Contents

- Netpoll and its origins
- Network driver primer
- Netpoll inner-workings
- Quick-start guide to the API
- Extending netconsole
- Moving forward
Netpoll Origins

- 2.4 kernel crash dump solution – netdump (Ingo Molnar)
  - netdump
  - remote syslog
  - netlog / netconsole

- Requirements
  - send / receive packets when kernel is crashed
  - send out log messages from interrupt context

- 2.6 – core architecture abstracted and generic API created (Matt Mackall)
  - kgdb support added
The Netpoll API

API which provides a means for implementing UDP clients and servers in the kernel.

- Operates *mostly* independently from the core network stack
- Used by “applications” which require network communications when the system is quiesced
  - netconsole
  - kgdb
  - netdump
- Each netpoll client describes a single connection (src/dst ip:port)
Network Driver Primer

- Sending packets: `hard_start_xmit`
  - When is it safe to call?
    - `irqs enabled`, bh's disabled
    - `dev->xmit_lock` held
    - `netif_queue_stopped` returns false (0)

- Device Output Queue
  - `netif_stop_queue`
    - out of TX descriptors
    - link down event
    - driver unload
Network Driver Primer (cont'd)

- netif_wake_queue
  - TX descriptors back to a sane level
  - link up event
- netif_queue_stopped
  - boolean test
Receiving Packets

- Interrupt routine
  - Process and ACK interrupts (duh!)
  - Schedule packets for delivery to the network stack
  - Clean up any free RX or TX descriptors*

*The diagram shows the flow from incoming packets to `netif_rx` and then to `irq routine`.
The New API

- Theory of operation
  - Faster network adapters cause many interrupts
    - Interrupts are bad, mm'kay?
    - Switch to polling mode until the “storm” passes

- Polling loop
  - NAPI polls are scheduled for the CPU on which the interrupt was received
  - Only one CPU can execute the poll routine at a time, and it is **not** reentrant!
  - Each interface is given a budget, whose default is set in the driver code (device weight)
Receiving Packets with NAPI

- Interrupt routine:
  - Process and ACK interrupts
  - Disable interrupts on this device
  - Schedule a NAPI poll if necessary

- `net_rx_action` (network bh handler) calls the NAPI poll routine, which:
  - delivers the packet to the net stack
  - cleans up any free RX or TX descriptors*

- Interrupts are re-enabled when the device has no more pending work
Receiving Packets (NAPI)

incoming packets

- irq routine
  - netif_rx_schedule
- device poll routine
  - netif_receive_skb
  - net_rx_action
Netpoll
Netpoll Implementation

- Driver Hooks
- Polling
- Sending Packets
  - Real network device
  - Bonded network device
- What to do when polling fails
- Receiving Packets
Netpoll – Driver Interface

- Polling mode
  - needs to work with irq's disabled
  - needs to work when the system is crashed
  - requires special hook(s) in network drivers

- Typical `poll_controller` hook:

```c
static void tg3_poll_controller(struct net_device *dev)
{
    struct tg3 *tp = netdev_priv(dev);
    tg3_interrupt(tp->pdev->irq, dev, NULL);
}
```
Sending Packets

- API Routine: netpoll_send_udp
  - Directly calls driver's hard_start_xmit routine

- Needs to handle the netif_queue_stopped case
  - dev->poll_controller
  - poll_napi(dev->poll)
Sending Packets

```plaintext
netpoll_send_udp

netpoll_send_skb
```
Sending Packets

- netpoll_send_udp
- netpoll_send_skb
- netpoll_poll

netif_queue_stopped ?
Sending Packets

- netpoll_send_udp
- netpoll_send_skb
- netpoll_poll
- dev->poll_controller
Sending Packets

- `netpoll_send_udp`
- `netpoll_send_skb`
- `netpoll_poll`
- `dev->poll_controller`
Sending Packets

- `netpoll_send_udp`
- `netpoll_send_skb`
- `netpoll_poll`
- `dev->poll_controller`
- `dev->poll`
Sending Packets

- `netpoll_send_udp`
- `netpoll_send_skb`
- `netpoll_poll`
- `dev->poll_controller`
- `dev->poll`
Sending Packets

- `netpoll_send_udp`
- `netpoll_send_skb`
- `netpoll_poll`
- `dev->poll_controller`
- `dev->poll`
Sending Packets

- netpoll_send_udp
- netpoll_send_skb
- dev->hard_start_xmit
- netpoll_poll
- dev->poll_controller
- dev->poll
Sending Packets – Bonding Driver

- netpoll_send_udp
- netpoll_send_skb
- netpoll_poll
- dev->poll_controller
- dev->poll
Sending Packets – Bonding Driver

- netpoll_send_udp
- netpoll_send_skb
- dev->hard_start_xmit
- bond_3ad_xmit_xor
- netpoll_poll
- dev->poll_controller
- dev->poll
Sending Packets – Bonding Driver

- netpoll_send_udp
- netpoll_send_skb
- dev->hard_start_xmit
  - bond_3ad_xmit_xor
  - bond_dev_queue_xmit
- netpoll_poll
  - dev->poll
  - dev->poll_controller
Sending Packets – Bonding Driver

- `netpoll_send_udp`
- `netpoll_send_skb`
- `dev->hard_start_xmit`
- `bond_dev_queue_xmit`
- `netpoll_poll`
- `dev->poll_controller`
- `dev->poll`
Sending Packets – Bonding Driver

- netpoll_send_udp
- netpoll_send_skb
- dev->hard_start_xmit
  - e.g. e100_xmit_frame
- bond_dev_queue_xmit
- netpoll_poll
- dev->poll_controller
- dev->poll
When Polling Fails...

- `netif_queue_stopped` returns true when:
  - no TX descriptors
  - link is down

- Sending packets synchronously can fail!

- Drop routine:
  - can do whatever the module author wants it to do
  - `netpoll_queue` is provided as a means to queue the packet for later delivery (in process context)
  - if not specified, the packet will be dropped
Receiving Packets (non-NAPI)
Receiving Packets (NAPI)
Sending Packets in the Receive Path

- netif_receive_skb
- dev->poll
- net_rx_action
- netpoll_send_udp
- netpoll_send_skb
- dev->hard_start_xmit
- netpoll_rx
- netpoll_poll
- dev->poll_controller
- dev->poll
Using the API

- Initialization
- Sending Packets
- Receiving Packets
- Specifying a *drop* routine
Client Data Structure

```c
struct netpoll {
    struct net_device *dev;
    char dev_name[16], *name;
    void (*rx_hook)(struct netpoll *, int, char *, int);
    void (*drop)(struct sk_buff *skb);
    u32 local_ip, remote_ip;
    u16 local_port, remote_port;
    unsigned char local_mac[6], remote_mac[6];
};
```
Netpoll Module Initialization

```
int netpoll_parse_options(struct netpoll *np, char *opt);

np: struct netpoll with name, drop, and rx_hook filled in
opt: 
    "[src-port]@[src-ip]/[dev],[tgt-port]<tgt-ip>/[tgt-macaddr]"

Returns 0 on success, -1 on failure
```

```
int netpoll_setup(struct netpoll *np);

np: struct netpoll, initialized via a call to
    netpoll_parse_options

Returns: 0 on success, -1 on failure
```
API – Sending & Receiving Packets

void netpoll_send_udp(struct netpoll *np, const char *msg, int len);

  msg: byte stream to be sent
  len: length of byte stream contained in msg

void rx_hook(struct netpoll *np, short source, char *data, int dlen);

  data: contents of received packet; UDP headers stripped
  dlen: length of data

Called in BH context for NAPI drivers, interrupt context for old drivers.

void drop(struct sk_buff *skb);

  skb: socket buffer that could not be sent.

void netpoll_queue(struct sk_buff *skb);

  queues the packet for later delivery, in process context
Extending Netconsole

- **Goals**
  - allow remote user to issue sysrq commands via netconsole

- **Non-goals**
  - Support a full interactive console
Extending Netconsole (cont'd)

static struct netpoll np = {
    .name = "netconsole",
    .dev_name = "eth0",
    .local_port = 6665,
    .remote_port = 6666,
    .remote_mac = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff},
    .drop = netpoll_queue,
    .rx_hook = netconsole_rx;
};

void netconsole_rx(struct netpoll *nps, short source, char *data, int dlen)
{
    while (count < dlen) {
        if (data[count] < 'a' || data[count] > 'Z' ||
            data[count] == '\n') {
            count++;
            continue;
        }
        handle_sysrq(msg->msg[count], NULL, NULL);
        count++;
    }
}
Netpoll TODO

- Allow more than one netpoll client to register an rx hook
- Netpoll calls drivers in improper context
  - Implement separate hard_start_xmit routine for every network driver?
- Fix locking so that queuing is not necessary all of the time
References

- netdev mailing list <netdev@vger.kernel.org>
- Linux kernel sources, versions 2.4 and 2.6  http://www.kernel.org/
- http://people.redhat.com/jmoyer/