NICs, offloading, IPv6, mbufs, and checksum (TODO ideas)

or: how to simplify, enhance and make faster

FreeBSD DevSummit along BSDCan 2012-05

Bjoern A. Zeeb <bz@FreeBSD.org>

new mbuf fields

- For more effective offload support we want the following (reduce cache misses, less code duplication, ...):
  - L2 offset
  - L3 offset
  - L4 offset
  - L5+?
- mbuf functions should automatically adjust these as we prepend, etc.

L2 offset

- TX: find the link-layer header
- RX: fill in the link-layer header

- Remember PPP(oEoA) still exists.
- On Ethernet etc. make it easy to handle stacked VLANs (QinQ), etc.
- Allows up-in-the-stack to access it more easily (think 802.1p PCP, ..)

L3 offset

- TX: find the IP header in drivers.
- RX: NICs can provide that offset already, makes LRO, etc easier.
- Consider tunneling protocols (IPIP, IPsec, ..), esp on TX (where to point to?)

L4 offset

- TX: find the upper layer protocol (ULP) header in drivers.
- RX: NICs can provide that offset already.
- Can help LRO performance, etc.
- Makes it easier to skip a lot of IPv6 extension headers.

L5+ offset ?

- What if I know the stream and/or do not care about all the things below my payload?
- Maybe not a lot today but a lot more in the future?

Why on TX? See:

```
+       if (eh->evl_encap_proto == htons(ETHERTYPE_VLAN)) {
                ehdrlen = ETHER_HDR_LEN + ETHER_VLAN_ENCAP_LEN;
-       else
+               eh_type = eh->evl_proto;
+       } else {
                ehdrlen = ETHER_HDR_LEN;
+               eh_type = eh->evl_encap_proto;
+       }
-        if (mp->m_len < ehdrlen + sizeof(struct ip) + sizeof(struct tcphdr))
-               return FALSE;
+       len = ehdrlen + sizeof(struct tcphdr);
+       switch (ntohs(eh_type)) {
+#ifdef INET6
+       case ETHERTYPE_IPV6:
+               if (mp->m_len < len + sizeof(struct ip6_hdr))
+                       return FALSE;
+               ip6 = (struct ip6_hdr *)(mp->m_data + ehdrlen);
+               /* XXX-BZ For now we do not pretend to support ext. hdrs. */
+               if (ip6->ip6_nxt != IPPROTO_TCP)
+                       return FALSE;
+               ip_hlen = sizeof(struct ip6_hdr);
+               th = (struct tcphdr *)((caddr_t)ip6 + ip_hlen);
+               th->th_sum = in6_cksum_pseudo(ip6, 0, IPPROTO_TCP, 0);
+               type_tucmd_mlhl |= IXGBE_ADVTXD_TUCMD_IPV6;
+               break;
+#endif
+#ifdef INET
+       case ETHERTYPE_IP:
+               if (mp->m_len < len + sizeof(struct ip))
+                       return FALSE;
+               ip = (struct ip *)(mp->m_data + ehdrlen);
+               if (ip->ip_p != IPPROTO_TCP)
+                       return FALSE;
+               ip->ip_sum = 0;
+               ip_hlen = ip->ip_hl << 2;
+               th = (struct tcphdr *)((caddr_t)ip + ip_hlen);
+               th->th_sum = in_pseudo(ip->ip_src.s_addr,
+                   ip->ip_dst.s_addr, htons(IPPROTO_TCP));
+               type_tucmd_mlhl |= IXGBE_ADVTXD_TUCMD_IPV4;
+               /* Tell transmit desc to also do IPv4 checksum. */
+               *olinfo_status |= IXGBE_TXD_POPTS_IXSM << 8;
+               break;
+#endif
+       default:
+               panic("%s: CSUM_TSO but no supported IP version (0x%04x)",
+                   __func__, ntohs(eh_type));
+               break;
+       }
```

Why oh RX? See:

```
+       eh = mtod(m, struct ether_header *);
+       eh_type = ntohs(eh->ether_type);
+       switch (eh_type) {
+#ifdef INET6
+       case ETHERTYPE_IPV6:
+               l3hdr = ip6 = (struct ip6_hdr *)(eh + 1);
+               error = tcp_lro_rx_ipv6(lc, m, ip6, &th);
+               if (error != 0)
+                       return (error);
+               tcp_data_len = ntohs(ip6->ip6_plen);
+               ip_len = sizeof(*ip6) + tcp_data_len;
+               break;
+#endif
+#ifdef INET
+       case ETHERTYPE_IP:
+               l3hdr = ip4 = (struct ip *)(eh + 1);
+               error = tcp_lro_rx_ipv4(lc, m, ip4, &th);
+               if (error != 0)
+                       return (error);
+               ip_len = ntohs(ip4->ip_len);
+               tcp_data_len = ip_len - sizeof(*ip4);
+               break;
+#endif
+       /* XXX-BZ what happens in case of VLAN(s)? */
+       default:
+               return (TCP_LRO_NOT_SUPPORTED);
        }
```

"IFCAP6exthdr"

- Offloading and IPv6 fragmentation.

  more general:

- Offloading and IPv6 extension headers.
- Currently no way to know which IPv6 extension headers each NIC supports, thus have to always do in software. Need per-NIC meta-information in the upper stack.

"Call-Through"

(gibbs)

- Assume virtualization, a bridge, a physical interface and offloading.
- Can we make the first-in-the-chain know the details about the last-in-the-chain to know if we could use hardware offloading, ..., etc?

LRO & fwding/bridging

(gallatin)

- Problem: LRO/TSO extra work; resegmentation, what if not TSO?
- Simple: forwarding on -> LRO off.
- More advanced: improve LRO code.

deferred cksums

(bz+pyun)

- Some drivers require "special" pseudo-hdr cksums. We duplicate the work.
- Move it all into a library.
- Can we do it down the stack (in the driver) more easily without too much cache hits?
- Or: do we need flags on which to calculate in upper layers?

checksums

- We need a better set of (library of) checksum manipulation functions.
- We have manually rolled-out code.
- We have in_cksum and in6_cksum.
- We have LRO, NAT, ... who want to do partial checksum updates as do delayed checksum, pseudohdr calculaltions, ...
- Would MD code help performance?