Introduzione ai driver di rete per Linux

Vincenzo Maffione <v.maffione@gmail.com>

22/10/2016

Outline

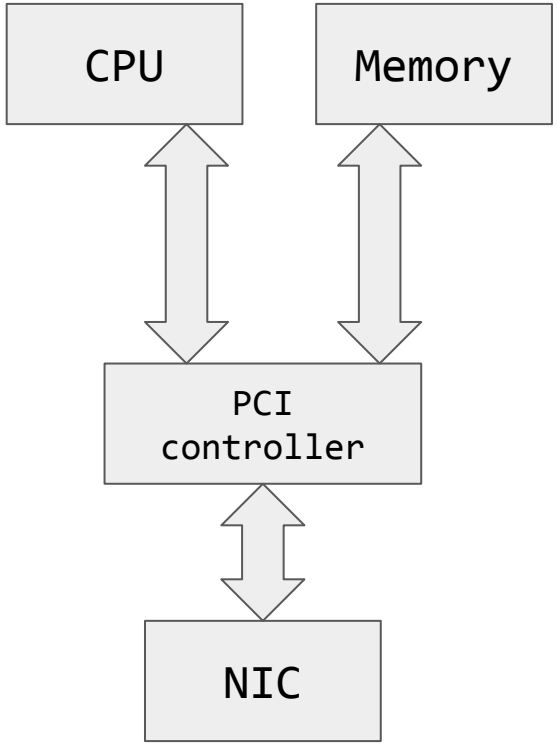
- Funzionamento delle schede di rete Ethernet
- Struttura dei driver di schede Ethernet per Linux

Funzionamento delle schede di rete (1)

- Le schede di rete Ethernet (Network Interface Cards, NICs) vengono usate per connettere un computer ad una Local Area Network.
- Come funzionano?
- Come sono gestite dal Sistema Operativo?

Funzionamento delle schede di rete (2)

- Sono dispositivi montati sul bus PCI.
- Vengono programmate dal S.O. tramite numerosi registri.
- Lettura e scrittura dei registri avviene tramite l'istruzione **MOV** della CPU, come per accedere alla RAM.
- Ogni modello di scheda ha un set di registri diverso.
- Sono capaci di accedere direttamente alla memoria per scrivere e leggere pacchetti di rete
    - *Direct Memory Access*, avviene senza l'intervento della CPU



La scheda di rete e1000 (1)

- 82540EM (a.k.a. e1000) è una scheda da 1 Gbps prodotta da Intel.
  - DeviceID = 0x100E, VendorID = 0x8086
- e1000 espone una regione di memoria contenente registri da 4 bytes, per un totale di 128KB.

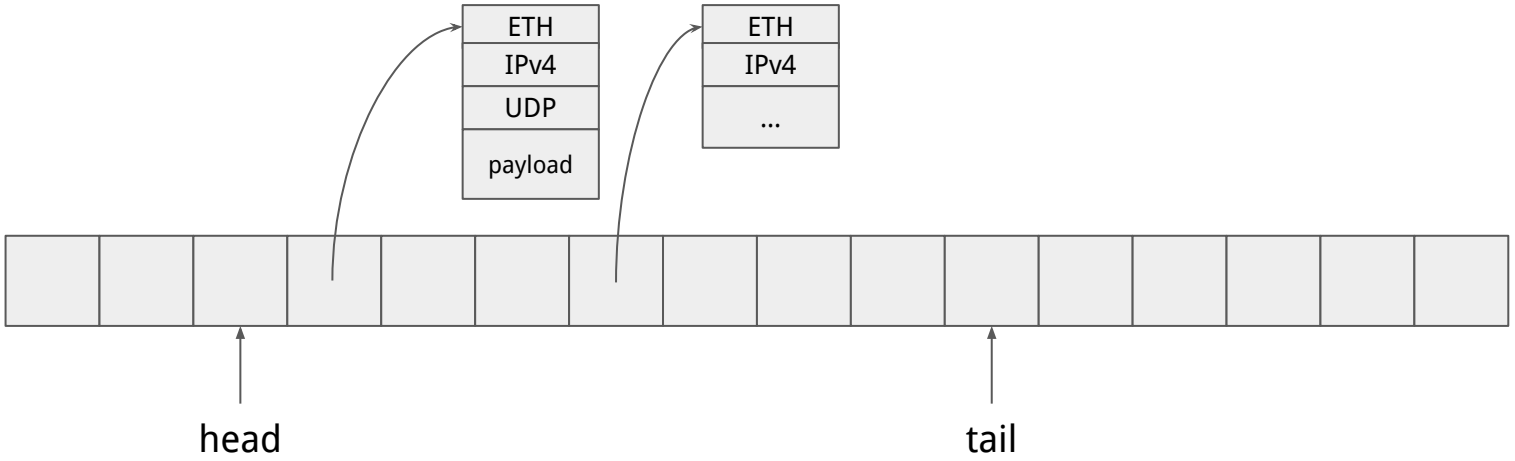
- La scheda può inviare interruzioni alla CPU per informare il S.O. che ha ricevuto un nuovo pacchetto o completato la trasmissione di un pacchetto.

La scheda di rete e1000 (2)

- Le schede sono complesse per via della configurazione
  - Power management, PHY, EEPROM, FLASH …
  - Centinaia di registri
  - Migliaia di pagine di documentazione
- Il *datapath* è invece l'interfaccia software che il S.O. usa per ricevere/trasmettere pacchetti.
  - Riguarda pochi registri e strutture dati
- I datapath di modelli diversi di scheda tendono ad essere simili

La scheda di rete e1000 (3)

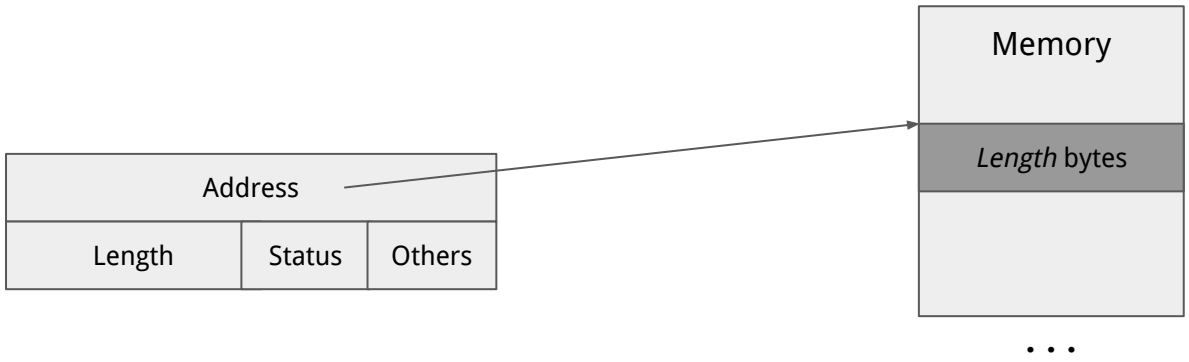
- Il S.O. scambia pacchetti con la scheda tramite delle strutture dati chiamate *rings*, allocate nella RAM del computer
- Un ring è un array circolare di descrittori
- Un descrittore descrive un buffer allocato nella RAM.
- Un buffer può essere usato per trasmettere o ricevere pacchetti.
- e1000 ha un ring di trasmissione (TX) ed un ring di ricezione (RX)



Trasmissione (1)

Ogni descrittore di trasmissione è lungo 16 byte e contiene

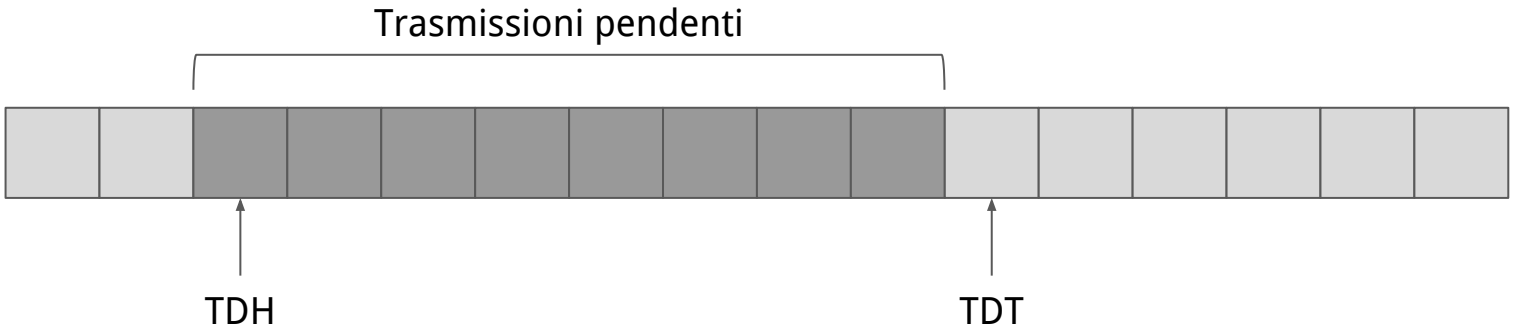
- Indirizzo (fisico) del buffer
- Lunghezza del buffer
- Un bit di stato (Descriptor Done) che vale 1 se il pacchetto è stato trasmesso.



Trasmissione (2)

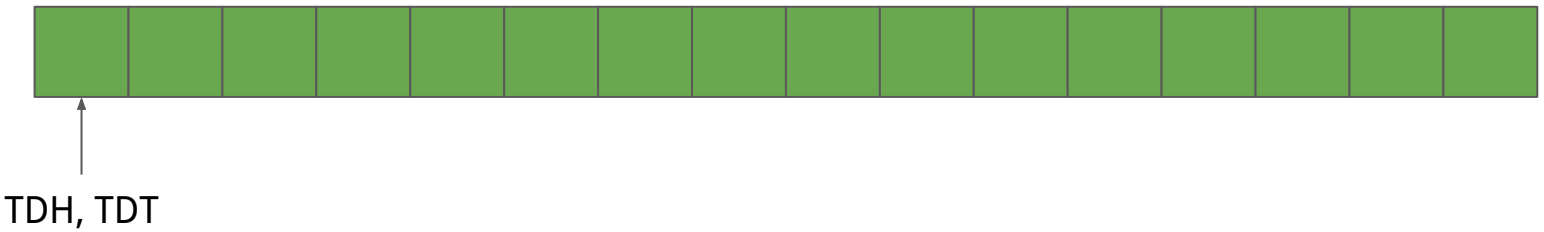
La sincronizzazione con il S.O. avviene tramite due registri, che fanno da indici nel TX ring:

- Transmit Descriptor Head (TDH): punta al prossimo descrittore pronto per la trasmissione sul cavo.
- Transmit Descriptor Tail (TDT): punta al primo descrittore *non* pronto. Indica alla scheda quando smettere di processare i descrittori.



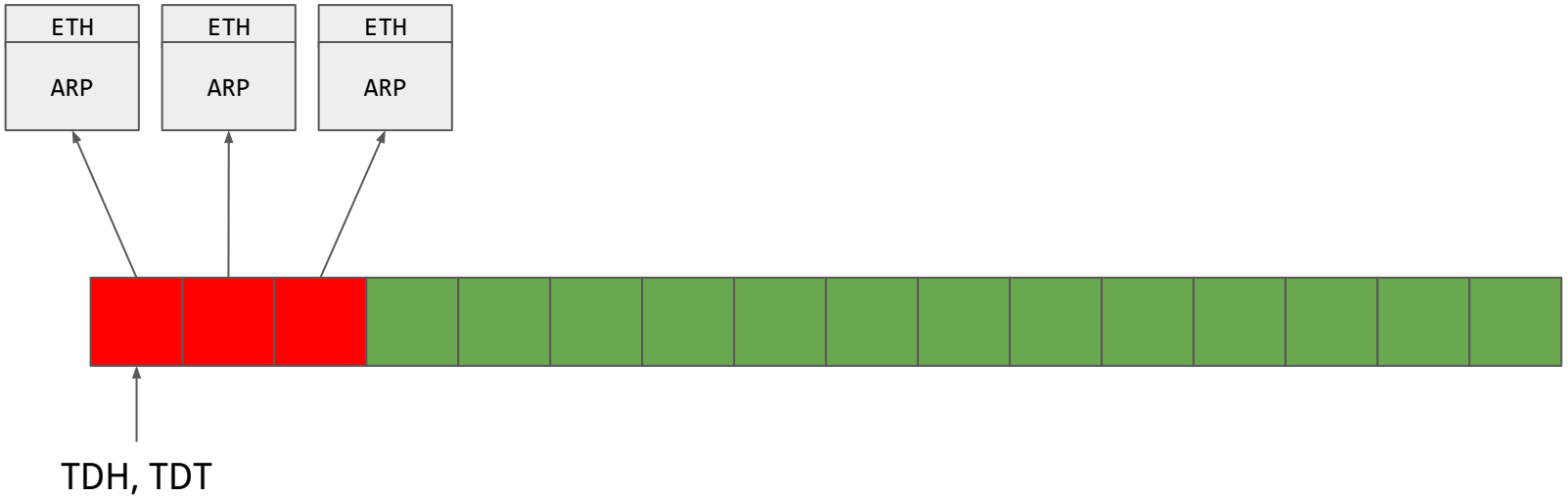
Trasmissione (3)

- All’inizio TDT = TDH = 0
- Non ci sono pacchetti pendenti
- Tutti i descrittori sono a disposizione del S.O. per nuove trasmissioni.



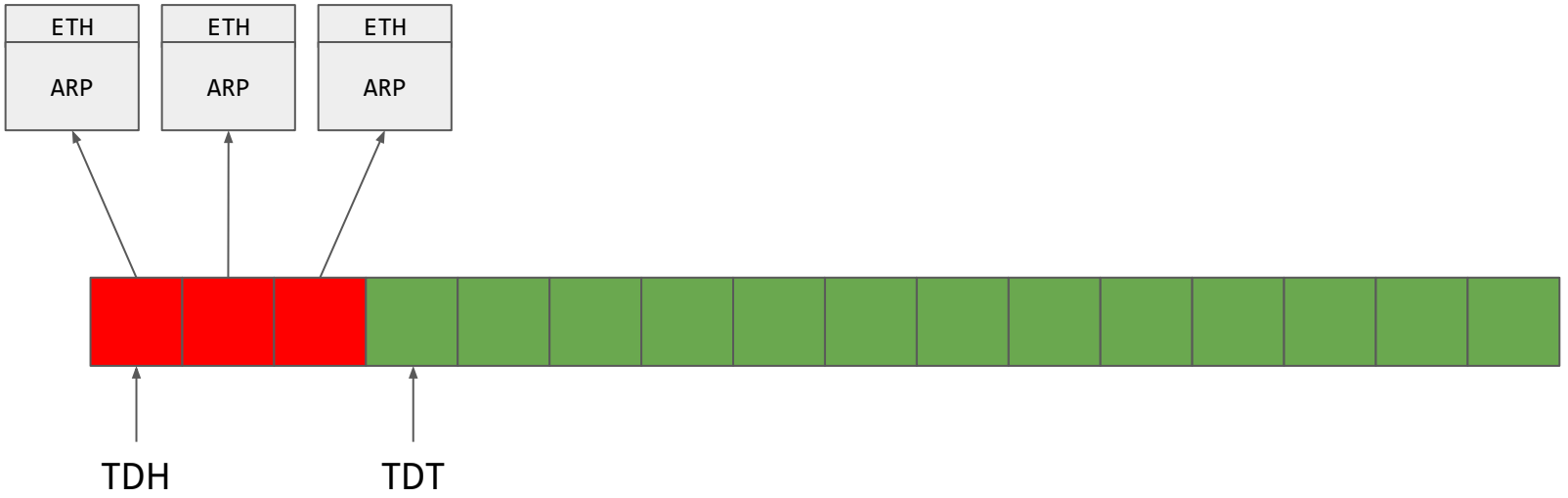
Trasmissione (4)

- Il S.O. vuole trasmettere tre pacchetti → prepara i primi tre descrittori a partire da TDT.
- Ogni descrittore punta ad un buffer che contiene il contenuto del pacchetto.



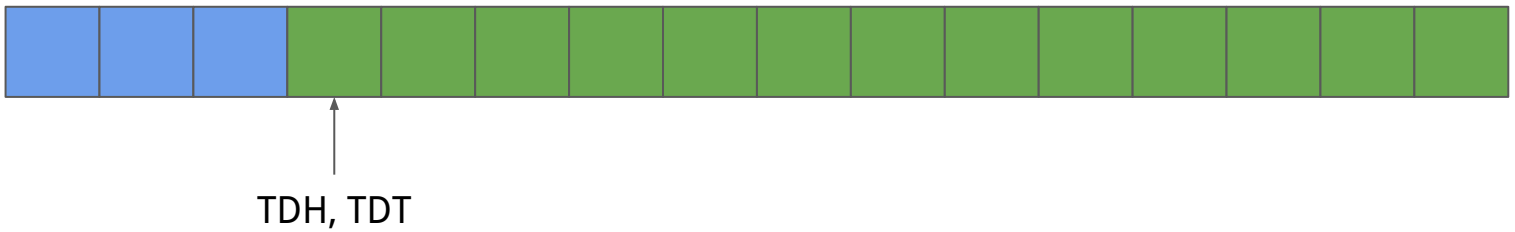
Trasmissione (5)

- Il S.O. incrementa il registro TDT di 3 unità, per notificare la scheda che ci sono nuovi pacchetti pendenti.



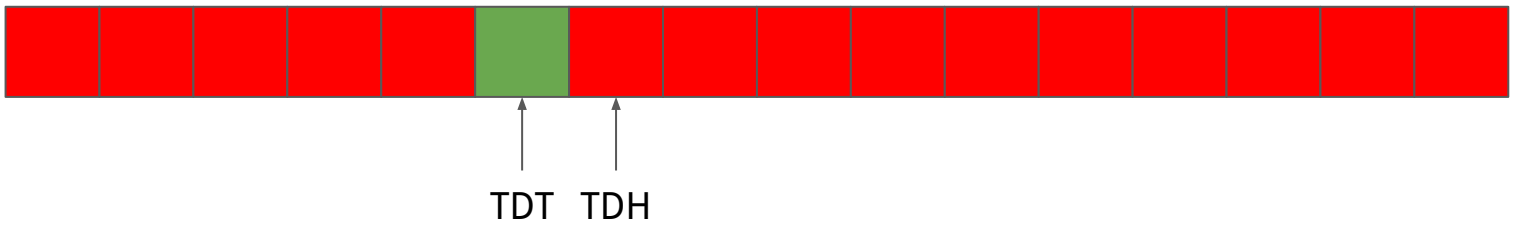
Trasmissione (6)

- In seguito alla scrittura in TDT, la scheda (hardware) comincia a processare i descrittori pendenti, partendo da quello puntato da TDH.
- Per ogni descrittore pendente:
  - legge il buffer dalla RAM
  - Lo trasmette sul cavo
  - Incrementa il TDH
  - Emette un segnale di interruzione per segnalare al S.O. che la trasmissione è completa.



Trasmissione (7)

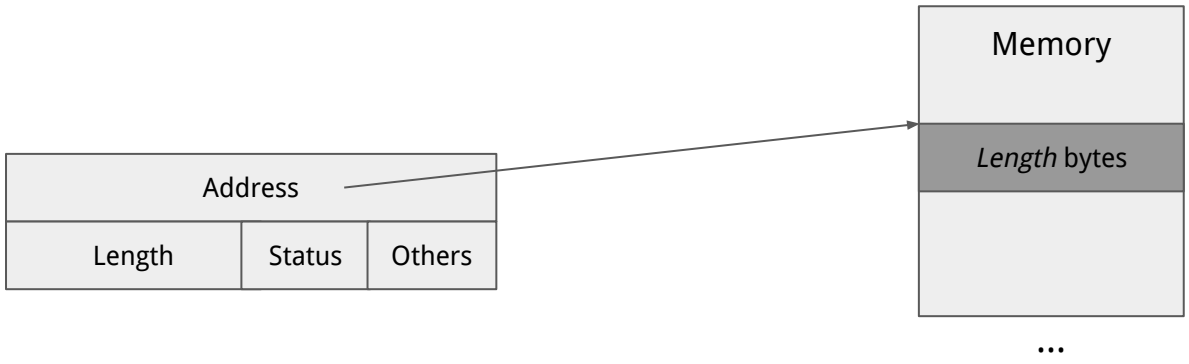
- Il S.O. e la scheda lavorano *in parallelo*:
  - Il S.O. produce nuovi descrittori e avanza il TDT
  - La scheda consuma i descrittori preparati ed avanza il TDH
- La scheda si ferma quando TDT == TDH
- Il S.O. si ferma quando c'è un solo descrittore libero.



Ricezione (1)

Ogni descrittore di ricezione (lungo 16 bytes) contiene:

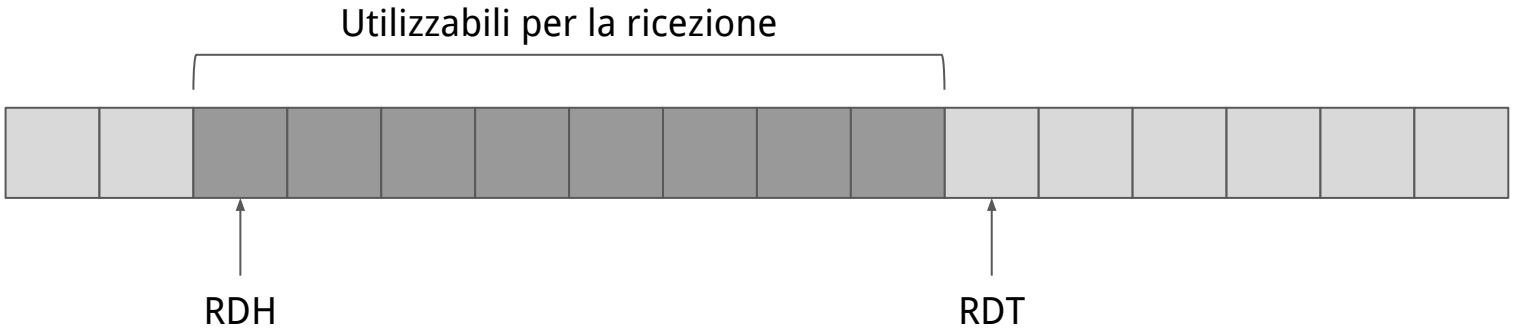
- L’indirizzo (fisico) di un buffer che contiene (o conterrà) un pacchetto arrivato dal cavo.
- La lunghezza del buffer.
- Un bit di stato (Descriptor Done) che vale 1 se il buffer è stato riempito.



Ricezione (2)

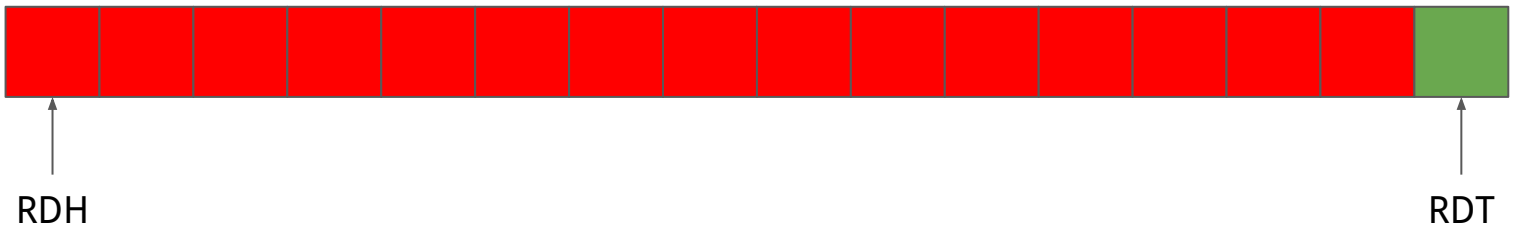
La sincronizzazione tra il S.O. e la scheda avviene tramite due registri, indice nel RX ring:

- Receive Descriptor Head (RDH): punta al primo descrittore che la scheda può usare per memorizzare il prossimo pacchetto che arriva dal cavo.
- Receive Descriptor Tail (RDT): punta al primo descrittore non pronto per la ricezione. Indica alla scheda quando deve fermarsi nella ricezione.



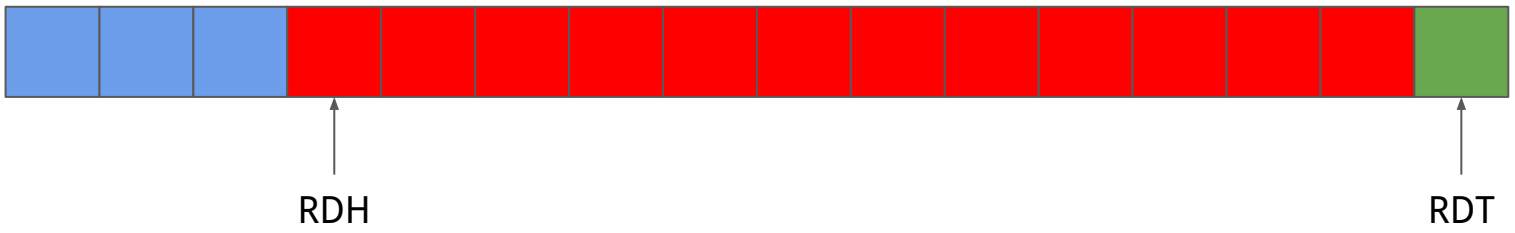
Ricezione (3)

- All’inizio RDH = 0
- Il S.O. prepara tutti i descrittori (tranne uno)
  - RDT viene quindi inizializzato a *N-1*
- In questa situazione (quasi) tutti i descrittori sono disponibili per essere usati dalla scheda



Ricezione (4)

- Supponiamo 3 pacchetti arrivino sul cavo, l'uno subito dopo l'altro
- La scheda
  - Usa 3 descrittori disponibili, a partire da RDH, e copia i pacchetti ricevuti nei relativi buffer
  - Incrementa RDH di 3 unità
  - Emette un segnale di interruzione verso la CPU per indicare al S.O. l'arrivo di nuovi pacchetti

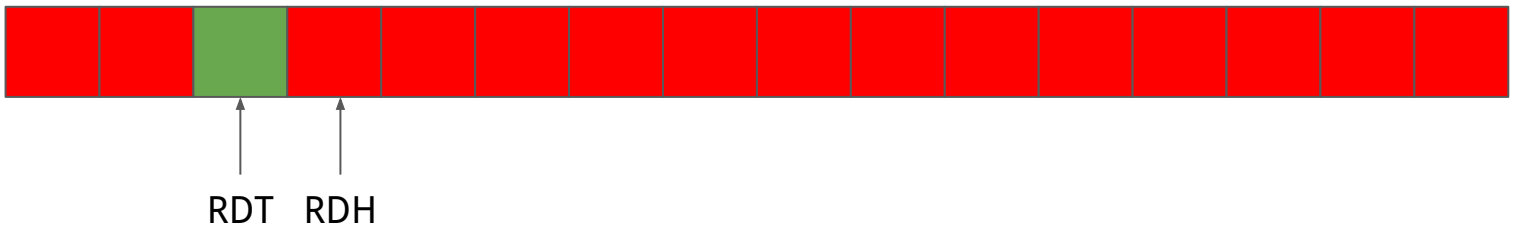


Ricezione (5)

- Il S.O. reagisce all'interruzione effettuando una scansione del ring RX.
- Per ogni pacchetto ricevuto:
  - Il pacchetto viene processato dai protocollo di rete (e.g. IP, TCP, etc)
  - Il relativo descrittore viene riciclato → associato ad un nuovo buffer

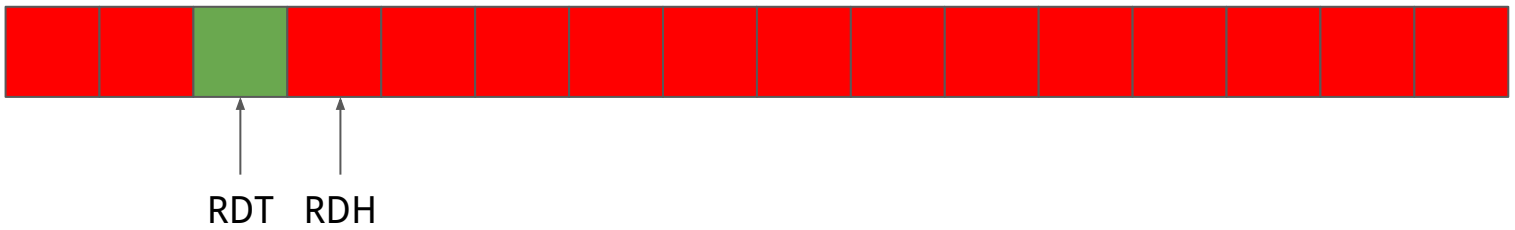
Ricezione (6)

- Una volta che i 3 descrittori sono stati riciclati, il S.O. incrementa RDT di 3 unità.
- La scrittura informa la scheda che ci sono nuovi descrittori disponibili per la ricezione.



Ricezione (7)

- Il S.O. e la scheda lavorano *in parallelo*:
  - Il S.O. produce nuovi descrittori e avanza RDT
  - La scheda consuma i descrittori preparati ed avanza RDH
- La scheda si ferma quando RDH == RDT
- Il S.O. si ferma quando c'è un solo descrittore inutlizzato.

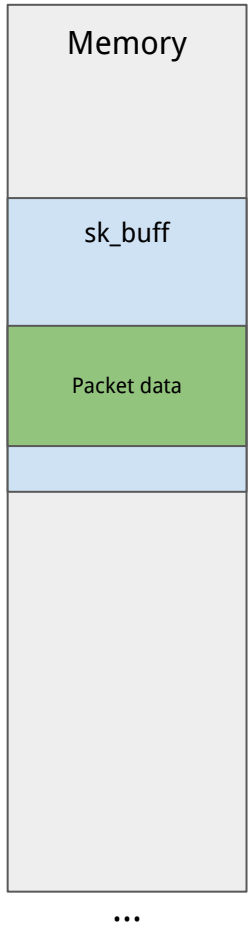


Driver Ethernet per Linux (1)

- I driver sono tipicamente moduli kernel caricabili on-demand.
- Il kernel espone una API che i driver Ethernet utilizzano per scambiare pacchetti con il resto del kernel.
- Un driver di rete registra un *interfaccia di rete* per ogni scheda che gestisce, associandogli un nome (e.g. `eth0`).
- Il comando `ifconfig` (or `ip link`) mostra la lista delle interfacce di rete di un computer.
- Per gestire la complessità di un S.O., Linux utilizza la programmazione orientata agli oggetti.
    - L'interazione tra kernel e driver di rete avviene tramite una classe astratta (`struct net_device`)
    - Quando un driver registra un interfaccia, specifica un insieme di puntatori a funzione (`struct net_device_ops` object) che contengono l'implementazione dei metodi.

Driver Ethernet per Linux (2)

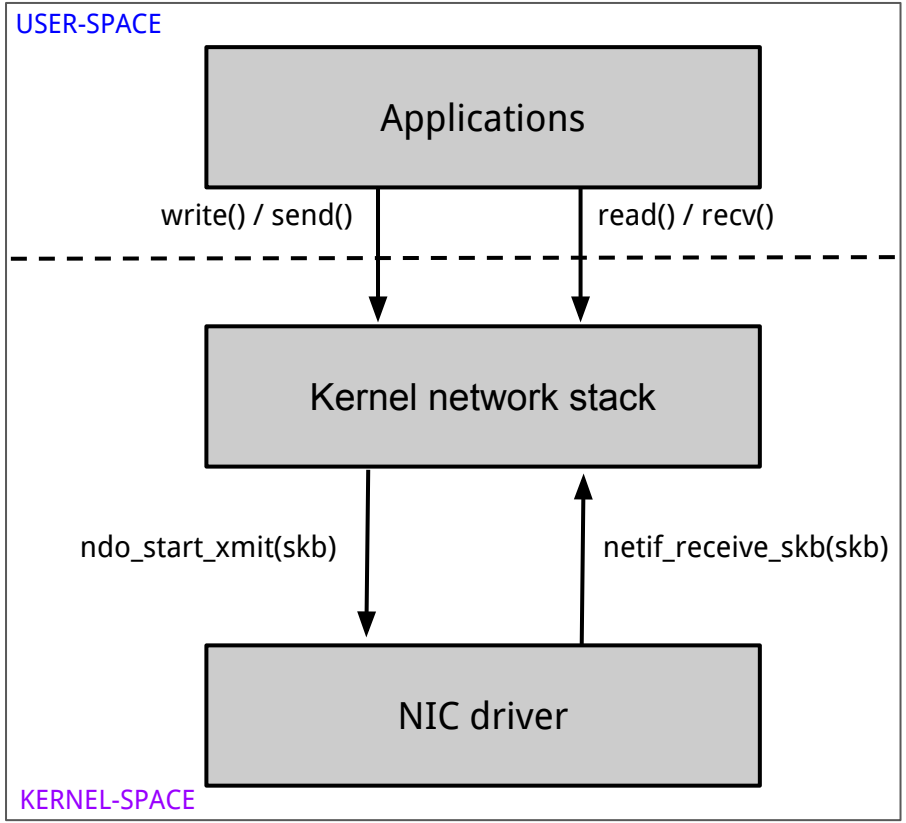
- Linux memorizza i pacchetti di rete utilizzando oggetti di tipo `struct sk_buff`, che memorizzano
  - il contenuto del pacchetto
  - metadati.
- Gli skb vengono allocati e deallocati dinamicamente, e scambiati tra kernel e driver.



Driver Ethernet per Linux (3)

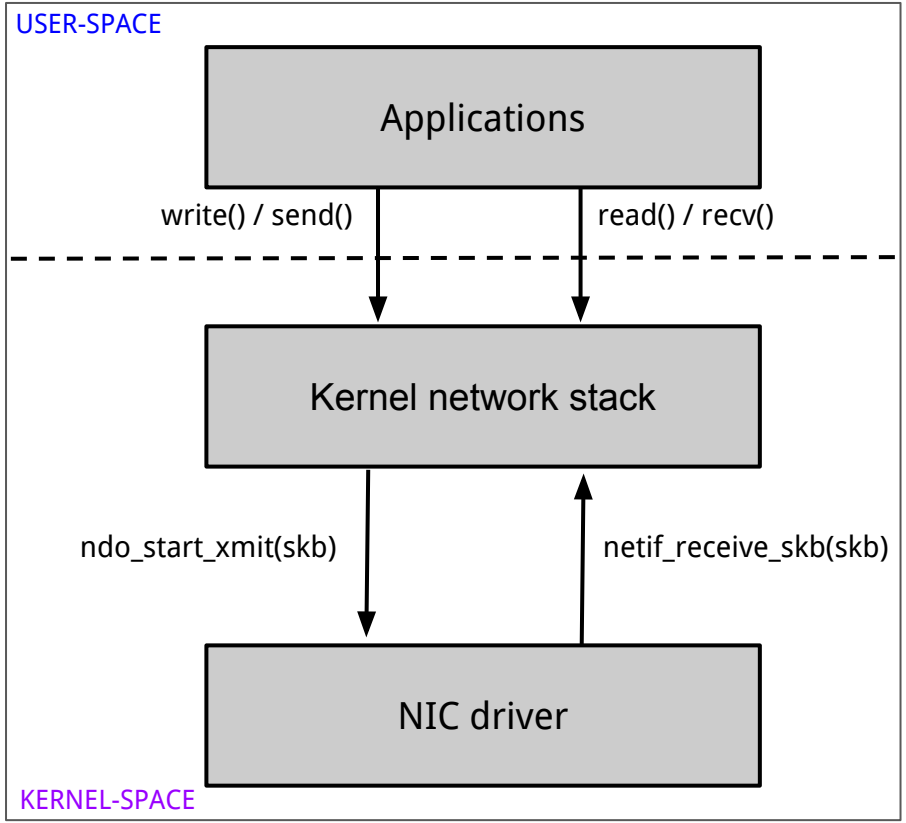
Alcuni metodi interessanti:

1. `ndo_open(dev)` - Il kernel chiede al driver di attivare l'interfaccia. Il driver alloca ed inizializza i ring TX/RX ed abilita la scheda.
2. `ndo_close(dev)` - Il kernel chiede al driver di disattivare l'interfaccia. Il driver dealloca i ring, libera i buffer pendenti e disabilita l'interfaccia.
3. `ndo_start_xmit(dev, skb)` - Il kernel chiede al driver di trasmettere il pacchetto contenuto in skb. Il driver programma un descrittore del ring TX.



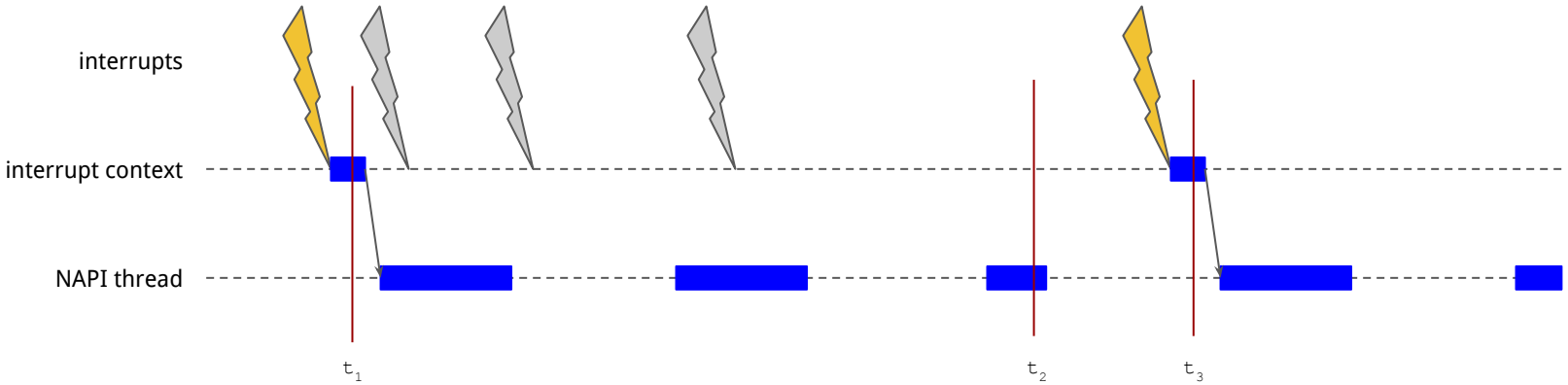
Driver Ethernet per Linux (4)

- Quando la scheda riceve un pacchetto, il driver lo passa allo stack di rete invocando la funzione `netif_receive_skb(skb)`.
- A seconda del contenuto del pacchetto, il kernel effettua una azione appropriata:
  - ➢ Lo inserisce nella coda di ricezione di un socket.
  - ➢ Lo inoltra ad un'altra scheda di rete
  - ➢ Lo butta
  - ➢ ...



NAPI

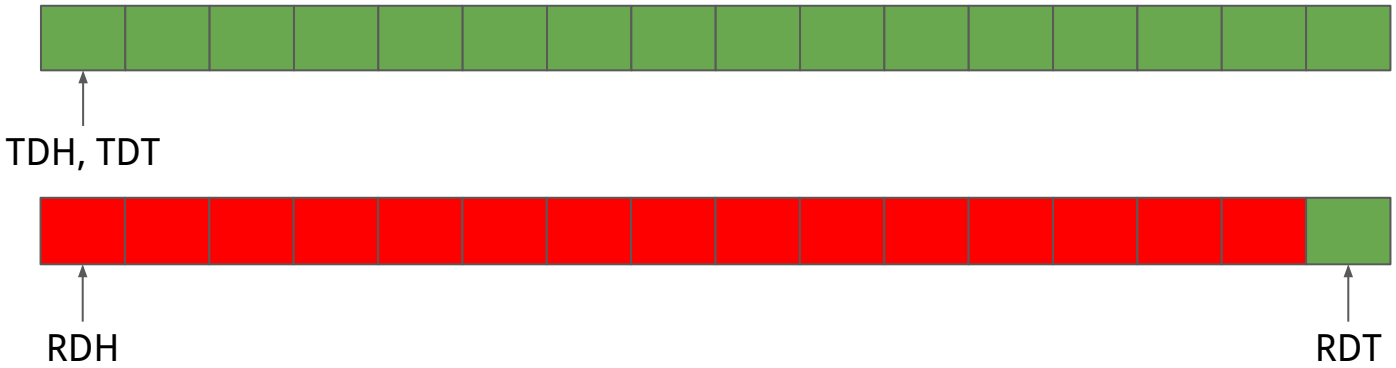
- La gestione delle interruzioni da parte del S.O. è costosa e delicata
- La routine di interruzione deve fare il minor lavoro possibile
- Per migliorare efficienza e responsività, Linux usa un meccanismo noto come NAPI.
- Quando riceve un'interruzione, ulteriori interruzioni vengono temporaneamente disabilitate
- I ring vengono processati da un kernel thread.
- Quando il kernel thread non ha più nulla da fare, riabilita le interruzioni e termina.



e1000 driver

Implementazione di `ndo_open` per e1000 (`e1000_open`):

- Allocazione dei ring TX e RX
- Inizializzazione di `TDH`,`TDT`, `RDT`, `RDH` a zero.
- Riempimento del ring RX con sk_buffs, ed avanzamento di RDT.
- Abilitazione degli interrupt

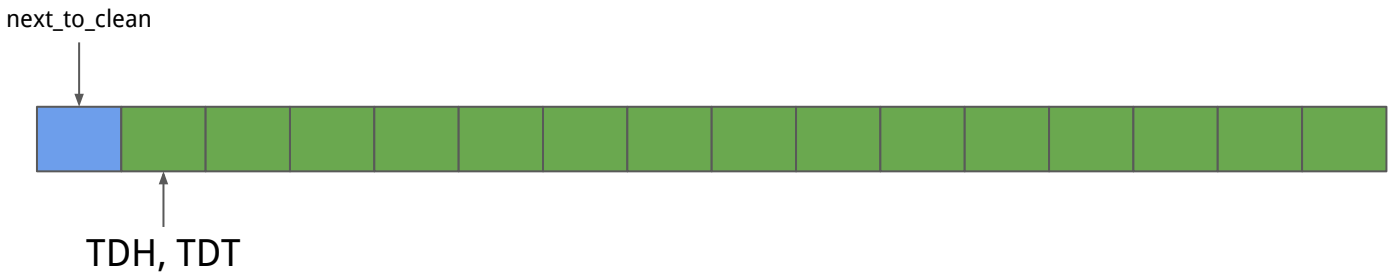


e1000 driver: trasmissione (1)

- Il metodo `ndo_start_xmit` di e1000 (`e1000_xmit_frame`) si occupa di:
  - Preparare un descrittore TX con l'indirizzo e lunghezza del buffer contenuto nello skb ricevuto come argomento.
  - Incrementare il TDT, per informare alla scheda del nuovo descrittore

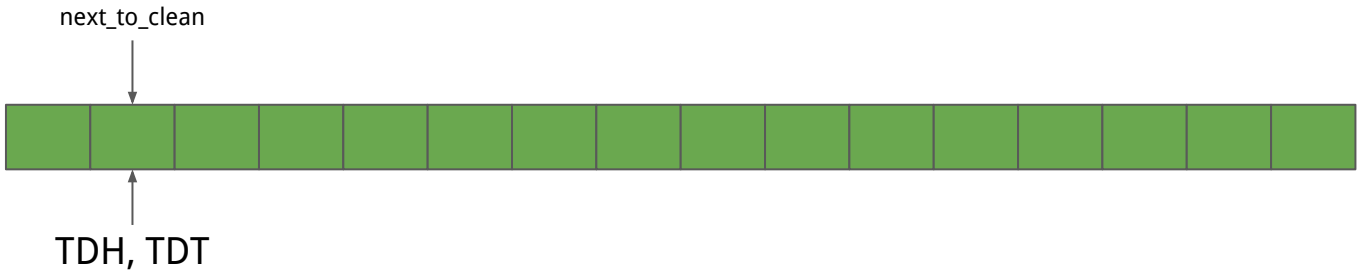
e1000 driver: trasmissione (2)

- Quando la scheda termina la trasmissione del pacchetto, emette un interrupt di trasmissione.
- La routine di interruzione effettua l'acknowledgement dell'interrupt e schedula il NAPI thread.



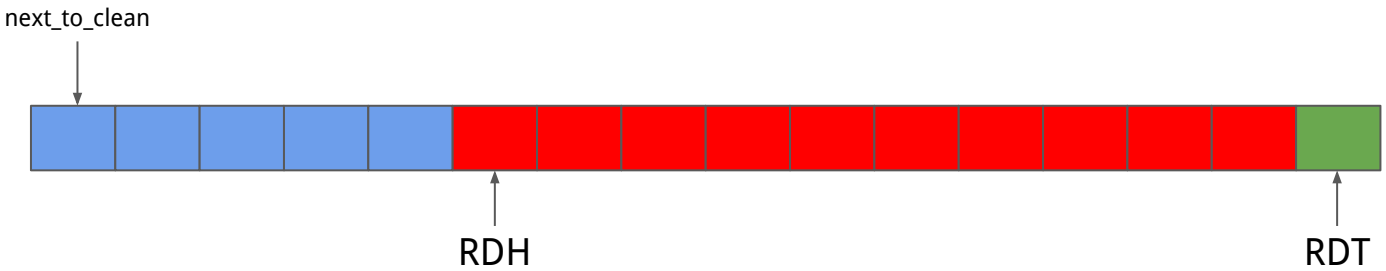
e1000 driver: trasmissione (3)

- Il NAPI thread esegue la routine `e1000_clean_tx_irq` che si occupa di liberare gli skb associati a tutti i descrittori trasmessi → ossia fino al TDH.
- Il driver mantiene una variabile di stato (next_to_clean) per tenere traccia del prossimo descrittore da liberare.



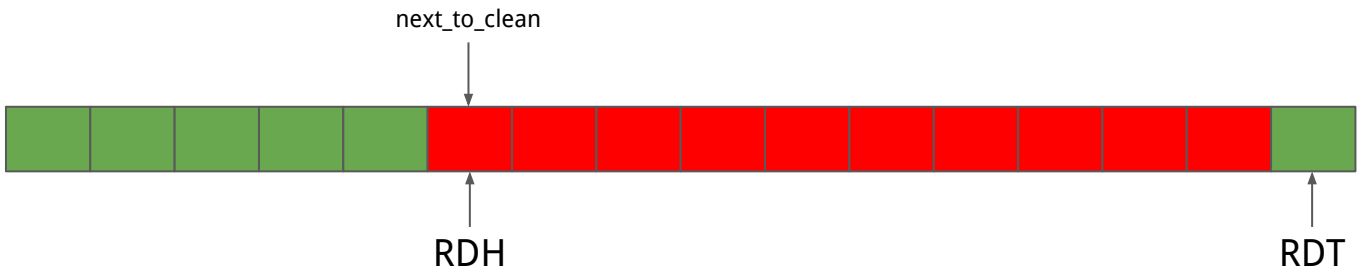
e1000 driver: ricezione (1)

- Quando la scheda riceve dei pacchetti, emette un’interruzione.
- La routine di interruzione schedula il NAPI thread.



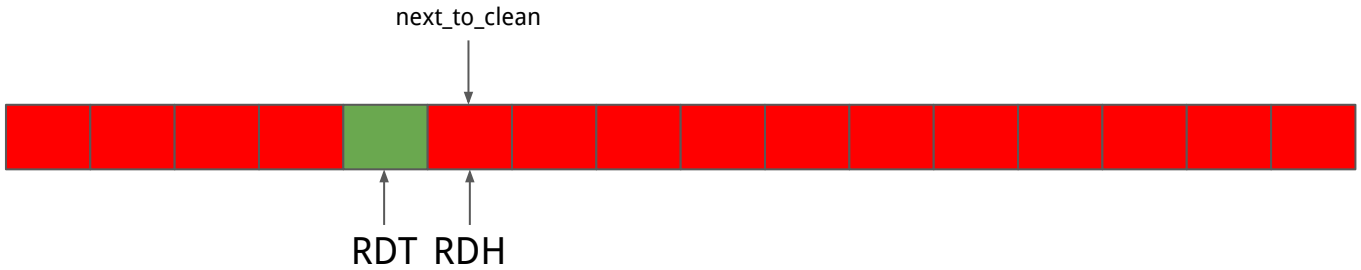
e1000 driver: ricezione (2)

- Il NAPI thread esegue la routine `e1000_clean_rx_irq`, che scandisce il ring RX per cercare descrittori consumati → ossia fino all'indice RDH.
- Per ogni descrittore consumato, lo skb associato viene passato allo stack di rete invocando la routine `netif_receive_skb`.
- Il driver mantiene una variabile di stato (next_to_clean) per tenere traccia del prossimo descrittore da processare.



e1000 driver: ricezione (3)

- Ogni descrittore consumato viene rimpiazzato utilizzando un nuovo skb, che viene allocato sul momento
- Il driver infine incrementa RDT per segnalare alla scheda i nuovi buffer disponibili per la ricezione



Domande ?