Stacked Vlan

- Performance Improvement and Challenges

Toshiaki Makita
NTT Open Source Software Center

Today's topics

- Stacked vlan
- Performance Improvement and Challenges
- Interoperability Problem

Who is Toshiaki Makita?

- **Linux kernel engineer at NTT Open Source Software Center**

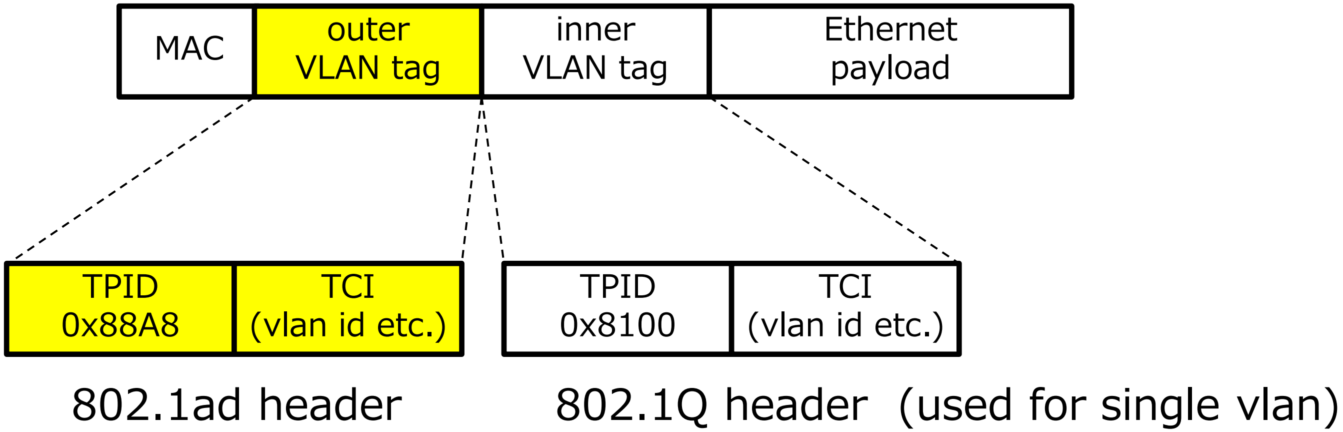
- **Providing technical support for NTT group companies**

- **Active patch submitter on kernel networking subsystem**
  - bridge, vlan, etc.

Stacked Vlan

What is stacked vlan?

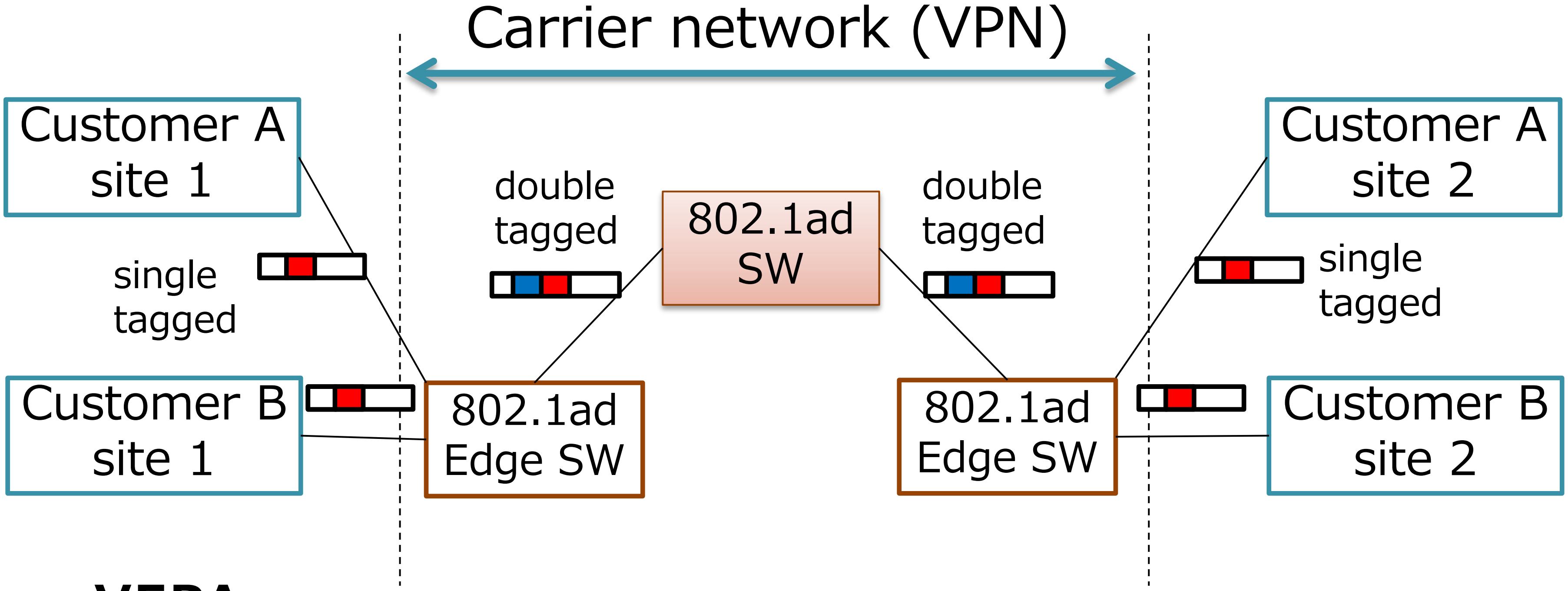
- **Stacked vlan:**
  - Two (or more) vlan tags in packets



- Note: sometimes 802.1Q is also used for outer tag
  - Stacked vlan != 802.1ad

Where is stacked vlan used?

- **Ethernet VPN (Metro Ethernet)**
  - Outer tag is used to separate customers
  - Allow customers to use vlan (i.e. inner vlan) through VPN

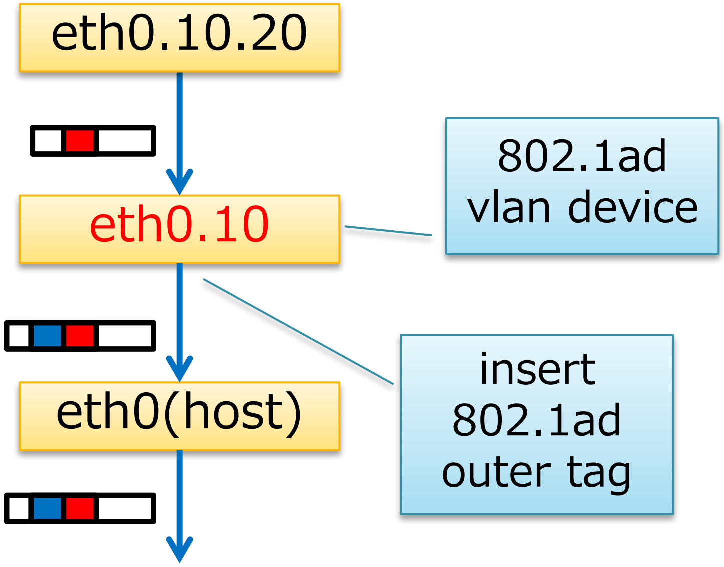


- **VEPA**
  - Offload packet-forwarding between VMs to external switch
  - Use 802.1ad to separate VMs

802.1ad in Linux

- **4 ways to use 802.1ad**
  - 802.1ad vlan device
    - since kernel 3.10
  - 802.1ad vlan-aware bridge
    - since kernel 3.16
  - openvswitch
    - still in net-next
  - 802.1ad capable SR-IOV device
    - still in net-next
    - mlx4_en

e.g. 802.1ad vlan device



Performance Improvement and Challenges

Features for stacked vlan

- **802.1ad was introduced in kernel 3.10**
- **At that time, acceleration features were…**
  - Normal 802.1Q vlan device on 3.10

```
# ethtool -k ens1f0.10
tx-checksumming: on
scatter-gather: on
tcp-segmentation-offload: on
generic-segmentation-offload: on
```

  - Stacked vlan device (802.1Q on 802.1ad) on 3.10

```
# ethtool -k ens1f0.10.20
tx-checksumming: off
scatter-gather: off
tcp-segmentation-offload: off
generic-segmentation-offload: off
```

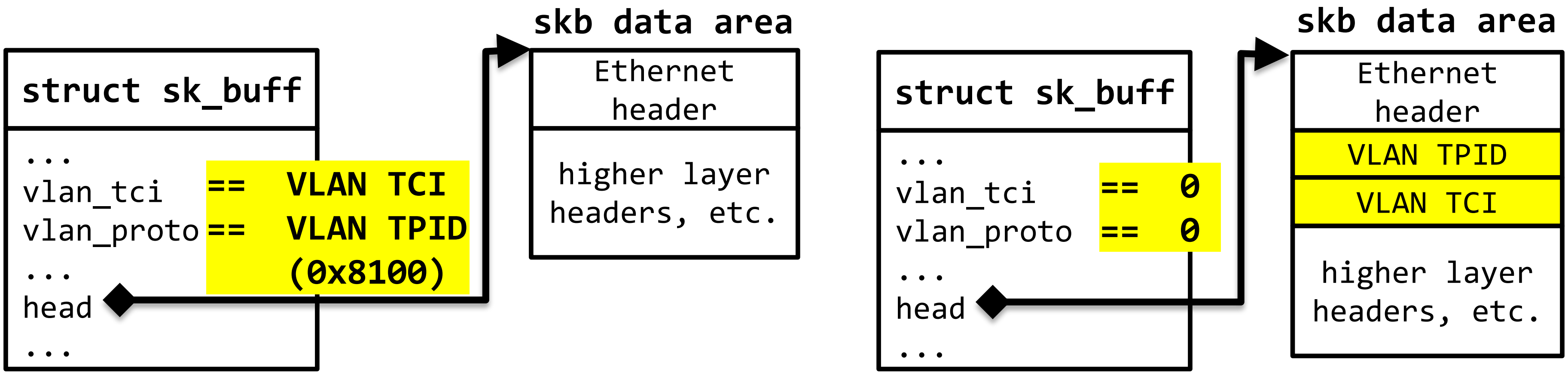
Features for stacked vlan

- Few features because of missing vlan_features of vlan device

- Let's enable the features then, but...

Assumptions about vlan packets

- **Data structure of VLAN (Normal 802.1Q packet)**



in kernel network stack or
in drivers with vlan-tag offload enabled

struct sk_buff

...
vlan_tci == VLAN TCI
vlan_proto == VLAN TPID (0x8100)
...
head
...

skb data area

Ethernet header

higher layer headers, etc.

in drivers with vlan-tag offload disabled

struct sk_buff

...
vlan_tci == 0
vlan_proto == 0
...
head
...

skb data area

Ethernet header

VLAN TPID

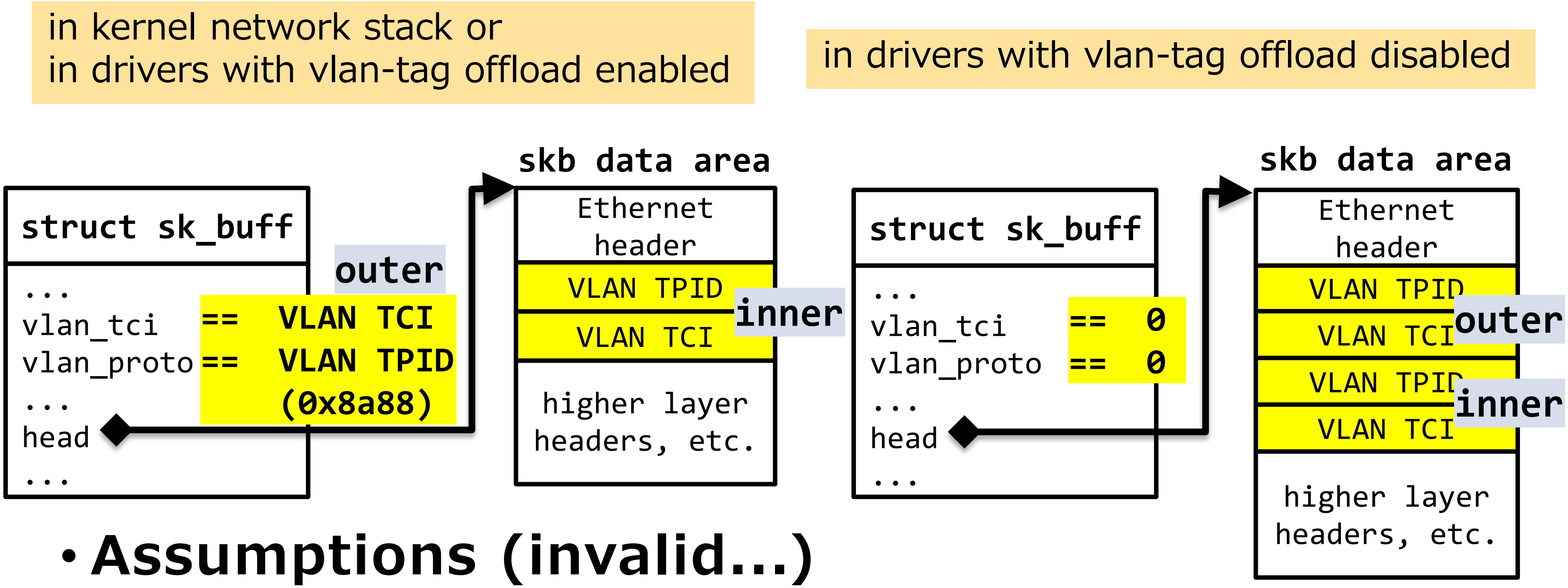
VLAN TCI

higher layer headers, etc.

- **Assumptions (valid in most cases)**
  - kernel network stack handles only vlan-tag stripped skb
  - there is at most one vlan tag in skb data
  - VLAN TPID is 0x8100

Assumptions about stacked vlan packets

- Data structure of VLAN (802.1Q in 802.1ad packet)



- Assumptions (invalid...)
  - ~~kernel network stack handles only vlan-tag stripped skb~~
  - ~~there is at most one vlan tag in skb data~~
  - ~~VLAN TPID is 0x8100~~

Assumptions on stacked vlan packets

- **Some examples of breaking assumptions**
  - Tx-Checksum of IP_CSUM devices

Example: Tx-checksum of IP_CSUM

- Example (igb) of Tx IP_CSUM in kernel 3.10

```
netdev_tx_t igb_xmit_frame_ring(...)
         ...
         __be16 protocol = vlan_get_protocol(skb);
         ...
         first->protocol = protocol;
         ...
                  igb_tx_csum(tx_ring, first);

static void igb_tx_csum(...)
         ...
                  switch (first->protocol) {
                  case __constant_htons(ETH_P_IP):
                  ...
                  default:
                           if (unlikely(net_ratelimit())) {
                                    dev_warn(tx_ring->dev,
                                             "partial checksum but proto=%x!¥n",
                                             first->protocol);
                           }
```

try to get network protocol

packet gets corrupted
if protocol is not IPv4/IPv6

Tx-checksum of IP_CSUM

- vlan_get_protocol() in kernel 3.10

```
static inline __be16 vlan_get_protocol(const struct sk_buff *skb)
{
         __be16 protocol = 0;

         if (vlan_tx_tag_present(skb) ||
              skb->protocol != cpu_to_be16(ETH_P_8021Q))
                   protocol = skb->protocol;
         else {
                   __be16 proto, *protop;
                   protop = skb_header_pointer(skb, offsetof(struct vlan_ethhdr,
                                                      h_vlan_encapsulated_proto),
                                                      sizeof(proto), &proto);

                   if (likely(protop))
                             protocol = *protop;
         }

         return protocol;
}
```

handle only 0x8100

handle at most one vlan tag in skb data

- **So igb corrupted stacked vlan packets...**
- **Fixed in 3.19**
  - so vlan_get_protocol() can handle any number of vlan tag

Tx-checksum of IP_CSUM

- Another example (ixgbe) of Tx IP_CSUM in 3.10

```
netdev_tx_t ixgbe_xmit_frame_ring(...)
        ...
        __be16 protocol = skb->protocol;
        ...
        if (vlan_tx_tag_present(skb)) {
                ...
        } else if (protocol == __constant_htons(ETH_P_8021Q)) {
                struct vlan_hdr *vhdr, _vhdr;
                vhdr = skb_header_pointer(skb, ETH_HLEN, sizeof(_vhdr), &_vhdr);
                if (!vhdr)
                        goto out_drop;

                protocol = vhdr->h_vlan_encapsulated_proto;
        ...
        first->protocol = protocol;
        ...
                ixgbe_tx_csum(tx_ring, first);
```

handle only 0x8100

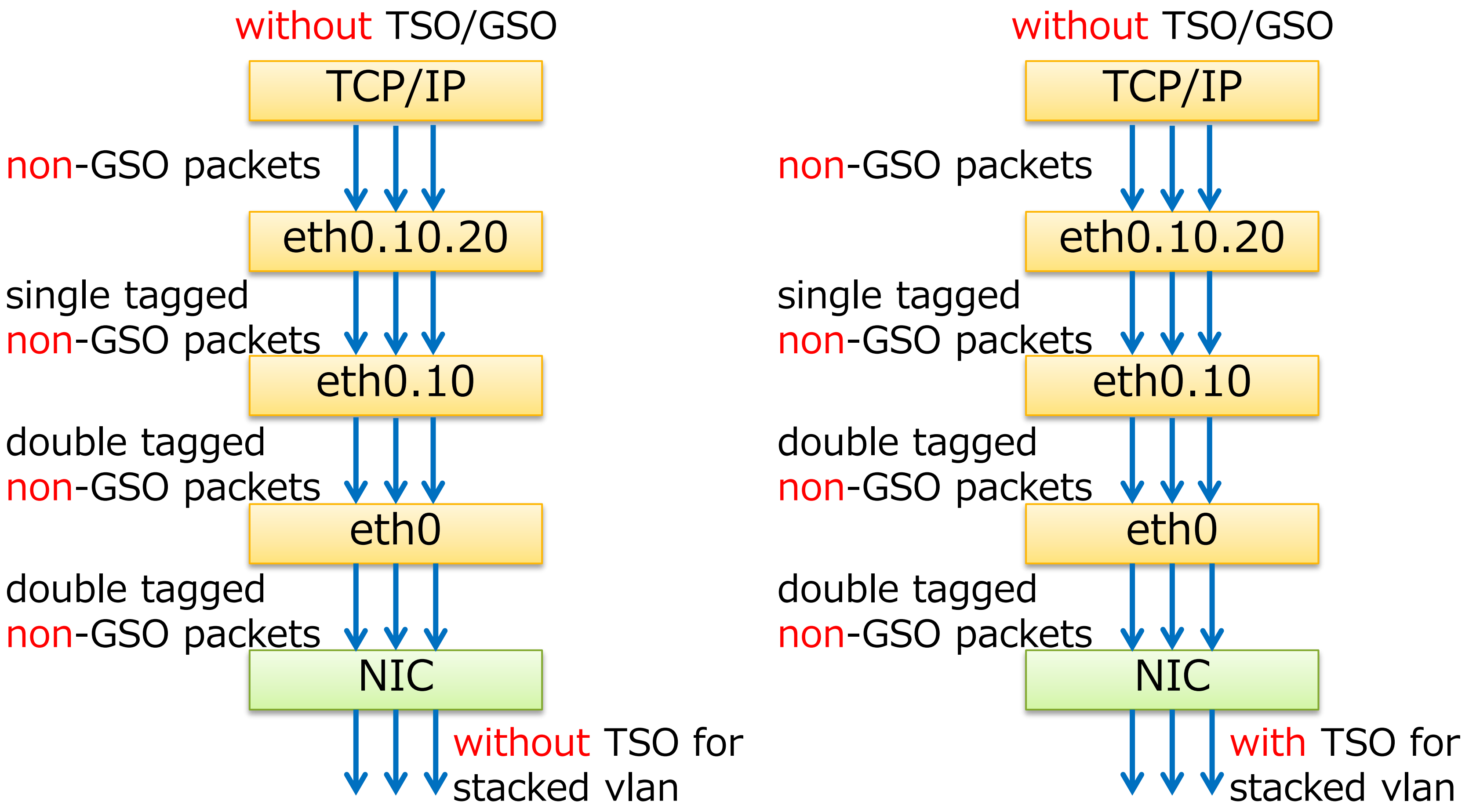
handle at most one vlan tag in skb data

- Fixed in 3.19 (using vlan_get_protocol())
- Intel drivers are just examples. All IP_CSUM drivers should be careful with this failure

Missing key infrastructure for stacked vlan

- **TSO**
  - could not be performed for stacked vlan
  - No in-kernel infrastructure to determine if device can segment stacked vlan packets

TSO/GSO in kernel 3.10



without TSO/GSO

TCP/IP

non-GSO packets

eth0.10.20

single tagged
non-GSO packets

eth0.10

double tagged
non-GSO packets

eth0

double tagged
non-GSO packets

NIC

without TSO for
stacked vlan

without TSO/GSO

TCP/IP

non-GSO packets

eth0.10.20

single tagged
non-GSO packets

eth0.10

double tagged
non-GSO packets

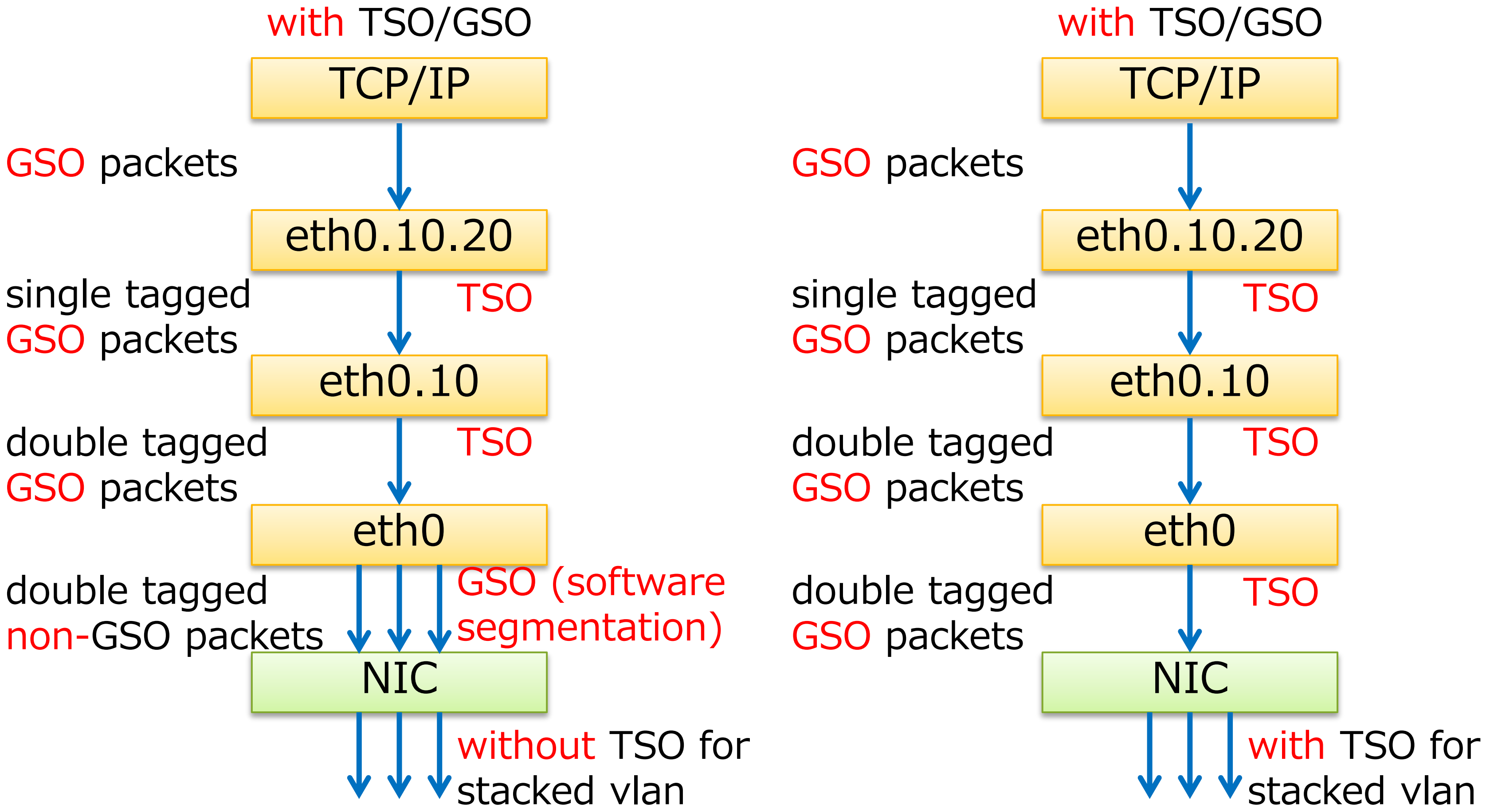
eth0

double tagged
non-GSO packets

NIC

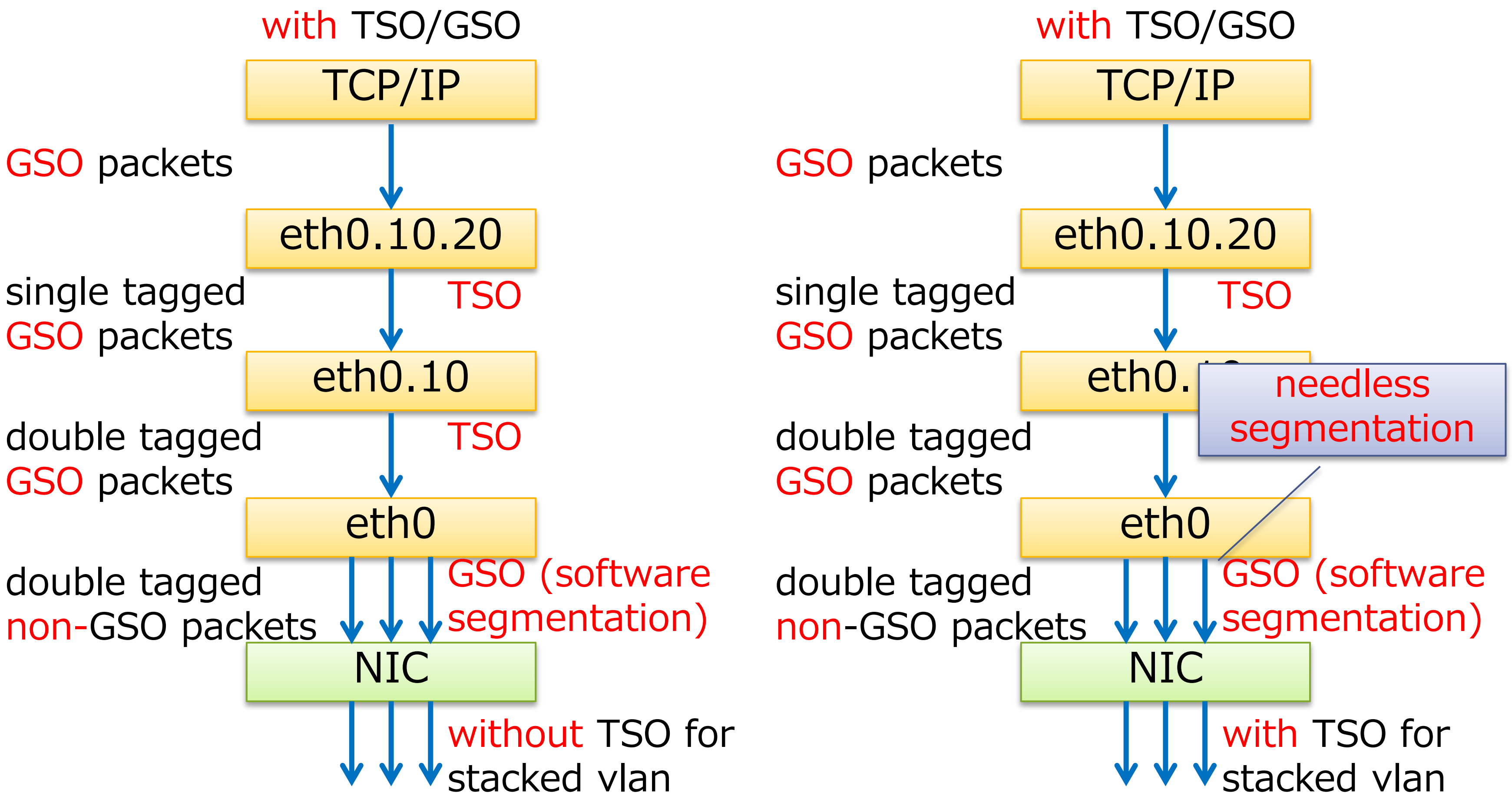
with TSO for
stacked vlan

TSO/GSO in ideal world



TSO/GSO in kernel 3.10 (with TSO enabled)

- **Added TSO bit on eth0.10 & eth0.10.20**


with TSO/GSO
TCP/IP
GSO packets
eth0.10.20
single tagged GSO packets
TSO
eth0.10
double tagged GSO packets
TSO
eth0
double tagged non-GSO packets
GSO (software segmentation)
NIC
without TSO for stacked vlan
with TSO/GSO
TCP/IP
GSO packets
eth0.10.20
single tagged GSO packets
TSO
eth0.10
needless segmentation
double tagged GSO packets
eth0
double tagged non-GSO packets
GSO (software segmentation)
NIC
with TSO for stacked vlan

TSO for stacked vlan

- **How to determine if NIC is capable of TSO**

```
netdev_features_t netif_skb_features(...)
         ...
-        if (skb_vlan_tagged_multi(skb))
-                 features = netdev_intersect_features(features,
-                                                       NETIF_F_SG |
-                                                       NETIF_F_HIGHDMA |
-                                                       ...
         ...
         if (dev->netdev_ops->ndo_features_check)
                  features &= dev->netdev_ops->ndo_features_check(skb, dev,
                                                                   features);
+        else
+                 features &= dflt_features_check(skb, dev, features);
         ...
+static netdev_features_t dflt_features_check(...)
+        return vlan_features_check(skb, features);
```

had disabled TSO
just because of double tag

move into

Disable TSO only if drivers do not
have ndo_features_check()

- **Introduced in 4.1**

TSO for stacked vlan

- **Example of TSO-capable drivers (igb)**

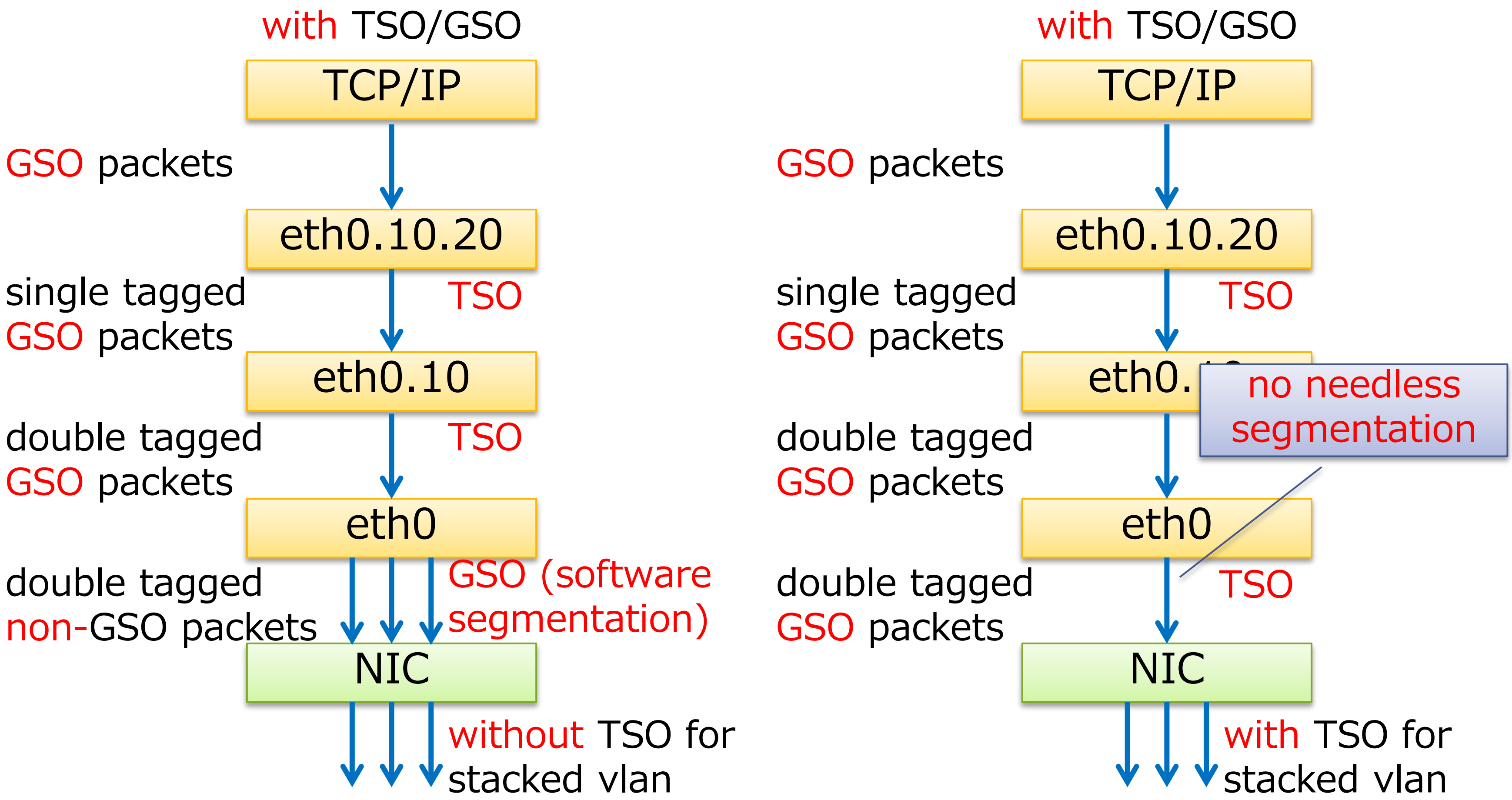
```
static const struct net_device_ops igb_netdev_ops = {
         ...
+        .ndo_features_check         = passthru_features_check,

+netdev_features_t passthru_features_check(...)
+{
+        return features;
+}
```

just for skipping vlan checking

TSO for stacked vlan

- **Everything works**


with TSO/GSO
TCP/IP
GSO packets
eth0.10.20
single tagged GSO packets
TSO
eth0.10
double tagged GSO packets
TSO
eth0
double tagged non-GSO packets
GSO (software segmentation)
NIC
without TSO for stacked vlan
with TSO/GSO
TCP/IP
GSO packets
eth0.10.20
single tagged GSO packets
TSO
eth0.
no needless segmentation
double tagged GSO packets
eth0
double tagged GSO packets
TSO
NIC
with TSO for stacked vlan

TSO for stacked vlan

- **If your driver does not support TSO for stacked vlan but implements ndo_features_check()...**

- **Example**

```
static netdev_features_t xxx_features_check(...)
{
        features = vlan_features_check(skb, features);
        ...
        return features;
}
```

make sure to call vlan_features_check()

Rx side?

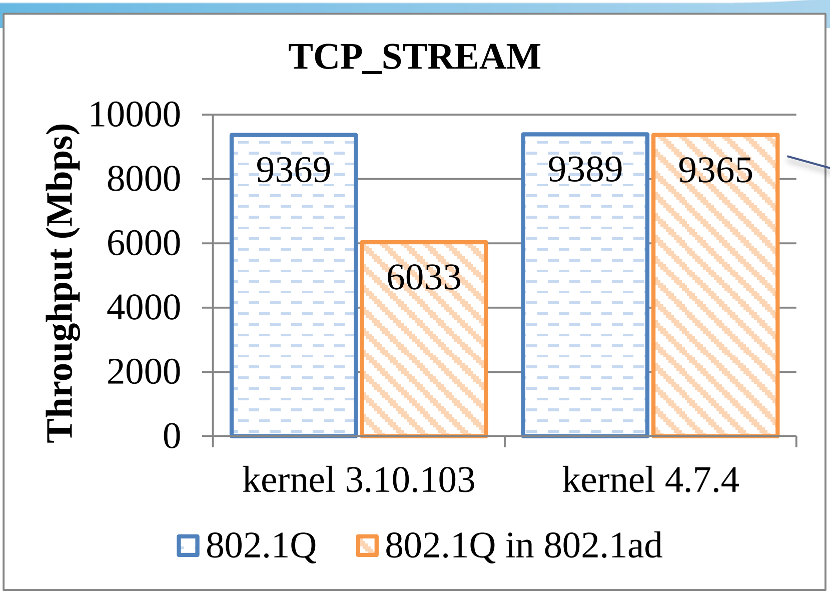
- **Rx-checksum IP_CSUM**
- **RSS**
  - totally depend on devices' capability to parse vlan

- **RPS**
- **GRO**
  - did not work in kernel 3.10 but now fixed

Performance change

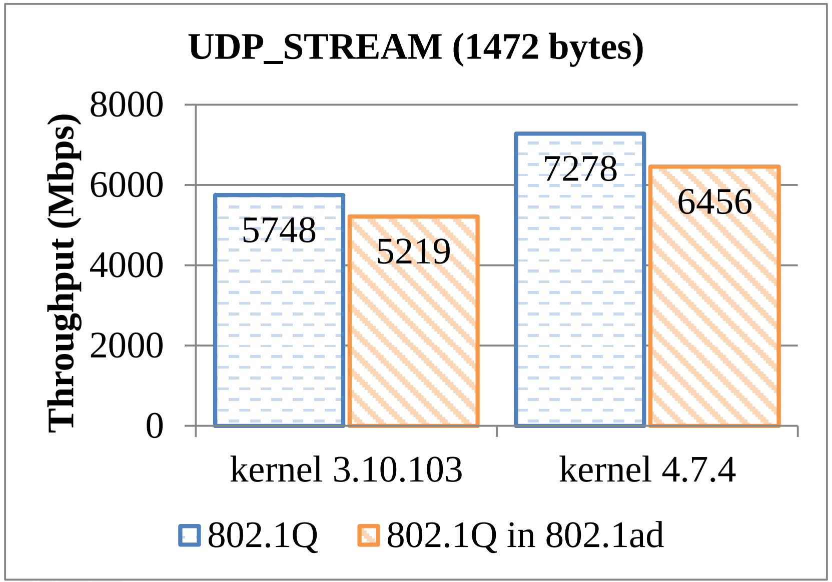
- **kernel**
  - 3.10.103
  - 4.7.4
- **Test environment**
  - 2 machines connected back to back
  - CPU: Intel Xeon E5-2650v3, 2.30GHz, 2-socket, 10-core each
  - NIC: Intel 82599ES 10-Gigabit (ixgbe)
    - Tx-checksum/TSO-capable for stacked vlan
    - Cannot Rx-checksum/RSS for stacked vlan
- **Tool**
  - netperf
  - super_netperf

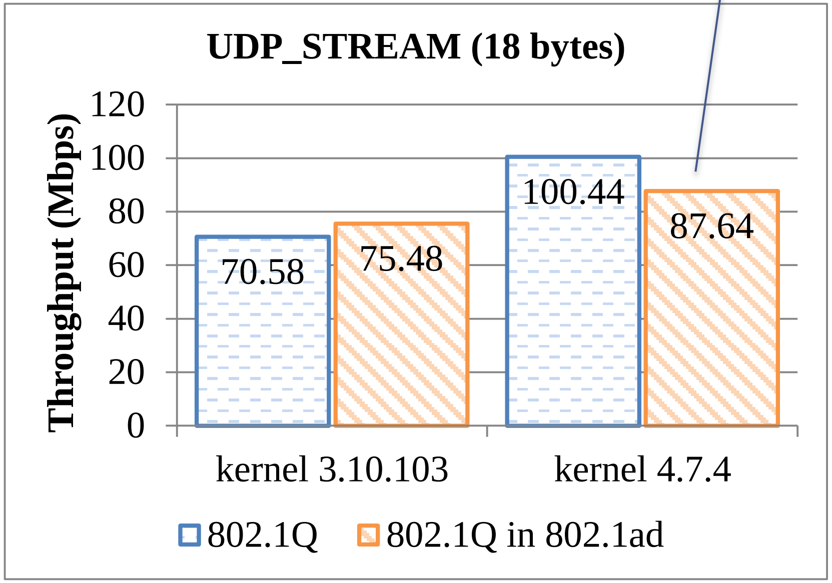
Performance change (Single flow)



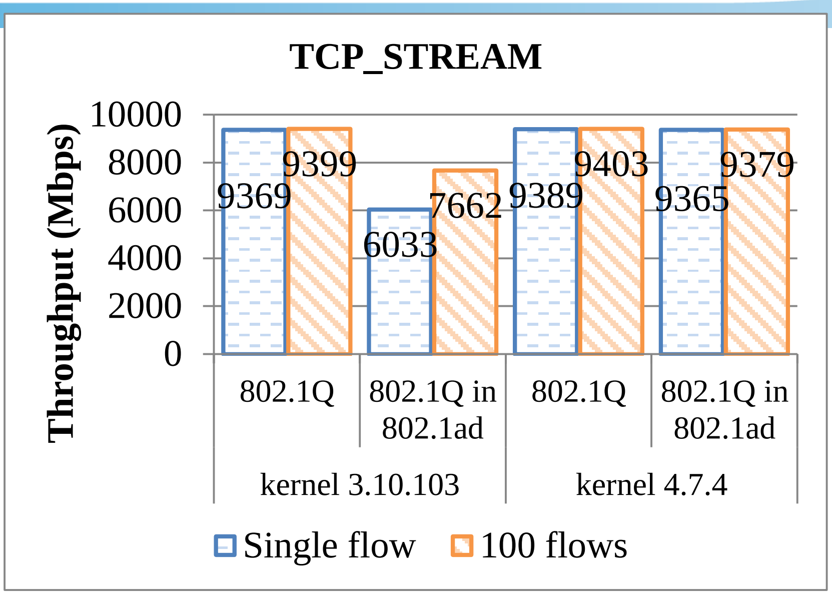
10G wire speed with single core

improved but worse than single vlan



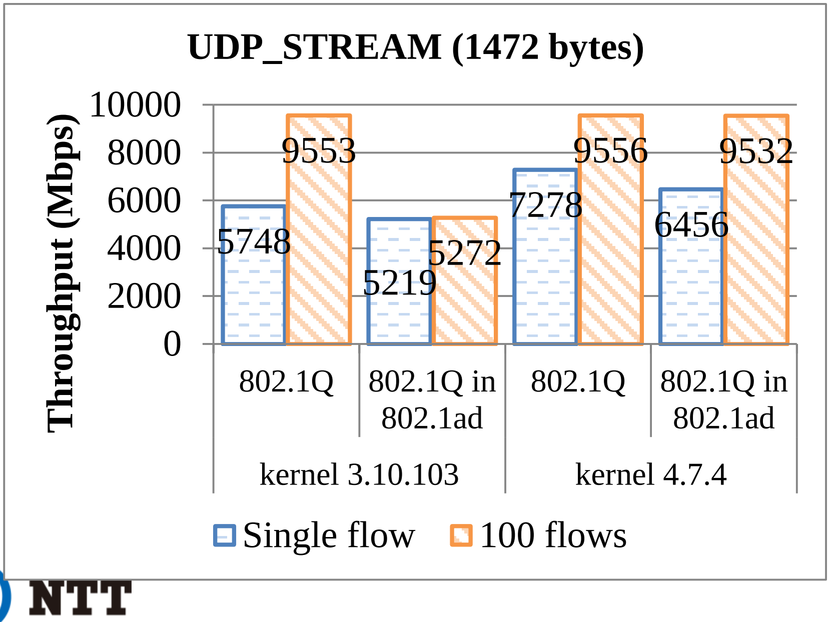


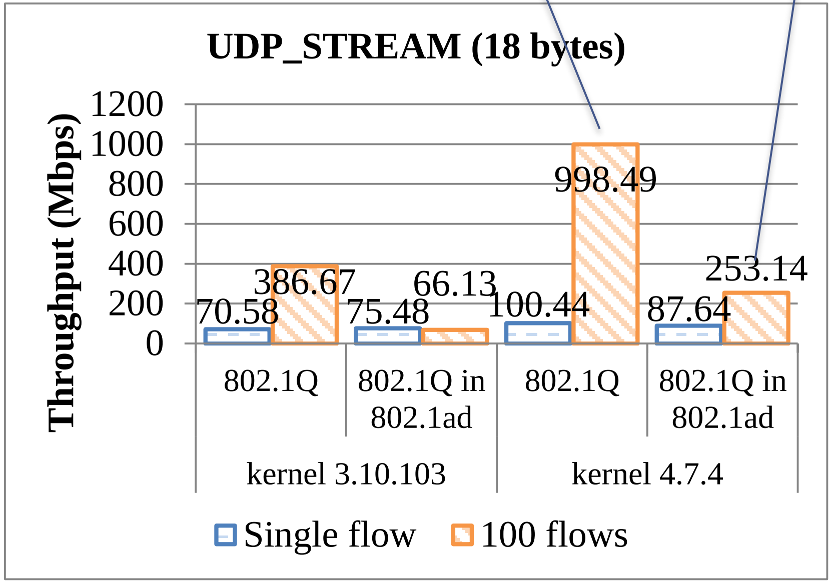
Performance change (Multi flow)



scales by RSS

scales by RPS





Summary for features improvement

	Feature		kernel 3.10	kernel 4.7
Tx	Tx-Checksum	HW_CSUM	✗	✓
		IP_CSUM	✗	depends on drivers
	GSO		✗	✓
	TSO		✗	depends on drivers
Rx	Rx-Checksum	HW_CSUM	✓	✓
		IP_CSUM	depends on devices	depends on devices
	RSS		depends on devices	depends on devices
	RPS		✗	✓
	GRO		✗	✓

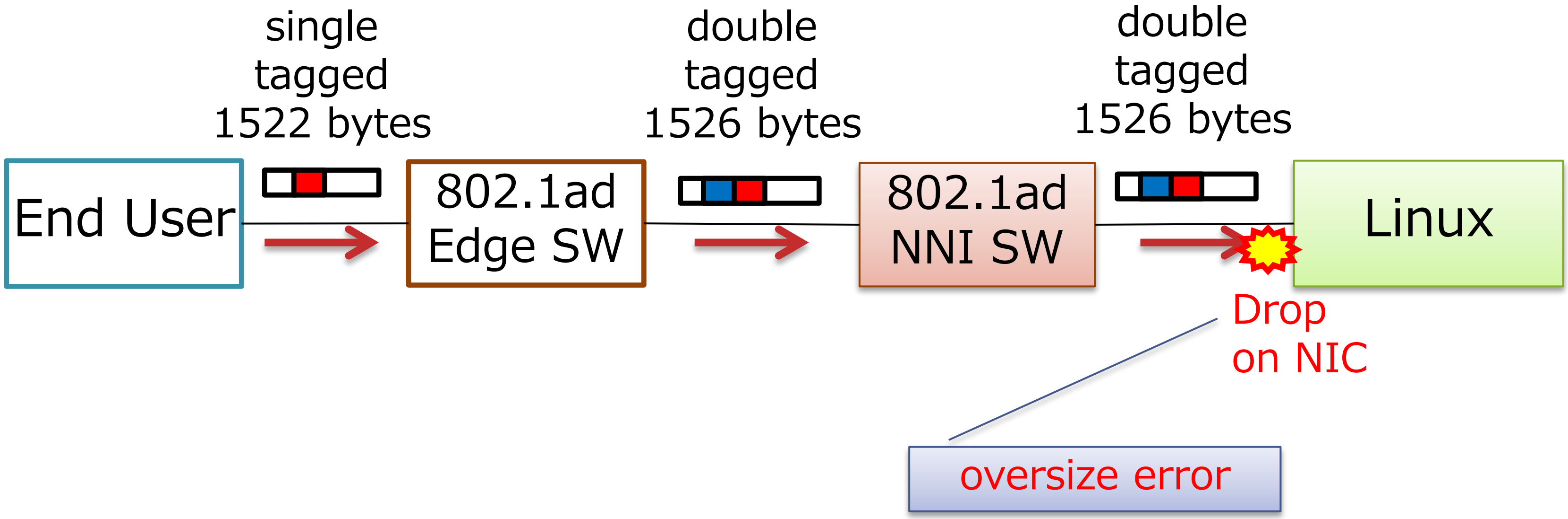
For further improvement

- **Tx-checksum IP_CSUM**
  - Some NICs still may corrupt double tagged packets
  - Use vlan_get_protocol() in drivers if possible
- **TSO**
  - Implement ndo_features_check(), if your driver is capable of performing TSO on stacked vlan packets
- **Rx-checksum IP_CSUM, RSS**
  - Some NICs like intel 10G have a feature to enter "stacked vlan mode", which can parse only stacked vlan packets
  - Need a feature to turn it on

Interoperability problem

Size of vlan packets

- **Single tagged packets have 1522 bytes**
  - Most NICs accept them
- **Double tagged packets have 1526 bytes**
  - Some NICs do not accept them
  - This size is needed to provide transparent Ethernet VPN



Examples from our lab

- **e1000e**
  - Drop packets > 1522 bytes
- **bnx2x, ixgbe**
  - Drop 802.1Q vlan-tagged packets > 1522 bytes
  - Drop other packets > 1518 bytes
- **mlx4_en, sfc**
  - Accept packets <= 1526 (+α) bytes
  - Drop packets > 1526 (+α) bytes
    - take into account double tag or alignment restriction
- **igb**
  - Accept packets <= 9xxx bytes
  - Drop packets > 9xxx bytes
    - Accept jumbo frames by default

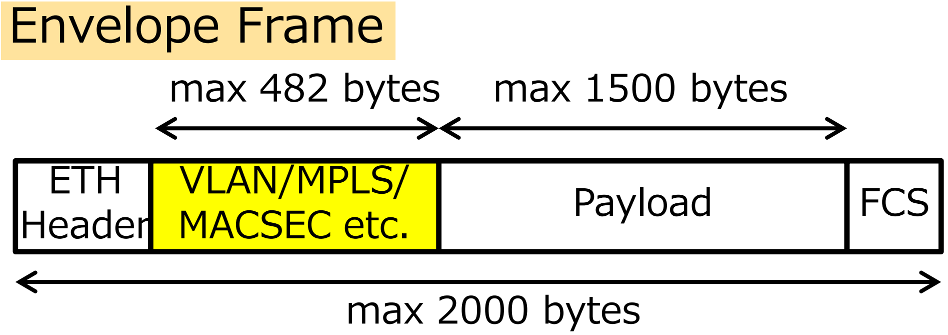
Approaches

- **Reduce vlan device MTU?**
  - Does not change size of received packet
- **Increase real device MTU?**
  - OK, but this allows untagged jumbo frames
- **Accept larger sized packets by default?**
  - Like igb
  - Some NICs change their behavior when accepting larger size
    - e1000e, qlge, etc.

Approaches: Envelope frames

- **Introduce envelope frames (802.3as)**

- **IEEE 802.3-2012 says:**
  - A MAC frame that carries a Length/Type field with the Type interpretation that may indicate additional encapsulation information within the MAC client data and has a maximum length of 2000 octets. The envelope frame is intended to allow inclusion of additional prefixes and suffixes required by higher layer encapsulation protocols. The encapsulation protocols may use up to 482 octets.



Approaches: Envelope frames

- **Envisioned implementation**
  - Introduce variable max envelope header length

```
# ip link set eth0 envhdrlen 8
```

  - Expand acceptable packet size of NIC by envhdrlen
    - Replace dev->mtu with dev->mtu + envhdrlen in drivers
  - Do not change dev->mtu
  - RFC posted
    - http://marc.info/?t=147496691500005&r=1&w=2
    - http://marc.info/?t=147496691500003&r=1&w=2
    - http://marc.info/?t=147496691500002&r=1&w=2
    - http://marc.info/?t=147496691500004&r=1&w=2
    - http://marc.info/?t=147496691500001&r=1&w=2
- **In the future**
  - Inform real device drivers of envhdrlen on creating 802.1ad vlan device

Summary

- **Performance**
  - Improved since 3.10
  - TCP reaches 10G with 1 core
  - Need more work in drivers for further improvement

- **Interoperability**
  - Stacked vlan packets get dropped due to oversize
  - Approach: Envelope frames
    - RFC posted

Thank you!