USL

Unstoppable Session Layer

Alexander Aring

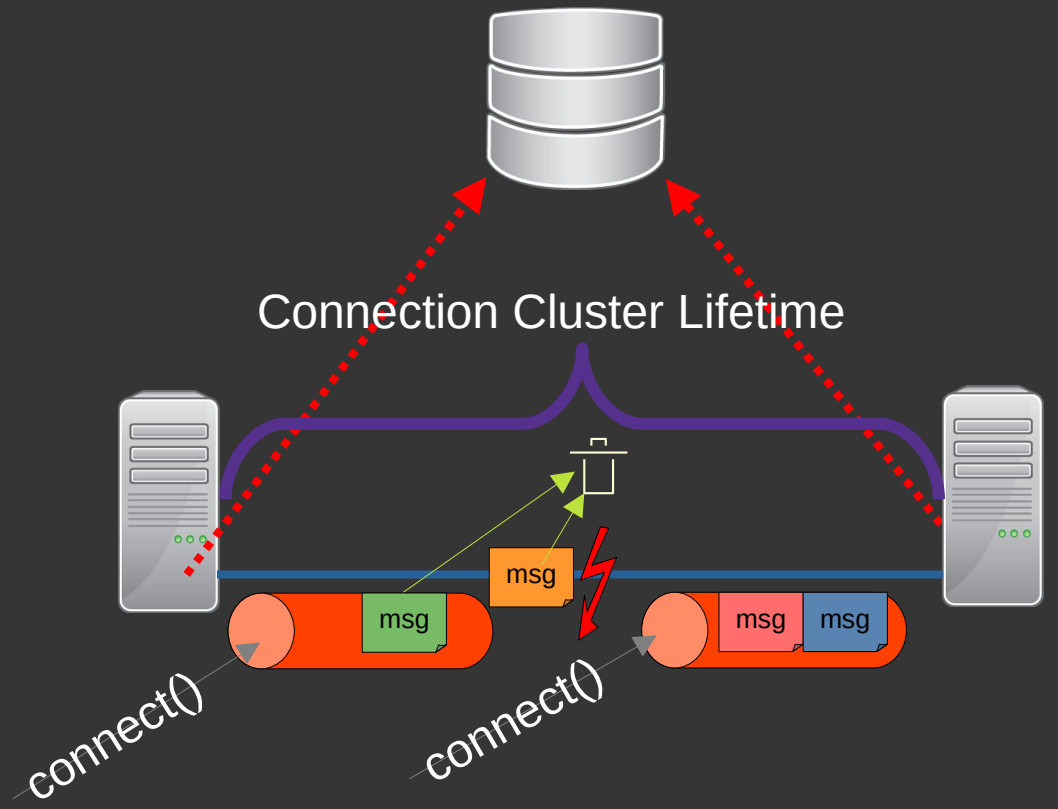
Red Hat

What is the Unstoppable Session Layer?

Red Hat

Problem is the Connection-oriented Socket Lifetime

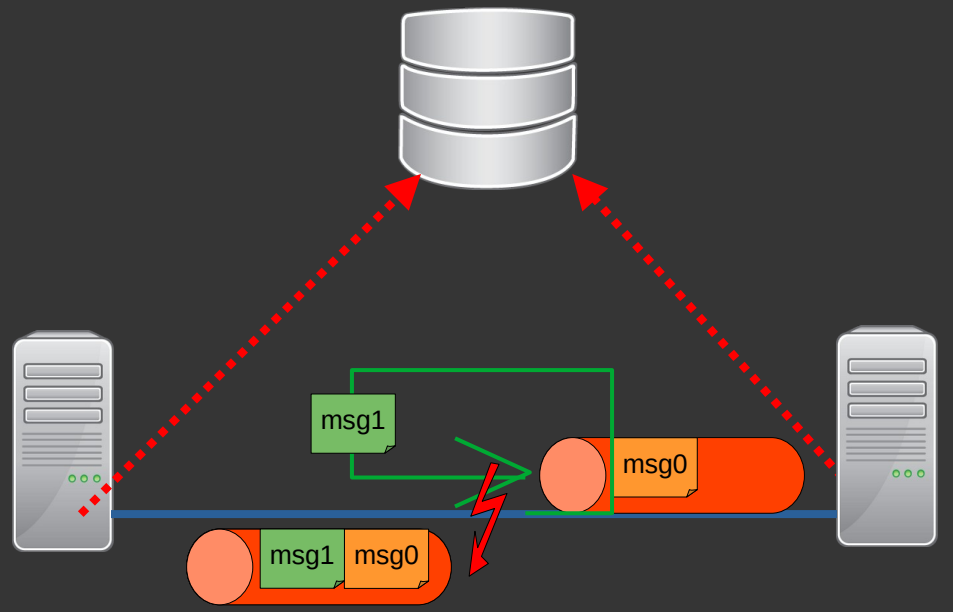
- First Priority is Reliability
- E.g. AF_TCP/SCTP
- Cluster Join Resource
- Problem: Socket Lifetime
  - Error → Reconnect
  - May end into data losses



Red Hat

Retransmit on Reconnect

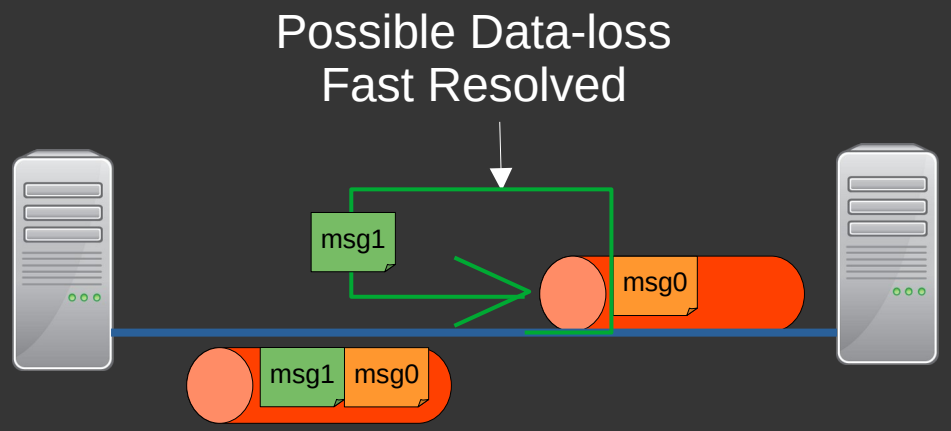
- Encapsulated Header
- Simple Seq and Ack based
- Acks are Piggybacked on Messages
- Retransmit non Acked Messages after Reconnect



Red Hat

Why Retransmit on Reconnect?

- Only a Jitter occurs in Hotpath
- No full Application re-synchronization
  - Takes too much time, may block pending Application due re-synchronize with others
  - Only necessary if Application/USL state got lost, e.g. Power Outage (Fence)



Red Hat

How to transparent USLify your Application?

Red Hat

Idea: Transparent* USLify your Application

- Do you know torsocks[1)] ?
- Example: `torsocks ssh $DESTINATION`
  - Redirects to local SOCKS5 Tor daemon server
  - Outgoing traffic will be done through Tor
- Main mechanism is LD_PRELOAD Env
- Torsocks isn’t easy – e.g. DNS resolving

Source:
1) https://gitlab.torproject.org/tpo/core/torsocks/

Red Hat

The LD_PRELOAD Environment Variable

LD_PRELOAD

A list of additional, user-specified, ELF shared objects to be loaded before all others. This feature can be used to selectively override functions in other shared objects.

1)

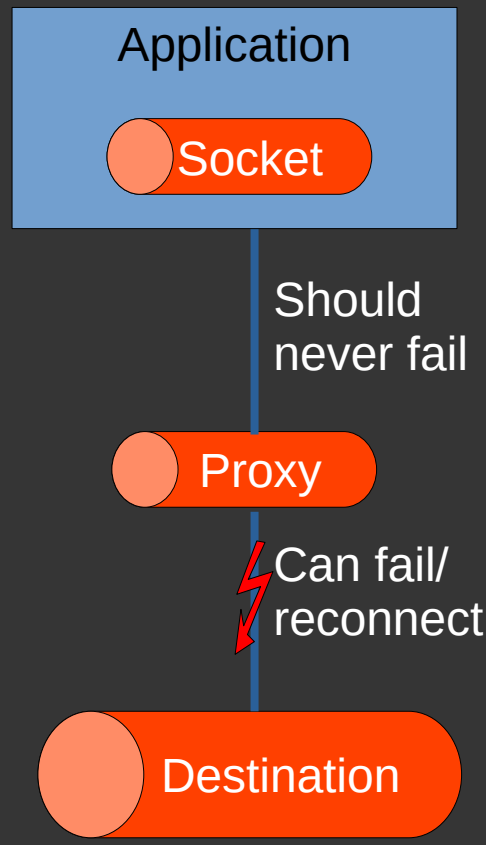
- Idea override the networking related Syscalls
- LD_PRELOAD=libusl.so nc -4t 127.0.0.1 1337
- Hide it in an Script: usl nc -4t 127.0.0.1 1337
- Pass Parameters: usl -foo bar nc -4t 127.0.0.1 1337

Source:
1) https://www.man7.org/linux/man-pages/man8/ld.so.8.html

Red Hat

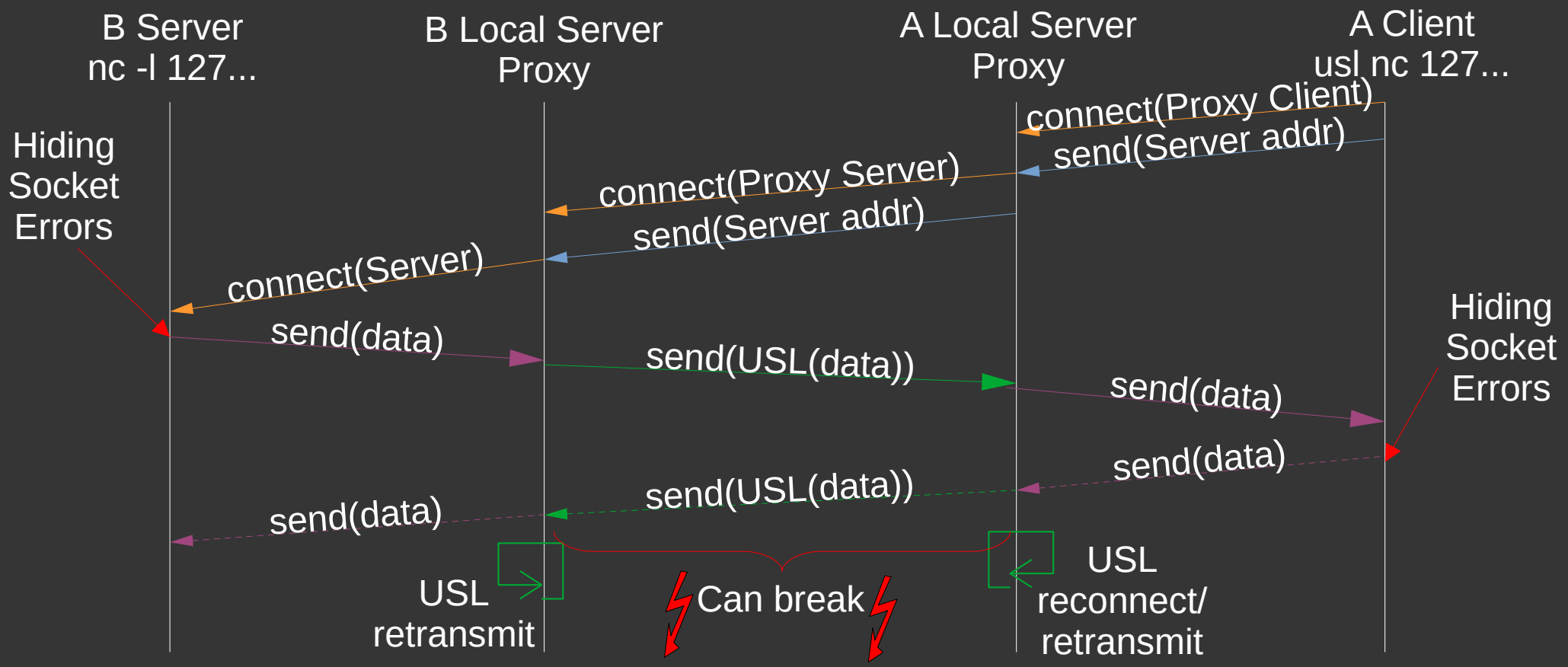
Problem with just Override everything

- Override send(), recv(), etc? DON'T!
  - Difficult to catch all Socket recv()/send() operation methods...
  - We need to Hide Socket Errors
  - The Application may not just reconnect
- Solution: Override connect(2)
- Socket Proxy hiding Errors to Application Socket!



Red Hat

SOCKS* Example Transparent USL Tunneling



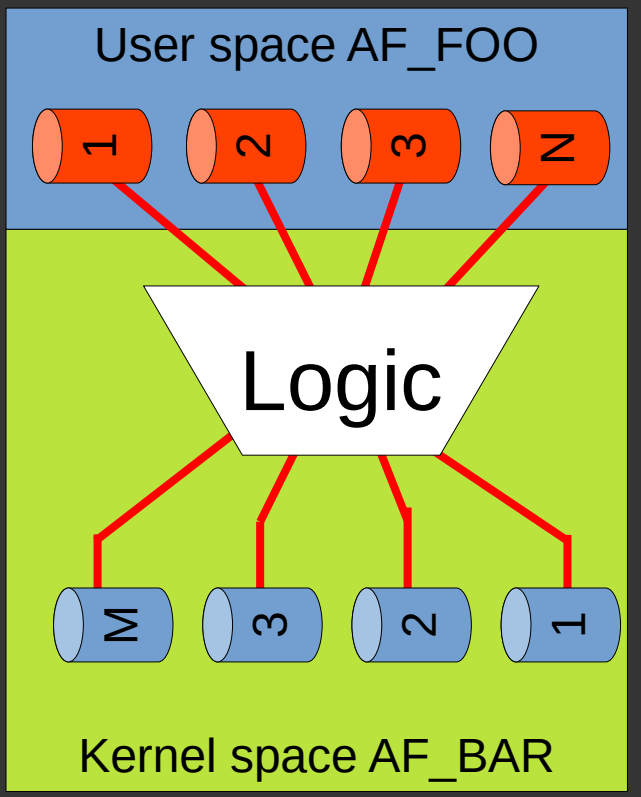
Red Hat

LD_PRELOAD is difficult to maintain!

- Think about torsocks every time you introduce a new method to do e.g. connect(2)!
- New Project provide similar functionality in a more “stable” way?
- It is a nice toy…, we look into other options! But maybe you got some new ideas. ;-)

Red Hat

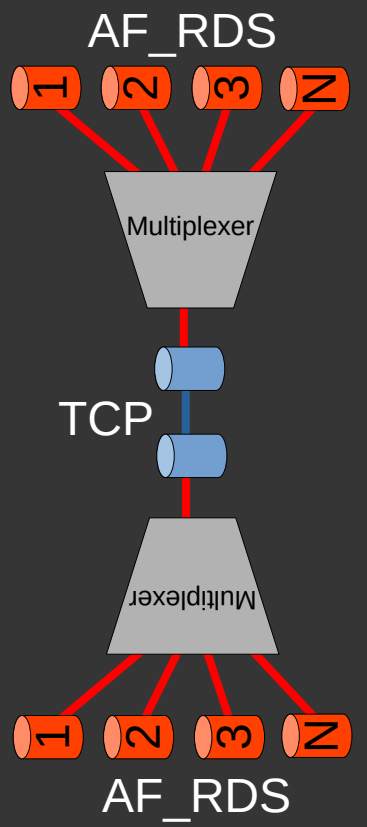
Compared to Socket based Solutions (Multiplexers*)



Red Hat

Reliable Datagram Sockets

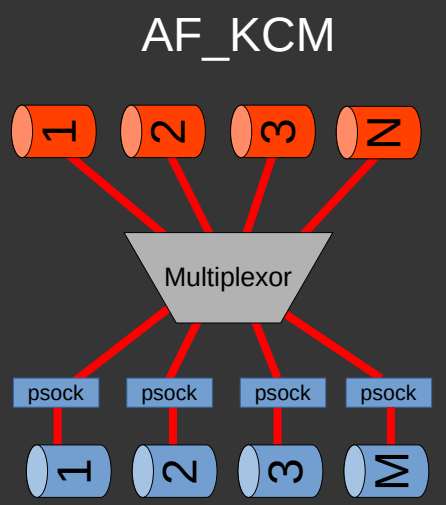
- AF_RDS N (Process Independent)
- 100% Reliable like USL (Encapsulated Header)
- Own Port handling (Multiplexing into One Socket)
- Needs Adaptation (TCP/RDMA/…)
- Multipath* (not MPTCP) available



Red Hat

Kernel Connection Multiplexer

- Multiplexer (NxM)
- Message Framing (psock)
- Multiplexor Logic (Load Balancing)
  - AF_KCM → Deliver when App blocks in recv() or use next sk
  - IPPROTO_TCP → If congested use next sk
- Where is the Counterpart? Designed for Server only? No Encapsulated Header!



Red Hat

USL could solve a note of AF_KCM kdoc!

Error handling

An application should include a thread to monitor errors raised on the TCP connection. Normally, this will be done by placing each TCP socket attached to a KCM multiplexor in epoll set for POLLERR event. If an error occurs on an attached TCP socket, KCM sets an EPIPE on the socket thus waking up the application thread. When the application sees the error (which may just be a disconnect) it should unattach the socket from KCM and then close it. It is assumed that once an error is posted on the TCP socket the data stream is unrecoverable (i.e. an error may have occurred in the middle of receiving a message).

1)

Source:
1) https://www.kernel.org/doc/html/latest/networking/kcm.html#error-handling

Red Hat

Moonshot Idea!

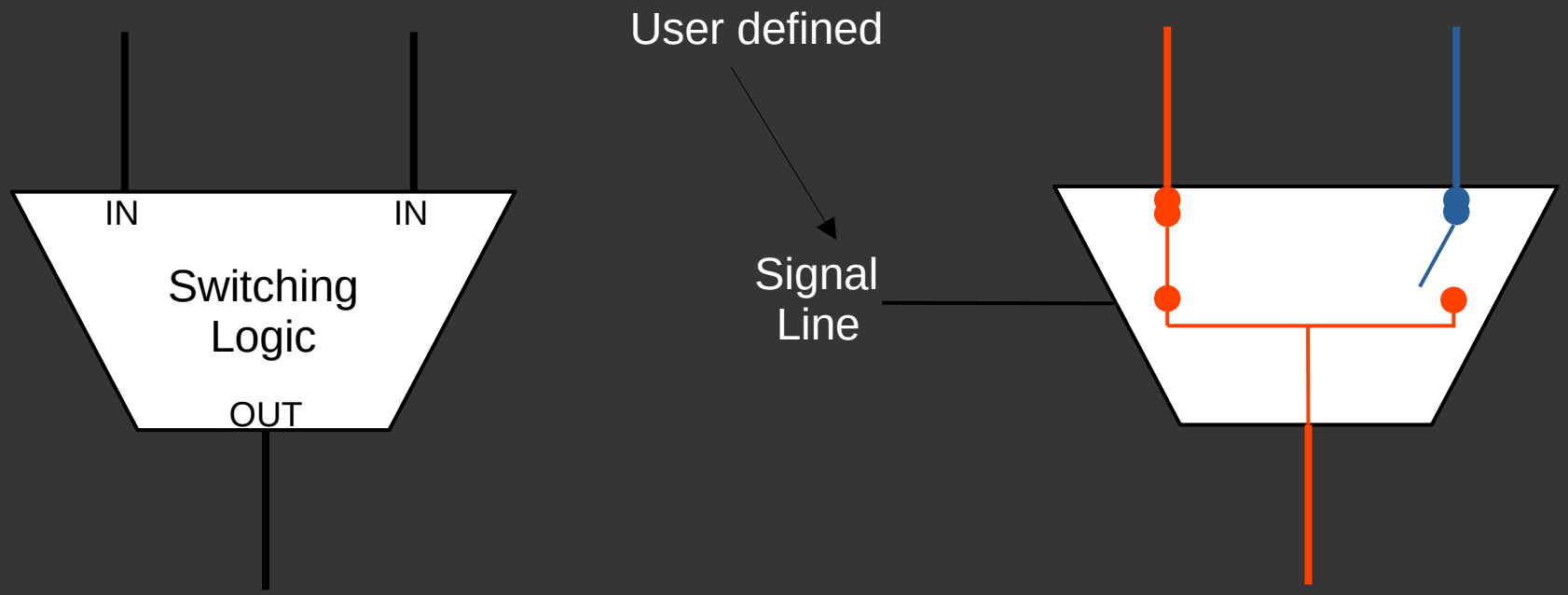
Introducing: “Socket Control”

(An approach how to make AF_KCM “programmable”)

Red Hat

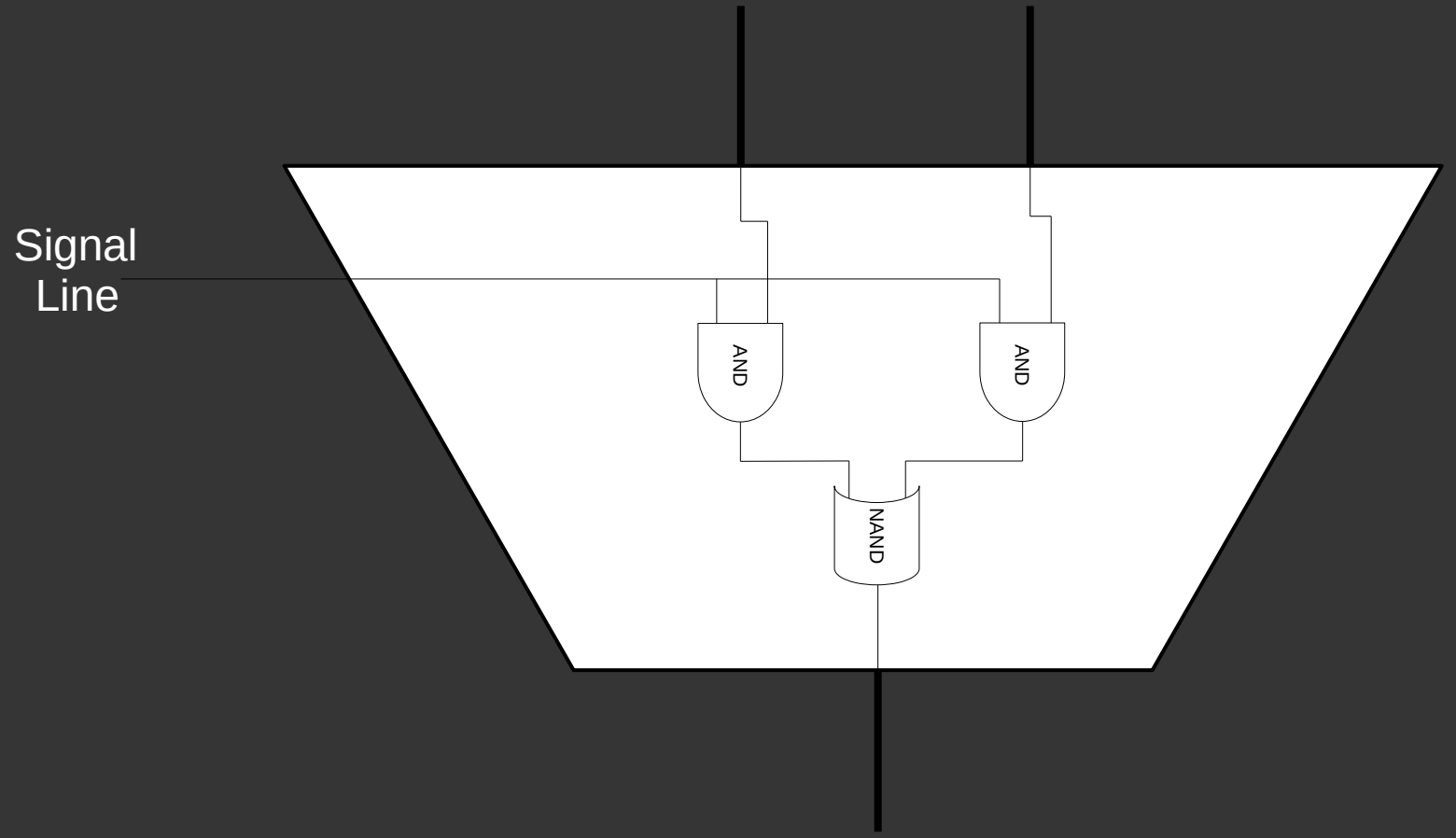
What is a Multiplexer?

Electronic Basics – 2x1 Multiplexer



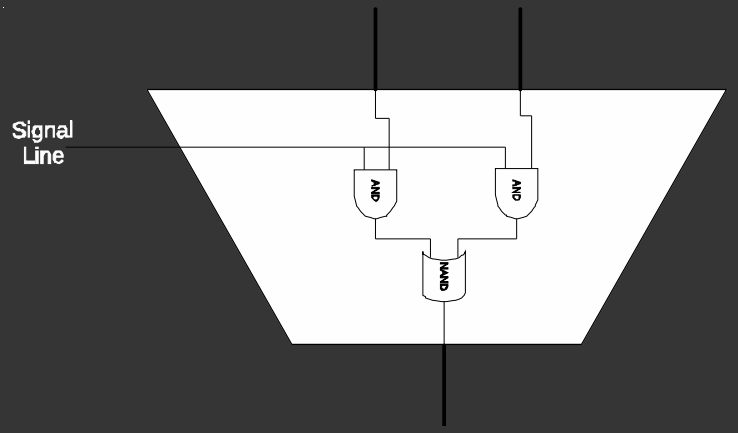
Red Hat

Multiplexer Logic are just basic AND, NAND, etc. Gates!

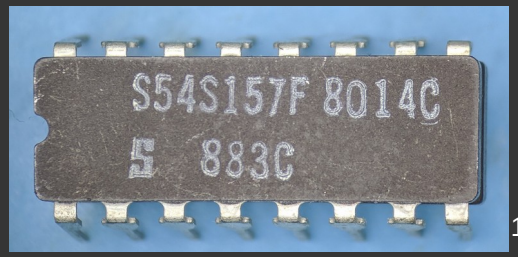


Red Hat

In a Multiplexer IC – Logic is static!



==



1)

- User cannot change Switching Logic
- Multiplexer IC is specific for the Use-Case

Source:
1) By Robert.Baruch - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=63802402

Red Hat

The existing Socket Multiplexers are like ICs

Their Logic is static!

1)

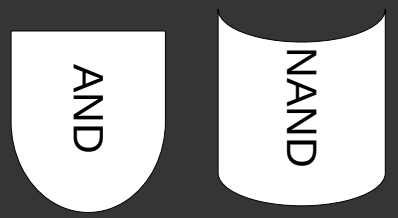
Those Multiplexers are specific for a special Use-Case defined by it’s Authors (and you need to take it)

Source:
1) By Robert.Baruch - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=63802402

Red Hat

How to put the static Logic out of IC Multiplexers?

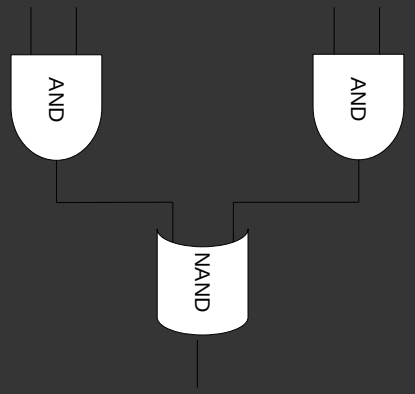
Use Primitives and Wire them together!

Primitives: 

...

Wiring: ——————

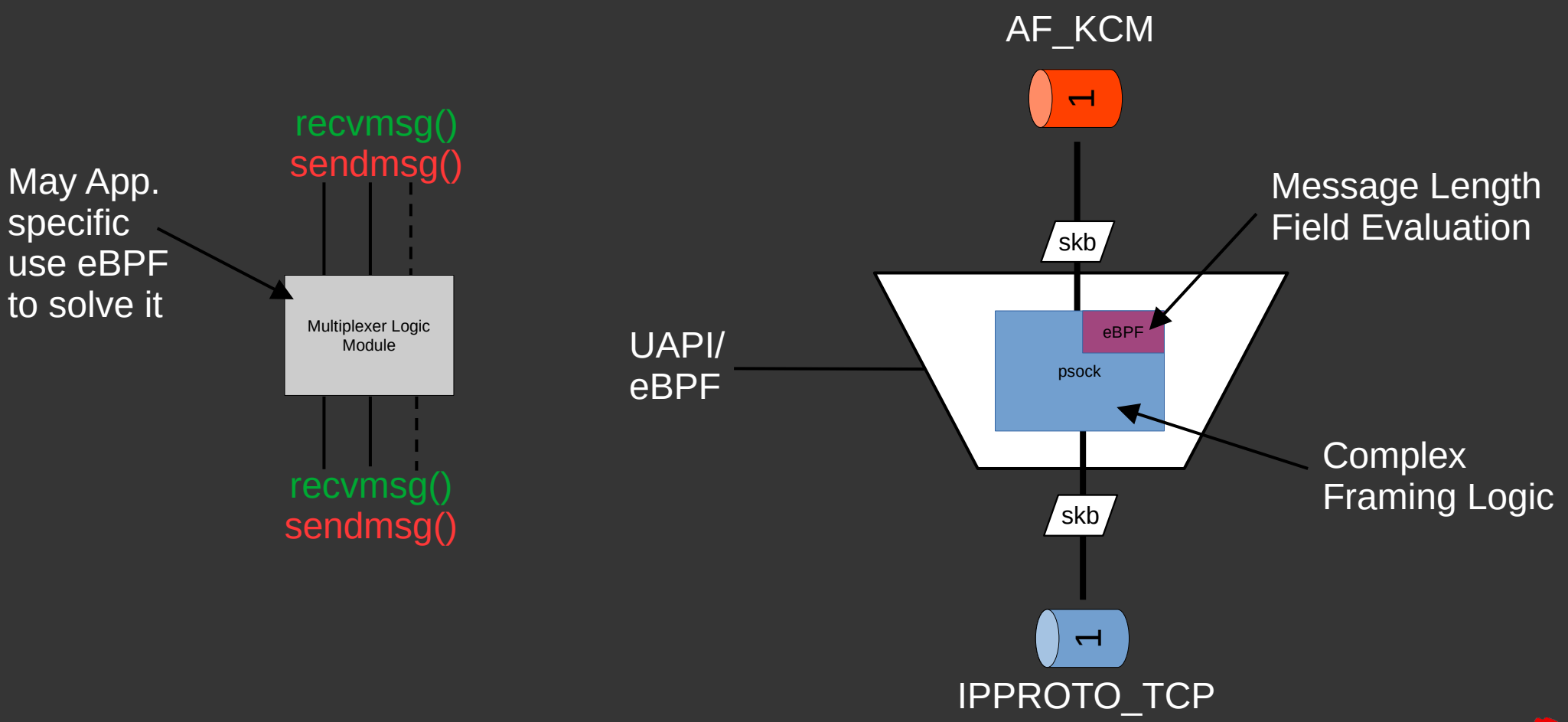
Constructed Complex Logic:



Red Hat

Simple Example 1x1 – Message Framing only

(AF_KCM can already do that)



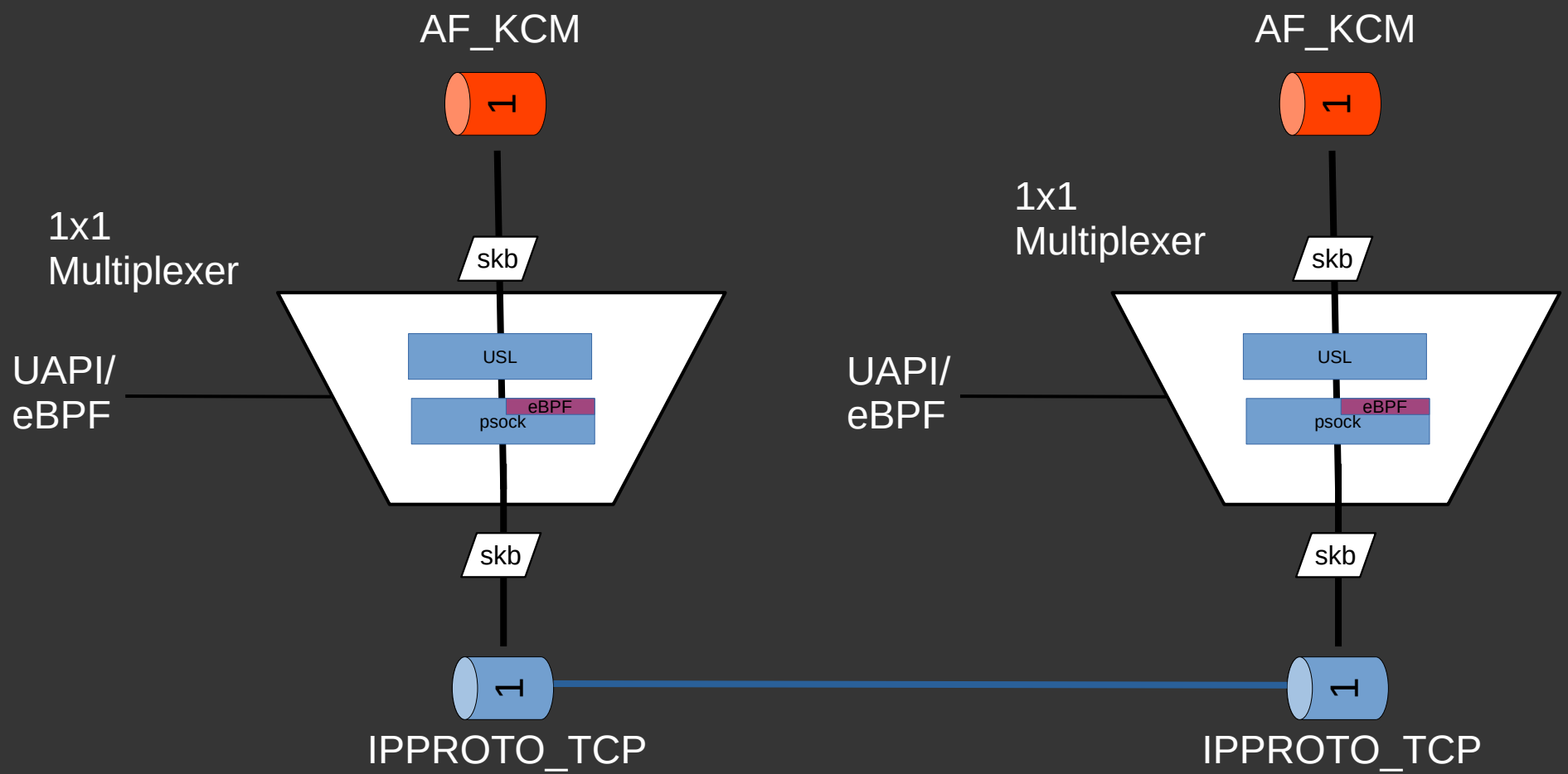
Red Hat

Reuse existing AF_KCM implementation

- AF_KCM is good into defining Multiplexer Sockets
  - N AF_KCM Sockets (clone)
  - M AF_TCP Sockets (attach/unattach)
- Multiplexer Logic
  - Default Logic is the current AF_KCM Logic
  - Can be flushed and reprogrammed by User

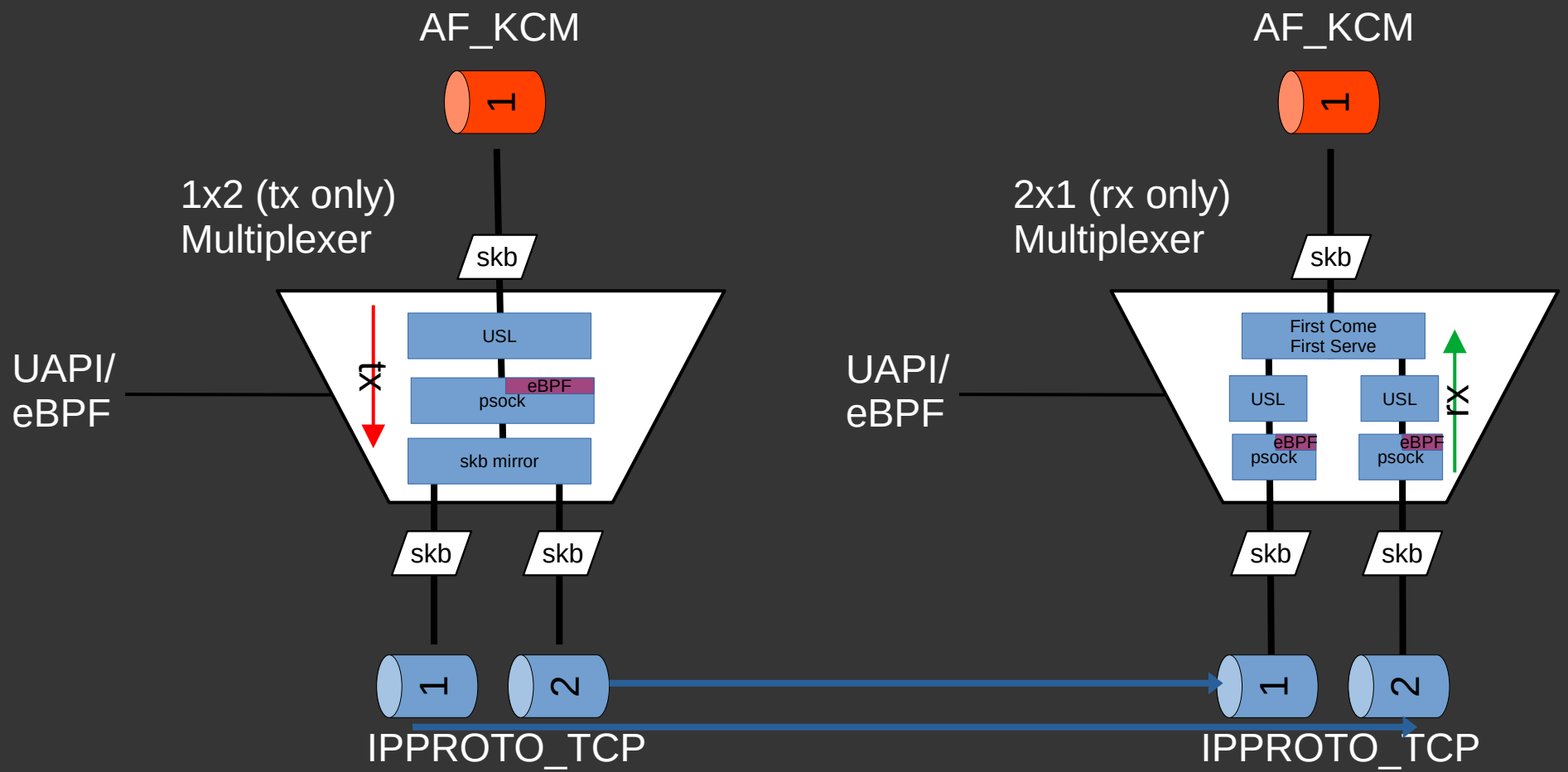
Red Hat

How this fits with USL?



Red Hat

USL: Redundant Multipath*



Red Hat

Future Work

- Transport Layers? QUIC/MPTCP
- USL existing kind of in “fs/dlm/” (No Multipath)
- I try to move it out from my Application Layer
- Maybe start to look into “programmable” AF_KCM?
- Update “fs/dlm” to use AF_KCM and program my needs e.g. psock, USL, ?multipath mirroring?

Red Hat

I hope you enjoyed this talk

- May the LD_PRELOAD trick was new for you
- You may liked the idea of “Socket Control”
- I missed in my talk… (Sorry)
  - Talking about Half-Closed issue
  - The Demo using Proxys and Netcat

Red Hat