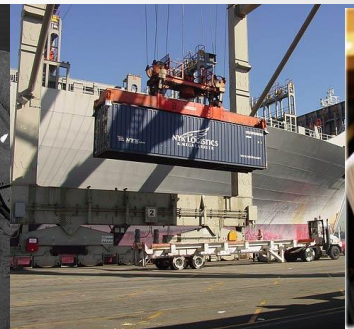
UDP Encapsulation in Linux

netdev0.1 Conference
February 16, 2015

Tom Herbert <therbert@google.com>





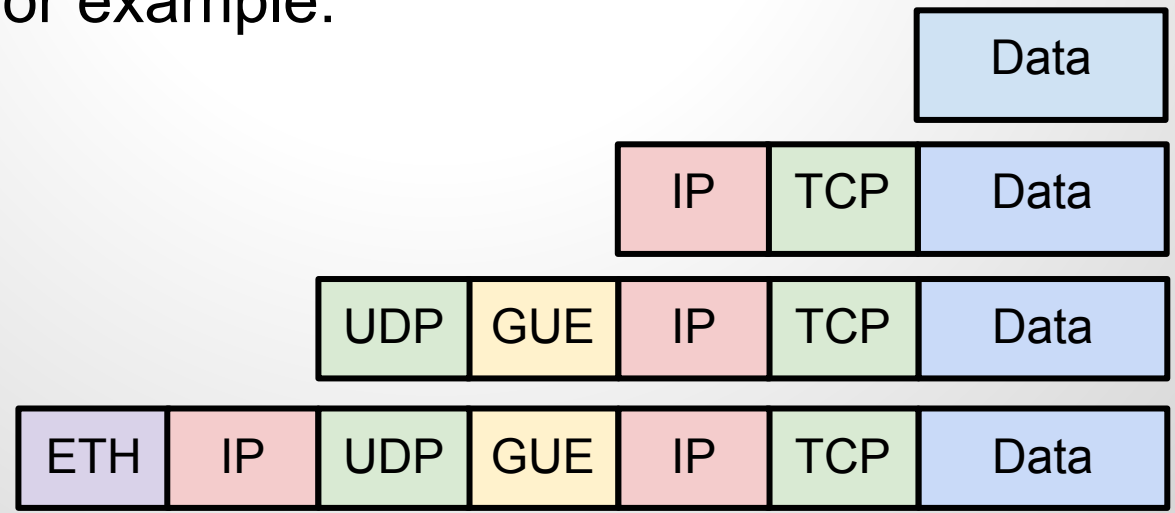



Topics

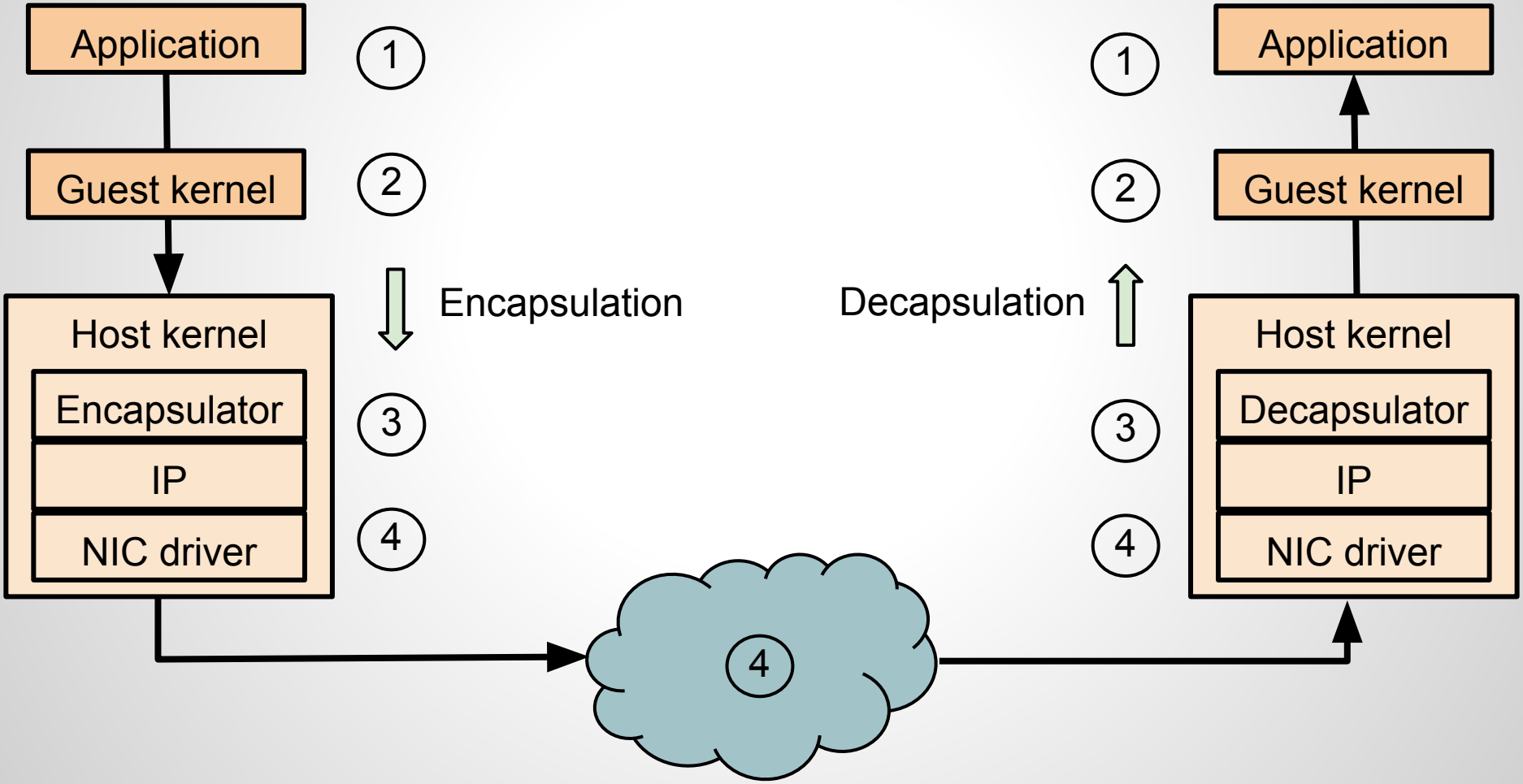
- UDP encapsulation
- Common offloads
- Foo over UDP (FOU)
- Generic UDP Encapsulation (GUE)

Basic idea of UDP encap

- Put network packets into UDP payload
- Two general methods
  - No encapsulation header: protocol of packet is inferred from port number
  - Encapsulation header: extra header between UDP header and packet. Protocol and other data can be there. For example:



VM encap example

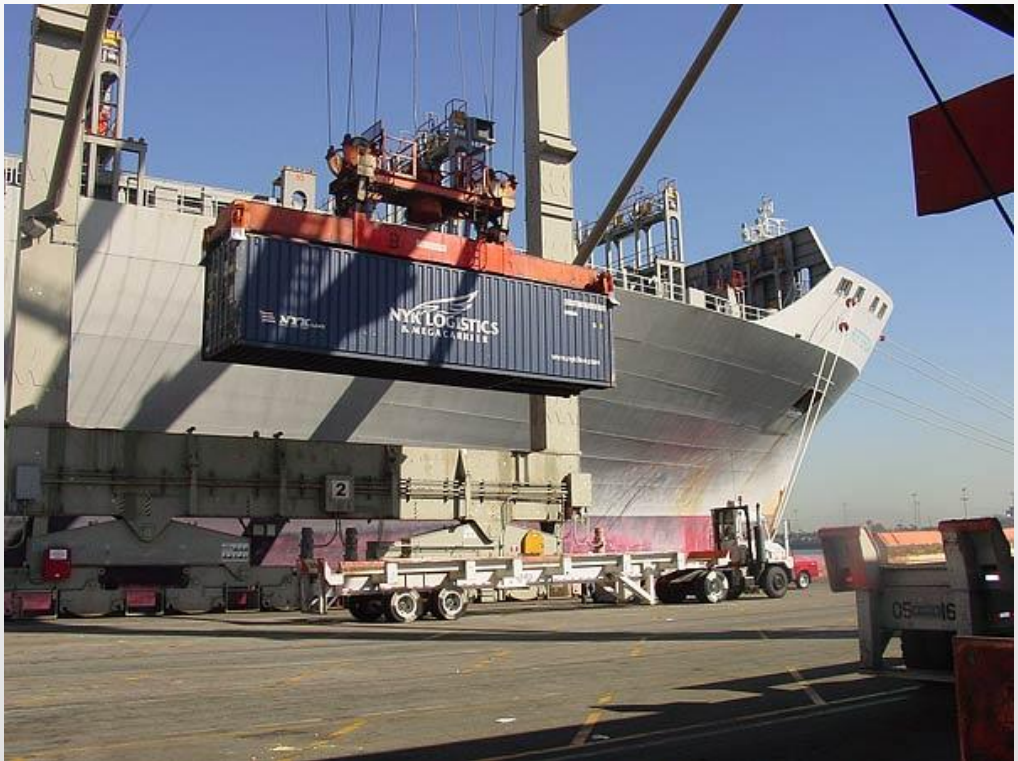


UDP encap popularity

- UDP works with existing HW infrastructure
  - RSS in NICs, ECMP in switches
  - Checksum offload
- Used in nearly all encap, NV data protocols
  - VXLAN, LISP, MPLS, GUE, Geneve, NSH, L2TP
- Likelihood UDP based encapsulation becomes ubiquitous
  - In time most packets in DC could be UDP!

Offloads

- Load balancing
- Checksum offload
- Segmentation offload



Load balancing

- For ECMP, RSS, LAG port selection
- Probably all switches can 5-tuple over UDP/IP packets
- Solution: use source port to represent hash of inner flow
  - ~14 bits of entropy
  - udp_src_flow_port function

TX Checksum offload

- NETIF_HW_CSUM
  - Initialize checksum to pseudo header csum
  - Input to device *start* and *offset*
  - HW checksums from start to end of packet and writes result at offset
- NETIF_IP_CSUM
  - HW can only checksum with certain protocol hdrs
  - Typically UDP/IP and TCP/IP
  - HW handle pseudo hdr csum also

RX Checksum offload

- CHECKSUM_COMPLETE
  - HW returns checksum calculation across whole packet
  - Host uses returned value to validate checksum(s) in the packet
- CHECKSUM_UNNECESSARY
  - HW verfies and returns “checksum okay”
  - Protocol specific, HW needs to parse packet
  - csum_level allows HW to checksum within encapsulation, multiple checksums

Checksum offload for encapsulation

- Need to offload inner checksum like TCP
- UDP also has it’s own checksum, this makes things interesting!

The MIGHTHY UDP Checksum for Encaps

- Want set to zero for “performance” (particularly switch vendors), **but...**
- UDP checksum is *required* for IPv6, **and…**
- UDP checksum covers more of packet than inner checksum, **but...**
- RFC6935, RFC6936, and a lot more requirements in encapsulation protocol drafts to allow it, **but…**
- UDP checksum is actually a **good** idea for both v4 and v6 when you’re using Linux hosts to do encapsulation, **let me explain...**

Leveraging UDP checksum offload

- Probably every deployed NIC supports simple UDP checksum for TX and RX
- Only new NICs support offload of encapsulated checksums
- Solution: Enable UDP checksum for encap and use it to offload inner checksums
  - Receive: checksum-unnecessary conversion
  - Transmit: remote checksum offload

Checksum unnecessary conversion

- Device returns “checksum unnecessary” for non-zero outer UDP checksum
- Complete checksum of packet starting from the UDP header is ~pseudo_hdr_csum
- So convert checksum unnecessary to checksum complete
- Inner checksum(s) verified using checksum complete
- No checksum computation on host!

Remote checksum offload

- Defer TX checksum offload to remote
- Encapsulation header with *start* and *offset* data referring to inner checksum
- Offload outer UDP checksum and send
- At receive
  - Do what device does: determine checksum from start to end of packet and write to offset
  - Aleady have complete checksum so we can easily find this
  - Write checksum into packet, validate like normal
- No checksum calculation in host

Segmentation offload

- Stack operates on bigger than MTU sized packets
- Offloads in receive and transmit

Transmit segmentation offload

- Split big TCP packet into small ones
- GSO (stack), TSO (HW)
- For each created packet
  - Copy headers from big one
  - Adjust lengths, checksums, sequence number that must be set per packet

GSO for UDP encapsulation

- UDP GSO function calls skb_udp_tunnel_segment
- Call GSO segment for next layer: gso_inner_segment
- Adjust UDP length and checksum per packet
- For encapsulation header, **just copy** those bytes*

*Assuming encapsulation header does not have fields that must be set per packet

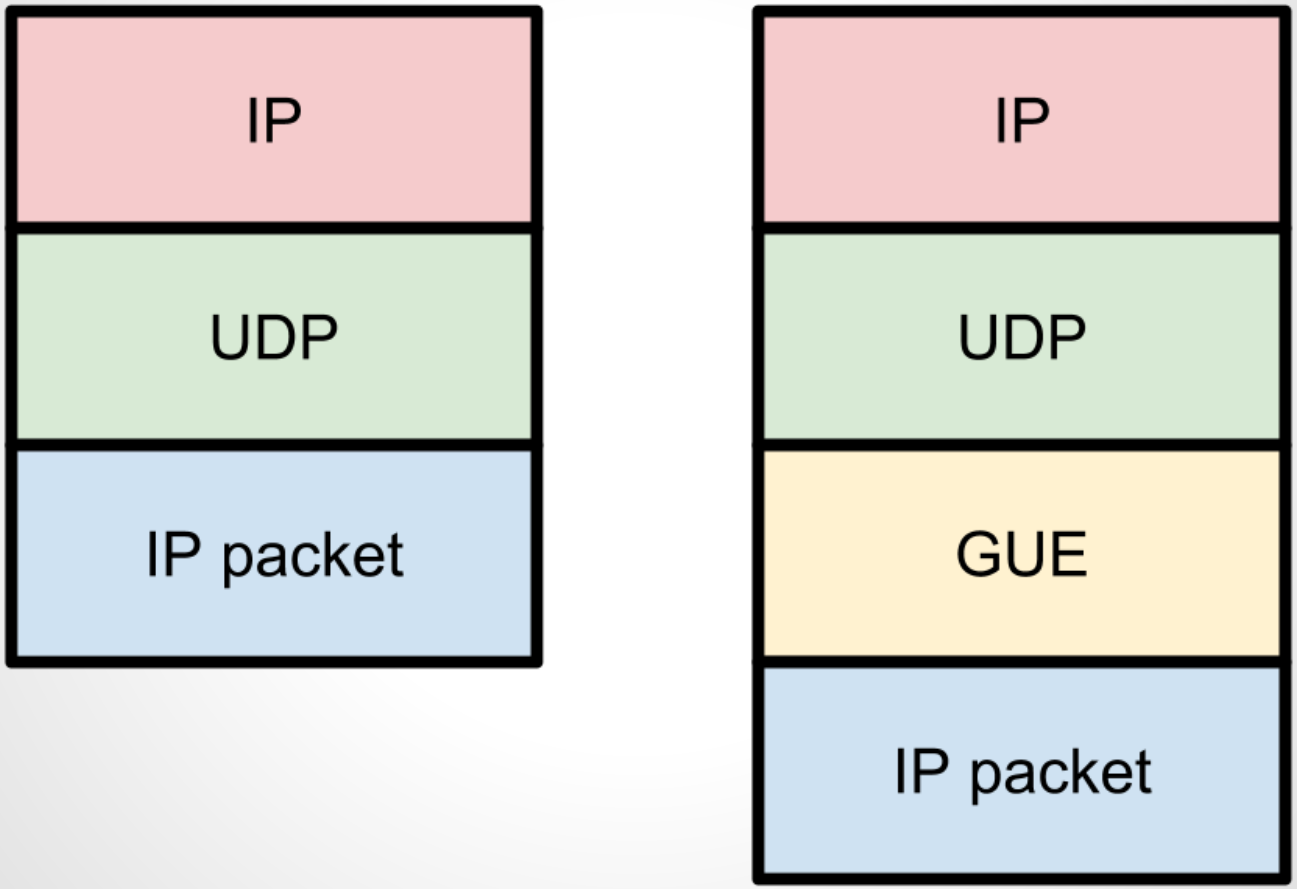
Receive segmentation offload

- Build large TCP packet from small ones
- GRO operation is to match packets to same flow for coalesing
- GRO (stack), LRO (HW)

GRO for UDP encapsulation

- UDP GRO receive path (udp_gro_receive)
- Encapsulation specific GRO functions
  - Call GRO function per port
  - Facility to register offloads per port
  - Call GRO receive for next protocol

FOU and GUE



FOU and GUE encapsulating IP

Foo over UDP

- Packets of IP protocol over UDP
- Destination port maps to IP protocol
  - e.g. IP (IPIP), IPv6, (sit), GRE, ESP, etc
  - Example: IPIP on port 5555

FOU support

- Logically, a header **inserted** to facilitate transport
- fou.c implements RX.
  - encap_rcv in socket
  - Remove UDP and reinject IP packet as protocol associated with port
- Ip tunnel implements FOU for IPIP, SIT, GRE
  - Insert UDP header between IP and payload
  - Source port from flow_hash

FOU example

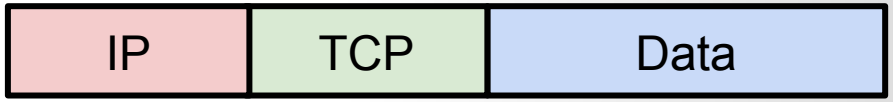
- Set up receive

```
ip fou add port 5555 ipproto 4
```

- Set up transmit

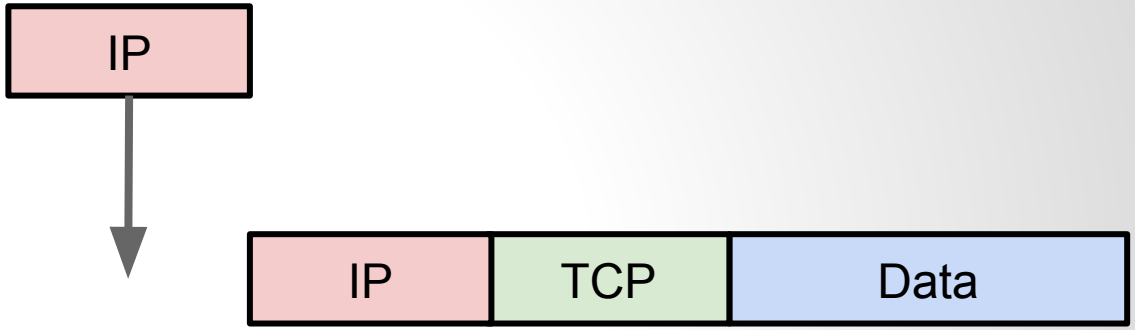
```
ip link add name tun1 type ipip \
    remote 192.168.1.1 \
    local 192.168.1.2 \
    ttl 225 \
    encap fou \
        encap-sport auto \
        encap-dport 5555
```

IP in FOU transmit



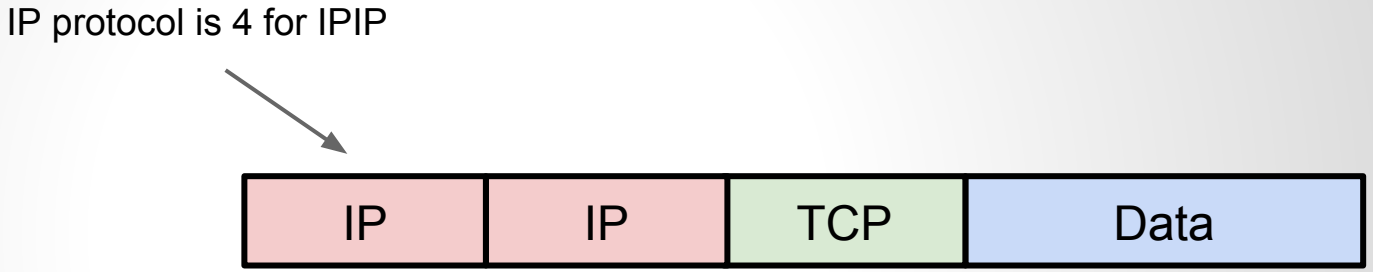
Start with a plain TCP/IP packet sent on tun1

IP in FOU transmit



Logically prepend IP header

IP in FOU transmit

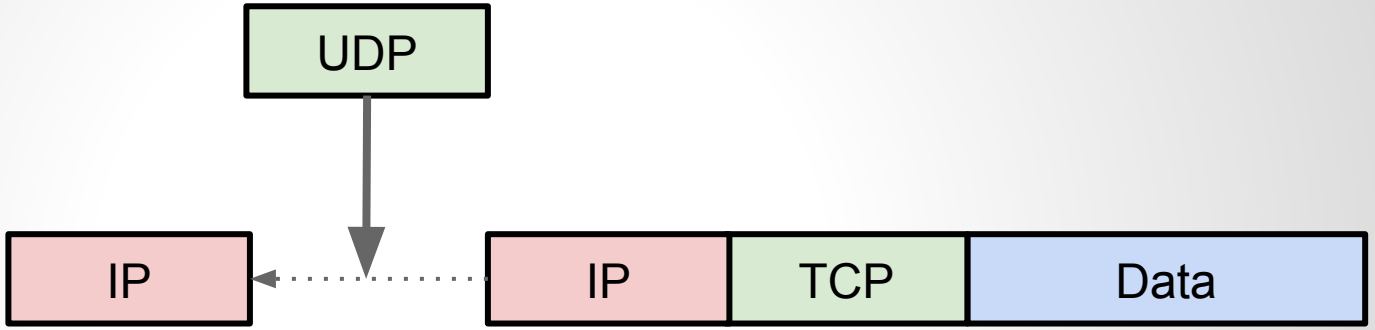


This is IPIP encapsulation

IP in FOU transmit

UDP destination port set to 5555 for IP/UDP

UDP port set to hassh value for inner IP/TCP headers



Insert UDP header

IP in FOU transmit

IP packet with encapsulation

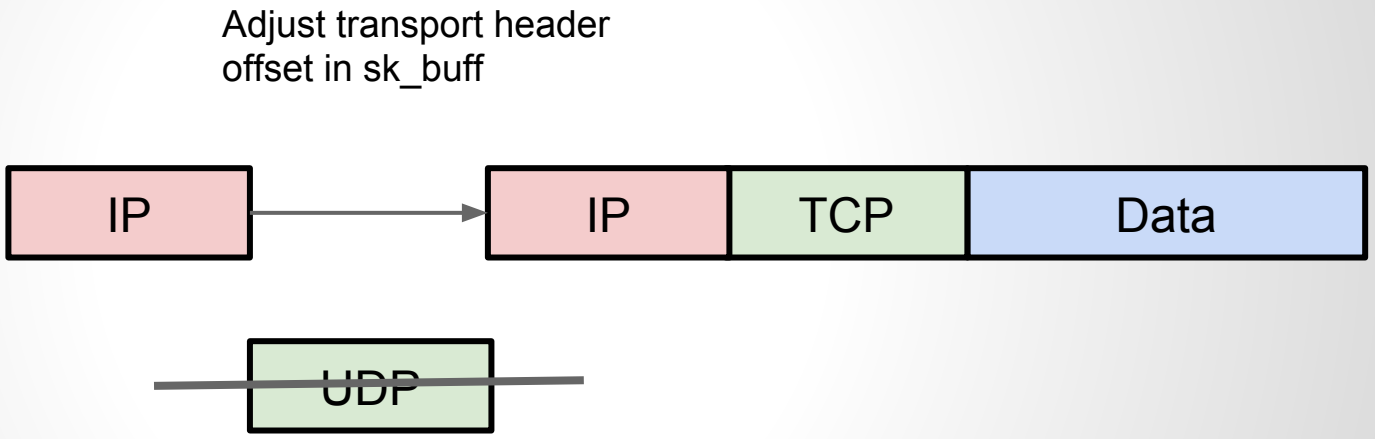
IP in FOU transmit

Add Ethernet header and send

IP in FOU receive

Receiver processes UDP packet based on destination port

IP in FOU receive



Remove UDP header

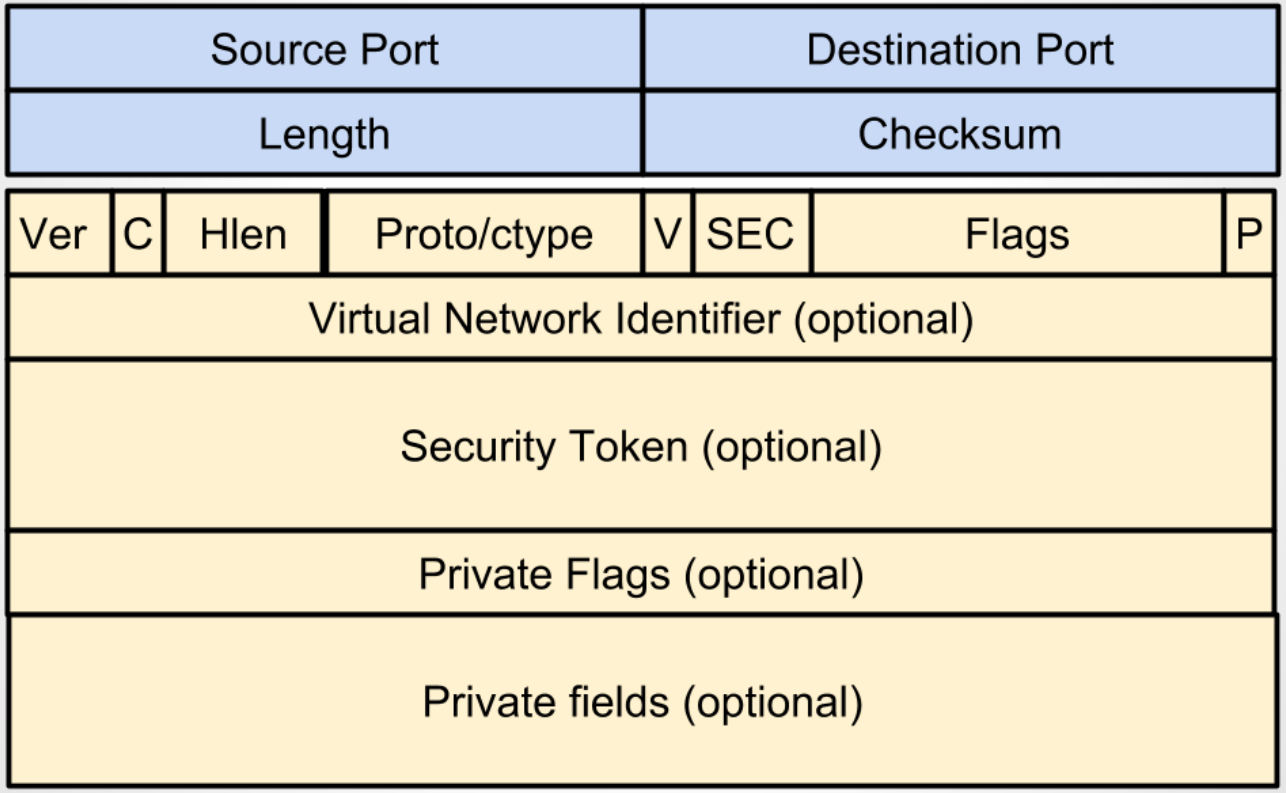
IP in FOU receive

Now have original IPIP packet. Reinject this into kernel, next protocol to prcess is 4

Generic UDP encapsulation (GUE)

- Extensible and generic encapsulation proto
- Encapsulation header for carrying packets of IP protocol
- Type field, header length, 8 bit IP protocol
- 16 bit flags and optional fields indicated by them. More can be defined in extension
- Private/extension flag

GUE headers



UDP and GUE headers

GRE/GUE example

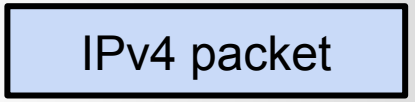
- Set up receiver

```
ip fou add port 7777 gue
```

- Set up transmit

```
ip link add name tun1 type ipip \
    remote 192.168.1.1 \
    local 192.168.1.2 \
    ttl 225 \
    encap gue \
        encap-sport auto \
        encap-dport 7777 \
        encap-udp-csum \
        encap-remcsum
```

GRE in GUE transmit

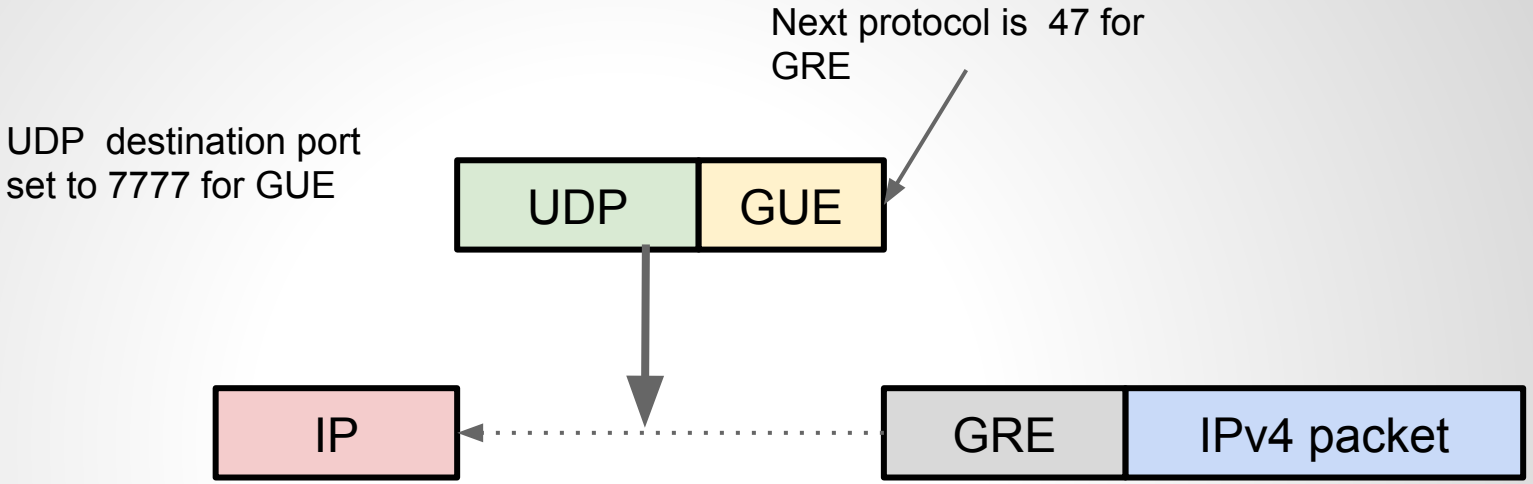


Application sends packet on tun1

GRE in GUE transmit

Logically prepend IP header for GRE/IP tunneling

GRE in GUE transmit



Insert UDP/GUE headers

GRE in GUE transmit

Insert UDP/GUE headers

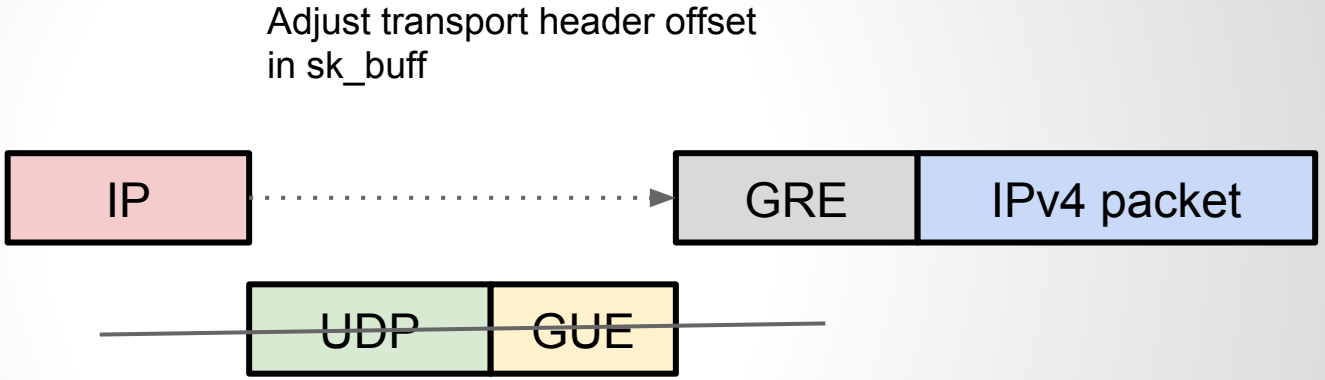
GRE in GUE transmit

Add Ethernet and IP headers and send

GRE in GUE receive

Process packet based on UDP port (GUE port)

GRE in GUE receive



Remove UDP/GUE headers

GRE in GUE receive

Now have original GRE/IP packet. Reinject this into kernel, next protocol to prcess is 47 (GRE)

Thanks, and looking forward

- Good support for UDP encapsulation is the result of a broad community effort
- Still a lot of intersting work to do in security, control, and performance