an indication that the Linux operating system in the Granites is likely not properly responding to Ethernet *link* (the physical connection between two network devices) state transitions—from *link up* (*carrier on*) to *link down* (*carrier off*) and from link down (carrier off) to link up (carrier on). (Link state is referred to as carrier state in the Linux kernel.) This happens, for example, when a Granite's Ethernet cable is unplugged or plugged in or the device on the opposite end of a Granite's Ethernet cable powers off or powers on.

Linux `netlink` `NETLINK_ROUTE` event notification messages provide a mechanism for user-space programs to monitor the link state of a network interface. Using this mechanism, a Linux network link monitoring daemon, such as `netplugd`, can be used to adapt the IP networking environment to a new link state by running a script to bring an interface up when the link is established and to bring an interface down when the link is lost.

Link state-change event notification messages are sent to `netlink` subscribers when a network device driver notifies the Linux kernel of a change in link (carrier) state. However, the Granite Ethernet drivers do not detect link state changes. Without link state-change event notification messages, the `netplugd` daemon is not useful.

This report describes the IP networking behavior of a Granite system and provides modifications to the Granite Ethernet device drivers to send Ethernet link (carrier) state-change event notifications to the Linux kernel. With these modifications, the Linux `netplugd` daemon can be configured to properly reconfigure Granite IP networking when the Ethernet interface link state changes.

It is expected that these modifications will solve the problem of the disappearance of the IP network default route and there will no longer be a need to run the Kinemetrics `netwatcher` daemon. However, because the failure cannot be reproduced on demand, that is not certain. The Linux `netplugd` daemon can coexist with the Kinemetrics `netwatcher` daemon. If the problem is solved, there should be no `netwatcher` "`Restarting network . . .`" messages in the system log files. When the Ethernet link state changes, there should instead be "`eth0: 10Base-T (RJ-45) has no cable`" messages when the link is lost, and "`eth0: using half-duplex 10Base-T (RJ-45)`" messages when the link is reestablished, and the IP network default route should no longer disappear unexpectedly.

1.2. Electronic Supplement

The electronic supplement to this report contains the following:

- `/etc/init.d/ipmonitor`—The shell script for the `ipmonitor` service in section 4.1, "How to Enable `netlink NETLINK_ROUTE` Message Logging," which logs Linux kernel `netlink NETLINK_ROUTE` event notification messages.
- `/usr/local/src/cs89x0.patch`—The patch file in section 6, "Modifications to the `cs89x0_x` *`Device Driver`*," to fix the Granite Ethernet device drivers so IP networking can be properly reconfigured when the Ethernet interface link state changes.
- `/lib/modules/2.6.35.14/kernel/drivers/net/cs89x0_0 and cs89x0_1`—The modified Granite Ethernet Linux kernel driver modules.

2. Linux Networking on a Granite/Slate

2.1. Linux Networking Software on a Granite/Slate

Granites, and their close siblings, Slates (a Granite without data acquisition hardware), run the Linux Long Term Support (LTS) kernel, version 2.6.35.14 (https://www.kernel.org/pub/linux/kernel/v2.6/long-term/v2.6.35/linux-2.6.35.14.tar.gz, accessed August 1, 2016):

```
# uname -a
Linux BakerTest 2.6.35.14 #423 PREEMPT Thu Aug 21 12:48:14 PDT 2014 armv5tel GNU/Linux
```

Granites/Slates use the Debian (https://www.debian.org, accessed August 1, 2016) ARMEL Linux version 6.0.2 *squeeze* distribution:

```
# cat /etc/debian_version
6.0.2
```

The Ethernet interfaces (`eth0` on Granites and Slates, `eth1` on Slates only) are configured using the Kinemetrics `netconfig` command. `netconfig` creates the network interfaces configuration file, `/etc/network/interfaces`, which is used by the Linux `networking` service to start each networking interface.

USGS NSMP configures the Granite `eth0` Ethernet interface to use a fixed IP address:

```
# cat /etc/network/interfaces
# /etc/network/interfaces config file

auto lo eth0

iface lo inet loopback

iface eth0 inet static
    address 192.168.99.11
    netmask 255.255.255.0
    broadcast 192.168.99.255
    gateway 192.168.99.1
```

Note that `eth0` *is configured as* `auto`*, not as* `allow-hotplug`*.*

Debian Linux manages network connections using the `ifup` and `ifdown` commands from the `ifupdown` package. As their names imply, `ifup` brings a network interface up and `ifdown` brings a network interface down. (See section 3.4, "The Linux Network Interface Up (`ifup`) and Interface Down (`ifdown`) Commands.")

Network interfaces are dynamically created when first referenced by the Linux kernel device manager, `udev`. `udev` loads the device driver for `eth0` and the network interface is registered (displayed by `ifconfig`, for example) when the Linux `networking` service starts. There is no attempt to create `eth1` because there is no entry for `eth1` in the `networking` service configuration file. An attempt to create `eth1` on a Granite would fail in any case because the hardware for `eth1` is not fully implemented.

More detailed information about Linux networking is available in Understanding Linux Network Internals (Benvenuti, 2005), especially on:

- Link State Change Detection
  - Chapter 8, p. 163, describes the calls a network device driver makes to `netif_carrier_on()` and `netif_carrier_off()` when it detects the presence or absence of carrier (link up or link down for an Ethernet interface), respectively.
- Routing
  - Chapter 30, p. 795, explains the difference between the routing table and the routing cache.
  - Chapter 32, p. 856, describes the impact of changes in device status on the routing tables.
  - Chapter 33, p. 881, describes `link_failure()` (for example, `ipv4_link_failure()` in `net/ipv4/route.c`), which is called when a destination becomes unreachable.

2.2. Ethernet Networking Hardware on a Granite/Slate

Note that the Granite/Slate Ethernet interfaces do not auto-negotiate. Any Ethernet device connected to a Granite/Slate must be configured for 10Base-T, half-duplex operation.

Granites/Slates use Cirrus Logic CS8900 low-power 10 Mbit Ethernet controller chips (https://www.cirrus.com/en/products/pro/detail/P46.html, accessed August 1, 2016). The CS8900 does not auto-negotiate link-level settings (duplex, flow control).

The Cirrus Logic CS8900A Product Data Sheet (Cirrus Logic, 2015), section 5.8 Auto-Negotiation Considerations, discusses auto-negotiation and its effect on the assertion of Link-OK status:

> The original IEEE 802.3 specification requires the MAC to wait until 4 valid link-pulses are received before asserting Link-OK. Any time an invalid link-pulse is received, the count is restarted.

When auto-negotiation occurs, a transmitter sends FLPs (auto-negotiation Fast Link Pulses) bursts instead of the original IEEE 802.3 NLP (Normal Link Pulses).

If the hub is attempting to auto-negotiate with the CS8900A, the CS8900A will never get more than 1 "valid" link pulse (valid NLP). This is not a problem if the CS8900A is already sending link-pulses, because when the hub receives NLPs from the CS8900A, the hub is required to stop sending FLPs and start sending NLPs. The NLP transmitted by the hub will put the CS8900A into Link-OK.

However, if the CS8900A is in Auto-Switch mode, the CS8900A will never send any link-pulses, and the hub will never change from sending FLPs to sending NLPs.

The CS8900A does not understand auto-negotiation Fast Link Pulses (FLPs). This is understandable, because auto-negotiation did not exist prior to 100Base-TX Ethernet. When connected to certain switch hardware, the result can be the CS8900 never asserts Link-OK. The solution is to try different switch hardware.

Each Ethernet interface (`eth0` on Granites and Slates, `eth1` on Slates only) has its own loadable kernel driver module—`cs89x0_0` for `eth0` and `cs89x0_1` for `eth1`:

```
# lsmod | grep cs
cs89x0_0                6889  0
```

`cs89x0_0` and `cs89x0_1` are custom versions of the standard Linux device driver for the Cirrus Logic CS8900, `cs89x0` (`rock1_cs89.tgz`; Jason DiDonato, Kinemetrics, Inc., written commun., June 2016). The Kinemetrics' modified drivers specify the I/O addresses used in a Granite/Slate, remove DMA support, and force 10Base-T, half-duplex operation.

Neither the standard Linux `cs89x0` driver, nor the modified `cs89x0_0` and `cs89x0_1` drivers, include support for Ethernet link state query, or any options to query or change media options, such as duplex and flow control. There is also no support for `mii-tool` or `ethtool`. Furthermore, none of these drivers detect and notify the Linux kernel of link state changes. The hardware has no interrupt for Link-OK state transitions, and the drivers do not poll the Link-OK status bit to detect link state changes.

3. Investigations

To understand the Linux IP network behavior on a Granite, I booted the Granite with and without an Ethernet cable plugged in, removed it and inserted it, and observed the IP network configuration before and after the changes. This section describes the tests performed and some of the key components involved.

3.1. Test of a Granite With an Ethernet Cable Plugged In

I first booted the Granite with an Ethernet cable plugged in (link up), then removed it (link down), and observed the IP network configuration before and after unplugging the Ethernet cable.

Note that the Kinemetrics `netwatcher` *daemon is not enabled.*

Display any Linux kernel messages containing the device name of an Ethernet interface:

```
# dmesg | grep -w eth[0-9]
Rock Option: eth0 set to power save mode.
Rock Option: eth1 set to power save mode.
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
eth0: using half-duplex 10Base-T (RJ-45)
eth0: no IPv6 routers present
```

Display any Linux kernel messages containing the module name of an Ethernet interface device driver:

```
# dmesg | grep cs89.0
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
cs89x0 media RJ-45, IRQ 139, programmed I/O, MAC 00:50:c2:5a:c5:73
```

Display the Linux kernel IP network routing table:

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.99.0    *               255.255.255.0   U     0      0        0 eth0
default         192.168.99.1    0.0.0.0         UG    0      0        0 eth0
```

Both the route to the local subnet, destination `192.168.99.0` using interface `eth0`, and the default route, through the gateway at `192.168.99.1`, are shown.

The `ifupdown` run state is maintained in `/etc/network/run/ifstate`:

```
# cat /etc/network/run/ifstate
lo=lo
eth0=eth0
```

Unplug the Ethernet cable here.

When the cable is unplugged, the console displays:

```
NETDEV WATCHDOG: eth0: transmit queue 0 timed out
```

The `NETDEV WATCHDOG` message now appears in the Linux kernel messages containing the device name of an Ethernet interface:

```
# dmesg | grep -w eth[0-9]
Rock Option: eth0 set to power save mode.
Rock Option: eth1 set to power save mode.
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
eth0: using half-duplex 10Base-T (RJ-45)
eth0: no IPv6 routers present
NETDEV WATCHDOG: eth0: transmit queue 0 timed out
```

There is no change to the Linux kernel messages containing the module name of an Ethernet interface device driver:

```
# dmesg | grep cs89.0
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
cs89x0 media RJ-45, IRQ 139, programmed I/O, MAC 00:50:c2:5a:c5:73
```

The Linux kernel IP network routing table is the same:

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.99.0    *               255.255.255.0   U     0      0        0 eth0
```

Note that it takes several seconds to display the `default` *route.*

```
default         192.168.99.1    0.0.0.0         UG    0      0        0 eth0
```

The `ifupdown` run state still includes `eth0`:

```
# cat /etc/network/run/ifstate
lo=lo
eth0=eth0
```

This result implies `ifdown` has not run. Why might that be the case?:

- No network link monitoring daemon is running that would run `ifdown` when the cable is unplugged.
- `NetCheck()` in the Kinemetrics `kwdping` daemon (see section 3.3, "The Kinemetrics `kwdping` Daemon") thinks the interface is fine as long as `ifconfig` shows an assigned `inet addr`, not whether the interface is in the `RUNNING` state. Otherwise, `NetCheck()` would have run `ifdown`.

The faulty `NetCheck()` test does not matter in any case, because `ifconfig` continues to show `eth0` is `RUNNING`, even though the cable is unplugged.

The Granite was left in this state overnight waiting for the default route to be removed, but that never happened.

3.2. Test of a Granite Without an Ethernet Cable Plugged In

Next, I booted the Granite with the Ethernet cable unplugged (link down), then plugged it in (link up). I observed the IP network configuration before and after plugging in the Ethernet cable.

Note that the Kinemetrics `netwatcher` daemon is not enabled.

The results are inconsistent when booting without an Ethernet cable plugged in. Usually `dmesg` contains the "`eth0: no network cable attached to configured media`" messages. However, sometimes `dmesg` shows exactly the same messages as when booting with an Ethernet cable plugged in, with an immediate "`NETDEV WATCHDOG: eth0: transmit queue 0 timed out`" message. In that case, `route` behaves as it did above when booting with an Ethernet cable plugged in, after the Ethernet cable has been unplugged.

Display any Linux kernel messages containing the device name of an Ethernet interface:

```
# dmesg | grep -w eth[0-9]
Rock Option: eth0 set to power save mode.
Rock Option: eth1 set to power save mode.
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
```

Display any Linux kernel messages containing the module name of an Ethernet interface device driver:

```
# dmesg | grep cs89.0
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
cs89x0 media RJ-45, IRQ 139, programmed I/O, MAC 00:50:c2:5a:c5:73
```

Display the Linux kernel IP network routing table:

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```

Because the `eth0` interface has never been up, no routes have ever been configured for it. Once a minute the console displays:

```
eth0: no network cable attached to configured media
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
```

The `ifupdown` run state is maintained in `/etc/network/run/ifstate`:

```
# cat /etc/network/run/ifstate
lo=lo
```

Plug in an Ethernet cable here.

When the cable is plugged in, the console displays:

```
eth0: using half-duplex 10Base-T (RJ-45)
```

The "`eth0: using half-duplex 10Base-T (RJ-45)`" message now appears in the Linux kernel messages containing the device name of an Ethernet interface:

```
# dmesg | grep -w eth[0-9]
Rock Option: eth0 set to power save mode.
Rock Option: eth1 set to power save mode.
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
```

The last two lines repeat many times.

```
eth0: using half-duplex 10Base-T (RJ-45)
eth0: no IPv6 routers present
```

There is no change to the Linux kernel messages containing the module name of an Ethernet interface device driver:

```
# dmesg | grep cs89.0
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
cs89x0 media RJ-45, IRQ 139, programmed I/O, MAC 00:50:c2:5a:c5:73
```

The Linux kernel IP network routing table now contains the route to the local subnet, destination `192.168.99.0` using interface `eth0`, and the default route, through the gateway at `192.168.99.1`—the same routes that are shown when the Granite is booted with an Ethernet cable plugged in:

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.99.0    *               255.255.255.0   U     0      0        0 eth0
default         192.168.99.1    0.0.0.0         UG    0      0        0 eth0
```

The `ifupdown` run state now includes `eth0`:

```
# cat /etc/network/run/ifstate
lo=lo
eth0=eth0
```

This result implies `ifup` has run. Why might that be the case?:

- No network link monitoring daemon is running that would run `ifup` when the cable is plugged in.
- `NetCheck()` in the Kinemetrics' `kwdping` daemon (see section 3.3, "The Kinemetrics `kwdping` Daemon") checks whether `ifconfig` shows an assigned `inet addr` for the interface. It does not. Therefore, `NetCheck()` runs `ifup`, which causes "`eth0: using half-duplex 10Base-T (RJ-45)`" to be displayed on the console.

The Granite was left in its initial state overnight prior to plugging in the cable, but that made no difference.

3.3. The Kinemetrics `kwdping` Daemon

When an Ethernet cable is not plugged in, once a minute the console displays:

```
eth0: no network cable attached to configured media
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
```

These messages are coming from the `NetCheck()` function in the Kinemetrics `kwdping` daemon, `/sbin/kwdpingd`, which is run by the `kwdping` service. The `NetCheck()` function is:

```
NetCheck ()
{
    if [ ! -f "$NET_CFG_FILE" ]; then
        return
    fi

    local cfgInfo=`cat $NET_CFG_FILE | grep auto`
    local active=`ifconfig`
    local ethDev=""

    ##### eth0
    ethDev="eth0"

    echo "$cfgInfo" | grep -q "$ethDev"

    if [ $? = 0 ]; then

        echo "$active" | grep -q "$ethDev"

        if [ $? = 1 ]; then
            ifdown "$ethDev" > /dev/null 2>&1; sleep 1;  ifup "$ethDev"
        else

            ifconfig "$ethDev" 2> /dev/null | grep -q "inet addr"

            if [ $? = 1 ]; then
                ifdown "$ethDev" > /dev/null 2>&1; sleep 1;  ifup "$ethDev"
            fi
        fi
    fi

    ##### eth1
    ethDev="eth1"

    echo "$cfgInfo" | grep -q "$ethDev"

    if [ $? = 0 ]; then

        echo "$active" | grep -q "$ethDev"

        if [ $? = 1 ]; then
            ifdown "$ethDev" > /dev/null 2>&1; sleep 1;  ifup "$ethDev"
```

```
        else

            ifconfig "$ethDev" 2> /dev/null | grep -q "inet addr"

            if [ $? = 1 ]; then
                ifdown "$ethDev" > /dev/null 2>&1; sleep 1;  ifup "$ethDev"
            fi
        fi
    fi
}
```

`NetCheck()` is one of several tasks run once a minute by the `kwdping` daemon. That part of the `kwdping` daemon is:

```
while true
do
    [ -f "$PROC_FILE" ] && echo ":WDPING:" > "$PROC_FILE"
    FSTest "/"
    FSTest "/opt/"
    RHPingCheck
    sleep 60
    NetCheck
Done
```

Among the tasks performed by the `kwdping` daemon is `RHPingCheck()`, which refreshes the RockHound hardware watchdog timer. (RockHound is the Kinemetrics data acquisition software that runs atop Linux on a Granite/Slate.) The watchdog timer will reboot the system if it is not refreshed before expiring, such as when the RockHound software is hung.

3.4. The Linux Network Interface Up (`ifup`) and Interface Down (`ifdown`) Commands

Debian Linux uses the `ifup` and `ifdown` commands from the `ifupdown` package to manage network interfaces. `ifup` brings a network interface up and `ifdown` brings a network interface down. The Linux `networking` service, for example, runs `ifup` to start any network interfaces configured as `auto` in `/etc/network/interfaces`.

For interfaces configured as `static inet` in `/etc/network/interfaces`, `ifup` executes the following commands in `inet.defn` (from the Debian `ifupdown` source archive; Debian.org, 2016):

```
up
    [[ ifconfig %iface% hw %hwaddress%]]
    ifconfig %iface% %address% netmask %netmask% [[broadcast %broadcast%]] \
      [[pointopoint %pointopoint%]] [[media %media%]] [[mtu %mtu%]] \
      up
    route add -net %network%  \
        if ( mylinuxver() < mylinux(2,1,100) )
    [[ route add default gw %gateway% [[metric %metric%]] %iface% ]]
```

For `static inet` interfaces, `ifdown` executes the following commands in `inet.defn`:

```
down
    [[ route del default gw %gateway% [[metric %metric%]] %iface% ]]
    ifconfig %iface% down
```

Options defined in the network interfaces file are enclosed in percent signs (`%…%`). Expressions enclosed in double square brackets (`[[…]]`) are optional; they are included only if the options contained within the brackets are defined.

`ifup` calls `iface_up()` in `execute.c` (from the same Debian source archive; Debian.org, 2016):

```
int iface_up(interface_defn *iface) {
```

```
        if (!iface->method->up(iface,check)) return -1;

        set_environ(iface, "start", "pre-up");
        if (!execute_all(iface,doit,"pre-up")) return 0;

        if (!iface->method->up(iface,doit)) return 0;

        set_environ(iface, "start", "post-up");
        if (!execute_all(iface,doit,"up")) return 0;

        return 1;
}
```

The `iface_up()` function:

- Calls `check()` to verify the command string is not empty (failure returns `-1`),
- Runs all the scripts in `/etc/network/if-pre-up.d/` (failure returns `0`),
- Calls `doit()` to execute the commands to bring up the interface (failure returns `0`), and
- Runs all the scripts in `/etc/network/if-up.d/` (failure returns `0`).

On success, `iface_up()` returns `1`.

The commands are executed by `doit()` in `execute.c`:

```
static int doit(char *str) {
        assert(str);

        if (verbose || no_act) {
                fprintf(stderr, "%s\n", str);
        }
        if (!no_act) {
                pid_t child;
                int status;

                fflush(NULL);
                switch(child = fork()) {
                    case -1: /* failure */
                       return 0;
                    case 0: /* child */
                       execle("/bin/sh", "/bin/sh", "-c", str, NULL, environ);
                       exit(127);
                    default: /* parent */
                       break;
                }
                waitpid(child, &status, 0);
                if (!WIFEXITED(status) || WEXITSTATUS(status) != 0)
                       return 0;
        }
        return 1;
}
```

`ifup` sets `failed` to `1` if `iface_up()` fails; otherwise, `ifup` sets `failed` to `0`.

If `failed` is `1`, `ifup` prints an error message. Otherwise, the interface is added to the `ifupdown` run state file, `/etc/network/run/ifstate`. `ifup` then exits:

```
                        if (failed == 1) {
                                printf("Failed to bring up %s.\n", liface);
                        } else {
                                add_to_state(&state, &n_state, &max_state, newiface);
                        }
```

For interfaces configured as `dhcp inet` in `/etc/network/interfaces`, `ifup` looks for `dhclient3` (from the `dhcp3-client` package), `dhclient` (from the `dhcp3-client` package), `pump` (from the `pump` package), `udhcpc` (from the `udhcpc` package), and `dhcpcd` (from the `dhcpcd` package), in order, to provide the IP address for the interface.

Both the `dhcp3-client` and `pump` DHCP client packages are installed on a Granite:

```
# dpkg --list | grep dhcp
ii  dhcp3-client                    4.1.1-P1-15+squeeze2         ISC DHCP serve)
ii  dhcp3-common                    4.1.1-P1-15+squeeze2         ISC DHCP commo)
ii  isc-dhcp-client                 4.1.1-P1-15+squeeze2         ISC DHCP client
ii  isc-dhcp-common                 4.1.1-P1-15+squeeze2         common files us
ii  isc-dhcp-server                 4.1.1-P1-15+squeeze2         ISC DHCP servet

# dpkg --list | grep pump
ii  pump                            0.8.24-7                     BOOTP and DHCPn

# ls /sbin/{*dhc*,pump}
/sbin/dhclient  /sbin/dhclient-script  /sbin/pump
```

Thus, the `pump` DHCP client is superfluous.

3.5. Test of Linux Hotplug Support

I examined whether Linux *hotplug* device support can be configured to recognize when Ethernet link state changes occur and to reconfigure the IP network accordingly.

The Linux device manager, `udev`, supports the dynamic addition and removal of devices, such as USB devices, through what is called hotplug device support. `udev` hotplug events will trigger `ifup` and `ifdown` using the `net` rule (`SUBSYSTEM=="net"`) in `/lib/udev/rules.d/80-drivers.rules` when, for example, a USB Ethernet adapter is inserted or removed.

The `net` rule for hotplug events is:

```
SUBSYSTEM=="net",                       RUN+="net.agent"
```

When the `net` rule is invoked, `udev` runs the `net.agent` script (`RUN+="net.agent"`):

```
# cat /lib/udev/net.agent
#!/bin/sh -e
#
# run /sbin/{ifup,ifdown} with the --allow=hotplug option.
#

. /lib/udev/hotplug.functions

if [ -z "$INTERFACE" ]; then
    mesg "Bad net.agent invocation: \$INTERFACE is not set"
    exit 1
fi

check_program() {
    [ -x $1 ] && return 0

    mesg "ERROR: $1 not found. You need to install the ifupdown package."
    mesg "net.agent $ACTION event for $INTERFACE not handled."
    exit 1
}

wait_for_interface() {
    local interface=$1

    while :; do
        local state="$(cat /sys/class/net/$interface/operstate 2>/dev/null || true)"
        if [ "$state" != down ]; then
                return 0
        fi
        sleep 1
```

```
    done
}

net_ifup() {
    check_program /sbin/ifup

    if grep -q '^auto[[:space:]].*\<'"$INTERFACE"'\>' \
            /etc/network/interfaces; then
        # this $INTERFACE is marked as "auto"
        IFUPARG='\('$INTERFACE'\|-a\|--all\)'
    else
        IFUPARG=$INTERFACE
    fi

    if ps -C ifup ho args | grep -q "$IFUPARG"; then
        debug_mesg "Already ifup-ing interface $INTERFACE"
        exit 0
    fi

    wait_for_interface lo
    if [ -e /bin/systemctl ]; then
      wait_for_file /dev/log
    fi

    exec ifup --allow=hotplug $INTERFACE
}

net_ifdown() {
    check_program /sbin/ifdown

    if ps -C ifdown ho args | grep -q $INTERFACE; then
        debug_mesg "Already ifdown-ing interface $INTERFACE"
        exit 0
    fi

    exec ifdown --allow=hotplug $INTERFACE
}

do_everything() {

case "$ACTION" in
    add)
    # these interfaces generate hotplug events *after* they are brought up
    case $INTERFACE in
        ppp*|ippp*|isdn*|plip*|lo|irda*|ipsec*)
        exit 0 ;;
    esac

    net_ifup
    ;;

    remove)
    # the pppd persist option may have been used, so it should not be killed
    case $INTERFACE in
        ppp*)
        exit 0 ;;
    esac

    net_ifdown
    ;;

    *)
    debug_mesg "NET $ACTION event not supported"
    exit 1
    ;;
esac

}

# When udev_log="debug" stdout and stderr are pipes connected to udevd.
# They need to be closed or udevd will wait for this process which will
# deadlock with udevsettle until the timeout.
do_everything > /dev/null 2> /dev/null &
```

```
exit 0
```

In the `net.agent` script, `udev` hotplug support is enabled for a network interface listed in an `allow-hotplug` entry in the network interfaces configuration file, `/etc/network/interfaces`. As mentioned earlier, `allow-hotplug` is missing from the Granite network interfaces configuration file, thus disabling hotplugging for `eth0`.

I determined that `udev` hotplug events are not triggered by network interface link (carrier) state-change events, only by the insertion or removal of a removable network device, such as a USB Ethernet adapter. (See section 4.3, "How to Enable the `udev net.agent` Logging.") This is unfortunate, because the `net.agent` script would execute the proper `ifup` and `ifdown` commands that should be triggered by link up and link down state changes. Instead, a network link monitoring daemon, such as `ifplugd` or `netplugd`, must be used to provide this capability (see the following section 3.6, "The Linux Network Link-Monitoring Daemons").

In addition, for this method to work properly the Ethernet driver must notify the kernel of link (carrier) state changes by calling `netif_carrier_on()` and `netif_carrier_off()` when it detects the presence or absence of carrier (link up or link down), respectively. Because the Granite `cs89x0` Ethernet drivers do not detect link state changes, `udev` hotplug support would be ineffective in any case.

3.6. The Linux Network Link-Monitoring Daemons

Debian Linux has two network link-monitoring daemon packages to manage IP network reconfiguration when Ethernet cables are plugged in or unplugged: `ifplugd` and `netplug`.

The `ifplugd` package is older. The `ifplugd` daemon polls Ethernet devices for changes in link state, using either the `mii-tool` or `ethtool` support in the driver. If the driver does not support one of those methods and the kernel thinks the device is in the `RUNNING` state (`IFF_RUNNING` is `1`; for example, `ifconfig` shows the interface is `RUNNING`), that is taken as evidence for link state up.

Because the Granite `cs89x0` Ethernet drivers do not support any Ethernet link-state query, `ifplugd` relies on the interface `IFF_RUNNING` flag. `IFF_RUNNING` is set (`=1`) when the Ethernet cable is plugged in, but is not cleared (`=0`) if the cable is later unplugged, rendering `ifplugd` ineffective.

The `netplug` package is newer. Instead of polling, the `netplug` daemon monitors the link state of Ethernet interfaces by listening for kernel `netlink` events. Kernel `netlink` events are signaled by calls in the Ethernet driver when the link state changes. Again, because the Granite `cs89x0` Ethernet drivers do not detect link state changes, `netplug` is ineffective.

In a Granite, the `netplug` network link-monitoring daemon package is installed, but is not enabled:

```
# dpkg --list | grep plug
ii  netplug                        1.2.9.1-2                      network link mn
ii  udev                           164-3                          /dev/ and hotpn

# ls /etc/*/*netplug
/etc/init.d/netplug  /etc/netplug/netplug
```

4. Monitoring Linux Networking Components

Various Linux network components have monitoring or debugging features to log their activity. The following sections describe the logging features I enabled and the IP network behavior of the Granite Linux system that I observed:

- *How to Enable `netlink NETLINK_ROUTE` Message Logging*—How to create a Linux `ipmonitor` service to log the output of the `ip monitor` command, which monitors Linux kernel `netlink` `NETLINK_ROUTE` event notification messages.
- *How to Set the `udev` Log Level to `debug`*—How to change the `udev` default log level from `err` to `debug`.
- *How to Enable the `udev net.agent` Logging*—How to log every call to the `udev net.agent`, and every time `ifup` and `ifdown` are run, to `/var/log/net.agent`.

- *How to Enable `ifup` and `ifdown` Verbose Messages*—How to enable verbose messages for `ifup` and `ifdown` in the Linux `networking` service script and the `udev` `net.agent` script.
- *How to Enable `ifupdown` Script Debugging Messages*—How to turn on debugging messages in `ifupdown` scripts.
- *How to Enable the `cs89x0_x` Device Driver Debugging Option*—How to enable the debugging option (`debug=1`) for the `cs89x0` device driver. (Requires a new driver; see section 5, "How to Build a New `cs89x0_x` Device Driver.")

4.1. How to Enable `netlink` `NETLINK_ROUTE` Message Logging

I created an IP-monitoring service that logs the output from the `ip monitor` command, which monitors Linux kernel `netlink` `NETLINK_ROUTE` event notification messages (from Stack Overflow, 2016):

Create the following script for the `ipmonitor` service and make it executable:

```
# vi /etc/init.d/ipmonitor

#!/bin/sh -e
### BEGIN INIT INFO
# Provides:          ipmonitor
# Required-Start:    $local_fs $syslog
# Required-Stop:     $local_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: ip monitor logging
# Description:
#      ipmonitor logs "ip monitor all" IP state change messages

### END INIT INFO

THIS="ip monitor logging"

PATH=/sbin:/bin:/usr/sbin:/usr/bin

. /lib/lsb/init-functions

PID_FILE=/var/run/ipmonitor

# ip monitor all stdout/stderr ...
DAEMON=`which ip`
DAEMON_ARGS="monitor all"

# ... piped to logger -p user.info -t ipmonitor
LOGGER=`which logger`
LOGGER_ARGS="-p user.info -t ipmonitor"

[ -n "$DAEMON" ] || exit 5
[ -n "$LOGGER" ] || exit 5

case $1 in

   start)

      # Note: There is no advantage to using the bash exec command
      # since the command is a pipeline

```

```
      log_daemon_msg "Starting $THIS"
      $LOGGER $LOGGER_ARGS "Starting $THIS"
      start-stop-daemon --start --background \
                        --make-pidfile --pidfile $PID_FILE \
                        --exec /bin/bash -- -c \
                               "$DAEMON $DAEMON_ARGS 2>&1 | \
                                $LOGGER $LOGGER_ARGS"
      log_end_msg $?

      ;;

   stop)

      # Kill all the processes in the daemon process group
      # Note: This has to be done manually

      [ -f "$PID_FILE" ] || exit 5
      TOP_PID=`cat $PID_FILE`
      [ -f "/proc/$TOP_PID/stat" ] || exit 5
      GPID=`cut -f 5 -d " " /proc/$TOP_PID/stat`
      PIDS=`pgrep -g $GPID`
      log_daemon_msg "Stopping $THIS"
      $LOGGER $LOGGER_ARGS "Stopping $THIS"
      start-stop-daemon --stop --pidfile $PID_FILE && \
      kill $PIDS 2>/dev/null || /bin/true
      log_end_msg $?
      ;;

   restart)

      $0 stop && sleep 2 && $0 start
      ;;

   status)

      status_of_proc -p $PID_FILE "/bin/bash" "$THIS"
      ;;

   *)
      echo "Usage: $0 {start|stop|restart|status}"
      exit 2
      ;;

esac

# chmod +x /etc/init.d/ipmonitor
```

Then, enable the `ipmonitor` service:

```
# update-rc.d ipmonitor defaults
```

Use `ipmonitor` to log `netlink` `NETLINK_ROUTE` messages triggered by Ethernet link state changes.

Compare this original Granite behavior with the behavior in section 7, "How to Enable the `netplugd` Ethernet Network Link Monitor," using the modified Granite Ethernet device drivers.

Boot with an Ethernet cable plugged in.

Issue the `ifconfig`, `route`, and `ip link` commands to observe the results:

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:50:c2:5a:c5:73
          inet addr:192.168.99.11  Bcast:192.168.99.255  Mask:255.255.255.0
          inet6 addr: fe80::250:c2ff:fe5a:c573/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:182 errors:0 dropped:0 overruns:0 frame:0
          TX packets:188 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17223 (16.8 KiB)  TX bytes:16151 (15.7 KiB)
          Interrupt:139 Base address:0x300

# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.99.0    *               255.255.255.0   U     0      0        0 eth0
default         192.168.99.1    0.0.0.0         UG    0      0        0 eth0

# ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UNKNOWN qlen 1000
    link/ether 00:50:c2:5a:c5:73 brd ff:ff:ff:ff:ff:ff
```

Unplug the Ethernet cable here.

When the cable is unplugged, the console displays:

```
NETDEV WATCHDOG: eth0: transmit queue 0 timed out
```

Observe the results.

Note that the Ethernet link state remains `UP`*.*
The interface is still in the `RUNNING` *state.*

This is not the correct behavior.

```
# ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UNKNOWN qlen 1000
    link/ether 00:50:c2:5a:c5:73 brd ff:ff:ff:ff:ff:ff

# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:50:c2:5a:c5:73
          inet addr:192.168.99.11  Bcast:192.168.99.255  Mask:255.255.255.0
          inet6 addr: fe80::250:c2ff:fe5a:c573/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:184 errors:0 dropped:0 overruns:0 frame:0
          TX packets:190 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17368 (16.9 KiB)  TX bytes:16764 (16.3 KiB)
          Interrupt:139 Base address:0x300

# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```

```
192.168.99.0    *               255.255.255.0   U     0      0        0 eth0
default         192.168.99.1    0.0.0.0         UG    0      0        0 eth0
```

Use the `ip link` command to force the Ethernet link state to `DOWN`:

```
# ip link set dev eth0 down
```

Observe the results:

```
# ip link show eth0
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen
1000
    link/ether 00:50:c2:5a:c5:73 brd ff:ff:ff:ff:ff:ff
```

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:50:c2:5a:c5:73
          inet addr:192.168.99.11  Bcast:192.168.99.255  Mask:255.255.255.0
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:184 errors:0 dropped:0 overruns:0 frame:0
          TX packets:190 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17368 (16.9 KiB)  TX bytes:18486 (18.0 KiB)
          Interrupt:139 Base address:0x300
```

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```

Attempt to set the Ethernet link state to `UP` (the Ethernet cable is still unplugged):

```
# ip link set dev eth0 up
```

The Ethernet link state remains `DOWN`.

The console displays:

```
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
RTNETLINK answers: Resource temporarily unavailable
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
```

Observe the results:

```
# ip link show eth0
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen
1000
    link/ether 00:50:c2:5a:c5:73 brd ff:ff:ff:ff:ff:ff
```

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:50:c2:5a:c5:73
          inet addr:192.168.99.11  Bcast:192.168.99.255  Mask:255.255.255.0
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:184 errors:0 dropped:0 overruns:0 frame:0
          TX packets:190 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:17368 (16.9 KiB)  TX bytes:18486 (18.0 KiB)
```

```
          Interrupt:139 Base address:0x300

# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```

Plug the Ethernet cable back in here.

When the cable is plugged in, the console displays:

```
eth0: using half-duplex 10Base-T (RJ-45)
```

Observe the results.

Verify that the Ethernet link state changes to `UP`.

```
# ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UNKNOWN qlen 1000
    link/ether 00:50:c2:5a:c5:73 brd ff:ff:ff:ff:ff:ff

# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:50:c2:5a:c5:73
          inet addr:192.168.99.11  Bcast:192.168.99.255  Mask:255.255.255.0
          inet6 addr: fe80::250:c2ff:fe5a:c573/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:201 errors:0 dropped:0 overruns:0 frame:0
          TX packets:213 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:19178 (18.7 KiB)  TX bytes:20117 (19.6 KiB)
          Interrupt:139 Base address:0x300

# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.99.0    *               255.255.255.0   U     0      0        0 eth0
default         192.168.99.1    0.0.0.0         UG    0      0        0 eth0
```

The `ipmonitor` service logs the following `netlink` `NETLINK_ROUTE` messages for the transactions above (tagged `[ROUTE]`):

```
Jul 21 17:20:33 BakerTest ipmonitor: [ROUTE]Deleted fe80::/64 dev eth0  proto kernel  met-
ric 256  mtu 1500 advmss 1440 hoplimit 0
Jul 21 17:20:33 BakerTest ipmonitor: [ROUTE]Deleted ff00::/8 dev eth0  table local  metric
256  mtu 1500 advmss 1440 hoplimit 0
Jul 21 17:20:33 BakerTest ipmonitor: [ROUTE]Deleted local fe80::250:c2ff:fe5a:c573 via ::
dev lo  table local  proto none  metric 0  mtu 16436 advmss 16376 hoplimit 0
Jul 21 17:22:36 BakerTest ipmonitor: [ROUTE]broadcast 192.168.99.255 dev eth0  table local
proto kernel  scope link  src 192.168.99.11
Jul 21 17:22:36 BakerTest ipmonitor: [ROUTE]192.168.99.0/24 dev eth0  proto kernel  scope
link  src 192.168.99.11
Jul 21 17:22:36 BakerTest ipmonitor: [ROUTE]broadcast 192.168.99.0 dev eth0  table local
proto kernel  scope link  src 192.168.99.11
Jul 21 17:22:36 BakerTest ipmonitor: [ROUTE]ff00::/8 dev eth0  table local  metric 256
mtu 1500 advmss 1440 hoplimit 0
Jul 21 17:22:36 BakerTest ipmonitor: [ROUTE]fe80::/64 dev eth0  proto kernel  metric 256
mtu 1500 advmss 1440 hoplimit 0
Jul 21 17:22:36 BakerTest ipmonitor: [ROUTE]default via 192.168.99.1 dev eth0
Jul 21 17:22:38 BakerTest ipmonitor: [ROUTE]local fe80::250:c2ff:fe5a:c573 via :: dev lo
table local  proto none  metric 0  mtu 16436 advmss 16376 hoplimit 0
```

4.2. How to Set the udev Log Level to debug

The Linux device manager, udev, responds to kernel requests through netlink uevent messages. udev kernel uevent messages can be monitored using the udevadm monitor command:

```
# udevadm monitor
monitor will print the received events for:
UDEV - the event which udev sends out after rule processing
KERNEL - the kernel uevent
```

udevadm monitor options (not used here) do not affect which udev kernel uevent messages will be logged. They only affect what kernel uevent messages are displayed.

The udev logging level can be altered on-the-fly using the udevadm control command (the default log level is err):

```
# udevadm control --log-priority=info
```

Edit /etc/udev/udev.conf to change the default udev log level to debug:

```
# vi /etc/udev/udev.conf
```

```
#udev_log="err"
udev_log="debug"
```

udev info messages appear in /var/log/daemon.log, whereas udev debug messages appear on the console.

With the Ethernet cable plugged in, at boot there are udev debug messages for calls to net.agent for eth0. When the cable is unplugged, there are no udev debug messages, and no udev info messages.

With the Ethernet cable unplugged, at boot there are udev debug messages for calls to net.agent for eth0. When the cable is plugged in, there are no udev debug messages, and no udev info messages.

Before plugging in the Ethernet cable, the tail of /var/log/daemon.log is:

```
# tail /var/log/daemon.log
Jun 16 19:19:52 BakerTest ntpd_intres[944]: host name not found: 2.pool.ntp.org
Jun 16 19:19:52 BakerTest ntpd_intres[944]: host name not found: 3.pool.ntp.org
Jun 16 19:20:54 BakerTest ntpd_intres[944]: host name not found: 0.pool.ntp.org
Jun 16 19:20:54 BakerTest ntpd_intres[944]: host name not found: 1.pool.ntp.org
Jun 16 19:20:55 BakerTest ntpd_intres[944]: host name not found: 2.pool.ntp.org
Jun 16 19:20:55 BakerTest ntpd_intres[944]: host name not found: 3.pool.ntp.org
Jun 16 19:22:57 BakerTest ntpd_intres[944]: host name not found: 0.pool.ntp.org
Jun 16 19:22:57 BakerTest ntpd_intres[944]: host name not found: 1.pool.ntp.org
Jun 16 19:22:57 BakerTest ntpd_intres[944]: host name not found: 2.pool.ntp.org
Jun 16 19:22:57 BakerTest ntpd_intres[944]: host name not found: 3.pool.ntp.org
```

After plugging in the Ethernet cable, the tail of /var/log/daemon.log is:

```
# tail /var/log/daemon.log
Jun 16 19:22:57 BakerTest ntpd_intres[944]: host name not found: 3.pool.ntp.org
Jun 16 19:26:09 BakerTest udevd[374]: worker [403] exit
Jun 16 19:26:09 BakerTest udevd[374]: worker [403] cleaned up
Jun 16 19:26:09 BakerTest udevd[374]: worker [404] exit
Jun 16 19:26:09 BakerTest udevd[374]: worker [404] cleaned up
Jun 16 19:26:59 BakerTest ntpd_intres[944]: host name not found: 0.pool.ntp.org
Jun 16 19:26:59 BakerTest ntpd_intres[944]: host name not found: 1.pool.ntp.org
Jun 16 19:26:59 BakerTest ntpd_intres[944]: host name not found: 2.pool.ntp.org
Jun 16 19:26:59 BakerTest ntpd_intres[944]: host name not found: 3.pool.ntp.org
Jun 16 19:27:07 BakerTest ntpdate[1701]: the NTP socket is in use, exiting
```

udevadm monitor does not show any kernel uevents or triggered events when the cable is plugged in.

These results confirm that neither the Granite `cs89x0` Ethernet drivers, nor any other daemon or kernel task, notify the Linux kernel of an Ethernet link state change.

4.3. How to Enable the `udev net.agent` Logging

I modified the `udev net.agent` to log calls to `/var/log/net.agent`. The `logger` program cannot be used because the network is not up; these messages are written directly to the log file:

```
# vi /lib/udev/net.agent

do_everything() {

echo >>/var/log/net.agent "`date` net.agent: $ACTION event for $INTERFACE."

case "$ACTION" in

esac

echo >>/var/log/net.agent "`date` net.agent: status = $?."

}
```

I created `debug` shell scripts to also include a message in the log every time `ifup` and `ifdown` are run:

```
# cat >/etc/network/if-pre-up.d/debug <<'EOF'
#!/bin/sh
echo >>/var/log/net.agent "`date` if-pre-up: $IFACE."
EOF
# chmod +x /etc/network/if-pre-up.d/debug

# cat >/etc/network/if-post-down.d/debug <<'EOF'
#!/bin/sh
echo >>/var/log/net.agent "`date` if-post-down: $IFACE."
EOF
# chmod +x /etc/network/if-post-down.d/debug
```

With the Ethernet cable plugged in, at boot there are `net.agent` messages for `eth0`:

```
# tail /var/log/net.agent
Mon Aug 15 16:44:01 UTC 2016 if-post-down: eth0.
Thu Jan  1 00:00:37 UTC 1970 net.agent: add event for lo.
Mon Aug 15 16:45:41 UTC 2016 if-pre-up: lo.
Mon Aug 15 16:45:42 UTC 2016 if-pre-up: eth0.
Mon Aug 15 16:45:43 UTC 2016 net.agent: add event for eth0.
```

Unplug the Ethernet cable here.

When the cable is unplugged, there are no new `net.agent` messages (the time of the last message is unchanged):

```
# tail /var/log/net.agent
Mon Aug 15 16:44:01 UTC 2016 if-post-down: eth0.
Thu Jan  1 00:00:37 UTC 1970 net.agent: add event for lo.
Mon Aug 15 16:45:41 UTC 2016 if-pre-up: lo.
Mon Aug 15 16:45:42 UTC 2016 if-pre-up: eth0.
Mon Aug 15 16:45:43 UTC 2016 net.agent: add event for eth0.
```

With the Ethernet cable unplugged, at boot there are `net.agent` messages for `eth0`:

```
# tail /var/log/net.agent
Mon Aug 15 17:09:40 UTC 2016 if-post-down: eth0.
Thu Jan  1 00:00:37 UTC 1970 net.agent: add event for lo.
Mon Aug 15 17:11:20 UTC 2016 if-pre-up: lo.
Mon Aug 15 17:11:21 UTC 2016 if-pre-up: eth0.
Mon Aug 15 17:11:22 UTC 2016 net.agent: add event for eth0.
Mon Aug 15 17:13:39 UTC 2016 if-pre-up: eth0.
```

The last line repeats every minute.

```
Mon Aug 15 17:14:48 UTC 2016 if-pre-up: eth0.
Mon Aug 15 17:15:56 UTC 2016 if-pre-up: eth0.
```

Every time a new `net.agent` message appears in `/var/log/net.agent`, the console displays:

```
eth0: no network cable attached to configured media
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
```

Plug in an Ethernet cable here.

When the cable is plugged in, the console displays:

```
eth0: using half-duplex 10Base-T (RJ-45)
```

And there is a new `net.agent` message at the end of `/var/log/net.agent`:

```
# tail /var/log/net.agent
Mon Aug 15 17:09:40 UTC 2016 if-post-down: eth0.
Thu Jan  1 00:00:37 UTC 1970 net.agent: add event for lo.
Mon Aug 15 17:11:20 UTC 2016 if-pre-up: lo.
Mon Aug 15 17:11:21 UTC 2016 if-pre-up: eth0.
Mon Aug 15 17:11:22 UTC 2016 net.agent: add event for eth0.
Mon Aug 15 17:13:39 UTC 2016 if-pre-up: eth0.
Mon Aug 15 17:14:48 UTC 2016 if-pre-up: eth0.
Mon Aug 15 17:15:56 UTC 2016 if-pre-up: eth0.
Mon Aug 15 17:17:05 UTC 2016 if-pre-up: eth0.
```

These results show that `udev` calls `net.agent` only at boot time. However, there are multiple entries from `ifup` when the Ethernet cable is unplugged at boot. There are never entries from `ifdown` when the Ethernet cable is plugged in at boot and then unplugged. There are also never any entries from `ifup` or `ifdown` for hotplug events, which means that cable insertions/removals are not being treated as `udev` hotplug events.

4.4. How to Enable `ifup` and `ifdown` Verbose Messages

I modified `/etc/init.d/networking` and `/lib/udev/net.agent` to enable verbose messages for `ifup` and `ifdown`:

```
# vi /etc/init.d/networking

        log_action_begin_msg "Configuring network interfaces"
        if ifup -v -a; then

        log_action_begin_msg "Deconfiguring network interfaces"
        if ifdown -v -a --exclude=lo; then

        log_action_begin_msg "Reconfiguring network interfaces"
```

```
        ifdown -v -a --exclude=lo || true
        if ifup -v -a --exclude=lo; then

# vi /lib/udev/net.agent

    exec ifup --verbose --allow=hotplug $INTERFACE

    exec ifdown --verbose --allow=hotplug $INTERFACE
```

4.5. How to Enable `ifupdown` Script Debugging Messages

I modified `/etc/default/ifupdown` to turn on debugging messages in scripts:

```
# vi /etc/default/ifupdown

# Set to "yes" to turn on debugging messages in scripts
# DEBUG=""
DEBUG="yes"
```

4.6. How to Enable the `cs89x0_x` Device Driver Debugging Option

To enable the `cs89x0_x` device driver debugging option, save the original Granite `modprobe` network drivers options file (if not done already) and add an entry enabling the debugging option (`debug=1`):

```
# mv -n /etc/modprobe.d/net.conf{,.orig}
# cat /etc/modprobe.d/net.conf.orig - >/etc/modprobe.d/net.conf <<EOF
options cs89x0_0 debug=1
options cs89x0_1 debug=1
EOF
```

Then, reboot.

Note that this fails on the Granites; the modified **`cs89x0`** *drivers are compiled without debugging support. They have to be recompiled with debugging support enabled. See the following section, "How to Build a New* ***`cs89x0_x`*** *Device Driver."*

5. How to Build a New `cs89x0_x` Device Driver

Granites have sufficient resources to self-host Linux kernel module development; `make` and `gcc` are already installed. You must run `make` from the top directory of the Linux source tree (`/usr/src/linux/`). Be sure the kernel configuration file is copied to `/usr/src/linux/.config`.

Note that the root file system on the Granites is not large enough to contain the Linux kernel sources. Use the `/opt` file system instead.

Extract the Linux 2.6.35.14 LTS kernel sources from the compressed archive file that has been copied to `/usr/local/src/linux-2.6.35.14.tar.gz`:

```
# cd /opt
# tar -xzf /usr/local/src/linux-2.6.35.14.tar.gz
```

Extract the Granite Ethernet drivers and Granite hardware-specific (Intel PXA processor) support files from the compressed archive file that has been copied to `/usr/local/src/rock1_cs89.tgz`:

```
# cd linux-2.6.35.14
# tar -xzf /usr/local/src/rock1_cs89.tgz
```

The Debian ARM gcc 4.4.5 compiler on the Granites emits ARM `R_REL32` relocation records, but the ARM Linux kernel module loader cannot handle them. The symptom is an "`unknown relocation: 3`" error when loading (`modprobe`) a kernel module.

Patch the Linux kernel `arch/arm/Makefile` to add the `gcc -fno-dwarf2-cfi-asm` option to disable them (from Michal Marek, 2010; also see http://linux-arm-kernel.infradead.narkive.com/KIU3o209/patch-arm-r-arm-rel32-relocation-support, accessed August 1, 2016):

```
# vi /usr/local/src/arch-arm-Makefile.patch
```

```
--- a/arch/arm/Makefile
+++ b/arch/arm/Makefile
@@ -21,6 +21,9 @@ GZFLAGS         :=-9
 # Explicitly specify 32-bit ARM ISA since toolchain default can be -mthumb:
 KBUILD_CFLAGS      +=$(call cc-option,-marm,)

+# Never generate .eh_frame
+KBUILD_CFLAGS      += $(call cc-option,-fno-dwarf2-cfi-asm)
+
 # Do not use arch/arm/defconfig - it's always outdated.
 # Select a platform that is kept up-to-date
 KBUILD_DEFCONFIG := versatile_defconfig
```

```
# patch -b -p 1 </usr/local/src/arch-arm-Makefile.patch
```

Copy the Granite Linux kernel configuration file to `.config` and prepare the configuration-specific kernel header files:

```
# cp config.kmi .config
# make oldconfig && make prepare && make modules_prepare
```

Build the modified `cs89x0_x` driver modules:

```
# make M=drivers/net/ CONFIG_CS89x0=m

  WARNING: Symbol version dump /opt/linux-2.6.35.14/Module.symvers
           is missing; modules will have no dependencies and modversions.

  CC [M]  drivers/net/cs89x0_0.o
  CC [M]  drivers/net/cs89x0_1.o
  Building modules, stage 2.
  MODPOST 2 modules
  LD [M]  drivers/net/cs89x0_0.ko
  LD [M]  drivers/net/cs89x0_1.ko
```

The new `cs89x0_x` driver modules will be in `drivers/net/`:

```
# ls -l drivers/net/cs89x0_?.ko
-rw-r--r-- 1 root root 12952 Jul 14 17:49 drivers/net/cs89x0_0.ko
-rw-r--r-- 1 root root 12952 Jul 14 17:49 drivers/net/cs89x0_1.ko
```

The original Granite cs89x0_x driver modules are in `/lib/modules/2.6.35.14/kernel/drivers/net/`:

```
# ls -l /lib/modules/2.6.35.14/kernel/drivers/net
total 28
-rw-rw-r-- 1 root root 13398 Aug 21  2014 cs89x0_0.ko
-rw-rw-r-- 1 root root 13398 Aug 21  2014 cs89x0_1.ko
```

`lsmod` displays the currently loaded modules:

```
# lsmod | grep cs89x0
cs89x0_0                6889  0
```

As a precaution, edit `/etc/modprobe.d/net.conf` and `blacklist` the `cs89x0_x` driver modules. The following change prevents the driver modules from automatically loading when Linux boots, in case there are problems (save the original Granite `modprobe` network drivers options file, if not done already):

```
# mv -n /etc/modprobe.d/net.conf{,.orig}
# vi /etc/modprobe.d/net.conf

blacklist cs89x0_0
blacklist cs89x0_1
```

Reboot.

Verify that the original Granite `cs89x0_x` driver modules are no longer loaded:

```
# lsmod | grep cs89x0
```

Load the original Granite `cs89x0_0` driver module for `eth0`:

```
# modprobe cs89x0_0
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
cs89x0 media RJ-45, IRQ 139, programmed I/O, MAC 00:50:c2:5a:c5:73
eth0: using half-duplex 10Base-T (RJ-45)
```

Reboot.

Save the original Granite `cs89x0_x` driver modules and verify the new driver modules work on the running kernel:

```
# cd /lib/modules/2.6.35.14/kernel/drivers/net
# mv -n cs89x0_0.ko{,.orig}
# mv -n cs89x0_1.ko{,.orig}
# cp /opt/linux-2.6.35.14/drivers/net/cs89x0_?.ko ./
# modprobe cs89x0_0
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
cs89x0 media RJ-45, IRQ 139, programmed I/O, MAC 00:50:c2:5a:c5:73
eth0: using half-duplex 10Base-T (RJ-45)
```

When the new drivers are working properly, edit `/etc/modprobe.d/net.conf` and disable the `blacklist` entries:

```
# vi /etc/modprobe.d/net.conf

#blacklist cs89x0_0
#blacklist cs89x0_1
```

Finally, reboot again.

6. Modifications to the `cs89x0_x` Device Driver

I made the following modifications to the Granite `cs89x0` Ethernet drivers to enable the device driver debugging option described above and to issue link state-change notifications to the Linux kernel for network link monitoring support:

- Reconcile the places where the `cs89x0_0` and `cs89x0_1` drivers have different amounts of white space.
- Move the "`cs89x0_probe1() successful`" debug message to the correct location.
- Enable the `cs89x0` driver `debug` option (`DEBUGGING=1`) and restore the code from the original `cs89x0` driver to ignore any `debug` option specified when the driver is loaded, instead of failing to load the driver, when the `cs89x0` driver `debug` option is disabled (`DEBUGGING=0`).
- Add a kernel timer to poll the interface for changes in link state and notify the Linux kernel by calling `netif_carrier_on()` and `netif_carrier_off()` for link up (carrier on) and link down (carrier off) transitions, respectively.
- Add interrupt request (IRQ) spinlock protection to `net_close()`.
- Move the code to stop the hardware to a new routine, `cs_stop_hw()`.

With these modifications, the `netplugd` daemon can be configured to properly reconfigure Granite IP networking when the Ethernet interface link state changes.

Create a patch file to modify `cs89x0_0.c` and `cs89x0_1.c`, as follows:

```
# vi /usr/local/src/cs89x0.patch

--- kmi/drivers/net/cs89x0_0.c
+++ usgs/drivers/net/cs89x0_0.c
@@ -22,3 +22,3 @@
  */
-#define DEBUGGING   0
+#define DEBUGGING   1

@@ -107,2 +107,5 @@

+#define CS_TIMER_LINK             (60 * HZ)
+#define CS_TIMER_NO_LINK   (5 * HZ)
+
 /* Information that need to be kept for each board. */
@@ -124,2 +127,3 @@
     spinlock_t lock;
+    struct timer_list      media_timer;
 };
@@ -149,2 +153,3 @@
 #endif
+static void cs_media_timer(unsigned long data);

@@ -472,4 +477,6 @@
 #endif
-
     }
+    init_timer(&lp->media_timer);
+    lp->media_timer.function = cs_media_timer;
+    lp->media_timer.data = (unsigned long) dev;

@@ -763,4 +770,2 @@
     printk("\n");
-    if (net_debug)
-        printk("cs89x0_probe1() successful\n");

@@ -769,3 +774,7 @@
         goto out3;
+
+    if (net_debug)
+        printk("cs89x0_probe1() successful\n");
     return (0);
+
     out3:
@@ -914,2 +923,59 @@

+static void cs_stop_hw(struct net_device *dev)
```

```
+{
+
+    udelay(5);
+
+    writereg(dev, PP_RxCFG, 0);
+    writereg(dev, PP_TxCFG, 0);
+    writereg(dev, PP_BufCFG, 0);
+    writereg(dev, PP_BusCTL, 0);
+
+    udelay(10);
+}
+
+static void cs_link_up(struct net_device *dev)
+{
+    if (!netif_carrier_ok(dev)) {
+      netif_carrier_on(dev);
+      printk(KERN_INFO "%s: link up\n", dev->name);
+    }
+}
+
+static void cs_link_down(struct net_device *dev)
+{
+    if (netif_carrier_ok(dev)) {
+      netif_carrier_off(dev);
+      printk(KERN_INFO "%s: link down\n", dev->name);
+    }
+}
+
+static void cs_media_timer(unsigned long data)
+{
+    struct net_device *dev = (struct net_device *) data;
+    struct net_local *lp = netdev_priv(dev);
+    unsigned int carrier;
+
+    /* Normally multiple media checks would be done, but */
+    /* KMI_ROCK_PXA forces 10Base-T, half-duplex         */
+    carrier = ( readreg(dev, PP_LineST) & ( LINK_OK | TENBASET_ON ) ) ==
+            ( LINK_OK | TENBASET_ON );
+
+    if (carrier) {
+      if (!netif_carrier_ok(dev))
+          cs_link_up(dev);
+      else if (net_debug > 1)
+          printk("%s: link ok\n", dev->name);
+      lp->media_timer.expires = jiffies + CS_TIMER_LINK;
+      add_timer(&lp->media_timer);
+    } else {
+      if (netif_carrier_ok(dev))
+          cs_link_down(dev);
+      else if (net_debug > 1)
+          printk(KERN_INFO "%s: no link, try later\n", dev->name);
+      lp->media_timer.expires = jiffies + CS_TIMER_NO_LINK;
+      add_timer(&lp->media_timer);
+    }
+}
+
 /* send a test packet - return true if carrier bits are ok */
@@ -1029,2 +1095,5 @@

+    if (net_debug)
+      printk("%s: enabling interface\n", dev->name);
+
     if (dev->irq < 2)
@@ -1207,3 +1276,6 @@
             );
+
     netif_start_queue(dev);
+    mod_timer(&lp->media_timer, jiffies + CS_TIMER_NO_LINK);
```

```
+
     if (net_debug > 1)
@@ -1211,2 +1283,3 @@
     return(0);
+
     bad_out:
@@ -1425,8 +1498,15 @@
 {
-    netif_stop_queue(dev);
+    struct net_local *lp = netdev_priv(dev);
+    unsigned long flags;

-    writereg(dev, PP_RxCFG, 0);
-    writereg(dev, PP_TxCFG, 0);
-    writereg(dev, PP_BufCFG, 0);
-    writereg(dev, PP_BusCTL, 0);
+    if (net_debug)
+      printk("%s: disabling interface\n", dev->name);
+
+    del_timer_sync(&lp->media_timer);
+
+    spin_lock_irqsave(&lp->lock, flags);
+    cs_stop_hw(dev);
+    netif_stop_queue(dev);
+    netif_carrier_off(dev);
+    spin_unlock_irqrestore(&lp->lock, flags);

@@ -1508,2 +1588,16 @@

+/*
+ * Support the 'debug' module parm even if we're compiled for non-debug to
+ * avoid breaking someone's startup scripts
+ */
+
+static int debug;
+
+module_param(debug, int, 0);
+#if DEBUGGING
+MODULE_PARM_DESC(debug, "cs89x0 debug level (0-6)");
+#else
+MODULE_PARM_DESC(debug, "(ignored)");
+#endif
+
 MODULE_AUTHOR("Mike Cruse, Russwll Nelson <nelson@crynwr.com>, Andrew Morton");
@@ -1517,2 +1611,8 @@

+#if DEBUGGING
+    net_debug = debug;
+#else
+    debug = 0;
+#endif
+
     dev = alloc_etherdev(sizeof(struct net_local));
--- kmi/drivers/net/cs89x0_1.c
+++ usgs/drivers/net/cs89x0_1.c
@@ -22,3 +22,3 @@
  */
-#define DEBUGGING   0
+#define DEBUGGING   1

@@ -107,2 +107,5 @@

+#define CS_TIMER_LINK             (60 * HZ)
+#define CS_TIMER_NO_LINK    (5 * HZ)
+
 /* Information that need to be kept for each board. */
@@ -124,2 +127,3 @@
     spinlock_t lock;
```

```
+    struct timer_list      media_timer;
 };
@@ -149,2 +153,3 @@
 #endif
+static void cs_media_timer(unsigned long data);

@@ -472,4 +477,6 @@
 #endif
-
     }
+    init_timer(&lp->media_timer);
+    lp->media_timer.function = cs_media_timer;
+    lp->media_timer.data = (unsigned long) dev;

@@ -763,4 +770,2 @@
     printk("\n");
-    if (net_debug)
-        printk("cs89x0_probe1() successful\n");

@@ -769,3 +774,7 @@
         goto out3;
-    return(0);
+
+    if (net_debug)
+        printk("cs89x0_probe1() successful\n");
+    return (0);
+
     out3:
@@ -775,3 +784,3 @@
     out1:
-    return(retval);
+    return (retval);
 }
@@ -914,2 +923,59 @@

+static void cs_stop_hw(struct net_device *dev)
+{
+
+    udelay(5);
+
+    writereg(dev, PP_RxCFG, 0);
+    writereg(dev, PP_TxCFG, 0);
+    writereg(dev, PP_BufCFG, 0);
+    writereg(dev, PP_BusCTL, 0);
+
+    udelay(10);
+}
+
+static void cs_link_up(struct net_device *dev)
+{
+    if (!netif_carrier_ok(dev)) {
+      netif_carrier_on(dev);
+      printk(KERN_INFO "%s: link up\n", dev->name);
+    }
+}
+
+static void cs_link_down(struct net_device *dev)
+{
+    if (netif_carrier_ok(dev)) {
+      netif_carrier_off(dev);
+      printk(KERN_INFO "%s: link down\n", dev->name);
+    }
+}
+
+static void cs_media_timer(unsigned long data)
+{
+    struct net_device *dev = (struct net_device *) data;
+    struct net_local *lp = netdev_priv(dev);
```

```
+    unsigned int carrier;
+
+    /* Normally multiple media checks would be done, but */
+    /* KMI_ROCK_PXA forces 10Base-T, half-duplex         */
+    carrier = ( readreg(dev, PP_LineST) & ( LINK_OK | TENBASET_ON ) ) ==
+            ( LINK_OK | TENBASET_ON );
+
+    if (carrier) {
+      if (!netif_carrier_ok(dev))
+          cs_link_up(dev);
+      else if (net_debug > 1)
+          printk("%s: link ok\n", dev->name);
+      lp->media_timer.expires = jiffies + CS_TIMER_LINK;
+      add_timer(&lp->media_timer);
+    } else {
+      if (netif_carrier_ok(dev))
+          cs_link_down(dev);
+      else if (net_debug > 1)
+          printk(KERN_INFO "%s: no link, try later\n", dev->name);
+      lp->media_timer.expires = jiffies + CS_TIMER_NO_LINK;
+      add_timer(&lp->media_timer);
+    }
+}
+
 /* send a test packet - return true if carrier bits are ok */
@@ -1029,2 +1095,5 @@

+    if (net_debug)
+      printk("%s: enabling interface\n", dev->name);
+
     if (dev->irq < 2)
@@ -1207,3 +1276,6 @@
             );
+
     netif_start_queue(dev);
+    mod_timer(&lp->media_timer, jiffies + CS_TIMER_NO_LINK);
+
     if (net_debug > 1)
@@ -1211,2 +1283,3 @@
     return(0);
+
     bad_out:
@@ -1425,8 +1498,15 @@
 {
-    netif_stop_queue(dev);
+    struct net_local *lp = netdev_priv(dev);
+    unsigned long flags;

-    writereg(dev, PP_RxCFG, 0);
-    writereg(dev, PP_TxCFG, 0);
-    writereg(dev, PP_BufCFG, 0);
-    writereg(dev, PP_BusCTL, 0);
+    if (net_debug)
+      printk("%s: disabling interface\n", dev->name);
+
+    del_timer_sync(&lp->media_timer);
+
+    spin_lock_irqsave(&lp->lock, flags);
+    cs_stop_hw(dev);
+    netif_stop_queue(dev);
+    netif_carrier_off(dev);
+    spin_unlock_irqrestore(&lp->lock, flags);

@@ -1508,2 +1588,16 @@

+/*
+ * Support the 'debug' module parm even if we're compiled for non-debug to
+ * avoid breaking someone's startup scripts
```

```
+ */
+
+static int debug;
+
+module_param(debug, int, 0);
+#if DEBUGGING
+MODULE_PARM_DESC(debug, "cs89x0 debug level (0-6)");
+#else
+MODULE_PARM_DESC(debug, "(ignored)");
+#endif
+
 MODULE_AUTHOR("Mike Cruse, Russwll Nelson <nelson@crynwr.com>, Andrew Morton");
@@ -1517,2 +1611,8 @@

+#if DEBUGGING
+    net_debug = debug;
+#else
+    debug = 0;
+#endif
+
     dev = alloc_etherdev(sizeof(struct net_local));
```

Restore the Granite `cs89x0_x` driver module source files and patch them:

```
# cd /opt/linux-2.6.35.14
# tar -xzf /usr/local/src/rock1_cs89.tgz drivers/net/cs89x0_{0,1}.c
# patch -p 1 -b </usr/local/src/cs89x0.patch
patching file drivers/net/cs89x0_0.c
patching file drivers/net/cs89x0_1.c
```

Build the modified `cs89x0_x` driver modules:

```
# make M=drivers/net/ CONFIG_CS89x0=m

  WARNING: Symbol version dump /opt/linux-2.6.35.14/Module.symvers
           is missing; modules will have no dependencies and modversions.

  CC [M]  drivers/net/cs89x0_0.o
  CC [M]  drivers/net/cs89x0_1.o
  Building modules, stage 2.
  MODPOST 2 modules
  LD [M]  drivers/net/cs89x0_0.ko
  LD [M]  drivers/net/cs89x0_1.ko
```

Save the original Granite `cs89x0_x` driver modules (if not done already), and copy the new driver modules to the network driver modules directory for the running kernel:

```
# cd /lib/modules/2.6.35.14/kernel/drivers/net
# mv -n cs89x0_0.ko{,.orig}
# mv -n cs89x0_1.ko{,.orig}
# cp /opt/linux-2.6.35.14/drivers/net/cs89x0_?.ko ./
```

As a precaution, edit `/etc/modprobe.d/net.conf` and `blacklist` the `cs89x0_x` driver modules. This prevents the driver modules from automatically loading when Linux boots, in case there are problems (save the original Granite `modprobe` network drivers options file, if not done already):

```
# mv -n /etc/modprobe.d/net.conf{,.orig}
# vi /etc/modprobe.d/net.conf

blacklist cs89x0_0
blacklist cs89x0_1
```

Verify the new driver modules work on the running kernel.

Note that one must reboot first if the `cs89x0_0` *driver module is already loaded (*`lsmod | grep cs89x0`*).*

```
# modprobe cs89x0_0
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
cs89x0 media RJ-45, IRQ 139, programmed I/O, MAC 00:50:c2:5a:c5:73
eth0: using half-duplex 10Base-T (RJ-45)
```

Reboot.

Add an entry to enable the `cs89x0_x` device driver `debug` option (see section 4.6, "How to Enable the `cs89x0_x` Device Driver Debugging Option"):

```
# cat /etc/modprobe.d/net.conf.orig - >/etc/modprobe.d/net.conf <<EOF
options cs89x0_0 debug=1
options cs89x0_1 debug=1
EOF

# modprobe cs89x0_0
cs89x0.c: v2.4.3-pre1-kmi1 Russell Nelson <nelson@crynwr.com>, Andrew Morton
eth0: cs8900 rev K found at 0xf0200300 [Cirrus EEPROM]
cs89x0 media RJ-45, IRQ 139, programmed I/O, MAC 00:50:c2:5a:c5:73
cs89x0_probe1() successful
eth0: using half-duplex 10Base-T (RJ-45)
```

When the new drivers are working properly, edit `/etc/modprobe.d/net.conf` and disable the `blacklist` and `options` entries:

```
# vi /etc/modprobe.d/net.conf

#blacklist cs89x0_0
#blacklist cs89x0_1

#options cs89x0_0 debug=1
#options cs89x0_1 debug=1
```

Finally, reboot again.

7. How to Enable the Linux `netplugd` Ethernet Network Link Monitor

The Linux `netplugd` daemon monitors the link state of Ethernet interfaces by listening for kernel `netlink` events and runs a script to bring an interface up when link is established, and to bring an interface down when link is lost. Kernel `netlink` events are signaled when the Ethernet driver notifies the kernel of link state changes. Follow the instructions in section 6, "Modifications to the `cs89x0_x` Device Driver," to add the necessary calls to notify the kernel.

The `netplugd` configuration file, `/etc/netplug/netplugd.conf`, has already been configured for the two Ethernet interfaces on a Granite/Slate:

```
# cat /etc/netplug/netplugd.conf
eth0
eth1
```

Boot with an Ethernet cable plugged in.

Test the `netplugd` daemon:

```
# service netplug start
Starting network plug daemon: netplugd.
```

Issue the `ifconfig`, `route`, and `ip link` commands to observe the results:

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:50:c2:5a:c5:73
          inet addr:192.168.99.11  Bcast:192.168.99.255  Mask:255.255.255.0
          inet6 addr: fe80::250:c2ff:fe5a:c573/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:785 errors:0 dropped:0 overruns:0 frame:0
          TX packets:783 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:62982 (61.5 KiB)  TX bytes:63603 (62.1 KiB)
          Interrupt:139 Base address:0x300
```

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.99.0    *               255.255.255.0   U     0      0        0 eth0
default         192.168.99.1    0.0.0.0         UG    0      0        0 eth0
```

```
# ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UNKNOWN qlen 1000
    link/ether 00:50:c2:5a:c5:73 brd ff:ff:ff:ff:ff:ff
```

Unplug the Ethernet cable here.

When the cable is unplugged, the console displays:

```
NETDEV WATCHDOG: eth0: transmit queue 0 timed out
eth0: link down
eth0: 10Base-T (RJ-45) has no cable
eth0: no network cable attached to configured media
```

Observe the results.

Note that the Ethernet link state is `DOWN` *(not* `UP`*).*
The interface is no longer in the `RUNNING` *state.*

```
# ip link show eth0
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast state DOWN qlen
1000
    link/ether 00:50:c2:5a:c5:73 brd ff:ff:ff:ff:ff:ff
```

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:50:c2:5a:c5:73
          inet addr:192.168.99.11  Bcast:192.168.99.255  Mask:255.255.255.0
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:786 errors:0 dropped:0 overruns:0 frame:0
          TX packets:784 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:63067 (61.5 KiB)  TX bytes:63688 (62.1 KiB)
          Interrupt:139 Base address:0x300
```

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
```

Plug the Ethernet cable back in here.

When the cable is plugged in, the console displays:

```
eth0: using half-duplex 10Base-T (RJ-45)
ADDRCONF(NETDEV_UP): eth0: link is not ready
eth0: link up
ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
```

Observe the results.

The Ethernet link state is back `UP`*.*
The interface is in the `RUNNING` *state again.*

```
# ip link show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether 00:50:c2:5a:c5:73 brd ff:ff:ff:ff:ff:ff
```

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:50:c2:5a:c5:73
          inet addr:192.168.99.11  Bcast:192.168.99.255  Mask:255.255.255.0
          inet6 addr: fe80::250:c2ff:fe5a:c573/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:796 errors:0 dropped:0 overruns:0 frame:0
          TX packets:800 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:64312 (62.8 KiB)  TX bytes:64832 (63.3 KiB)
          Interrupt:139 Base address:0x300
```

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.99.0    *               255.255.255.0   U     0      0        0 eth0
default         192.168.99.1    0.0.0.0         UG    0      0        0 eth0
```

The `ipmonitor` service logs the following `netlink` `NETLINK_ROUTE` messages for the transactions above (tagged `[ROUTE]`):

```
Aug  2 00:44:18 BakerTest ipmonitor: [ROUTE]Deleted default via 192.168.99.1 dev eth0
Aug  2 00:44:19 BakerTest ipmonitor: [ROUTE]Deleted fe80::/64 dev eth0  proto kernel  met-
ric 256  mtu 1500 advmss 1440 hoplimit 0
Aug  2 00:44:19 BakerTest ipmonitor: [ROUTE]Deleted ff00::/8 dev eth0  table local  metric
256  mtu 1500 advmss 1440 hoplimit 0
Aug  2 00:44:19 BakerTest ipmonitor: [ROUTE]Deleted local fe80::250:c2ff:fe5a:c573 via ::
dev lo  table local  proto none  metric 0  mtu 16436 advmss 16376 hoplimit 0
Aug  2 00:50:19 BakerTest ipmonitor: [ROUTE]broadcast 192.168.99.255 dev eth0  table local
proto kernel  scope link  src 192.168.99.11
Aug  2 00:50:19 BakerTest ipmonitor: [ROUTE]192.168.99.0/24 dev eth0  proto kernel  scope
link  src 192.168.99.11
Aug  2 00:50:19 BakerTest ipmonitor: [ROUTE]broadcast 192.168.99.0 dev eth0  table local
proto kernel  scope link  src 192.168.99.11
Aug  2 00:50:19 BakerTest ipmonitor: [ROUTE]default via 192.168.99.1 dev eth0
Aug  2 00:50:24 BakerTest ipmonitor: [ROUTE]ff00::/8 dev eth0  table local  metric 256
mtu 1500 advmss 1440 hoplimit 0
Aug  2 00:50:24 BakerTest ipmonitor: [ROUTE]fe80::/64 dev eth0  proto kernel  metric 256
mtu 1500 advmss 1440 hoplimit 0
```

```
Aug  2 00:50:26 BakerTest ipmonitor: [ROUTE]local fe80::250:c2ff:fe5a:c573 via :: dev lo
table local  proto none  metric 0  mtu 16436 advmss 16376 hoplimit 0
```

Enable the `netplugd` daemon:

```
# update-rc.d netplug defaults
```

Finally, reboot.

References Cited

Benvenuti, Christian, 2005, Understanding Linux network internals—Guided tour to networking on Linux: O'Reilly Media, 1005 p. [Also available at http://shop.oreilly.com/product/9780596002558.do.]

Cirrus Logic, 2015, CS8900A Crystal LAN™ Ethernet Controller: Cirrus Logic product data sheet, 138 p., accessed August 1, 2016, at https://www.cirrus.com/en/pubs/proDatasheet/CS8900A_F6.pdf.

Debian.org, 2016, Debian `ifupdown` source archive:Debian.org Web site, accessed August 1, 2016, at http://archive.debian.org/debian-archive/debian/pool/main/i/ifupdown/ifupdown_0.6.7.tar.gz.

Marek, Michal, 2010, [PATCH v2] arm: Build with -fno-dwarf2-cfi-asm: Linux Kernel Mailing List posting, accessed August 1, 2016, at https://lkml.org/lkml/2010/7/26/154.

Stack Overflow, 2016, How can I log the stdout of a process started by start-stop-daemon?: Stack Overflow Web page, accessed August 1, 2016, http://stackoverflow.com/questions/8251933/how-can-i-log-the-stdout-of-a-process-started-by-start-stop-daemon.

Appendix. Linux Networking Packages, Commands, and Configuration Files Reference

All Web pages accessed August 1–2, 2016.

A.1. Debian Linux Networking Packages

`dhclient3, dhclient`
https://packages.debian.org/wheezy/dhcp3-client

`dhcpcd`	https://packages.debian.org/wheezy/dhcpcd
`ifplugd`	https://packages.debian.org/stable/net/ifplugd
`ifupdown`	https://packages.debian.org/stable/net/ifupdown
`netplug`	https://packages.debian.org/stable/main/netplug
`pump`	https://packages.debian.org/stable/pump
`udhcpc`	https://packages.debian.org/stable/udhcpc

A.2. Linux Networking Commands

`ethtool`	https://linux.die.net/man/8/ethtool
`ifconfig`	https://www.unix.com/man-page/linux/8/ifconfig
`ifdown`	https://www.unix.com/man-page/linux/8/ifdown
`ifup`	https://www.unix.com/man-page/linux/8/ifup
`ip`	https://www.unix.com/man-page/linux/8/ip
`mii-tool`	https://www.unix.com/man-page/linux/8/mii-tool
`netlink`	https://www.unix.com/man-page/linux/7/netlink
`netplugd`	https://linux.die.net/man/8/netplugd
`ping`	https://www.unix.com/man-page/linux/8/ping
`route`	https://www.unix.com/man-page/linux/8/route
`udev`	https://www.unix.com/man-page/linux/7/udev
`udevadm`	https://www.unix.com/man-page/linux/8/udevadm

A.3. Debian Linux Configuration Files

`/etc/network/interfaces`
https://www.unix.com/man-page/linux/5/interfaces

Menlo Park Publishing Service Center, California
Manuscript approved July 19, 2018
Edited by James Hendley II
Layout by Cory Hurd

ISSN 2331-1258 (online)
https://doi.org/10.3133/ofr20181117