Faster packet modification

Maxim Kharchenko, Cloudozer LLP

24/02/2014

1 Summary

- Packet modification as reimplemented in linc_max backend is 10x faster;
- Packet modification adds $1\mu s$ to the processing delay.

2 Status quo

The standard LINC switch backend — linc_us4 — does not distringuish between packet-modifying and simple forwarding flows. In linc_us4, each incoming packet undergoes a complete decapsulation. The matching machinery and actions operate on the `record` representation of the packet. The packet gets encapsulated just before exiting the switch. This makes a packet-modifying flow as fast (as slow) as a flow that does simple forwarding.

The decapsulation/encapsulation is a costly operation. Its latency cost was estimated at $4.1/27.0\mu s$.

3 Preparser and splicer

The linc_max backend matches packet without decapsulation. It does just enough parsing of an incoming packet to be able to apply the matching rules. A new module — linc_max_preparser — does the initial packet parsing.

The representation of the packet produced by the preparser is not suitable for Set-Field actions. The packet modification is based on a new module — linc_max_splicer. The splicer is optimized to keep amount of copying to the minimum.

4 Splicer capabilities and speed

The Table 1 summarizes capabilities and latency costs of the splicer. Some fields cannot be modified using the splicer because doing so will corrupt the packet. The example is the `ip_proto` field. Such fields are marked `protected`.

As mentioned earlier, the latency depends on necessity to recalculate checksums when the packet changes. The Ethernet header does not have a checksum, and its modification costs about 400ns. The IPv4 header modification is costlier, $1.6\mu s$, because it has a checksum.

The modification of TCP/UDP headers may require even more time because of the length of the data that require checksumming. SCTP is especially costly because it uses a non-trivial algorithm (CRC32).

Table 1: Splicer capabilities

Field	Support	Delay
eth_dst	yes	0.41μs
eth_src	yes	0.35μs
vlan_vid	yes	-
vlan_pcp	yes	-
ip_dscp	yes	1.65μs
ip_ecn	yes	1.56μs
ip_proto	protected	-
ipv4_src	yes	1.59μs
ipv4_dst	yes	1.62μs
tcp_src	yes	-
tcp_dst	yes	-
udp_src	yes	-
udp_dst	yes	-
sctp_src	yes	-
sctp_dst	yes	-
icmpv4_type	protected	-
icmpv4_code	protected	-
arp_op	yes	-
arp_spa	yes	-
arp_tpa	yes	-
arp_sha	yes	-
arp_tha	yes	-
ipv6_src	yes	-
ipv6_dst	yes	-
ipv6_flabel	yes	-
icmpv6_type	protected	-
icmpv6_code	protected	-
ipv6_nd_target	yes	-
ipv6_nd_sll	unimplemented	-
ipv6_nd_tll	unimplemented	-
mpls_label	yes	-
mpls_tc	yes	-
mpls_bos	protected	-
pbb_isid	yes	-
pbb_uca	yes	-