X.25 Device Driver Interface 1.1

Jonathan Naylor 26.12.96

This is a description of the messages to be passed between the X.25 Packet Layer and the X.25 device driver. They are designed to allow for the easy setting of the LAPB mode from within the Packet Layer.

The X.25 device driver will be coded normally as per the Linux device driver standards. Most X.25 device drivers will be moderately similar to the already existing Ethernet device drivers. However unlike those drivers, the X.25 device driver has a state associated with it, and this information needs to be passed to and from the Packet Layer for proper operation.

All messages are held in sk_buff's just like real data to be transmitted over the LAPB link. The first byte of the skbuff indicates the meaning of the rest of the skbuff, if any more information does exist.

Packet Layer to Device Driver

First Byte = 0x00 (X25_IFACE_DATA)

This indicates that the rest of the skbuff contains data to be transmitted over the LAPB link. The LAPB link should already exist before any data is passed down.

First Byte = 0x01 (X25_IFACE_CONNECT)

Establish the LAPB link. If the link is already established then the connect confirmation message should be returned as soon as possible.

First Byte = 0x02 (X25_IFACE_DISCONNECT)

Terminate the LAPB link. If it is already disconnected then the disconnect confirmation message should be returned as soon as possible.

First Byte = 0x03 (X25_IFACE_PARAMS)

LAPB parameters. To be defined.

Device Driver to Packet Layer

First Byte = 0x00 (X25_IFACE_DATA)

This indicates that the rest of the skbuff contains data that has been received over the LAPB link.

First Byte = 0x01 (X25_IFACE_CONNECT)

LAPB link has been established. The same message is used for both a LAPB link connect_confirmation and a connect_indication.

First Byte = 0x02 (X25_IFACE_DISCONNECT)

LAPB link has been terminated. This same message is used for both a LAPB link disconnect_confirmation and a disconnect_indication.

First Byte = 0x03 (X25_IFACE_PARAMS)

LAPB parameters. To be defined.

Possible Problems

(Henner Eisen, 2000-10-28)

The X.25 packet layer protocol depends on a reliable datalink service. The LAPB protocol provides such reliable service. But this reliability is not preserved by the Linux network device driver interface:

- With Linux 2.4.x (and above) SMP kernels, packet ordering is not preserved. Even if a device driver calls netif_rx(skb1) and later netif_rx(skb2), skb2 might be delivered to the network layer earlier that skb1.
- Data passed upstream by means of netif_rx() might be dropped by the kernel if the backlog queue is congested.

The X.25 packet layer protocol will detect this and reset the virtual call in question. But many upper layer protocols are not designed to handle such N-Reset events gracefully. And frequent N-Reset events will always degrade performance.

Thus, driver authors should make netif_rx() as reliable as possible:

SMP re-ordering will not occur if the driver's interrupt handler is always executed on the same CPU. Thus,

- Driver authors should use irq affinity for the interrupt handler.

The probability of packet loss due to backlog congestion can be reduced by the following measures or a combination thereof:

(1) Drivers for kernel versions 2.4.x and above should always check the return value of netif_rx(). If it returns NET_RX_DROP, the driver's LAPB protocol must not confirm reception of the frame to the peer.
This will reliably suppress packet loss. The LAPB protocol will automatically cause the peer to re-transmit the dropped packet later.
The lapb module interface was modified to support this. Its data_indication() method should now transparently pass the netif_rx() return value to the (lapb mopdule) caller.
(2) Drivers for kernel versions 2.2.x should always check the global variable netdev_dropping when a new frame is received. The driver should only call netif_rx() if netdev_dropping is zero. Otherwise the driver should not confirm delivery of the frame and drop it.

Alternatively, the driver can queue the frame internally and call
netif_rx() later when netif_dropping is 0 again. In that case, delivery
confirmation should also be deferred such that the internal queue
cannot grow to much.
This will not reliably avoid packet loss, but the probability
of packet loss in netif_rx() path will be significantly reduced.

(3) Additionally, driver authors might consider to support
CONFIG_NET_HW_FLOWCONTROL. This allows the driver to be woken up
when a previously congested backlog queue becomes empty again.
The driver could uses this for flow-controlling the peer by means
of the LAPB protocol's flow-control service.