1. 升级有 Rust API 支持的 Linux kernel

内核代码仓库：
https://github.com/elliott10/linux/commits/rust-v6.1.66/

内核支持Rust

内核配置文件使用工控机上自带的，例如（/boot/config-6.1.19-xxx）。将其复制到内核源码的 `arch/x86/configs/qcl_defconfig` 处。

编译并安装内核

```
make ARCH=x86_64 LLVM=1 O=build qcl_defconfig
make ARCH=x86_64 LLVM=1 O=build menuconfig # 自定义内核配置，如打开Rust Support；
make ARCH=x86_64 LLVM=1 O=build
make ARCH=x86_64 LLVM=1 O=build modules_install INSTALL_MOD_PATH=modules_install INSTALL_MOD_STRIP=1
```

内核及驱动模块部署

1. 将编译生成的 `build/arch/x86_64/boot/bzImage` 内核文件部署到项目物理机器上 `/boot`
2. 内核模块 `build/modules_install/lib/modules/6.1.66+` 文件部署到项目物理机器上 `/lib/modules`
3. *重新生成* *`initramfs`*。内核和模块文件生成后，在项目物理机器上执行如下命令重新创建 `initramfs`

```
NEW_KERN_VERSION="6.1.66+"
dracut /boot/initramfs-${NEW_KERN_VERSION}.img ${NEW_KERN_VERSION}
```

4. 更新 grub

```
grub2_-mkconfig -o /boot/efi/EFI/openEuler/grub.cfg
```

2. 对 E1000 的驱动进行修改，以支持物理机上运行

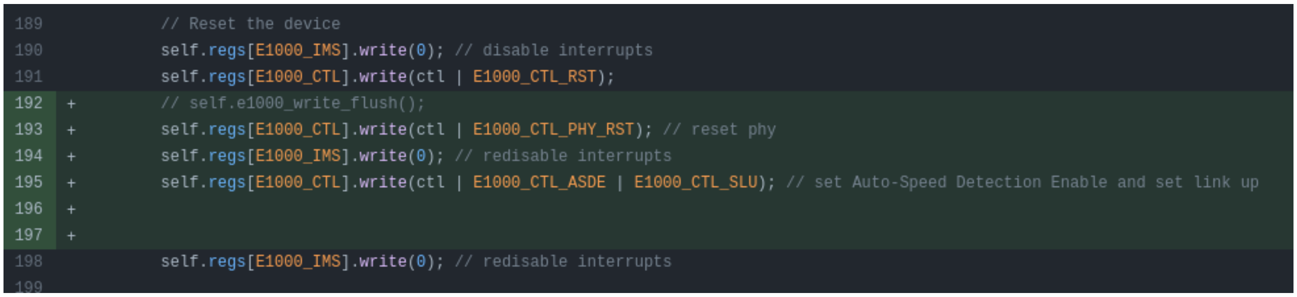
Rust 版工控机网卡驱动目前功能测试正常，可以正常连接外网进行通信

项目地址：
https://github.com/rcore-os/e1000-driver/discussions/1#discussioncomment-8217425

到物理机上迁移的工作：

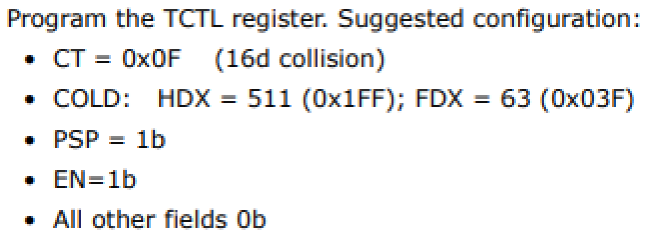
参考对比开源驱动和datasheet，整理出在驱动到物理机的迁移过程中比较重要的几个步骤
参考资料：
https://github.com/fujita/rust-e1000
https://courses.cs.washington.edu/courses/cse451/16au/readings/e1000e.pdf

1. 修改驱动初始化的代码，增加 PHY 芯片初始化的步骤，并启用 ASDE (用于自动协调PHY和MAC之间的通信速率)



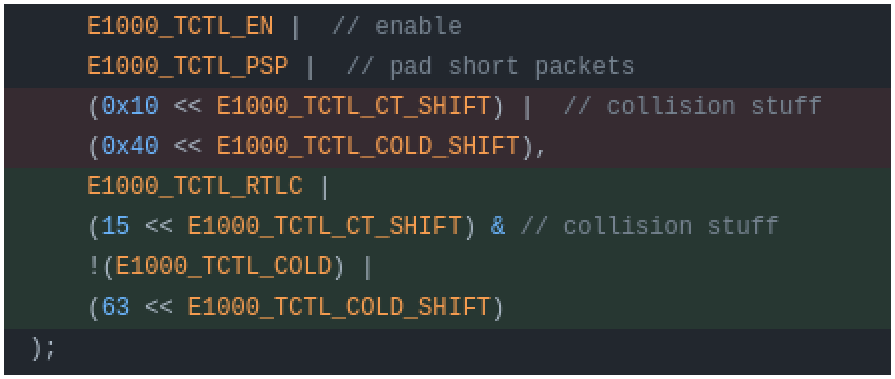
```
          // Reset the device
          self.regs[E1000_IMS].write(0); // disable interrupts
          self.regs[E1000_CTL].write(ctl | E1000_CTL_RST);
+         // self.e1000_write_flush();
+         self.regs[E1000_CTL].write(ctl | E1000_CTL_PHY_RST); // reset phy
+         self.regs[E1000_IMS].write(0); // redisable interrupts
+         self.regs[E1000_CTL].write(ctl | E1000_CTL_ASDE | E1000_CTL_SLU); // set Auto-Speed Detection Enable and set link up
+
+
          self.regs[E1000_IMS].write(0); // redisable interrupts
```

2. 参考 datasheet 更改 TCTL 寄存器的设置



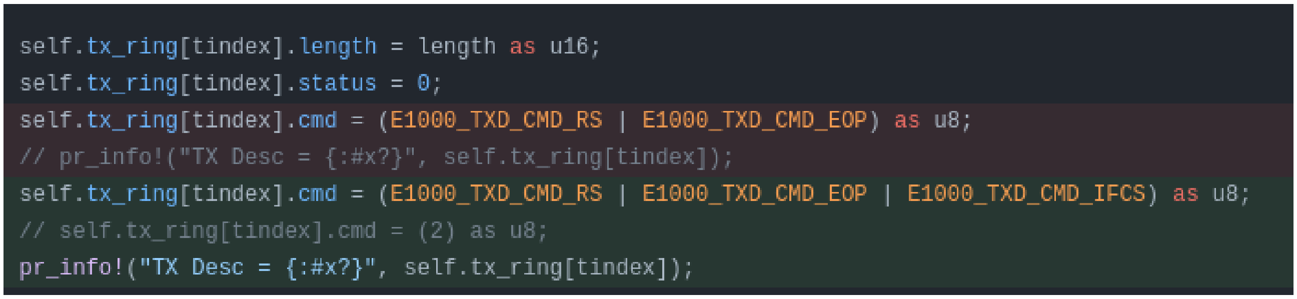
Program the TCTL register. Suggested configuration:

- CT = 0x0F   (16d collision)
- COLD: HDX = 511 (0x1FF); FDX = 63 (0x03F)
- PSP = 1b
- EN=1b
- All other fields 0b



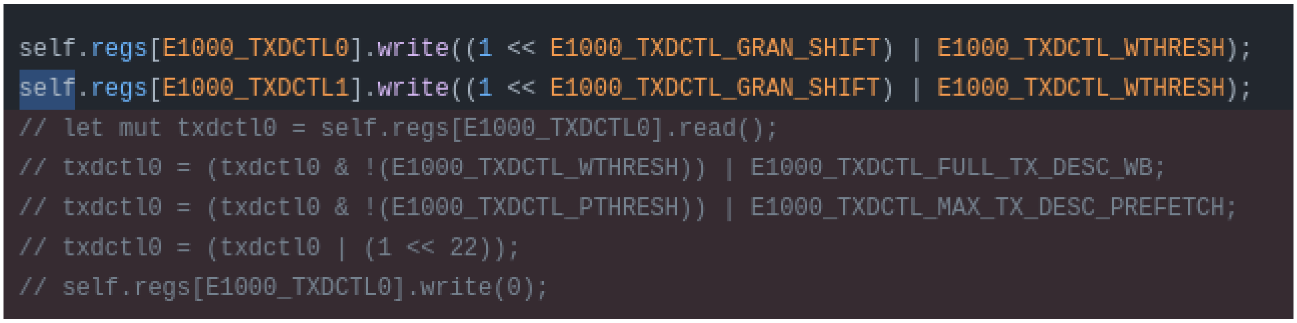
```
     E1000_TCTL_EN |  // enable
     E1000_TCTL_PSP |  // pad short packets
     (0x10 << E1000_TCTL_CT_SHIFT) |  // collision stuff
     (0x40 << E1000_TCTL_COLD_SHIFT),
     E1000_TCTL_RTLC |
     (15 << E1000_TCTL_CT_SHIFT) & // collision stuff
     !(E1000_TCTL_COLD) |
     (63 << E1000_TCTL_COLD_SHIFT)
 );
```

3. TxDesc 使能 IFCS 位（**重要**），不添加会导致网卡只能发 ARP 包，其他包无法正常发送



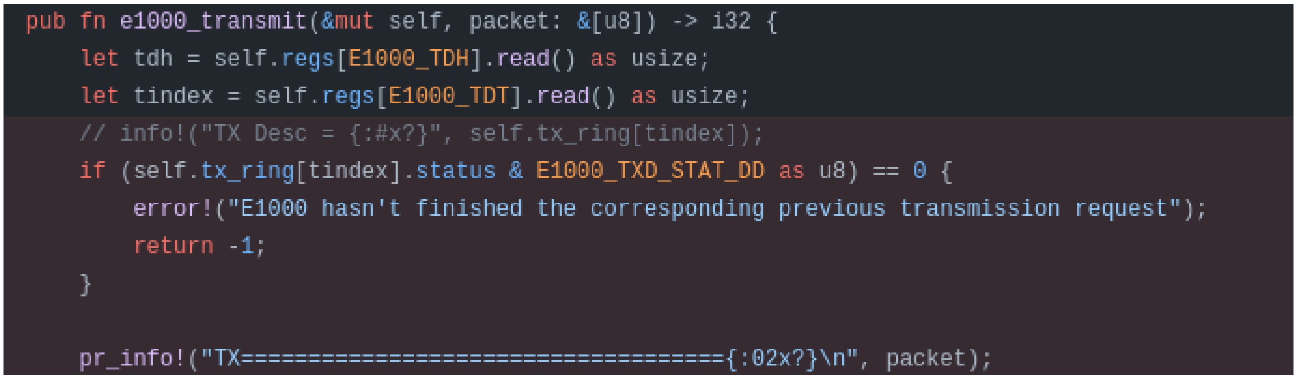
```
self.tx_ring[tindex].length = length as u16;
self.tx_ring[tindex].status = 0;
self.tx_ring[tindex].cmd = (E1000_TXD_CMD_RS | E1000_TXD_CMD_EOP) as u8;
// pr_info!("TX Desc = {:#x?}", self.tx_ring[tindex]);
self.tx_ring[tindex].cmd = (E1000_TXD_CMD_RS | E1000_TXD_CMD_EOP | E1000_TXD_CMD_IFCS) as u8;
// self.tx_ring[tindex].cmd = (2) as u8;
pr_info!("TX Desc = {:#x?}", self.tx_ring[tindex]);
```

4. 禁用 TxDesc 的 prefetch 机制（**重要**），开启 prefetch 会导致发包卡住，具体机制暂时未探明，目前先将其禁用



```
self.regs[E1000_TXDCTL0].write((1 << E1000_TXDCTL_GRAN_SHIFT) | E1000_TXDCTL_WTHRESH);
self.regs[E1000_TXDCTL1].write((1 << E1000_TXDCTL_GRAN_SHIFT) | E1000_TXDCTL_WTHRESH);
// let mut txdctl0 = self.regs[E1000_TXDCTL0].read();
// txdctl0 = (txdctl0 & !(E1000_TXDCTL_WTHRESH)) | E1000_TXDCTL_FULL_TX_DESC_WB;
// txdctl0 = (txdctl0 & !(E1000_TXDCTL_PTHRESH)) | E1000_TXDCTL_MAX_TX_DESC_PREFETCH;
// txdctl0 = (txdctl0 | (1 << 22));
// self.regs[E1000_TXDCTL0].write(0);
```

5. 修改发包队列满时处理的策略，避免发包队列满载时卡住
   移除以下代码，直接覆盖掉没发出去的包，由协议栈处理丢包重传

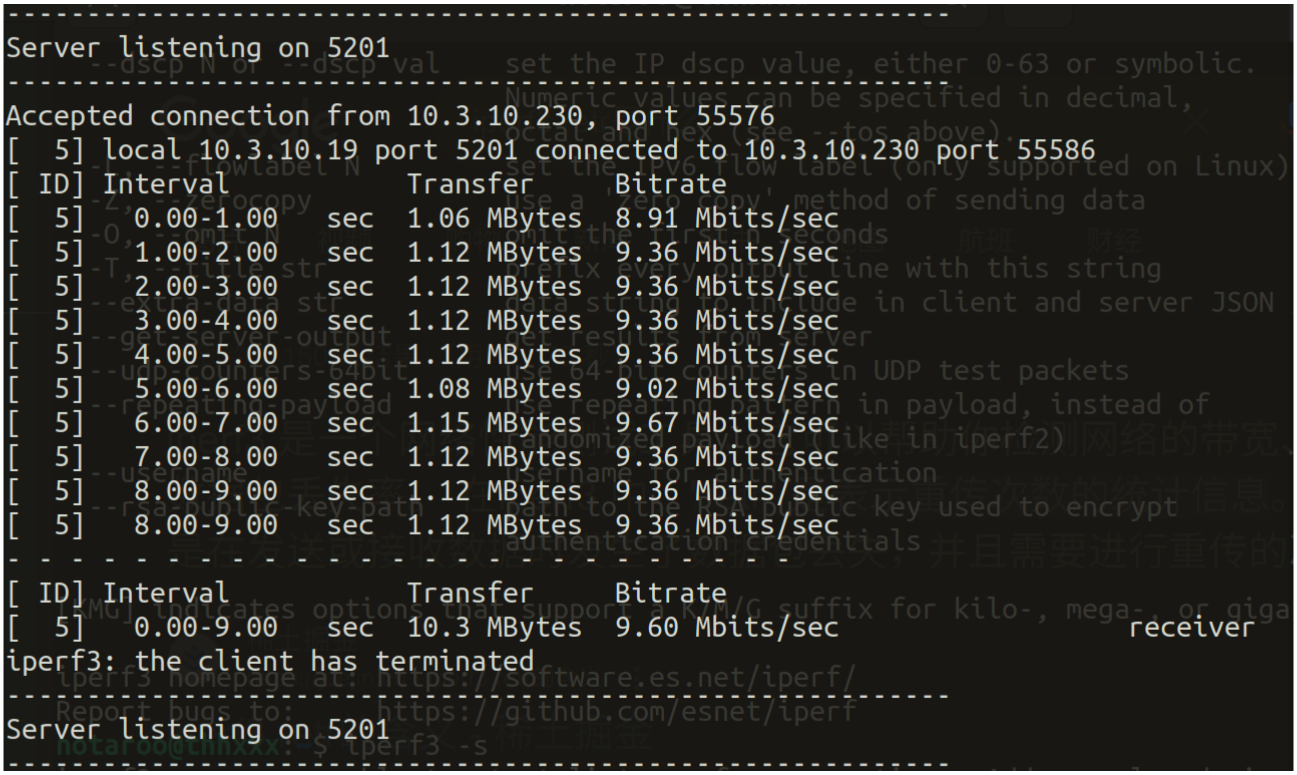


```
pub fn e1000_transmit(&mut self, packet: &[u8]) -> i32 {
    let tdh = self.regs[E1000_TDH].read() as usize;
    let tindex = self.regs[E1000_TDT].read() as usize;
    // info!("TX Desc = {:#x?}", self.tx_ring[tindex]);
    if (self.tx_ring[tindex].status & E1000_TXD_STAT_DD as u8) == 0 {
        error!("E1000 hasn't finished the corresponding previous transmission request");
        return -1;
    }

    pr_info!("TX===================================={:02x?}\n", packet);
```

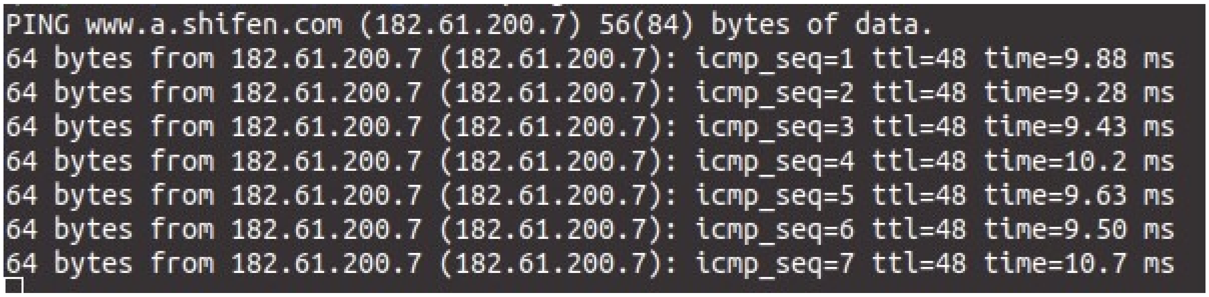
3.连接测试

1. 内网



```
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 10.3.10.230, port 55576
[  5] local 10.3.10.19 port 5201 connected to 10.3.10.230 port 55586
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec  1.06 MBytes  8.91 Mbits/sec
[  5]   1.00-2.00   sec  1.12 MBytes  9.36 Mbits/sec
[  5]   2.00-3.00   sec  1.12 MBytes  9.36 Mbits/sec
[  5]   3.00-4.00   sec  1.12 MBytes  9.36 Mbits/sec
[  5]   4.00-5.00   sec  1.12 MBytes  9.36 Mbits/sec
[  5]   5.00-6.00   sec  1.08 MBytes  9.02 Mbits/sec
[  5]   6.00-7.00   sec  1.15 MBytes  9.67 Mbits/sec
[  5]   7.00-8.00   sec  1.12 MBytes  9.36 Mbits/sec
[  5]   8.00-9.00   sec  1.12 MBytes  9.36 Mbits/sec
[  5]   8.00-9.00   sec  1.12 MBytes  9.36 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-9.00   sec  10.3 MBytes  9.60 Mbits/sec                  receiver
iperf3: the client has terminated
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

2. 外网



```
PING www.a.shifen.com (182.61.200.7) 56(84) bytes of data.
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=1 ttl=48 time=9.88 ms
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=2 ttl=48 time=9.28 ms
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=3 ttl=48 time=9.43 ms
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=4 ttl=48 time=10.2 ms
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=5 ttl=48 time=9.63 ms
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=6 ttl=48 time=9.50 ms
64 bytes from 182.61.200.7 (182.61.200.7): icmp_seq=7 ttl=48 time=10.7 ms
```