Processing the packets in the NIC

The journey of the packets…

I'm Yasamin :)

Index

- Intro to NIC
- Linux network stack
- A Packet
- Ingress
- DMA
- Kernel Space
- Bottom half
- Sk_buff
- Functions in sc_buff
- Bufferbloat
- Codel

What is NIC?

MAC Address: 12 Hex character

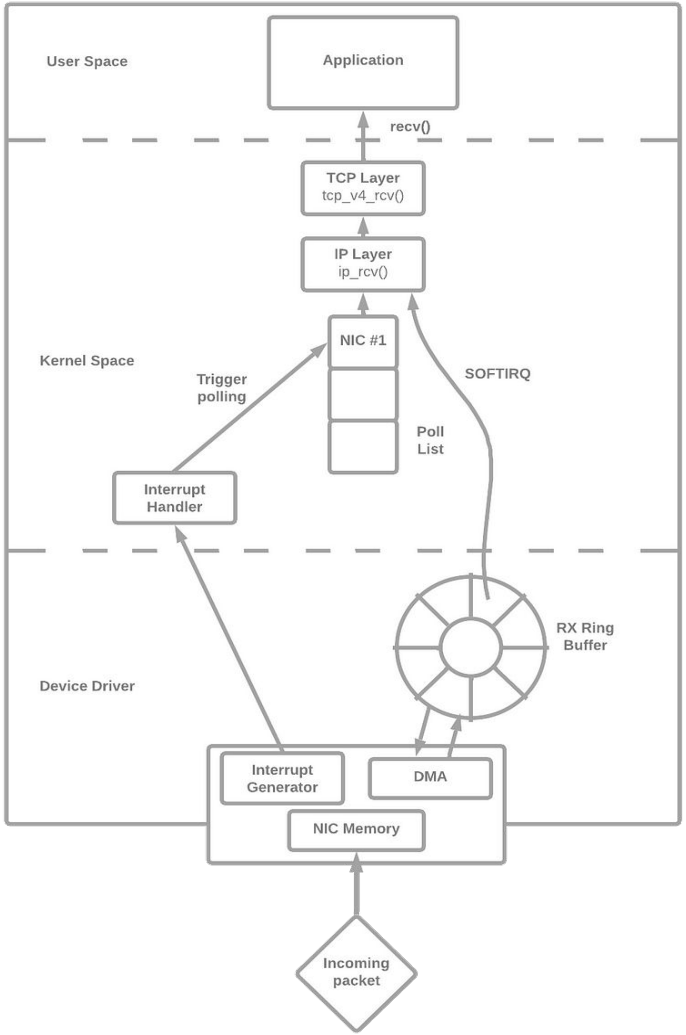
Example: 90:2e:16:d9:73:79

The whole picture

This is Linux Networking Stack

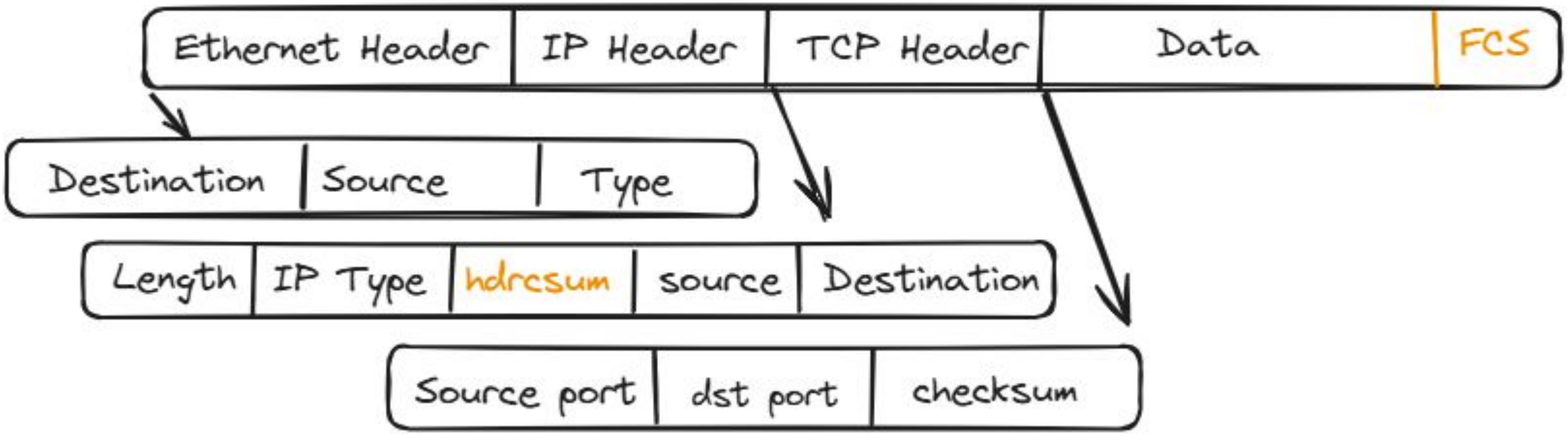
Stack:

1. **Device Driver**
2. **Kernel Space**
3. **User Space**



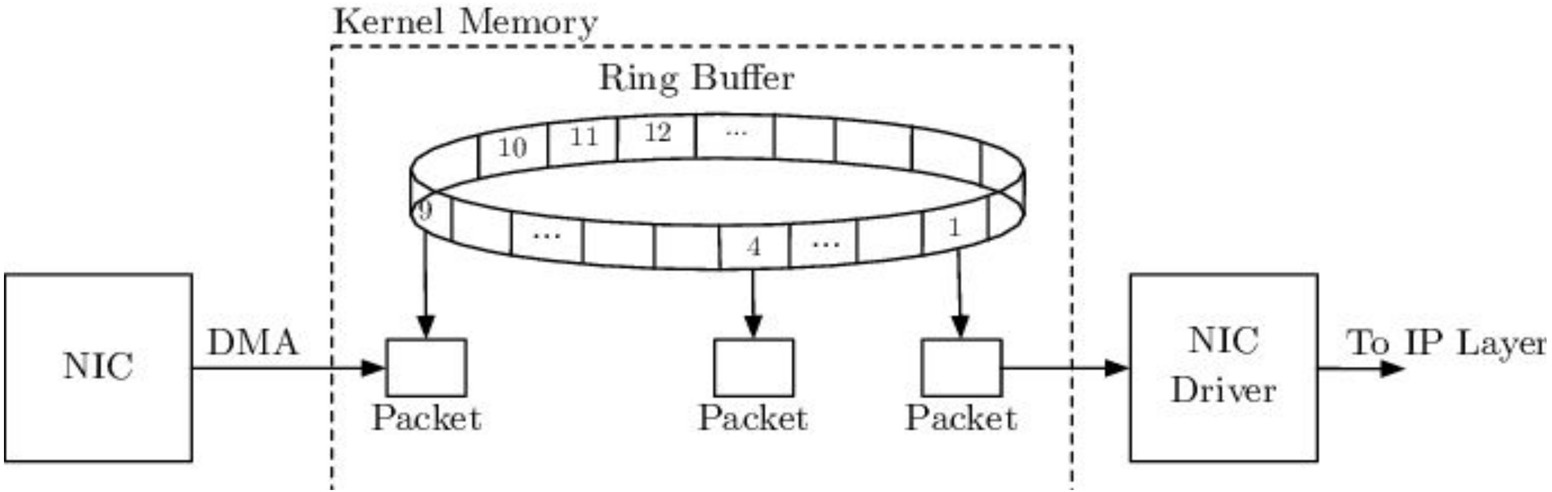
First point

- Packet structure:
- Checksum
- Headers
- Data



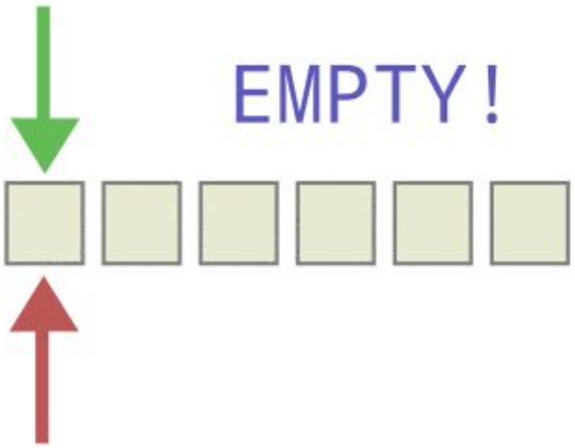
Ingress

- Network Interface Controller
- Receive the packet
- Compares against MAC address filter
- Verifies FCS: Ethernet csum
- Stores the packet to the system memory. to where? to a buffer!
- NIC triggers an interrupt



DMA

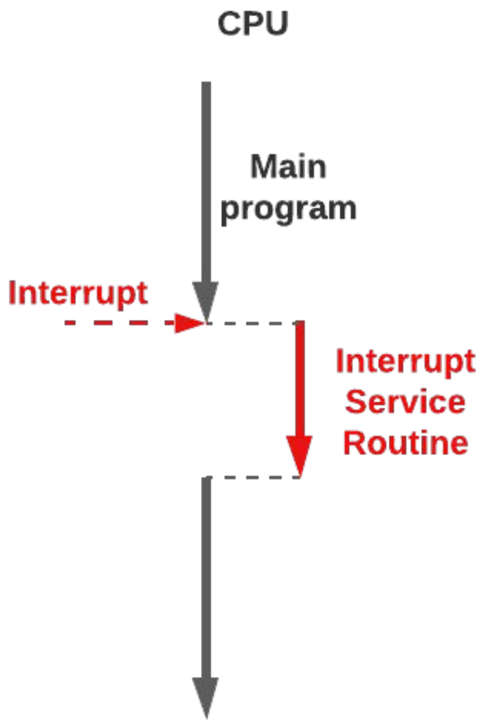
- NIC then uses DMA to transfer the packet to a Ring Buffer.
- a fixed-size circular FIFO queue .



Kernel Space

- After triggering the interrupt CPU transfers the control to the Interrupt Handler.

A software routine that hardware invokes in response to an interrupt.



Methods for Optimization of Interrupt

1. **Low processing in the Interrupt Context**
2. **Waking up the NAPI subsystem by SOFTIRQ Interrupt.**
3. **Poll List**
4. **Processing the packets by process_backlog() function and moves them to sk_buff**

Bottom Half

- Identifies the memory where the packet was stored, asks which buffer?
- Allocates an sk_buff

Wait...

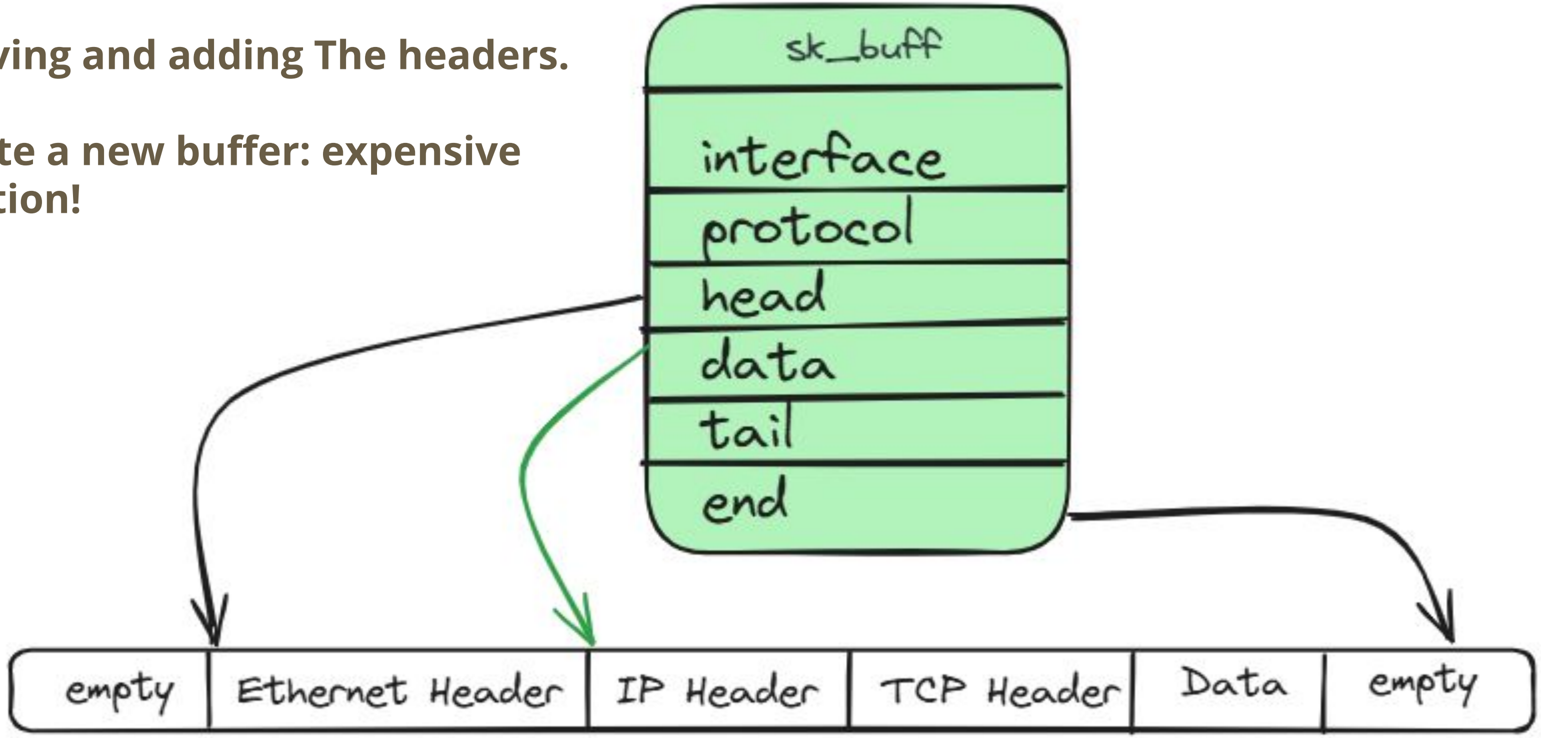
What is sk_buff?

sk_buff

- A structure in memory that includes metadata of packets.

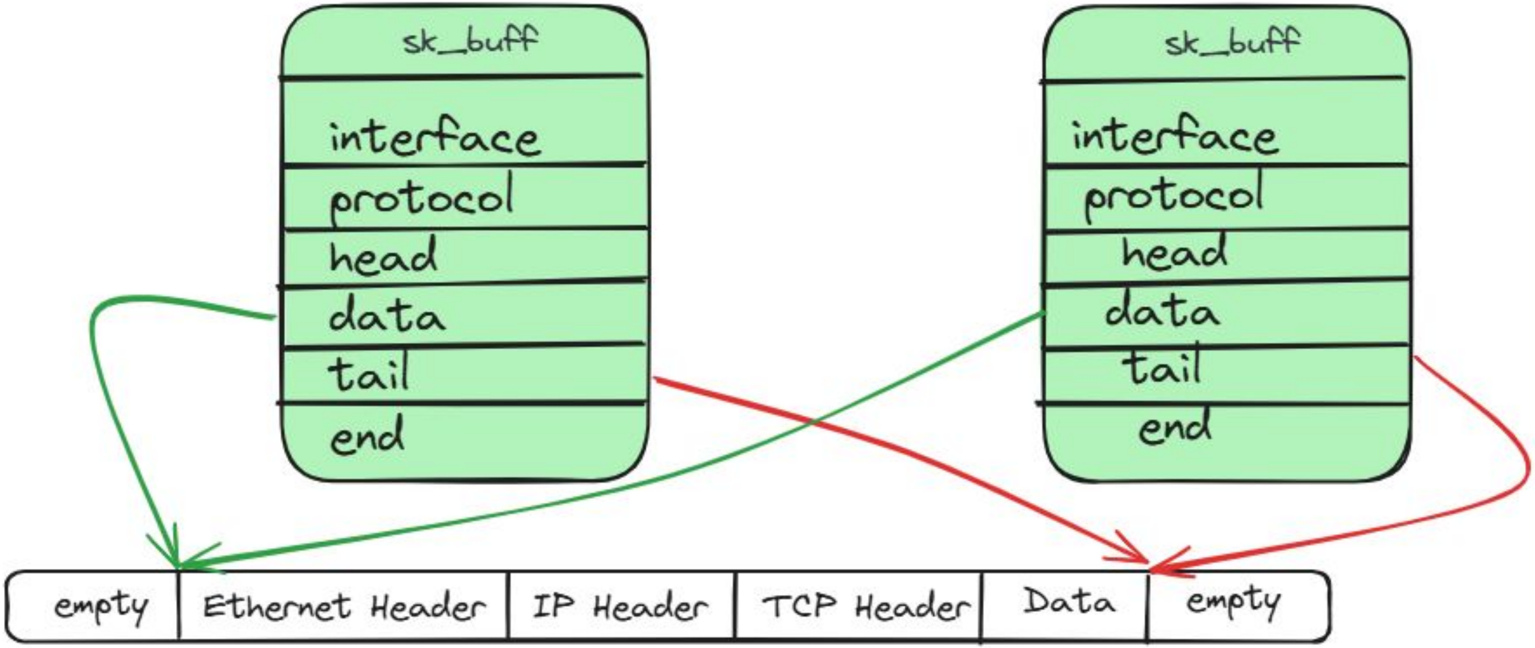
- Removing and adding The headers.

- Allocate a new buffer: expensive operation!



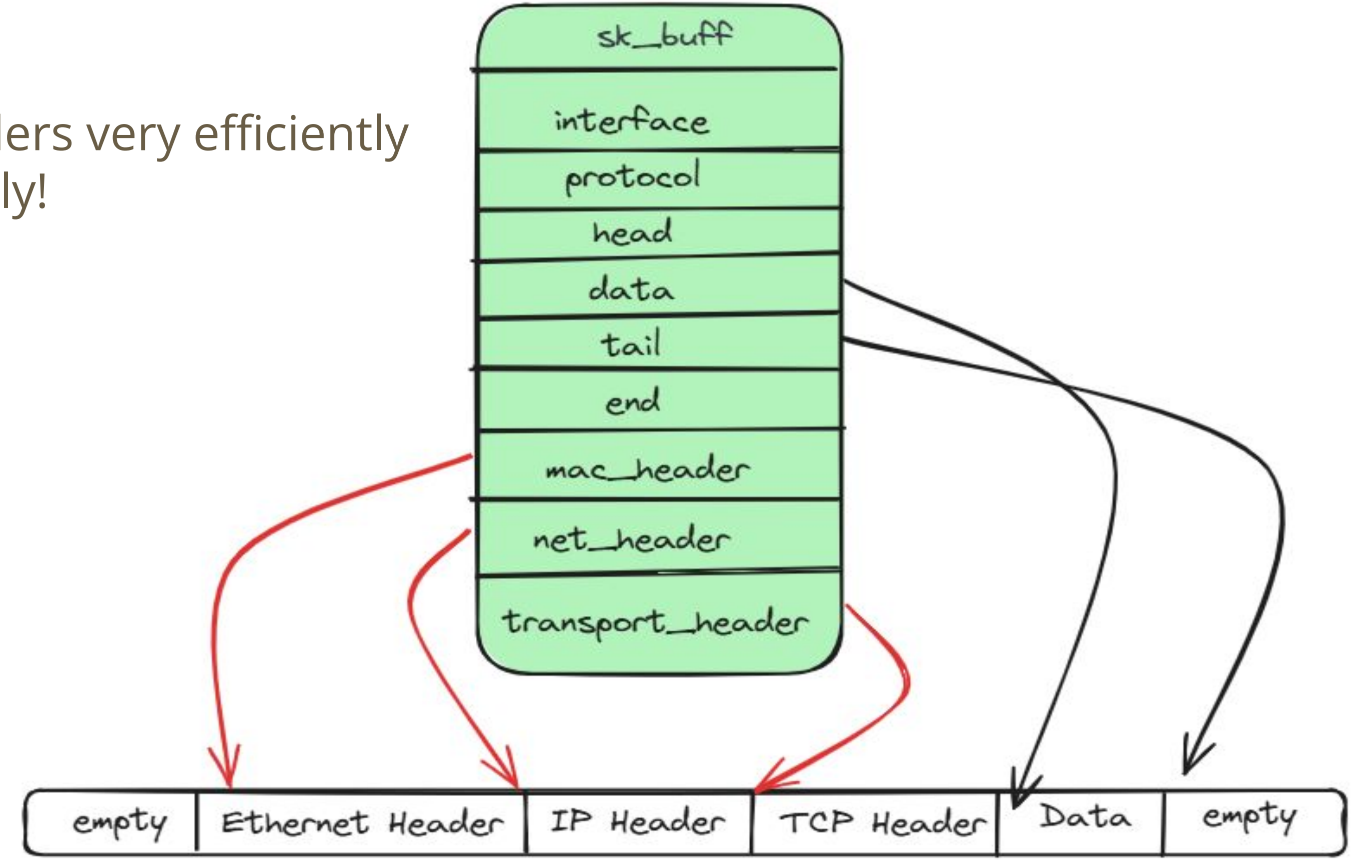
Sk_buff cloning

The Same content but different type pointers!



3 main pointers:

- Access to headers very efficiently
- And very quickly!



sk_buff

1- receiving the packet from sk_buff:

For each packet calls: netif_receive_skb() and direct it to a packet handler function for example refers it to ip_rcv() based on the type.

2. ip_rcv(): removes the ip header, defragments the packet, checks for errors and goes to ip_rcv_finish().

3. ip_rcv_finish(): route lookup for the packet and decides to delivered locally or forwarded.

5. dst_input():   it calls the ip_local_deliver() and defragments the packet, forward it locally and calls ip_local_deliver_finish() then calls the protocol specific function.

6.tcp_v4_rcv(): and pskb_may_pull(): valid TCP Header

7.csum

8. Looks up for open socket: __inet_lookup_skb()

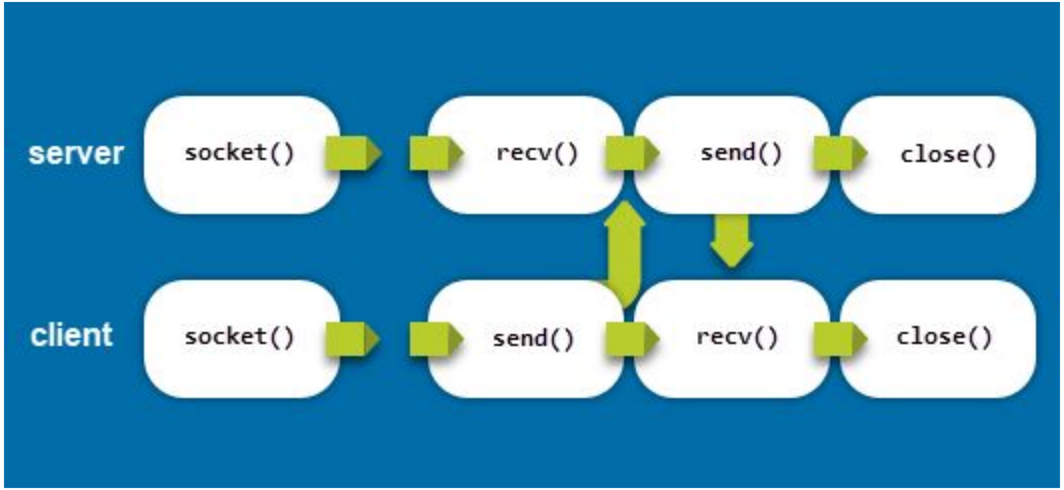
TCP socket status

- Sockets that are fully connected.
- Sockets that are waiting for a connection (in the listen state).
- Sockets that are in the process of establishing a connection (TCP 3-way handshake).

- Last calls at this point: tcp_v4-rcv calls tcp_v4_do_rcv(): branches

User Space

- **The only thing that is left to do is for the application to call the *accept()* function. This syscall takes out the connection from the accept queue, creates a new connected socket, and reads the packets/data corresponding to it using *read()*.**



Back to the journey ….

Bottom half:

- Identifies the memory where the packet was stored
- Allocates an sk_buff
- And fills in metadata
    - protocol (ETH metadata)
    - Receiving interface, packet type

- Sets MAC header

- Removes (pulls) the ETH header

- Passes the skb to the network stack

Bufferbloat: problem and solution

persistently full buffer problem: bufferbloat

1.cheap memory

2.Dynamically varying path characteristics

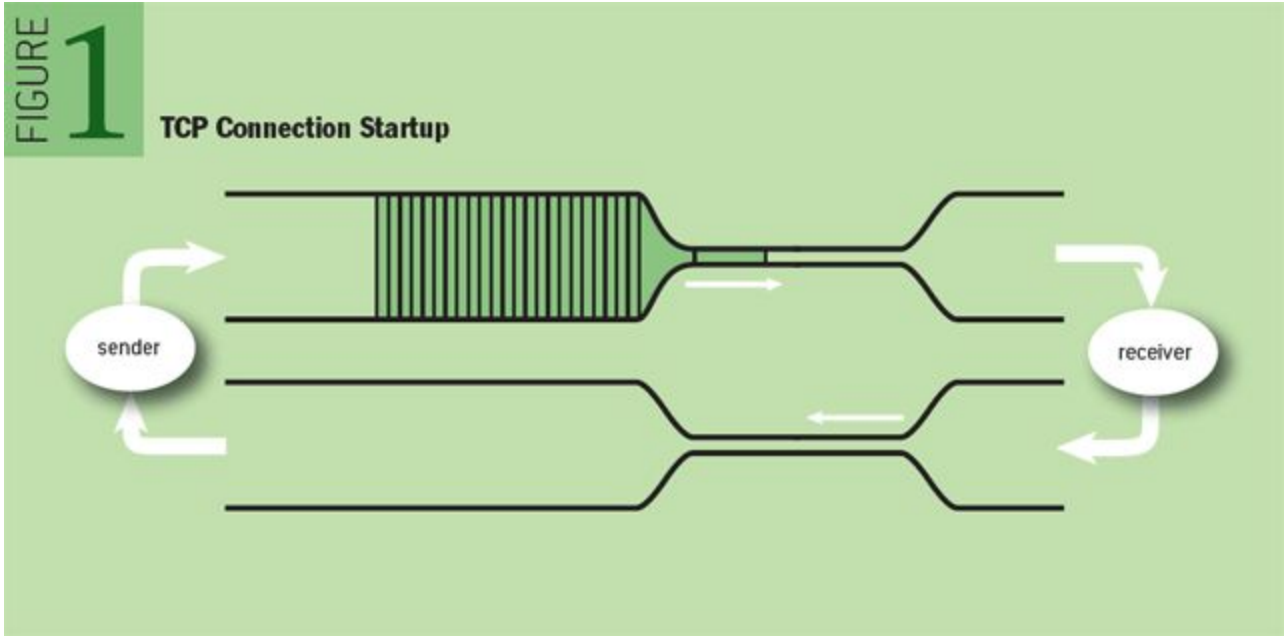
First solution for two decades was: AQM

And then RED was introduced.

New solution: CoDel

It adapts to changing link rates and is suitable for deployment and experimentation in Linux-based routers (as well as silicon).

Queues in Buffer: short-term mismatches in traffic arrival and departure



Codel

1. Focus on Queue Delay, Not Queue Size
2. 2. Detecting "Bad" Queues vs. "Good" Queues: minimum queue delay
3. Using Minimum Queue Delay for Decision Making: it’s fixed!
4. How CoDel Drops Packets

5. Adapting to Network Conditions

- CoDel’s operation is independent of link rates, round-trip times (RTTs), and traffic loads. It automatically adjusts to changes in these factors:
- It adapts to varying link speeds, which means it works well across different types of network connections (e.g., fiber, Wi-Fi, 4G, etc.).

Thank you

My social medias:

@MARYAMMIRZA
KHANICOLLEGE