链滴

linux 网络设备 watchdog 分析

作者：stepforwards

原文链接：https://ld246.com/article/1637818896624

来源网站：链滴

WATCHDOG分析

linux 内核版本4.4.58

watchdog初始化

igb网卡驱动在igb_probe函数中调用register_netdev-->register_netdevice-->dev_init_scheduler完成网络设备watchdog的初始化，从这里可以看出watchdog只是一个定时器。

```
igb_probe:
    netdev->watchdog_timeo = 5 * HZ;
    err = register_netdev(netdev);

register_netdev:
    register_netdevice:
        dev_init_scheduler:

void dev_init_scheduler(struct net_device *dev)
{
    dev->qdisc = &noop_qdisc;
    netdev_for_each_tx_queue(dev, dev_init_scheduler_queue, &noop_qdisc);
    if (dev_ingress_queue(dev))
        dev_init_scheduler_queue(dev, dev_ingress_queue(dev), &noop_qdisc);

    setup_timer(&dev->watchdog_timer, dev_watchdog, (unsigned long)dev);
}
```

启用watchdog

igb网卡被UP时通过netif_carrier_on-->__netdev_watchdog_up启动watchdog。

原文链接：linux 网络设备 watchdog 分析

```
netif_carrier_on:
        __netdev_watchdog_up:

void __netdev_watchdog_up(struct net_device *dev)
{
    if (dev->netdev_ops->ndo_tx_timeout) {
        if (dev->watchdog_timeo <= 0)
            dev->watchdog_timeo = 5*HZ;
        if (!mod_timer(&dev->watchdog_timer,
                round_jiffies(jiffies + dev->watchdog_timeo)))
            dev_hold(dev);
    }
}
```

超时处理dev_watchdog

当watchdog发现网卡处于up状态并且发送队列处于停止时间大于5秒时将触发看门狗机制，

dev_watchdog先找到停止的队列，然后调用igb_tx_timeout整个网卡队列做超时处理，igb_tx_timeout中对网卡做了DOWN、UP的操作。

```
static void dev_watchdog(unsigned long arg)
{
    struct net_device *dev = (struct net_device *)arg;

    netif_tx_lock(dev);
    if (!qdisc_tx_is_noop(dev)) {        //txq->qdisc不是noop_qdisc
        if (netif_device_present(dev) && //网卡设备还存在
            netif_running(dev) &&        // 网卡设备还在运行
            netif_carrier_ok(dev)) {     //网卡设备在线
            int some_queue_timedout = 0;
            unsigned int i;
            unsigned long trans_start;

            for (i = 0; i < dev->num_tx_queues; i++) {
                struct netdev_queue *txq;

                txq = netdev_get_tx_queue(dev, i);
                /*
                 * old device drivers set dev->trans_start
                 */
                trans_start = txq->trans_start ? : dev->trans_start;
                if (netif_xmit_stopped(txq) && //发送队列是停止状态
                    time_after(jiffies, (trans_start +
                               dev->watchdog_timeo))) { //发送队列停止超过5秒
                    some_queue_timedout = 1;
                    txq->trans_timeout++;
                    break;
                }
            }

            if (some_queue_timedout) {
                WARN_ONCE(1, KERN_INFO "NETDEV WATCHDOG: %s (%s): transmit queue %u ti
ed out\n",
```

```
                dev->name, netdev_drivername(dev), i);
            dev->netdev_ops->ndo_tx_timeout(dev); //超时处理函数
        }
        if (!mod_timer(&dev->watchdog_timer,
                round_jiffies(jiffies +
                    dev->watchdog_timeo)))
            dev_hold(dev);
    }
  }
  netif_tx_unlock(dev);

  dev_put(dev);
}

static const struct net_device_ops igb_netdev_ops = {
  .ndo_open       = igb_open,
  .ndo_stop       = igb_close,
  .ndo_start_xmit     = igb_xmit_frame,
... ...
  .ndo_tx_timeout     = igb_tx_timeout,
... ...
}

igb_tx_timeout:
  schedule_work(&adapter->reset_task);

igb_reset_task:
  netdev_err(adapter->netdev, "Reset adapter\n");
  igb_reinit_locked:
    igb_down
    igb_up
```

dev_watchdog超时触发条件

条件：网卡处于up状态、发送队列处于停止状态并且停止时间大于5秒。

网卡发送队列停止的条件:当发送队列满(会有一定的预留值)的时候，将停止发送。

发送队列停止条件：

```
igb_xmit_frame_ring:
  igb_maybe_stop_tx(tx_ring, count + 3):

static inline int igb_maybe_stop_tx(struct igb_ring *tx_ring, const u16 size)
{
  if (igb_desc_unused(tx_ring) >= size) //当发送队列剩余空间大于最小预留值时，不停止发送队列
    return 0;
  return __igb_maybe_stop_tx(tx_ring, size);
}

//先停止发送队列，再次判断发送队列剩余空间，若剩余空间大于最小预留值，则重新启动发送队列
static int __igb_maybe_stop_tx(struct igb_ring *tx_ring, const u16 size)
{
  struct net_device *netdev = tx_ring->netdev;
```

```
    netif_stop_subqueue(netdev, tx_ring->queue_index);

    /* Herbert's original patch had:
     *  smp_mb__after_netif_stop_queue();
     * but since that doesn't exist yet, just open code it.
     */
    smp_mb();

    /* We need to check again in a case another CPU has just
     * made room available.
     */
    if (igb_desc_unused(tx_ring) < size)
        return -EBUSY;

    /* A reprieve! */
    netif_wake_subqueue(netdev, tx_ring->queue_index);

    u64_stats_update_begin(&tx_ring->tx_syncp2);
    tx_ring->tx_stats.restart_queue2++;
    u64_stats_update_end(&tx_ring->tx_syncp2);

    return 0;
}
```

发送队列停止时间如何确定：

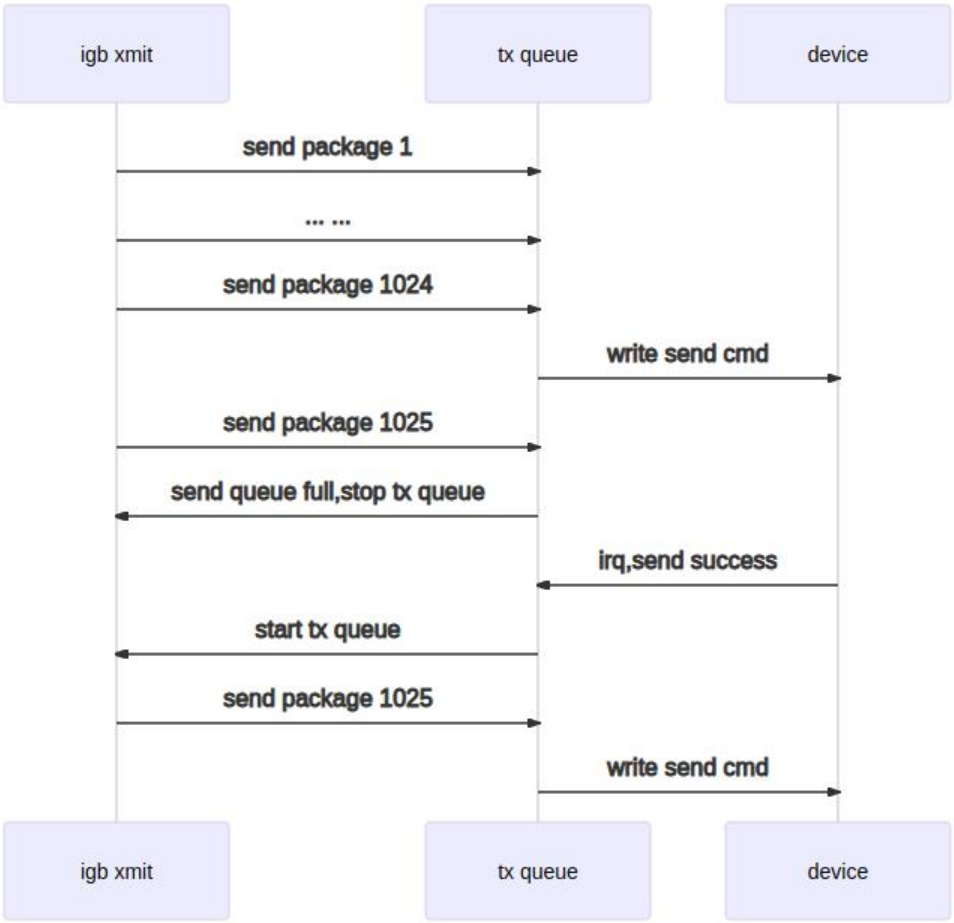
把最后一个数据包的发送时间作为发送队列停止的时间。每成功发送一个包都会记录当前的系统时间。

调用路径dev_hard_start_xmit-->xmit_one-->netdev_start_xmit-->txq_trans_update

```
static inline netdev_tx_t __netdev_start_xmit(const struct net_device_ops *ops,
                      struct sk_buff *skb, struct net_device *dev,
                      bool more)
{
    skb->xmit_more = more ? 1 : 0;
    return ops->ndo_start_xmit(skb, dev);
}

static inline netdev_tx_t netdev_start_xmit(struct sk_buff *skb, struct net_device *dev,
                    struct netdev_queue *txq, bool more)
{
    const struct net_device_ops *ops = dev->netdev_ops;
    int rc;

    rc = __netdev_start_xmit(ops, skb, dev, more);
    if (rc == NETDEV_TX_OK)
        txq_trans_update(txq);

    return rc;
}


static inline void txq_trans_update(struct netdev_queue *txq)
```

```
{
    if (txq->xmit_lock_owner != -1)
        txq->trans_start = jiffies;
}
```

发送队列停止原因



发送队列满了导致发送队列停止。

发送队列满的原因

1. 软件发送过快，瞬间将发送队列填满，但这种情况通常不会触发看门狗机制。
2. cpu繁忙对clean tx irq响应不及时。
3. 硬件网卡异常不再上报irq。

发送队列恢复

正常情况下，当igb网卡硬件完成发送之后，通过中断调用igb_msix_ring最终通过napi_schedule调用gb_poll，igb_poll函数通过调用igb_clean_tx_irq释放发送队列中发送完成的空间，当发送队列剩余间大于预定值时重新启用发送队列。

```
#if (65536/PAGE_SIZE + 1) < 16
#define MAX_SKB_FRAGS 16UL
#else
```

```
#define MAX_SKB_FRAGS (65536/PAGE_SIZE + 1)
#endif
#define DESC_NEEDED (MAX_SKB_FRAGS + 4)


static bool igb_clean_tx_irq(struct igb_q_vector *q_vector)
{
... ...
#define TX_WAKE_THRESHOLD (DESC_NEEDED * 2)
      if (unlikely(total_packets &&  //释放的包数量
        netif_carrier_ok(tx_ring->netdev) &&  //网卡设备在线
        igb_desc_unused(tx_ring) >= TX_WAKE_THRESHOLD)) { //剩余节点大于预定值
        /* Make sure that anybody stopping the queue after this
         * sees the new next_to_clean.
         */
        smp_mb();
        if (__netif_subqueue_stopped(tx_ring->netdev, //发送队列是停止状态
                     tx_ring->queue_index) &&
            !(test_bit(__IGB_DOWN, &adapter->state))) { //网口是UP状态
            netif_wake_subqueue(tx_ring->netdev,   // 启用发送队列
                        tx_ring->queue_index);

            u64_stats_update_begin(&tx_ring->tx_syncp);
            tx_ring->tx_stats.restart_queue++;
            u64_stats_update_end(&tx_ring->tx_syncp);
        }
    }
... ...
```