Generic XDP

From: David Miller <davem-AT-davemloft.net>
To: netdev-AT-vger.kernel.org
Subject: [PATCH v3 net-next RFC] Generic XDP
Date: Wed, 12 Apr 2017 14:54:15 -0400 (EDT)
Message-ID: <20170412.145415.1441440342830198148.davem@davemloft.net>
Cc: xdp-newbies-AT-vger.kernel.org
Archive-link: Article

```
This provides a generic SKB based non-optimized XDP path which is used
if either the driver lacks a specific XDP implementation, or the user
requests it via a new IFLA_XDP_FLAGS value named XDP_FLAGS_SKB_MODE.

It is arguable that perhaps I should have required something like
this as part of the initial XDP feature merge.

I believe this is critical for two reasons:

1) Accessibility.  More people can play with XDP with less
   dependencies.  Yes I know we have XDP support in virtio_net, but
   that just creates another depedency for learning how to use this
   facility.

   I wrote this to make life easier for the XDP newbies.

2) As a model for what the expected semantics are.  If there is a pure
   generic core implementation, it serves as a semantic example for
   driver folks adding XDP support.

This is just a rough draft and is untested.

One thing I have not tried to address here is the issue of
XDP_PACKET_HEADROOM, thanks to Daniel for spotting that.  It seems
incredibly expensive to do a skb_cow(skb, XDP_PACKET_HEADROOM) or
whatever even if the XDP program doesn't try to push headers at all.
I think we really need the verifier to somehow propagate whether
certain XDP helpers are used or not.

Signed-off-by: David S. Miller <davem@davemloft.net>
--

v3:
 - Make sure XDP program sees packet at MAC header, push back MAC
   header if we do XDP_TX.  (Alexei)
 - Elide GRO when generic XDP is in use.  (Alexei)
 - Add XDP_FLAG_SKB_MODE flag which the user can use to request generic
   XDP even if the driver has an XDP implementation.  (Alexei)
 - Report whether SKB mode is in use in rtnl_xdp_fill() via XDP_FLAGS
   attribute.  (Daniel)

v2:
 - Add some "fall through" comments in switch statements based
   upon feedback from Andrew Lunn
 - Use RCU for generic xdp_prog, thanks to Johannes Berg.

diff --git a/include/linux/netdevice.h b/include/linux/netdevice.h
index b0aa089..071a58b 100644
```

```
--- a/include/linux/netdevice.h
+++ b/include/linux/netdevice.h
@@ -1891,9 +1891,17 @@ struct net_device {
        struct lock_class_key   *qdisc_tx_busylock;
        struct lock_class_key   *qdisc_running_key;
        bool                    proto_down;
+       struct bpf_prog __rcu   *xdp_prog;
 };
 #define to_net_dev(d) container_of(d, struct net_device, dev)

+static inline bool netif_elide_gro(const struct net_device *dev)
+{
+       if (!(dev->features & NETIF_F_GRO) || dev->xdp_prog)
+               return true;
+       return false;
+}
+
 #define        NETDEV_ALIGN            32

 static inline
diff --git a/include/uapi/linux/if_link.h b/include/uapi/linux/if_link.h
index 8b405af..633aa02 100644
--- a/include/uapi/linux/if_link.h
+++ b/include/uapi/linux/if_link.h
@@ -887,7 +887,9 @@ enum {
 /* XDP section */

 #define XDP_FLAGS_UPDATE_IF_NOEXIST    (1U << 0)
-#define XDP_FLAGS_MASK                 (XDP_FLAGS_UPDATE_IF_NOEXIST)
+#define XDP_FLAGS_SKB_MODE             (2U << 0)
+#define XDP_FLAGS_MASK                 (XDP_FLAGS_UPDATE_IF_NOEXIST | \
+                                        XDP_FLAGS_SKB_MODE)

 enum {
        IFLA_XDP_UNSPEC,
diff --git a/net/core/dev.c b/net/core/dev.c
index ef9fe60e..9ed4569 100644
--- a/net/core/dev.c
+++ b/net/core/dev.c
@@ -95,6 +95,7 @@
 #include <linux/notifier.h>
 #include <linux/skbuff.h>
 #include <linux/bpf.h>
+#include <linux/bpf_trace.h>
 #include <net/net_namespace.h>
 #include <net/sock.h>
 #include <net/busy_poll.h>
@@ -4247,6 +4248,88 @@ static int __netif_receive_skb(struct sk_buff *skb)
        return ret;
 }

+static struct static_key generic_xdp_needed __read_mostly;
+
+static int generic_xdp_install(struct net_device *dev, struct netdev_xdp *xdp)
+{
+       struct bpf_prog *new = xdp->prog;
+       int ret = 0;
+
+       switch (xdp->command) {
+       case XDP_SETUP_PROG: {
+               struct bpf_prog *old = rtnl_dereference(dev->xdp_prog);
+
+               rcu_assign_pointer(dev->xdp_prog, new);
+               if (old)
+                       bpf_prog_put(old);
+
+               if (old && !new)
+                       static_key_slow_dec(&generic_xdp_needed);
+               else if (new && !old)
+                       static_key_slow_inc(&generic_xdp_needed);
+               break;
+       }
+
+       case XDP_QUERY_PROG:
+               xdp->prog_attached = !!rcu_access_pointer(dev->xdp_prog);
+               break;
+
+       default:
+               ret = -EINVAL;
+               break;
+       }
+
+       return ret;
+}
+
+static u32 netif_receive_generic_xdp(struct sk_buff *skb,
+                                    struct bpf_prog *xdp_prog)
+{
+       struct xdp_buff xdp;
+       u32 act = XDP_DROP;
+       void *orig_data;
+       int hlen, off;
+
+       if (skb_linearize(skb))
+               goto do_drop;
+
+       /* The XDP program wants to see the packet starting at the MAC
+        * header.
+        */
+       hlen = skb_headlen(skb) + skb->mac_len;
+       xdp.data = skb->data - skb->mac_len;
+       xdp.data_end = xdp.data + hlen;
+       xdp.data_hard_start = xdp.data - skb_headroom(skb);
+       orig_data = xdp.data;
+
+       act = bpf_prog_run_xdp(xdp_prog, &xdp);
+
+       off = xdp.data - orig_data;
+       if (off)
+               __skb_push(skb, off);
+
+       switch (act) {
+       case XDP_TX:
+               __skb_push(skb, skb->mac_len);
+               /* fall through */
+       case XDP_PASS:
+               break;
+
+       default:
+               bpf_warn_invalid_xdp_action(act);
+               /* fall through */
+       case XDP_ABORTED:
+               trace_xdp_exception(skb->dev, xdp_prog, act);
+               /* fall through */
+       case XDP_DROP:
+       do_drop:
+               kfree_skb(skb);
+               break;
+       }
+
+       return act;
+}
+
 static int netif_receive_skb_internal(struct sk_buff *skb)
 {
        int ret;
@@ -4258,6 +4341,21 @@ static int netif_receive_skb_internal(struct sk_buff *skb)

        rcu_read_lock();
```

```
+       if (static_key_false(&generic_xdp_needed)) {
+               struct bpf_prog *xdp_prog = rcu_dereference(skb->dev->xdp_prog);
+
+               if (xdp_prog) {
+                       u32 act = netif_receive_generic_xdp(skb, xdp_prog);
+
+                       if (act != XDP_PASS) {
+                               rcu_read_unlock();
+                               if (act == XDP_TX)
+                                       dev_queue_xmit(skb);
+                               return NET_RX_DROP;
+                       }
+               }
+       }
+
 #ifdef CONFIG_RPS
        if (static_key_false(&rps_needed)) {
                struct rps_dev_flow voidflow, *rflow = &voidflow;
@@ -4490,7 +4588,7 @@ static enum gro_result dev_gro_receive(struct napi_struct *napi, struct sk_buff
        enum gro_result ret;
        int grow;

-       if (!(skb->dev->features & NETIF_F_GRO))
+       if (netif_elide_gro(skb->dev))
                goto normal;

        if (skb->csum_bad)
@@ -6718,6 +6816,7 @@ EXPORT_SYMBOL(dev_change_proto_down);
  */
 int dev_change_xdp_fd(struct net_device *dev, int fd, u32 flags)
 {
+       int (*xdp_op)(struct net_device *dev, struct netdev_xdp *xdp);
        const struct net_device_ops *ops = dev->netdev_ops;
        struct bpf_prog *prog = NULL;
        struct netdev_xdp xdp;
@@ -6725,14 +6824,16 @@ int dev_change_xdp_fd(struct net_device *dev, int fd, u32 flags)

        ASSERT_RTNL();

-       if (!ops->ndo_xdp)
-               return -EOPNOTSUPP;
+       xdp_op = ops->ndo_xdp;
+       if (!xdp_op || (flags & XDP_FLAGS_SKB_MODE))
+               xdp_op = generic_xdp_install;
+
        if (fd >= 0) {
                if (flags & XDP_FLAGS_UPDATE_IF_NOEXIST) {
                        memset(&xdp, 0, sizeof(xdp));
                        xdp.command = XDP_QUERY_PROG;

-                       err = ops->ndo_xdp(dev, &xdp);
+                       err = xdp_op(dev, &xdp);
                        if (err < 0)
                                return err;
                        if (xdp.prog_attached)
@@ -6748,7 +6849,7 @@ int dev_change_xdp_fd(struct net_device *dev, int fd, u32 flags)
        xdp.command = XDP_SETUP_PROG;
        xdp.prog = prog;

-       err = ops->ndo_xdp(dev, &xdp);
+       err = xdp_op(dev, &xdp);
        if (err < 0 && prog)
                bpf_prog_put(prog);

@@ -7789,6 +7890,7 @@ EXPORT_SYMBOL(alloc_netdev_mqs);
 void free_netdev(struct net_device *dev)
 {
        struct napi_struct *p, *n;
+       struct bpf_prog *prog;
```

```
 	might_sleep();
 	netif_free_tx_queues(dev);
@@ -7807,6 +7909,12 @@ void free_netdev(struct net_device *dev)
 	free_percpu(dev->pcpu_refcnt);
 	dev->pcpu_refcnt = NULL;
 
+	prog = rcu_dereference(dev->xdp_prog);
+	if (prog) {
+		bpf_prog_put(prog);
+		static_key_slow_dec(&generic_xdp_needed);
+	}
+
 	/*  Compatibility with error handling in drivers */
 	if (dev->reg_state == NETREG_UNINITIALIZED) {
 		netdev_freemem(dev);
diff --git a/net/core/gro_cells.c b/net/core/gro_cells.c
index c98bbfb..814e58a 100644
--- a/net/core/gro_cells.c
+++ b/net/core/gro_cells.c
@@ -13,7 +13,7 @@ int gro_cells_receive(struct gro_cells *gcells, struct sk_buff *skb)
 	struct net_device *dev = skb->dev;
 	struct gro_cell *cell;
 
-	if (!gcells->cells || skb_cloned(skb) || !(dev->features & NETIF_F_GRO))
+	if (!gcells->cells || skb_cloned(skb) || netif_elide_gro(dev))
 		return netif_rx(skb);
 
 	cell = this_cpu_ptr(gcells->cells);
diff --git a/net/core/rtnetlink.c b/net/core/rtnetlink.c
index 58419da..958a2bf 100644
--- a/net/core/rtnetlink.c
+++ b/net/core/rtnetlink.c
@@ -896,15 +896,13 @@ static size_t rtnl_port_size(const struct net_device *dev,
 		return port_self_size;
 }
 
-static size_t rtnl_xdp_size(const struct net_device *dev)
+static size_t rtnl_xdp_size(void)
 {
 	size_t xdp_size = nla_total_size(0) +	/* nest IFLA_XDP */
-			  nla_total_size(1);	/* XDP_ATTACHED */
+			  nla_total_size(1) +	/* XDP_ATTACHED */
+			  nla_total_size(4);	/* XDP_FLAGS */
 
-	if (!dev->netdev_ops->ndo_xdp)
-		return 0;
-	else
-		return xdp_size;
+	return xdp_size;
 }
 
 static noinline size_t if_nlmsg_size(const struct net_device *dev,
@@ -943,7 +941,7 @@ static noinline size_t if_nlmsg_size(const struct net_device *dev,
 	       + nla_total_size(MAX_PHYS_ITEM_ID_LEN) /* IFLA_PHYS_PORT_ID */
 	       + nla_total_size(MAX_PHYS_ITEM_ID_LEN) /* IFLA_PHYS_SWITCH_ID */
 	       + nla_total_size(IFNAMSIZ) /* IFLA_PHYS_PORT_NAME */
-	       + rtnl_xdp_size(dev) /* IFLA_XDP */
+	       + rtnl_xdp_size() /* IFLA_XDP */
 	       + nla_total_size(1); /* IFLA_PROTO_DOWN */
 
 }
@@ -1251,23 +1249,35 @@ static int rtnl_fill_link_ifmap(struct sk_buff *skb, struct net_device *dev)
 
 static int rtnl_xdp_fill(struct sk_buff *skb, struct net_device *dev)
 {
-	struct netdev_xdp xdp_op = {};
 	struct nlattr *xdp;
+	u32 xdp_flags = 0;
+	u8 val = 0;
 	int err;
```

```
-	if (!dev->netdev_ops->ndo_xdp)
-		return 0;
	xdp = nla_nest_start(skb, IFLA_XDP);
	if (!xdp)
		return -EMSGSIZE;
-	xdp_op.command = XDP_QUERY_PROG;
-	err = dev->netdev_ops->ndo_xdp(dev, &xdp_op);
-	if (err)
-		goto err_cancel;
-	err = nla_put_u8(skb, IFLA_XDP_ATTACHED, xdp_op.prog_attached);
+	if (rcu_access_pointer(dev->xdp_prog)) {
+		xdp_flags = XDP_FLAGS_SKB_MODE;
+		val = 1;
+	} else if (dev->netdev_ops->ndo_xdp) {
+		struct netdev_xdp xdp_op = {};
+
+		xdp_op.command = XDP_QUERY_PROG;
+		err = dev->netdev_ops->ndo_xdp(dev, &xdp_op);
+		if (err)
+			goto err_cancel;
+		val = xdp_op.prog_attached;
+	}
+	err = nla_put_u8(skb, IFLA_XDP_ATTACHED, val);
	if (err)
		goto err_cancel;

+	if (xdp_flags) {
+		err = nla_put_u32(skb, IFLA_XDP_FLAGS, xdp_flags);
+		if (err)
+			goto err_cancel;
+	}
	nla_nest_end(skb, xdp);
	return 0;
```