

[patch/RFT 2.4.26-rc2] update usbnet matching 2.6.recent

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-04/2476.html>

From: David Brownell (*david-b_at_pacbell.net*)

Date: 04/12/04

Date: Mon, 12 Apr 2004 09:26:48 -0700

To: linux-kernel@vger.kernel.org, linux-usb-devel@lists.sourceforge.net

Title says it all; the 2.4 "usbnet" is missing features and fixes, this patch makes the delta from 2.6 be pretty small.

Its Config.in menu now looks more like 2.6, with per-minidriver config options. This matters mostly since the AX8817X driver will no longer be hidden (it's good for getting good 100BaseT speeds using EHCI).

On 2.6, "usbnet" handles CDC Ethernet ... so this patch disables "CDCEther", which may exceed some folk's expectations of what should happen in a "stable" kernel. However: (a) it's what Zaurus users have needed to do anyway (Zaurus claims CDC support, but lies), (b) this seems to work well on 2.6 systems, better on some hardware than "CDCEther", and (c) the CDCEther maintainer (GregKH) seemed to think making "usbnet" do this was OK last time this came up.

If you use "usbnet" or especially "CDCEther", and have the time to apply, please give this a whirl ... I don't think there should be many issues, there's not a lot of 2.4-specific code here.

- Dave

The "usbnet" driver has gotten a bit out of sync with the newer code (2.6.5+); this fixes that. Notable changes:

- Support more devices: Zaurus C-860, ALi M5632, and the new Linux-USB Ethernet/RNDIS gadget.
- Supports CDC Ethernet. 2.6 doesn't use CDCEther, and this probably supports some hardware that CDCEther won't.
- The kernel config setup changed to match 2.6. If in doubt, "y" for all the new options. Among other things, this means that

Linux–Kernel: [patch/RFT 2.4.26–rc2] update usbnet matching 2.6.recent

the ax8817x support is no longer invisible!

- Disables the old "CDCEther.c" driver in Config.in; probably not the ideal solution, but it'll simplify testing.
- Various chip–specific updates/fixes: net1080 flushes fifos to recover after framing errors; ax8817x gets lots of ethtool support, including WOL, MII, and EEPROM access.
- Cleanups, including using alloc_etherdev and ethtool_ops. Throttle down resubmit during temporary faults (and unplug).

--- 1.239/Documentation/Configure.help Thu Apr 1 02:19:49 2004
+++ edited/Documentation/Configure.help Mon Apr 12 08:16:38 2004
@@ -15847,24 +15847,35 @@

The module will be called dabusb.o. If you want to compile it as a module, say M here and read <file:Documentation/modules.txt>.

–Host–to–Host USB networking

+Multi–purpose USB Networking Framework

CONFIG_USB_USBNET

- This driver supports network links over USB with USB "Network" or "data transfer" cables, often used to network laptops to PCs.
- Such cables have chips from suppliers such as Belkin/eTEK, GeneSys (GeneLink), NetChip and Prolific. Intelligent USB devices could also use this approach to provide Internet access, using standard USB cabling. You can find these chips also on some motherboards with USB PC2PC support.
-
- These links will have names like "usb0", "usb1", etc. They act like two–node Ethernets, so you can use 802.1d Ethernet Bridging (CONFIG_BRIDGE) to simplify your network routing.
-
- This code is also available as a kernel module (code which can be inserted in and removed from the running kernel whenever you want).
- The module will be called usbnet.o. If you want to compile it as a module, say M here and read <file:Documentation/modules.txt>.
- + This driver supports several kinds of network links over USB, with "minidriver" built around a common network driver core.
- + The USB host runs "usbnet", and the other end of the link might be:
 - + – Another USB host, when using USB "network" or "data transfer" cables. These are often used to network laptops to PCs, like "Laplink" parallel cables or some motherboards. These rely on specialized chips from many suppliers.
 - + – An intelligent USB gadget, perhaps embedding a Linux system. These include PDAs running Linux (iPaq, Yopy, Zaurus, and others), and devices that interoperate using the standard CDC–Ethernet specification (including many cable modems).

- +
- + – Network adapter hardware (like those for 10/100 Ethernet) which
- + uses this driver framework.
- +
- + The link will appear with a name like "usb0", when the link is
- + a two-node link, or "eth0" for most Ethernet devices. Those
- + two-node links are most easily managed with Ethernet Bridging
- + (CONFIG_BRIDGE) instead of routing.
- +
- + For more information see <<http://www.linux-usb.org/usbnet/>>.
- +
- + To compile this driver as a module, choose M here: the
- + module will be called usbnet.

Freecom USB/ATAPI Bridge support

CONFIG_USB_STORAGE_FREECOM

--- 1.41/drivers/usb/Config.in Mon Jan 19 04:42:57 2004

+++ edited/drivers/usb/Config.in Mon Apr 12 08:16:38 2004

@@ -92,8 +92,24 @@

```
    dep_tristate 'USB Realtek RTL8150 based ethernet device support (EXPERIMENTAL)'
CONFIG_USB_RTL8150 $CONFIG_USB $CONFIG_NET $CONFIG_EXPERIMENTAL
    dep_tristate 'USB KLSI KL5USB101-based ethernet device support (EXPERIMENTAL)'
CONFIG_USB_KAWETH $CONFIG_USB $CONFIG_NET $CONFIG_EXPERIMENTAL
    dep_tristate 'USB CATC NetMate-based Ethernet device support (EXPERIMENTAL)'
CONFIG_USB_CATC $CONFIG_USB $CONFIG_NET $CONFIG_EXPERIMENTAL
- dep_tristate 'USB Communication Class Ethernet device support (EXPERIMENTAL)'
CONFIG_USB_CDCETHER $CONFIG_USB $CONFIG_NET $CONFIG_EXPERIMENTAL
- dep_tristate 'USB-to-USB Networking cables, Linux PDAs, ... (EXPERIMENTAL)'
CONFIG_USB_USBNET $CONFIG_USB $CONFIG_NET $CONFIG_EXPERIMENTAL
+ # dep_tristate 'USB Communication Class Ethernet device support (EXPERIMENTAL)'
CONFIG_USB_CDCETHER $CONFIG_USB $CONFIG_NET $CONFIG_EXPERIMENTAL
+ dep_tristate 'Multi-purpose USB Networking Framework (EXPERIMENTAL)' CONFIG_USB_USBNET
$CONFIG_USB $CONFIG_NET $CONFIG_EXPERIMENTAL
+ if [ "$CONFIG_USB_USBNET" = "y" -o "$CONFIG_USB_USBNET" = "m" ]; then
+ comment 'Host-to-Host cables'
+ bool 'ALi M5632 USB 2.0' CONFIG_USB_ALI_5632
+ bool 'AnchorChips 2720' CONFIG_USB_AN2720
+ bool 'Belkin/ETek' CONFIG_USB_BELKIN
+ bool 'GeneSys GL620USB-A' CONFIG_USB_GENESYS
+ bool 'NetChip 1080' CONFIG_USB_NET1080
+ bool 'Prolific PL-2301/2302' CONFIG_USB_PL2301
+ comment 'Embedded firmware'
+ bool 'ARM (and other) embedded Linux' CONFIG_USB_ARMLINUX
+ bool 'CDC Ethernet' CONFIG_USB_CDCETHER
+ bool 'EPSON sample firmware' CONFIG_USB_EPSON2888
+ bool 'Sharp Zaurus (many models)' CONFIG_USB_ZAURUS
+ comment 'Ethernet Adapters'
+ bool 'ASIX AX8817X based USB 2.0' CONFIG_USB_AX8817X
+ fi
fi
```

```
comment 'USB port drivers'
--- 1.27/drivers/usb/usbnet.c Thu Oct 23 08:02:44 2003
+++ edited/drivers/usb/usbnet.c Mon Apr 12 08:16:38 2004
@@ -1,7 +1,9 @@
/*
- * USB Host-to-Host Links
- * Copyright (C) 2000-2002 by David Brownell <dbrownell@users.sourceforge.net>
+ * USB Networking Links
+ * Copyright (C) 2000-2003 by David Brownell <dbrownell@users.sourceforge.net>
+ * Copyright (C) 2002 Pavel Machek <pavel@ucw.cz>
+ * Copyright (C) 2003 David Hollis <dhollis@davehollis.com>
+ * Copyright (c) 2002-2003 TiVo Inc.
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
@@ -19,63 +21,38 @@
*/

/*
- * This is used for "USB networking", connecting USB hosts as peers.
- *
- * It can be used with USB "network cables", for IP-over-USB communications;
- * Ethernet speeds without the Ethernet. USB devices (including some PDAs)
- * can support such links directly, replacing device-specific protocols
- * with Internet standard ones.
- *
- * The links can be bridged using the Ethernet bridging (net/bridge)
- * support as appropriate. Devices currently supported include:
+ * This is a generic "USB networking" framework that works with several
+ * kinds of full and high speed networking devices:
+ *
+ * + USB host-to-host "network cables", used for IP-over-USB links.
+ * These are often used for Laplink style connectivity products.
+ * - AnchorChip 2720
+ * - Belkin, eTEK (interopes with Win32 drivers)
- * - EPSON USB clients
+ * - GeneSys GL620USB-A
+ * - NetChip 1080 (interoperates with NetChip Win32 drivers)
+ * - Prolific PL-2301/2302 (replaces "plusb" driver)
- * - PXA-250 or SA-1100 Linux PDAs like iPAQ, Yopy, and Zaurus
+ *
+ * + Smart USB devices can support such links directly, using Internet
+ * standard protocols instead of proprietary host-to-device links.
+ * - Linux PDAs like iPaq, Yopy, and Zaurus
+ * - The BLOB boot loader (for diskless booting)
+ * - Linux "gadgets", perhaps using PXA-2xx or Net2280 controllers
+ * - Devices using EPSON's sample USB firmware
+ * - CDC-Ethernet class devices, such as many cable modems
+ *
+ * + Adapters to networks such as Ethernet.
+ * - AX8817X based USB 2.0 products
```

```
+ *
+ * Links to these devices can be bridged using Linux Ethernet bridging.
+ * With minor exceptions, these all use similar USB framing for network
+ * traffic, but need different protocols for control traffic.
+ *
+ * USB devices can implement their side of this protocol at the cost
+ * of two bulk endpoints; it's not restricted to "cable" applications.
+ * See the SA1110, Zaurus, or EPSON device/client support in this driver;
- * slave/target drivers such as "usb-eth" (on most SA-1100 PDAs) are
- * used inside USB slave/target devices.
- *
- *
- * Status:
- *
- * - AN2720 ... not widely available, but reportedly works well
- *
- * - Belkin/eTEK ... no known issues
- *
- * - Both GeneSys and PL-230x use interrupt transfers for driver-to-driver
- * handshaking; it'd be worth implementing those as "carrier detect".
- * Prefer generic hooks, not minidriver-specific hacks.
- *
- * - For Netchip, should use keventd to poll via control requests to detect
- * hardware level "carrier detect".
- *
- * - PL-230x ... the initialization protocol doesn't seem to match chip data
- * sheets, sometimes it's not needed and sometimes it hangs. Prolific has
- * not responded to repeated support/information requests.
- *
- * - SA-1100 PDAs ... the standard ARM Linux SA-1100 support works nicely,
- * as found in www.handhelds.org and other kernels. The Sharp/Lineo
- * kernels use different drivers, which also talk to this code.
- *
- * Interop with more Win32 drivers may be a good thing.
- *
- * Seems like reporting "peer connected" (carrier present) events may end
- * up going through the netlink event system, not hotplug ... so new links
- * would likely be handled with a link monitoring thread in some daemon.
- *
- * There are reports that bridging gives lower-than-usual throughput.
- *
- * Need smarter hotplug policy scripts ... ones that know how to arrange
- * bridging with "brctl", and can handle static and dynamic ("pump") setups.
- * Use those eventual "peer connected" events, and zeroconf.
+ * slave/target drivers such as "usb-eth" (on most SA-1100 PDAs) or
+ * "g_ether" (in the Linux "gadget" framework) implement that behavior
+ * within devices.
+ *
+ *
+ * CHANGELOG:
@@ -118,12 +95,24 @@
```

Linux-Kernel: [patch/RFT 2.4.26-rc2] update usbnet matching 2.6.recent

```
* for USB 2.0 TTs) and memory shortages (potential) too. (db)
* Use "locally assigned" IEEE802 address space. (Brad Hards)
* 18-oct-2002 Support for Zaurus (Pavel Machek), related cleanup (db).
- * 15-dec-2002 Partial sync with 2.5 code: cleanups and stubbed PXA-250
- * support (db), fix for framing issues on Z, net1080, and
- * gl620a (Toby Milne)
+ * 14-dec-2002 Remove Zaurus-private crc32 code (Pavel); 2.5 oops fix,
+ * cleanups and stubbed PXA-250 support (db), fix for framing
+ * issues on Z, net1080, and gl620a (Toby Milne)
+ *
+ * 31-mar-2003 Use endpoint descriptors: high speed support, simpler sa1100
+ * vs pxa25x, and CDC Ethernet. Throttle down log floods on
+ * disconnect; other cleanups. (db) Flush net1080 fifos
+ * after several sequential framing errors. (Johannes Erdfelt)
+ * 22-aug-2003 AX8817X support (Dave Hollis).
+ *
+ * 12-apr-2004 Resync 2.4.26 with 2.6.6: CDC Ethernet, C-860, M5632;
+ * alloc_etherdev, ethool_ops, net1080 flushes, ax8817x updates;
+ * cleanups, throttling, ether/rndis gadget.
*
*-----*/

+// #define DEBUG 1
+
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kmod.h>
@@ -147,28 +136,18 @@
#endif
#include <linux/usb.h>

-/* in 2.5 these standard usb ops take mem_flags */
-#define ALLOC_URB(n,flags) usb_alloc_urb(n)
-#define SUBMIT_URB(u,flags) usb_submit_urb(u)
-
-/* and these got renamed (may move to usb.h) */
-#define usb_get_dev usb_inc_dev_use
-#define usb_put_dev usb_dec_dev_use
-
-
-/* minidrivres _could_ be individually configured */
-#define CONFIG_USB_AN2720
-#define CONFIG_USB_AX8817X
-#define CONFIG_USB_BELKIN
-#define CONFIG_USB_EPSON2888
-#define CONFIG_USB_GENESYS
-#define CONFIG_USB_NET1080
-#define CONFIG_USB_PL2301
-#define CONFIG_USB_ARMLINUX
-#define CONFIG_USB_ZAURUS
+#define USB_DT_CS_INTERFACE 0x24
```

```

+/* in 2.6 these changed ... */
+#define usb_alloc_urb(n,flags) usb_alloc_urb(n)
+#define usb_submit_urb(u,flags) usb_submit_urb(u)
+#define usb_get_dev(u) ({ usb_inc_dev_use(u); u; })
+#define usb_put_dev(u) usb_dec_dev_use(u)
+#define usb_set_intfdata(intf,val) ((void)((intf)->private_data = val))
+#define URB_ASYNC_UNLINK USB_ASYNC_UNLINK;

-#define DRIVER_VERSION "18-Oct-2002"
+
+#define DRIVER_VERSION "12-Apr-2003/2.4"

/*-----*/

@@ -180,11 +159,11 @@
 * Ethernet packets (so queues should be bigger).
 */
#ifdef REALLY_QUEUE
-#define RX_QLEN 4
-#define TX_QLEN 4
+#define RX_QLEN(dev) (((dev)->udev->speed == USB_SPEED_HIGH) ? 60 : 4)
+#define TX_QLEN(dev) (((dev)->udev->speed == USB_SPEED_HIGH) ? 60 : 4)
#else
-#define RX_QLEN 1
-#define TX_QLEN 1
+#define RX_QLEN(dev) 1
+#define TX_QLEN(dev) 1
#endif

// packets are always ethernet inside
@@ -195,6 +174,10 @@
// reawaken network queue this soon after stopping; else watchdog barks
#define TX_TIMEOUT_JIFFIES (5*HZ)

+// throttle rx/tx briefly after some faults, so khubd might disconnect()
+// us (it polls at HZ/4 usually) before we report too many false errors.
+#define THROTTLE_JIFFIES (HZ/8)
+
+// for vendor-specific control operations
+#define CONTROL_TIMEOUT_MS (500) /* msec */
+#define CONTROL_TIMEOUT_JIFFIES ((CONTROL_TIMEOUT_MS * HZ)/1000)
@@ -204,10 +187,6 @@

/*-----*/

-// list of all devices we manage
-static DECLARE_MUTEX (usbnet_mutex);
-static LIST_HEAD (usbnet_list);
-
// randomly generated ethernet address

```

```

static u8 node_id [ETH_ALEN];

@@ -216,24 +195,20 @@
    // housekeeping
    struct usb_device *udev;
    struct driver_info *driver_info;
- struct semaphore mutex;
- struct list_head dev_list;
    wait_queue_head_t *wait;

    // i/o info: pipes etc
    unsigned in, out;
    unsigned maxpacket;
- //struct timer_list delay;
+ struct timer_list delay;

    // protocol/interface state
- struct net_device net;
+ struct net_device *net;
    struct net_device_stats stats;
    int msg_level;
- struct mii_if_info mii;
+ unsigned long data [5];

-#ifdef CONFIG_USB_NET1080
- u16 packet_id;
-#endif
+ struct mii_if_info mii;

    // various kinds of pending driver work
    struct sk_buff_head rxq;
@@ -253,14 +228,20 @@
    char *description;

    int flags;
+/* framing is CDC Ethernet, not writing ZLPs (hw issues), or optionally: */
#define FLAG_FRAMING_NC 0x0001 /* guard against device dropouts */
#define FLAG_FRAMING_GL 0x0002 /* genelink batches packets */
#define FLAG_FRAMING_Z 0x0004 /* zaurus adds a trailer */
+##define FLAG_FRAMING_RN 0x0008 /* RNDIS batches, plus huge header */
+
#define FLAG_NO_SETINT 0x0010 /* device can't set_interface() */
#define FLAG_ETHER 0x0020 /* maybe use "eth%d" names */

    /* init device ... can sleep, or cause probe() failure */
- int (*bind)(struct usbnet *, struct usb_device *);
+ int (*bind)(struct usbnet *, struct usb_interface *);
+
+ /* cleanup device ... can sleep, but can't fail */
+ void (*unbind)(struct usbnet *, struct usb_interface *);

```

Linux-Kernel: [patch/RFT 2.4.26-rc2] update usbnet matching 2.6.recent

```

    /* reset device ... can sleep */
    int (*reset)(struct usbnet *);
@@ -277,11 +258,11 @@

    // FIXME -- also an interrupt mechanism
    // useful for at least PL2301/2302 and GL620USB-A
+ // and CDC; use them to report 'is it connected' changes

    /* for new devices, use the descriptor-reading code instead */
    int in; /* rx endpoint */
    int out; /* tx endpoint */
- int epsize;

    unsigned long data; /* Misc driver specific data */
};
@@ -308,22 +289,39 @@
MODULE_PARM_DESC(msg_level, "Initial message level (default = 1)");

-#define mutex_lock(x) down(x)
-#define mutex_unlock(x) up(x)
-
#define RUN_CONTEXT(in_irq) ? "in_irq" \
    : (in_interrupt) ? "in_interrupt" : "can sleep")

-static struct ethtool_ops usbnet_ethtool_ops;
+#ifdef DEBUG
+#define devdbg(usbnet, fmt, arg...) \
+ printk(KERN_DEBUG "%s: " fmt "\n", (usbnet)->net->name, ## arg)
+#else
+#define devdbg(usbnet, fmt, arg...) do { } while(0)
+#endif
+
+#define devert(usbnet, fmt, arg...) \
+ printk(KERN_ERR "%s: " fmt "\n", (usbnet)->net->name, ## arg)
+#define devwarn(usbnet, fmt, arg...) \
+ printk(KERN_WARNING "%s: " fmt "\n", (usbnet)->net->name, ## arg)
+
+#define devinfo(usbnet, fmt, arg...) \
+ do { if ((usbnet)->msg_level >= 1) \
+ printk(KERN_INFO "%s: " fmt "\n", (usbnet)->net->name, ## arg); \
+ } while (0)
+
+/*-----*/

-/* mostly for PDA style devices, which are always present */
+static void usbnet_get_drvinfo (struct net_device *, struct ethtool_drvinfo *);
+static u32 usbnet_get_link (struct net_device *);
+static u32 usbnet_get_msglevel (struct net_device *);
+static void usbnet_set_msglevel (struct net_device *, u32);
+

```

Linux-Kernel: [patch/RFT 2.4.26-rc2] update usbnet matching 2.6.recent

```

+/* mostly for PDA style devices, which are always connected if present */
static int always_connected (struct usbnet *dev)
{
    return 0;
}

-/*-----*/
-
/* handles CDC Ethernet and many other network "bulk data" interfaces */
static int
get_endpoints (struct usbnet *dev, struct usb_interface *intf)
@@ -377,22 +375,45 @@
    return 0;
}

-/*-----*/

+static void skb_return (struct usbnet *dev, struct sk_buff *skb)
+{
+ int status;

-#ifdef DEBUG
-#define devdbg(usbnet, fmt, arg...) \
- printk(KERN_DEBUG "%s: " fmt "\n" , (usbnet)->net.name , ## arg)
-#else
-#define devdbg(usbnet, fmt, arg...) do { } while(0)
+ skb->dev = dev->net;
+ skb->protocol = eth_type_trans (skb, dev->net);
+ dev->stats.rx_packets++;
+ dev->stats.rx_bytes += skb->len;
+
+#ifdef VERBOSE
+ devdbg (dev, "< rx, len %d, type 0x%x",
+ skb->len + sizeof (struct ethhdr), skb->protocol);
+ #endif
+ memset (skb->cb, 0, sizeof (struct skb_data));
+ status = netif_rx (skb);
+ if (status != NET_RX_SUCCESS)
+ devdbg (dev, "netif_rx status %d", status);
+ }
+
+
+#ifdef CONFIG_USB_ALI_M5632
+#define HAVE_HARDWARE
+
+/*-----*/
+ *
+ * ALi M5632 driver ... does high speed
+ *
+ *-----*/

+static const struct driver_info ali_m5632_info = {

```

```

+ .description = "ALi M5632",
+};

-#define devinfo(usbnet, fmt, arg...) \
- do { if ((usbnet)->msg_level >= 1) \
- printk(KERN_INFO "%s: " fmt "\n" , (usbnet)->net.name , ## arg); \
- } while (0)
+
+#endif

#ifdef CONFIG_USB_AN2720
+#define HAVE_HARDWARE

/*-----
*
@@ -411,7 +432,6 @@
    // no check_connect available!

    .in = 2, .out = 2, // direction distinguishes these
- .epsize = 64,
};

#endif /* CONFIG_USB_AN2720 */
@@ -429,6 +449,8 @@
#define AX_CMD_READ_MII_REG 0x07
#define AX_CMD_WRITE_MII_REG 0x08
#define AX_CMD_SET_HW_MII 0x0a
+#define AX_CMD_READ_EEPROM 0x0b
+#define AX_CMD_WRITE_EEPROM 0x0c
#define AX_CMD_WRITE_RX_CTL 0x10
#define AX_CMD_READ_IPG012 0x11
#define AX_CMD_WRITE_IPG0 0x12
@@ -438,8 +460,15 @@
#define AX_CMD_READ_NODE_ID 0x17
#define AX_CMD_READ_PHY_ID 0x19
#define AX_CMD_WRITE_MEDIUM_MODE 0x1b
+#define AX_CMD_READ_MONITOR_MODE 0x1c
+#define AX_CMD_WRITE_MONITOR_MODE 0x1d
#define AX_CMD_WRITE_GPIOS 0x1f

+#define AX_MONITOR_MODE 0x01
+#define AX_MONITOR_LINK 0x02
+#define AX_MONITOR_MAGIC 0x04
+#define AX_MONITOR_HSFS 0x10
+
+
#define AX_MCAST_FILTER_SIZE 8
#define AX_MAX_MCAST 64

@@ -492,13 +521,13 @@
    int status;

```

```

struct urb *urb;

- if ((urb = ALLOC_URB(0, GFP_ATOMIC)) == NULL) {
+ if ((urb = usb_alloc_urb(0, GFP_ATOMIC)) == NULL) {
    devdbg(dev, "Error allocating URB in write_cmd_async!");
    return;
}

    if ((req = kmalloc(sizeof(struct usb_ctrlrequest), GFP_ATOMIC)) == NULL) {
- devdbg(dev, "Failed to allocate memory for control request");
+ deverr(dev, "Failed to allocate memory for control request");
        usb_free_urb(urb);
        return;
    }
@@ -514,8 +543,11 @@
        (void *)req, data, size,
        ax8817x_async_cmd_callback, req);

- if((status = SUBMIT_URB(urb, GFP_ATOMIC)) < 0)
- devdbg(dev, "Error submitting the control message: status=%d", status);
+ if((status = usb_submit_urb(urb, GFP_ATOMIC)) < 0) {
+ deverr(dev, "Error submitting the control message: status=%d", status);
+ kfree(req);
+ usb_free_urb(urb);
+ }
}

static void ax8817x_set_multicast(struct net_device *net)
@@ -531,34 +563,31 @@
    } else if (net->mc_count == 0) {
        /* just broadcast and directed */
    } else {
+ /* We use the 20 byte dev->data
+ * for our 8 byte filter buffer
+ * to avoid allocating memory that
+ * is tricky to free later */
+ u8 *multi_filter = (u8 *)&dev->data;
        struct dev_mc_list *mc_list = net->mc_list;
- u8 *multi_filter;
        u32 crc_bits;
        int i;

- multi_filter = kmalloc(AX_MCAST_FILTER_SIZE, GFP_ATOMIC);
- if (multi_filter == NULL) {
- /* Oops, couldn't allocate a buffer for setting the multicast
- filter. Try all multi mode. */
- rx_ctl |= 0x02;
- } else {
- memset(multi_filter, 0, AX_MCAST_FILTER_SIZE);
-
- /* Build the multicast hash filter. */

```

```

- for (i = 0; i < net->mc_count; i++) {
-   crc_bits =
-   ether_crc(ETH_ALEN,
-   mc_list->dmi_addr) >> 26;
-   multi_filter[crc_bits >> 3] |=
-   1 << (crc_bits & 7);
-   mc_list = mc_list->next;
- }
-
- ax8817x_write_cmd_async(dev, AX_CMD_WRITE_MULTI_FILTER, 0, 0,
- AX_MCAST_FILTER_SIZE, multi_filter);
+ memset(multi_filter, 0, AX_MCAST_FILTER_SIZE);

- rx_ctl |= 0x10;
+ /* Build the multicast hash filter. */
+ for (i = 0; i < net->mc_count; i++) {
+   crc_bits =
+   ether_crc(ETH_ALEN,
+   mc_list->dmi_addr) >> 26;
+   multi_filter[crc_bits >> 3] |=
+   1 << (crc_bits & 7);
+   mc_list = mc_list->next;
+   }
+
+ ax8817x_write_cmd_async(dev, AX_CMD_WRITE_MULTI_FILTER, 0, 0,
+ AX_MCAST_FILTER_SIZE, multi_filter);
+
+ rx_ctl |= 0x10;
+   }

    ax8817x_write_cmd_async(dev, AX_CMD_WRITE_RX_CTL, rx_ctl, 0, 0, NULL);
@@ -568,7 +597,7 @@
{
    struct usbnet *dev = netdev->priv;
    u16 res;
- u8 buf[4];
+ u8 buf[1];

    ax8817x_write_cmd(dev, AX_CMD_SET_SW_MII, 0, 0, 0, &buf);
    ax8817x_read_cmd(dev, AX_CMD_READ_MII_REG, phy_id, (__u16)loc, 2, (u16 *)&res);
@@ -581,14 +610,120 @@
{
    struct usbnet *dev = netdev->priv;
    u16 res = val;
- u8 buf[4];
+ u8 buf[1];

    ax8817x_write_cmd(dev, AX_CMD_SET_SW_MII, 0, 0, 0, &buf);
    ax8817x_write_cmd(dev, AX_CMD_WRITE_MII_REG, phy_id, (__u16)loc, 2, (u16 *)&res);
    ax8817x_write_cmd(dev, AX_CMD_SET_HW_MII, 0, 0, 0, &buf);
}

```

```

-static int ax8817x_bind(struct usbnet *dev, struct usb_device *intf)
+void ax8817x_get_wol(struct net_device *net, struct ethtool_wolinfo *wolinfo)
+{
+ struct usbnet *dev = (struct usbnet *)net->priv;
+ u8 opt;
+
+ if (ax8817x_read_cmd(dev, AX_CMD_READ_MONITOR_MODE, 0, 0, 1, &opt) < 0) {
+ wolinfo->supported = 0;
+ wolinfo->wlopts = 0;
+ return;
+ }
+ wolinfo->supported = WAKE_PHY | WAKE_MAGIC;
+ wolinfo->wlopts = 0;
+ if (opt & AX_MONITOR_MODE) {
+ if (opt & AX_MONITOR_LINK)
+ wolinfo->wlopts |= WAKE_PHY;
+ if (opt & AX_MONITOR_MAGIC)
+ wolinfo->wlopts |= WAKE_MAGIC;
+ }
+ }
+
+int ax8817x_set_wol(struct net_device *net, struct ethtool_wolinfo *wolinfo)
+{
+ struct usbnet *dev = (struct usbnet *)net->priv;
+ u8 opt = 0;
+ u8 buf[1];
+
+ if (wolinfo->wlopts & WAKE_PHY)
+ opt |= AX_MONITOR_LINK;
+ if (wolinfo->wlopts & WAKE_MAGIC)
+ opt |= AX_MONITOR_MAGIC;
+ if (opt != 0)
+ opt |= AX_MONITOR_MODE;
+
+ if (ax8817x_write_cmd(dev, AX_CMD_WRITE_MONITOR_MODE,
+ opt, 0, 0, &buf) < 0)
+ return -EINVAL;
+
+ return 0;
+ }
+
+int ax8817x_get_eeprom(struct net_device *net,
+ struct ethtool_eeprom *eeprom, u8 *data)
+{
+ struct usbnet *dev = (struct usbnet *)net->priv;
+ u16 *ebuf = (u16 *)data;
+ int i;
+
+ /* Crude hack to ensure that we don't overwrite memory
+ * if an odd length is supplied

```

```

+ */
+ if (eeprom->len % 2)
+ return -EINVAL;
+
+ /* ax8817x returns 2 bytes from eeprom on read */
+ for (i=0; i < eeprom->len / 2; i++) {
+ if (ax8817x_read_cmd(dev, AX_CMD_READ_EEPROM,
+ eeprom->offset + i, 0, 2, &ebuf[i]) < 0)
+ return -EINVAL;
+ }
+ return i * 2;
+}
+
+static void ax8817x_get_drvinfo (struct net_device *net,
+ struct ethtool_drvinfo *info)
+{
+ /* Inherit standard device info */
+ usbnet_get_drvinfo(net, info);
+ info->eedump_len = 0x3e;
+}
+
+static u32 ax8817x_get_link (struct net_device *net)
+{
+ struct usbnet *dev = (struct usbnet *)net->priv;
+
+ return (u32)mii_link_ok(&dev->mii);
+}
+
+static int ax8817x_get_settings(struct net_device *net, struct ethtool_cmd *cmd)
+{
+ struct usbnet *dev = (struct usbnet *)net->priv;
+
+ return mii_ethtool_gset(&dev->mii,cmd);
+}
+
+static int ax8817x_set_settings(struct net_device *net, struct ethtool_cmd *cmd)
+{
+ struct usbnet *dev = (struct usbnet *)net->priv;
+
+ return mii_ethtool_sset(&dev->mii,cmd);
+}
+
+/* We need to override some ethtool_ops so we require our
+ own structure so we don't interfere with other usbnet
+ devices that may be connected at the same time. */
+static struct ethtool_ops ax8817x_ethtool_ops = {
+ .get_drvinfo = ax8817x_get_drvinfo,
+ .get_link = ax8817x_get_link,
+ .get_msglevel = usbnet_get_msglevel,
+ .set_msglevel = usbnet_set_msglevel,
+ .get_wol = ax8817x_get_wol,

```

```

+ .set_wol = ax8817x_set_wol,
+ .get_eeprom = ax8817x_get_eeprom,
+ .get_settings = ax8817x_get_settings,
+ .set_settings = ax8817x_set_settings,
+};
+
+static int ax8817x_bind(struct usbnet *dev, struct usb_interface *intf)
{
    int ret;
    u8 buf[6];
@@ -602,12 +737,12 @@
    /* Toggle the GPIOs in a manufacturer/model specific way */
    for (i = 2; i >= 0; i--) {
        if ((ret = ax8817x_write_cmd(dev, AX_CMD_WRITE_GPIOS,
- (gpio_bits >> (i * 8)) & 0xff, 0, 0,
- buf) < 0)
+ (gpio_bits >> (i * 8)) & 0xff, 0, 0,
+ buf) < 0)
            return ret;
        wait_ms(5);
- }
-
+ }
+
    if ((ret = ax8817x_write_cmd(dev, AX_CMD_WRITE_RX_CTL, 0x80, 0, 0, buf) < 0) {
        dbg("send AX_CMD_WRITE_RX_CTL failed: %d", ret);
        return ret;
@@ -619,17 +754,7 @@
        dbg("read AX_CMD_READ_NODE_ID failed: %d", ret);
        return ret;
    }
- memcpy(dev->net.dev_addr, buf, ETH_ALEN);
-
- /* Get IPG values */
- if ((ret = ax8817x_read_cmd(dev, AX_CMD_READ_IPG012, 0, 0, 3, buf) < 0) {
-     dbg("Error reading IPG values: %d", ret);
-     return ret;
- }
-
- for(i = 0; i < 3; i++) {
-     ax8817x_write_cmd(dev, AX_CMD_WRITE_IPG0 + i, 0, 0, 1, &buf[i]);
- }
+ memcpy(dev->net->dev_addr, buf, ETH_ALEN);

    /* Get the PHY id */
    if ((ret = ax8817x_read_cmd(dev, AX_CMD_READ_PHY_ID, 0, 0, 2, buf) < 0) {
@@ -642,7 +767,7 @@
    }

    /* Initialize MII structure */
- dev->mii.dev = &dev->net;

```

```

+ dev->mii.dev = dev->net;
  dev->mii.mdio_read = ax8817x_mdio_read;
  dev->mii.mdio_write = ax8817x_mdio_write;
  dev->mii.phy_id_mask = 0x3f;
@@ -683,7 +808,8 @@
    return ret;
  }

- dev->net.set_multicast_list = ax8817x_set_multicast;
+ dev->net->set_multicast_list = ax8817x_set_multicast;
+ dev->net->ethtool_ops = &ax8817x_ethtool_ops;

    return 0;
  }
@@ -715,10 +841,13 @@
    .flags = FLAG_ETHER,
    .data = 0x001f1d1f,
  };
+
+ #endif /* CONFIG_USB_AX8817X */

+

+ #ifdef CONFIG_USB_BELKIN
+ #define HAVE_HARDWARE

+ /*-----
+ *
+ @ -736,7 +865,333 @@

+ /*-----
+ *
+ * Communications Device Class declarations.
+ * Used by CDC Ethernet, and some CDC variants
+ *-----*/
+
+ #ifdef CONFIG_USB_CDCETHER
+ #define NEED_GENERIC_CDC
+ #endif
+
+ #ifdef CONFIG_USB_ZAURUS
+ /* Ethernet variant uses funky framing, broken ethernet addressing */
+ #define NEED_GENERIC_CDC
+ #endif
+
+ #ifdef CONFIG_USB_RNDIS
+ /* ACM variant uses even funkier framing, complex control RPC scheme */
+ #define NEED_GENERIC_CDC

```

```

+ #endif
+
+
+ #ifdef NEED_GENERIC_CDC
+
+ /* "Header Functional Descriptor" from CDC spec 5.2.3.1 */
+ struct header_desc {
+     u8 bLength;
+     u8 bDescriptorType;
+     u8 bDescriptorSubType;
+
+
+     u16 bcdCDC;
+ } __attribute__((packed));
+
+ /* "Union Functional Descriptor" from CDC spec 5.2.3.X */
+ struct union_desc {
+     u8 bLength;
+     u8 bDescriptorType;
+     u8 bDescriptorSubType;
+
+
+     u8 bMasterInterface0;
+     u8 bSlaveInterface0;
+     /* ... and there could be other slave interfaces */
+ } __attribute__((packed));
+
+ /* "Ethernet Networking Functional Descriptor" from CDC spec 5.2.3.16 */
+ struct ether_desc {
+     u8 bLength;
+     u8 bDescriptorType;
+     u8 bDescriptorSubType;
+
+
+     u8 iMACAddress;
+     u32 bmEthernetStatistics;
+     u16 wMaxSegmentSize;
+     u16 wNumberMCFilters;
+     u8 bNumberPowerFilters;
+ } __attribute__((packed));
+
+ struct cdc_state {
+     struct header_desc *header;
+     struct union_desc *u;
+     struct ether_desc *ether;
+     struct usb_interface *control;
+     struct usb_interface *data;
+ };
+
+ static struct usb_driver usbnet_driver;
+
+
+ /*
+  * probes control interface, claims data interface, collects the bulk
+  * endpoints, activates data interface (if needed), maybe sets MTU.

```

```

+ * all pure cdc, except for certain firmware workarounds.
+ */
+static int generic_cdc_bind (struct usbnet *dev, struct usb_interface *intf)
+{
+ u8 *buf = intf->altsetting->extra;
+ int len = intf->altsetting->extralen;
+ struct usb_interface_descriptor *d;
+ struct cdc_state *info = (void *) &dev->data;
+ int status;
+ int rndis;
+
+ if (sizeof dev->data < sizeof *info)
+ return -EDOM;
+
+ /* expect strict spec conformance for the descriptors, but
+ * cope with firmware which stores them in the wrong place
+ */
+ if (len == 0 && dev->udev->actconfig->extralen) {
+ /* Motorola SB4100 (and others: Brad Hards says it's
+ * from a Broadcom design) put CDC descriptors here
+ */
+ buf = dev->udev->actconfig->extra;
+ len = dev->udev->actconfig->extralen;
+ if (len)
+ pr_debug ("CDC descriptors on config\n");
+ }
+
+ /* this assumes that if there's a non-RNDIS vendor variant
+ * of cdc-acm, it'll fail RNDIS requests cleanly.
+ */
+ rndis = (intf->altsetting->bInterfaceProtocol == 0xff);
+
+ memset (info, 0, sizeof *info);
+ info->control = intf;
+ while (len > 3) {
+ if (buf [1] != USB_DT_CS_INTERFACE)
+ goto next_desc;
+
+ /* use bDescriptorSubType to identify the CDC descriptors.
+ * We expect devices with CDC header and union descriptors.
+ * For CDC Ethernet we need the ethernet descriptor.
+ * For RNDIS, ignore two (pointless) CDC modem descriptors
+ * in favor of a complicated OID-based RPC scheme doing what
+ * CDC Ethernet achieves more simply.
+ */
+ switch (buf [2]) {
+ case 0x00: /* Header, mostly useless */
+ if (info->header) {
+ pr_debug ("extra CDC header\n");
+ goto bad_desc;
+ }

```

```

+ info->header = (void *) buf;
+ if (info->header->bLength != sizeof *info->header) {
+ pr_debug("CDC header len %u\n",
+ info->header->bLength);
+ goto bad_desc;
+ }
+ break;
+ case 0x06: /* Union (groups interfaces) */
+ if (info->u) {
+ pr_debug("extra CDC union\n");
+ goto bad_desc;
+ }
+ info->u = (void *) buf;
+ if (info->u->bLength != sizeof *info->u) {
+ pr_debug("CDC union len %u\n",
+ info->u->bLength);
+ goto bad_desc;
+ }
+
+ /* we need a master/control interface (what we're
+ * probed with) and a slave/data interface; union
+ * descriptors sort this all out.
+ */
+ info->control = usb_ifnum_to_if(dev->udev,
+ info->u->bMasterInterface0);
+ info->data = usb_ifnum_to_if(dev->udev,
+ info->u->bSlaveInterface0);
+ if (!info->control || !info->data) {
+ pr_debug("master #%u/%p slave #%u/%p\n",
+ info->u->bMasterInterface0,
+ info->control,
+ info->u->bSlaveInterface0,
+ info->data);
+ goto bad_desc;
+ }
+ if (info->control != intf) {
+ pr_debug("bogus CDC Union\n");
+ /* Ambit USB Cable Modem (and maybe others)
+ * interchanges master and slave interface.
+ */
+ if (info->data == intf) {
+ info->data = info->control;
+ info->control = intf;
+ } else
+ goto bad_desc;
+ }
+
+ /* a data interface altsetting does the real i/o */
+ d = info->data->altsetting;
+ if (d->bInterfaceClass != USB_CLASS_CDC_DATA) {
+ pr_debug("slave class %u\n",

```

```

+ d->bInterfaceClass);
+ goto bad_desc;
+ }
+ break;
+ case 0x0F: /* Ethernet Networking */
+ if (info->ether) {
+ pr_debug("extra CDC ether\n");
+ goto bad_desc;
+ }
+ info->ether = (void *) buf;
+ if (info->ether->bLength != sizeof *info->ether) {
+ pr_debug("CDC ether len %u\n",
+ info->u->bLength);
+ goto bad_desc;
+ }
+ dev->net->mtu = cpu_to_le16p (
+ &info->ether->wMaxSegmentSize)
+ - ETH_HLEN;
+ /* because of Zaurus, we may be ignoring the host
+ * side link address we were given.
+ */
+ break;
+ }
+next_desc:
+ len -= buf [0]; /* bLength */
+ buf += buf [0];
+ }
+
+ if (!info->header || !info->u || (!rndis && !info->ether)) {
+ pr_debug("missing cdc %s%s%sdescriptor\n",
+ info->header ? "" : "header ",
+ info->u ? "" : "union ",
+ info->ether ? "" : "ether ");
+ goto bad_desc;
+ }
+
+ /* claim data interface and set it up ... with side effects.
+ * network traffic can't flow until an altsetting is enabled.
+ */
+ if (usb_interface_claimed(info->data))
+ return -EBUSY;
+ usb_driver_claim_interface (&usbnet_driver, info->data, dev);
+ status = get_endpoints (dev, info->data);
+ if (status < 0) {
+ /* ensure immediate exit from usbnet_disconnect */
+ usb_set_intfdata(info->data, NULL);
+ usb_driver_release_interface (&usbnet_driver, info->data);
+ return status;
+ }
+ return 0;
+

```

```

+bad_desc:
+ pr_info("bad CDC descriptors\n");
+ return -ENODEV;
+}
+
+static void cdc_unbind (struct usbnet *dev, struct usb_interface *_intf)
+{
+ struct cdc_state *info = (void *) &dev->data;
+
+ if (info->data) {
+ usb_set_intfdata(info->data, NULL);
+ usb_driver_release_interface (&usbnet_driver, info->data);
+ info->data = 0;
+ }
+
+ if (info->control) {
+ usb_set_intfdata(info->control, NULL);
+ usb_driver_release_interface (&usbnet_driver, info->control);
+ info->control = 0;
+ }
+}
+
+ #endif /* NEED_GENERIC_CDC */
+
+ #ifdef CONFIG_USB_CDCETHER
+ #define HAVE_HARDWARE
+
+ /*-----*/
+ *
+ * Communications Device Class, Ethernet Control model
+ *
+ * Takes two interfaces. The DATA interface is inactive till an altsetting
+ * is selected. Configuration data includes class descriptors.
+ *
+ * This should interop with whatever the 2.4 "CDCEther.c" driver
+ * (by Brad Hards) talked with.
+ *-----*/
+
+ #include <linux/ctype.h>
+
+ static u8 nibble (unsigned char c)
+ {
+ if (likely (isdigit (c)))
+ return c - '0';
+ c = toupper (c);
+ if (likely (isxdigit (c)))
+ return 10 + c - 'A';
+ return 0;
+ }

```

```

+
+static inline int
+get_ethernet_addr (struct usbnet *dev, struct ether_desc *e)
+{
+ int tmp, i;
+ unsigned char buf [13];
+
+ tmp = usb_string (dev->udev, e->iMACAddress, buf, sizeof buf);
+ if (tmp < 0)
+ return tmp;
+ else if (tmp != 12)
+ return -EINVAL;
+ for (i = tmp = 0; i < 6; i++, tmp += 2)
+ dev->net->dev_addr [i] =
+ (nibble (buf [tmp]) << 4) + nibble (buf [tmp + 1]);
+ return 0;
+}
+
+static int cdc_bind (struct usbnet *dev, struct usb_interface *intf)
+{
+ int status;
+ struct cdc_state *info = (void *) &dev->data;
+
+ status = generic_cdc_bind (dev, intf);
+ if (status < 0)
+ return status;
+
+ status = get_ethernet_addr (dev, info->ether);
+ if (status < 0) {
+ usb_driver_release_interface (&usbnet_driver, info->data);
+ return status;
+ }
+
+ /* FIXME cdc-ether has some multicast code too, though it complains
+ * in routine cases. info->ether describes the multicast support.
+ */
+ return 0;
+}
+
+static const struct driver_info cdc_info = {
+ .description = "CDC Ethernet Device",
+ .flags = FLAG_ETHER,
+ // .check_connect = cdc_check_connect,
+ .bind = cdc_bind,
+ .unbind = cdc_unbind,
+};
+
+#endif /* CONFIG_USB_CDCETHER */
+
+
+
+

```

Linux-Kernel: [patch/RFT 2.4.26-rc2] update usbnet matching 2.6.recent

```
#ifdef CONFIG_USB_EPSON2888
+#define HAVE_HARDWARE

/*-----
 *
@@ -747,6 +1202,8 @@
 * implements this interface. Product developers can reuse or modify that
 * code, such as by using their own product and vendor codes.
 *
+ * Support was from Juro Bystricky <bystricky.juro@erd.epson.com>
+ *
 *-----*/

static const struct driver_info epson2888_info = {
@@ -754,13 +1211,13 @@
    .check_connect = always_connected,

    .in = 4, .out = 3,
- .epsize = 64,
};

#endif /* CONFIG_USB_EPSON2888 */

#ifdef CONFIG_USB_GENESYS
+#define HAVE_HARDWARE

/*-----
 *
@@ -778,6 +1235,9 @@
 * the transfer direction. (That's disabled here, partially coded.)
 * A control URB would block until other side writes an interrupt.
 *
+ * Original code from Jiun-Jie Huang <huangjj@genesyslogic.com.tw>
+ * and merged into "usbnet" by Stanislav Brabec <utx@penguin.cz>.
+ *
 *-----*/

// control msg write command
@@ -869,7 +1329,7 @@
    // issue usb interrupt read
    if (priv && priv->irq_urb) {
        // submit urb
- if ((retval = SUBMIT_URB (priv->irq_urb, GFP_KERNEL)) != 0)
+ if ((retval = usb_submit_urb (priv->irq_urb, GFP_KERNEL)) != 0)
        dbg ("gl_interrupt_read: submit fail - %X...", retval);
    else
        dbg ("gl_interrupt_read: submit success...");
@@ -888,18 +1348,18 @@
    // detect whether another side is connected
    if ((retval = gl_control_write (dev, GENELINK_CONNECT_WRITE, 0)) != 0) {
```

```

        dbg ("%s: genelink_check_connect write fail - %X",
- dev->net.name, retval);
+ dev->net->name, retval);
        return retval;
    }

    // usb interrupt read to ack another side
    if ((retval = gl_interrupt_read (dev)) != 0) {
        dbg ("%s: genelink_check_connect read fail - %X",
- dev->net.name, retval);
+ dev->net->name, retval);
        return retval;
    }

- dbg ("%s: genelink_check_connect read success", dev->net.name);
+ dbg ("%s: genelink_check_connect read success", dev->net->name);
    return 0;
}

@@ -911,14 +1371,14 @@
    // allocate the private data structure
    if ((priv = kmalloc (sizeof *priv, GFP_KERNEL)) == 0) {
        dbg ("%s: cannot allocate private data per device",
- dev->net.name);
+ dev->net->name);
        return -ENOMEM;
    }

    // allocate irq urb
- if ((priv->irq_urb = ALLOC_URB (0, GFP_KERNEL)) == 0) {
+ if ((priv->irq_urb = usb_alloc_urb (0, GFP_KERNEL)) == 0) {
        dbg ("%s: cannot allocate private irq urb per device",
- dev->net.name);
+ dev->net->name);
        kfree (priv);
        return -ENOMEM;
    }

@@ -967,7 +1427,6 @@
    struct gl_header *header;
    struct gl_packet *packet;
    struct sk_buff *gl_skb;
- int status;
    u32 size;

    header = (struct gl_header *) skb->data;
@@ -976,7 +1435,7 @@
    le32_to_cpus (&header->packet_count);
    if ((header->packet_count > GL_MAX_TRANSMIT_PACKETS)
        || (header->packet_count < 0)) {
- dbg ("genelink: illegal received packet count %d",
+ dbg ("genelink: invalid received packet count %d",

```

Linux-Kernel: [patch/RFT 2.4.26-rc2] update usbnet matching 2.6.recent

```
        header->packet_count);
    return 0;
}
@@ -993,7 +1452,7 @@

    // this may be a broken packet
    if (size > GL_MAX_PACKET_LEN) {
- dbg ("genelink: illegal rx length %d", size);
+ dbg ("genelink: invalid rx length %d", size);
        return 0;
    }

@@ -1002,21 +1461,8 @@
    if (gl_skb) {

        // copy the packet data to the new skb
- memcpy (gl_skb->data, packet->packet_data, size);
-
- // set skb data size
- gl_skb->len = size;
- gl_skb->dev = &dev->net;
-
- // determine the packet's protocol ID
- gl_skb->protocol = eth_type_trans (gl_skb, &dev->net);
-
- // update the status
- dev->stats.rx_packets++;
- dev->stats.rx_bytes += size;
-
- // notify os of the received packet
- status = netif_rx (gl_skb);
+ memcpy(skb_put(gl_skb, size), packet->packet_data, size);
+ skb_return (dev, skb);
    }

    // advance to the next packet
@@ -1032,7 +1478,7 @@
    skb_pull (skb, 4);

    if (skb->len > GL_MAX_PACKET_LEN) {
- dbg ("genelink: illegal rx length %d", skb->len);
+ dbg ("genelink: invalid rx length %d", skb->len);
        return 0;
    }
    return 1;
@@ -1063,6 +1509,8 @@
    skb2 = skb_copy_expand (skb, (4 + 4*1) , padlen, flags);
    dev_kfree_skb_any (skb);
    skb = skb2;
+ if (!skb)
+ return NULL;
```

```

    }

    // attach the packet count to the header
@@ -1087,7 +1535,6 @@
    .tx_fixup = genelink_tx_fixup,

    .in = 1, .out = 2,
- .epsize = 64,

#ifdef GENELINK_ACK
    .check_connect = genelink_check_connect,
@@ -1099,6 +1546,7 @@

#ifdef CONFIG_USB_NET1080
+#define HAVE_HARDWARE

/*-----
*
@@ -1107,6 +1555,9 @@
*
*-----*/

+#define dev_packet_id data[0]
+#define frame_errors data[1]
+
+/*
+ * NetChip framing of ethernet packets, supporting additional error
+ * checks for links that may drop bulk packets from inside messages.
@@ -1222,7 +1673,7 @@
    return;
}

- dbg ("%s registers:", dev->net.name);
+ dbg ("%s registers:", dev->net->name);
    for (reg = 0; reg < 0x20; reg++) {
        int retval;

@@ -1235,10 +1686,10 @@
        retval = nc_register_read (dev, reg, vp);
        if (retval < 0)
            dbg ("%s reg [0x%x] ==> error %d",
- dev->net.name, reg, retval);
+ dev->net->name, reg, retval);
        else
            dbg ("%s reg [0x%x] = 0x%x",
- dev->net.name, reg, *vp);
+ dev->net->name, reg, *vp);
    }
    kfree (vp);
}

```

```

@@ -1374,7 +1825,7 @@
    // nc_dump_registers (dev);

    if ((retval = nc_register_read (dev, REG_STATUS, vp)) < 0) {
- dbg ("can't read %s-%s status: %d",
+ devdbg(dev, "can't read %s-%s status: %d",
        dev->udev->bus->bus_name, dev->udev->devpath, retval);
        goto done;
    }
@@ -1382,7 +1833,7 @@
    // nc_dump_status (dev, status);

    if ((retval = nc_register_read (dev, REG_USBCTL, vp)) < 0) {
- dbg ("can't read USBCTL, %d", retval);
+ devdbg(dev, "can't read USBCTL, %d", retval);
        goto done;
    }
    usbctl = *vp;
@@ -1392,7 +1843,7 @@
        USBCTL_FLUSH_THIS | USBCTL_FLUSH_OTHER);

    if ((retval = nc_register_read (dev, REG_TTL, vp)) < 0) {
- dbg ("can't read TTL, %d", retval);
+ devdbg(dev, "can't read TTL, %d", retval);
        goto done;
    }
    ttl = *vp;
@@ -1400,7 +1851,7 @@

    nc_register_write (dev, REG_TTL,
        MK_TTL (NC_READ_TTL_MS, TTL_OTHER (ttl)) );
- dbg ("%s: assigned TTL, %d ms", dev->net.name, NC_READ_TTL_MS);
+ devdbg(dev, "assigned TTL, %d ms", NC_READ_TTL_MS);

    if (dev->msg_level >= 2)
        devinfo (dev, "port %c, peer %sconnected",
@@ -1426,7 +1877,7 @@
    status = *vp;
    kfree (vp);
    if (retval != 0) {
- dbg ("%s net1080_check_conn read - %d", dev->net.name, retval);
+ devdbg (dev, "net1080_check_conn read - %d", retval);

```