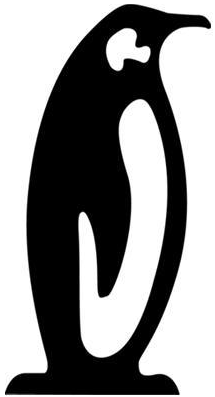
Google



東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Google

IDPF live-update support

Live Update MC

Brian Vazquez <brianvv@google.com>

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Agenda

01 Introduction

02 IDPF driver

03 Subsystem Integration

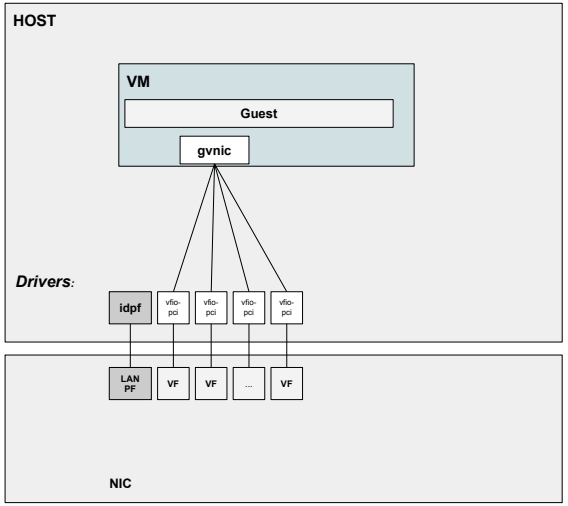
04 Open Questions & Discussion Topics

05 Q&A

Introduction

What is Live Update?

- **Definition:** A specialized reboot process preserving select device/kernel state.
- **Goal:** Minimize service disruption, especially in Cloud environments.
- **Key Use Case:** Hypervisor updates while keeping Virtual Machines (VMs) running with minimal network interruption.



Why Live Update for Network Drivers?

- Network I/O is critical for VM workloads.
- Traditional updates cause network downtime for VMs.
- Live Update aims to maintain or quickly restore network connectivity.

IDPF driver

What is IDPF?

Infrastructure Data-Plane Function. Modern Linux network driver for high-performance Intel Ethernet controllers.

Linux implementation: : https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/net/ethernet/intel/idpf/

IDPF spec

Key Characteristics & Complexities:

- SR-IOV (Single Root I/O Virtualization):
- Supports numerous Virtual Functions (VFs) for direct assignment to VMs
- Virtchnl Interface: communication between driver and firmware in the form of a dedicated set of hardware queues
- Data-Plane and Resource Management: Complex internal state for queues, filters, and interrupts.

Why is IDPF a Good Case Study for Live Update?

- Representative Complexity: Embodies the challenges of state preservation in modern, feature-rich NICs.
- SR-IOV Focus: The PF/VF architecture presents unique state synchronization challenges for Live Update.

IDPF driver: live update support

What needs to be saved and restored?

- Hardware registers (config, status, control).
- Transmit/Receive rings (descriptors, pointers, buffer states).
- Filter configurations (MAC/VLAN, Flow Director).
- Interrupt settings and coalescing.
- Internal driver software state.

What is actually needed to preserve VFs?

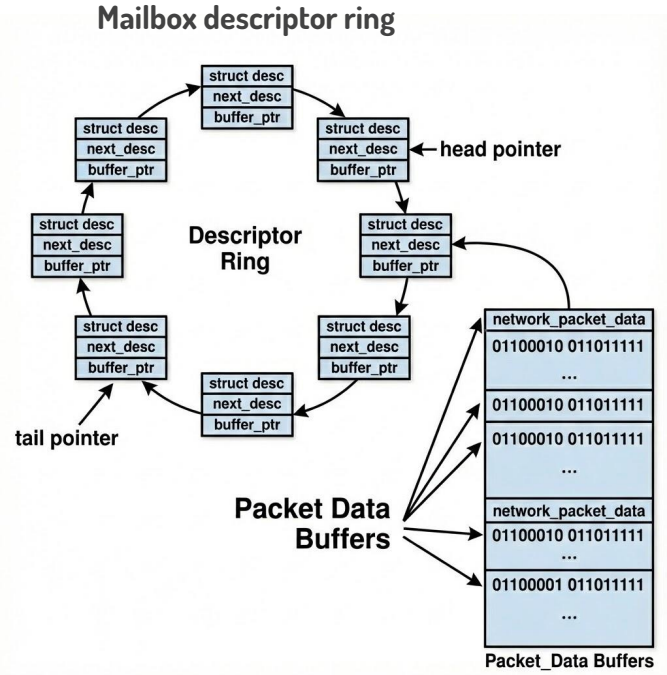
- Value of HW pointers to HEAD/TAIL of virtchannel: **it's the only part of the host's idpf state that really matters for the guests!**
- The rest of the state on the host can be torn down/reconstructed without impacting the guests

What makes liveupdate easy on IDPF?

- VF management is simple in IDPF, only two operations:
  - create/destroy VFs

What makes liveupdate hard on IDPF?

- PF can not be HW reset (it will trigger HW to destroy the VFs)
- Overhead of saving all PF state when only VFs need to be preserved



IDPF driver: live update support

```
--- a/drivers/net/ethernet/intel/libie/controlq.c
+++ b/drivers/net/ethernet/intel/libie/controlq.c
@@ -350,7 +350,10 @@ libie ctlq add(struct libie_ctlq_ctx *ctx,
                return ERR_PTR(-ENOMEM);
        }

-       libie ctlq init regs(ctlq);
+       if(liveupdate state updated())
+               libie_ctlq_restore_regs(ctlq);
+       else
+               libie_ctlq_init_regs(ctlq);

--- a/drivers/net/ethernet/intel/idpf/idpf lib.c
+++ b/drivers/net/ethernet/intel/idpf/idpf_lib.c
@@ -1902,6 +1902,12 @@ static void idpf_init_hard_reset(struct idpf_adapter *adapter)
        mutex_lock(&adapter->vport_ctrl_lock) ;

        dev info(dev,  "Device HW Reset initiated\n" );
+       if (liveupdate state updated()){
+               dev info(dev,  "liveupdate: Device HW Reset skipped\n" )
+               clear bit(IDPF HR DRV LOAD, adapter->flags);
+               goto liveupdate_skip_reset;
+       }
+

        /* Prepare for reset */
        if (test bit(IDPF_HR_DRV_LOAD, adapter->flags)) {
@@ -1928,6 +1934,7 @@ static void idpf_init_hard_reset(struct idpf_adapter *adapter)
                goto unlock_mutex;
        }

+liveupdate_skip_reset:
        /* Reset is complete and so start building the driver resources again */
        err = idpf init_dflt_mbx(adapter);
        if (err) {
```

Subsystem integration

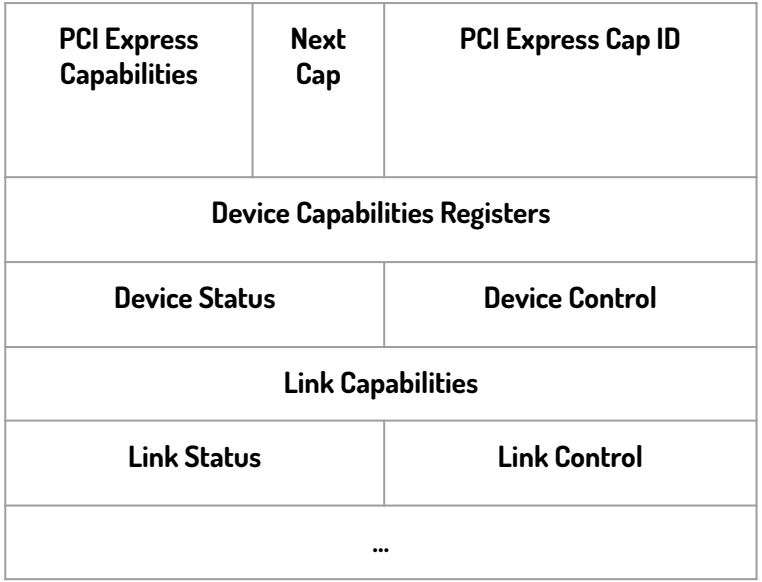
Live Update isn't just a driver problem. Requires coordination with core kernel components.

PCI

- How to prevent the PCI subsystem from fully resetting the device during the update
- Maintaining PCI configuration space, BARs.
- Notifying the driver about the special Live Update handling.

Challenges

- PCI changes save the entire config space and block any attempts to modify it, this forces a driver to save all the state, instead of just saving what is needed i.e. just save the VF portion, but allow PF to allocate new interrupts, new memory, etc.



PCI config space

Subsystem integration

PCI subsystem forwards live update callback to drivers

- hooks/interfaces needed:

```
struct dev_liveupdate_ops
{
        int (*prepare)(struct device *dev, u64 *data);
        int (*freeze)(struct device *dev, u64 *data);
        void (*cancel)(struct device *dev, u64 data);
        void (*finish)(struct device *dev, u64 data);
};
```

- **Prepare** (Normal → Prepared): Serialize all kernel state into memory to be passed to next kernel and mark folios to be preserved via KHO.
- **Freeze** (Prepared → Normal): Any final fixups before kexec.
- **Kexec** (Frozen → Updated): booting new kernel
- **Finish** (Updated → Normal): Validate that all preserved resources have been recovered by userspace.

Discussion: PCI preservation granularity

Context: pci preservation requires the device to persist everything: MSIX, PME, SR-IOV, etc

Problem: for a PF there's no value in persisting interrupts, and iommu mappings since host kernel is updating during liveupdate. Implementing the preservation of interrupts, saving the descriptor rings and dma mappings is a lot of overhead on a driver when requirement is only to preserve VF state if any.

Proposed solution: Allow drivers to advertise what is implemented by adding a new dev_liveupdate_ops and make pci honor that:

```
.pci_lu_features_check = idpf_lu_features_check,

static pci_lu_features_t  idpf_lu_features_check()
{
    return (pci_lu_features_t)1 << PCI_LU_SRIOV_BIT;
}

typedef u64 pci_lu_features_t;

enum {
        PCI_LU_MSIX_BIT,
        PCI_LU_SRIOV_BIT,
        ...
};
```

Discussion: pci_enable_sriov(pdev, num_vfs)

Problem: Although pci preserves the VFs, there's still need of an external actor triggering the enablement of SR-IOV feature

Possible solutions:

- PCI layer: can this be restored transparently by PCI layer?
- Driver: driver can store the preserved VFs value and call pci_enable_sriov(pdev, num_vfs?
- **Userspace:** Userspace is usually in charge of enabling VFs, should userspace restore them?

Discussion: pci_enable_sriov(pdev, num_vfs)

The snippet is run on every boot, but internally it behaves differently

- non Live update: PCI and IDPF inititialize SR-IOV resources
- Live update: code is treated as a nop by PCI or idpf at LU kexec, the call just triggers the restoration of what was saved

```
# Userspace script to enable SR-IOV
for i in "${device_paths[@]}"; do
    # make sure idpf will not be attached to all the LAN VFs
by default
    echo -n 0 > "${i}/sriov_drivers_autoprobe"

    # write contents from totalvfs to numvfs
    cp "${i}/sriov_totalvfs" "${i}/sriov_numvfs"

    # re-allow to bind drivers to LAN VFs
    echo -n 1 > "${i}/sriov_drivers_autoprobe"
 done
```

Open Questions

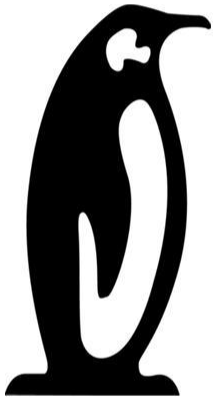
- Error handling if save/restore fails.
  - From device pov, if live update fails, driver needs to trigger a HW reset to clean internal state.
  - How to declare that an error occur in the live update? do we need error ops callback?
- Standardization efforts vs. driver-specific solutions.
  - Is this possible?
  - At a quick glance seems that each implementation requires deep knowledge of the device

Q&A

Conclusions

- Persisting drivers is complex, it requires very deep knowledge of the hardware to make sure there's no disruption
- More granular feature preservation at pci layer could help in making development more modular i.e implement device state preservation without interrupts preservation, etc.

Google



東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025