Implementing the Wireless Token Ring Protocol
As a Linux Kernel Module

Ruchira Datta
Web Over Wireless Group
University of California
Berkeley, California

September 28, 2001
Preliminary Groundwork: Fake Kernel

- Kernel implementation was derived from unified codebase
- Kernel functions and data structures used in non-kernel code
  - Scheduling functions
  - Socket buffer data structure and associated methods
Old Tale of WaveLAN
Initialization I: Insertion

- WaveLAN card inserted
- PCMCIA Card Services detects type
- Card Services loads device driver module wave1an2_cs.o
- Module initialization registers adapter_attach
- Card Services calls adapter_attach
Old Tale of WaveLAN
Initialization II: Attachment

• adapter_attach creates struct dev_link_t *link

• adapter_attach creates struct device *dev

• adapter_attach sets function pointers

• adapter_attach registers link with Card Services

• adapter_attach registers card event handler adapter_event with Card Services
Old Tale of WaveLAN
Initialization III: Hooks

- Interrupt Service Request:
  \[ \text{link->irq.handler} = \text{wvlan2_isr} \]

- Transmission: \[ \text{dev->hard_start_xmit} = \text{wvlan2_tx} \]

- Control: \[ \text{dev->do_ioctl} = \text{wvlan2_ioctl} \]

- Via call to \text{ether_setup}:
  - Transmission Header:
    \[ \text{dev->hard_header} = \text{eth_header} \]
  - Ditto, After Address Resolution:
    \[ \text{dev->rebuild_header} = \text{eth_rebuild_header} \]
Old Tale of WaveLAN
Initialization IV: Insertion Redux

• Card Services notifies adapter of card insertion event

• \texttt{adapter\_event} dispatches event to \texttt{wvlan\_insert}

• \texttt{wvlan\_insert} sets device IRQ and IO port address

• \texttt{wvlan\_insert} marks device as ready to transmit

• \texttt{wvlan\_insert} registers dev with kernel

• \texttt{wvlan\_insert} configures WaveLAN card
Old Tale of WaveLAN
Transmission I: Network Layer

- Network layer creates/modifies packet
- Network layer calls dev->hard_header to push link layer header on
- Network layer resolves address
- Network layer calls dev->rebuild_header to correct address
- Network layer calls dev_queue_xmit
Old Tale of WaveLAN

Transmission II: Kernel Scheduler

- `dev_queue_xmit` puts packet on device queue
- `dev_queue_xmit` calls `qdisc_wakeup` to start queue
- `qdisc_wakeup` checks if device is ready to transmit
- `qdisc_wakeup` calls `qdisc_restart`
- `qdisc_restart` calls `dev->hard_start_xmit`
Old Tale of WaveLAN
Transmission III: Device Driver

- `wvlan2.tx` checks if device is ready to transmit
- `wvlan2.tx` notifies: device is busy transmitting
- `wvlan2.tx` disables interrupts
- `wvlan2.tx` gives packet to card for transmission
- `wvlan2.tx` notifies: device ready to transmit, if appropriate
- `wvlan2.tx` reënables interrupts
- Card interrupts to notify: done transmitting
- `wvlan2_isr` notifies: device ready to transmit
- `wvlan2_isr` marks NET_BH
- `wvlan2_isr` restores interrupts
Old Tale of WaveLAN
Reception I: Device Driver

- Card interrupts to notify: received packet
- `wvlan2_isr` calls `wvlan2_rx`
- `wvlan2_rx` gets packet from card
- `wvlan2_rx` puts packet into `struct sk_buff *skb`
- `wvlan2_rx` decodes and “pulls” Ethernet header
- `wvlan2_rx` calls `netif_rx`
- `netif_rx` marks `NET_BH`
- `wvlan2_isr` restores interrupts
Old Tale of WaveLAN
Reception II: Bottom Half

• Kernel scheduler runs task queues
• Kernel scheduler runs networking task queue: net_bh

• net_bh runs pending transmissions
• net_bh pulls packet off received queue
• net_bh checks protocol type

• net_bh sends packet to functions registered to handle this type on all devices
  – Handlers bound to packet socket receive packet

• net_bh sends packet to functions registered to handle this type on this device
  – Network layer handlers such as ip_rcv or arp_rcv receive packet
Insinuating the Ring
Initialization I: Delegation

- `wavelan2_cs.o` module initialization requests BWOW glue manager module `bwow.o`
- `wavelan2_cs.o` registers `BWOW_wavelan2_setup` with BWOW
- `adapter_attach` does not create `struct device *dev`
- `adapter_attach` does not set hooks in `dev`
- `adapter_attach` calls `bwow_attach_dev`
Insinuating the Ring
Initialization II: Attachment

• `bwow_attach_dev` creates `struct device *dev`
• `bwow_attach_dev` registers `dev->init = bwow_init_dev`
• `wvlan2_insert` does not register `dev` with kernel
• `bwow_attach_dev` registers `dev` with kernel
• Kernel calls `bwow_init_dev`
• `bwow_attach_dev` creates `struct bwow_channel *ch`
• `bwow_attach_dev` creates `/proc` filesystem entries
  – `/proc/bwow/wavelan0/protocol`
  – `/proc/bwow/wavelan0/debug`
• `bwow_attach_dev` calls `BWOW_wavelan2_setup`
Insinuating the Ring
Initialization III: Hooks

- bwow_init_dev calls bwow_reset_dev
- bwow_reset_dev sets function pointers
  - Transmission:
    dev->hard_start_xmit = bwow_xmit
  - Control: dev->do_ioctl = bwow_ioctl
  - Transmission Header:
    dev->hard_header = bwow_header
  - Ditto, After Address Resolution:
    dev->rebuild_header = bwow_rebuild_header
Insinuating the Ring
Initialization IV: More Hooks

• BWOW_wavelan2_setup sets function pointers
  – Transmission: ch->HW_send_packet = wvlan2_tx
  – Control: ch->HW_ioctl = wvlan2_ioctl
Insinuating the Ring
Loading the Protocol

• User loads WTRP main protocol module wtrp.o

• wtrp.o module initialization registers wtrp.rcv with kernel to handle packets of type ETH_P_WTRP

• User loads protocol glue module bwow-proto-wtrp.o

• bwow-proto-wtrp.o module initialization registers bwow_wtrp_init with BWOW

• User writes wtrp to /proc/bwow/wavelan0/protocol

• bwow_write_proc sets ch->protocol and calls bwow_wtrp_init
Insinuating the Ring
Initializing the Protocol

- `bwow_wtrp_init` registers `dev`, `bwow_wtrp_up_to_network`, and `bwow_wtrp_data_transmit` with WTRP
- `wtrp_register` creates struct `station_struct *station` and calls `init_station`, associating it with `dev`
- `station` belongs to the common codebase
- `bwow_wtrp_init` sets function pointers
- `bwow_wtrp_init` creates `/proc` filesystem entries
  - State `state`
  - Ring Address `RA`
  - Number of Nodes `num_node`
  - Max Token Holding Time `max_token_holding_time`
  - Contention Time `contention_time`
Insinuating the Ring
Protocol Hooks

• Reception:
  ch->LINK_rx=bwow_wtrp_rx

• Transmission:
  ch->LINK_xmit=bwow_wtrp_xmit

• Notify Protocol Busy:
  ch->LINK_stop_queue = bwow_wtrp_stop_queue

• Notify Protocol Available:
  ch->LINK_wake_queue = bwow_wtrp_wake_queue
Insinuating the Ring
Headers on Data Packets

• WTRP may push just a standard Ethernet header onto data packets
  – Transmission Header:
    \[ \text{dev->hard\_header} = \text{eth\_header} \]
  – Ditto, After Address Resolution:
    \[ \text{dev->rebuild\_header} = \text{eth\_rebuild\_header} \]

• WTRP may push its own header onto data packets as well
  – Transmission Header:
    \[ \text{ch->LINK\_header} = \text{wtrp\_header} \]
  – Ditto, After Address Resolution:
    \[ \text{ch->LINK\_rebuild\_header} = \text{wtrp\_rebuild\_header} \]
Insinuating the Ring
Transmission I: Into WTRP

- If WTRP pushes its own data header:
  - Network layer calls dev->hard_header which points to bwow_header
  - bwow_header calls ch->LINK_header which points to wtrp_header

- Kernel scheduler calls dev->hard_start_xmit which points to bwow_xmit

- bwow_xmit calls ch->LINK_xmit which points to bwow_wtrp_xmit

- bwow_wtrp_xmit calls wtrp_data_request

- wtrp_data_request calls tok_tx_handler (part of the common codebase)
Insinuating the Ring
Transmission II: Out of WTRP

• From the common codebase the packet goes to the hook transmit
• transmit checks whether the device is ready
• transmit calls bwow_wtrp_data_transmit, registered previously
• bwow_wtrp_data_transmit calls ch->HW_send_packet, which points to wvlan2_tx
• When wvlan2_tx starts transmitting, it calls bwow_notify_hw_busy
• bwow_notify_hw_busy calls ch->LINK_stop_queue, which points to bwow_wtrp_stop_queue
• bwow_wtrp_stop_queue calls wtrp_notify_hw_busy
Insinuating the Ring
Transmission III: Done

- When transmission is done, 
bwow_notify_hw_available is called

- bwow_notify_hw_available calls 
  ch->LINK_wake_queue, which points to 
bwow_wtrp_wake_queue

- bwow_wtrp_wake_queue calls 
  wtrp_notify_hw_available

- wtrp_notify_hw_available queues wtrp_bh 
  on the immediate queue and marks 
  IMMEDIATE_BH

- Kernel scheduler runs task queues

- Kernel scheduler runs immediate task queue: 
  immediate_bh, which runs wtrp_bh

- wtrp_bh calls tx_done_handler, part of the 
  common codebase
Insinuating the Ring
Reception I: Into WTRP

- `wvlan2_rx` does not decode or pull the Ethernet header, set `skb->protocol`, or call `netif_rx`
- `wvlan2_rx` calls `ch->LINK_rx`, which points to `bwow_wtrp_rx`
- `bwow_wtrp_rx` decodes and pulls the Ethernet header
- If the protocol is `ETH_P_WTRP`, then `bwow_wtrp_rx` calls `wtrp_data_received`, otherwise it sets `skb->protocol` and calls `netif_rx`
- `wtrp_data_received` calls `process_packet`, part of the common codebase
Insinuating the Ring
Reception II: Out of WTRP

- From the common codebase the packet goes to the hook app_rx
- app_rx calls
  bwow_wtrp_up_to_network, registered previously
- bwow_wtrp_up_to_network sets skb->protocol and calls netif_rx