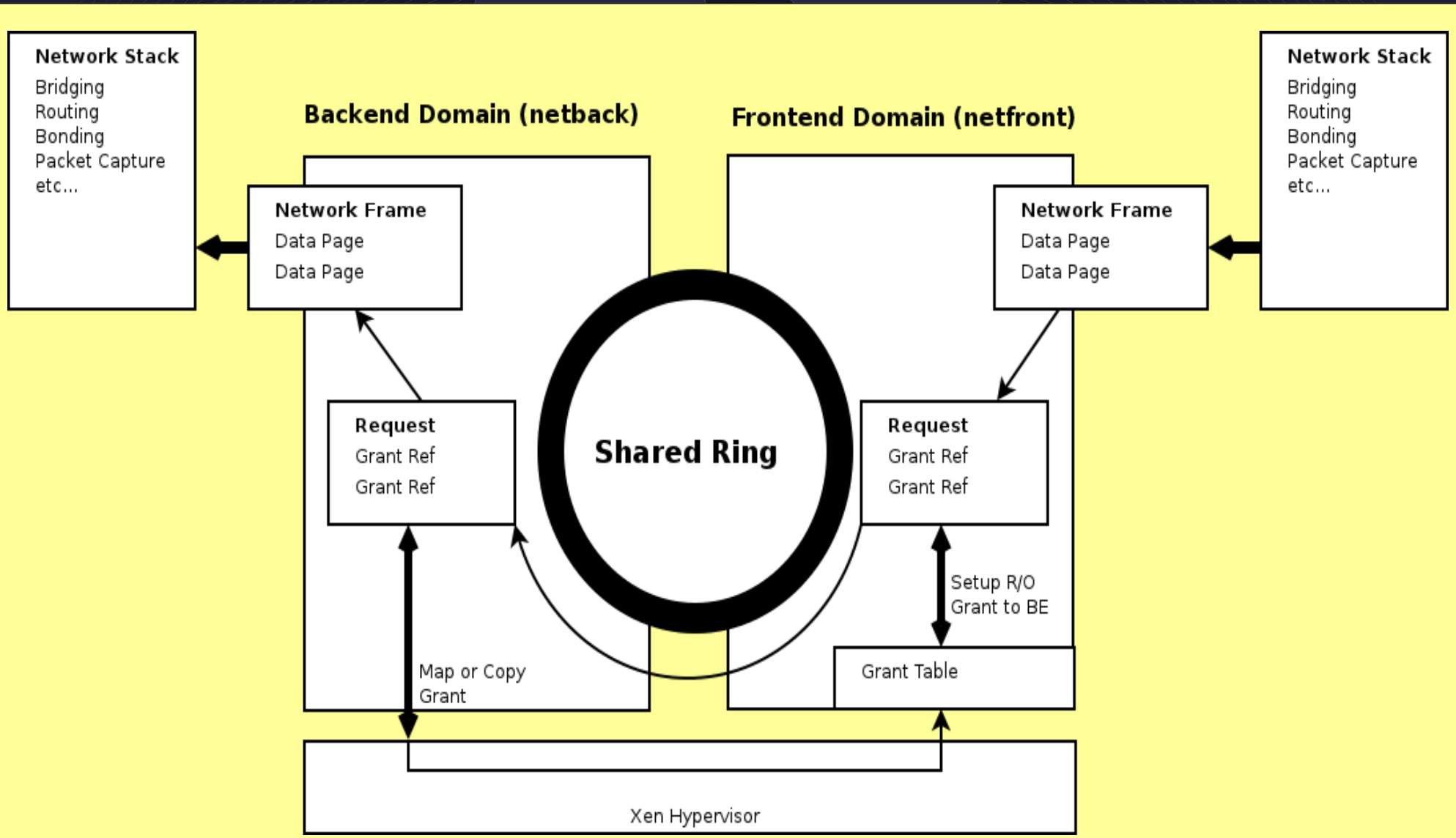
Zero-Copy Networking (or SKB fragment life-cycle tracking)

Ian Campbell
Citrix & Xen.org
<ian.campbell@citrix.com>

Agenda

- Xen PV network transmit operation
- Issues with supporting zero-copy
- Similar issues seen elsewhere
- How can we solve these problems
- SKB paged fragment destructors

Xen PV Network Transmit



Problems With Mapping

- Must eventually unmap and return to guest
- Must give up complete control of page to network stack
- Cannot do both...
  - “Foreign” memory free()d into standard allocation pools
  - Cannot manipulate mappings of foreign pages in the normal way (set_pte etc)
  - Badness occurs!
-

Underlying Issue

- Network stack does no tracking of page references itself
- Piggybacks on the core's reference counting of struct page
  - Uses raw get_page() and put_page()

“Classic” Xen Patch Solution

- **include/linux/page-flags.h:**

```
#ifdef CONFIG_XEN
    PG_foreign, /* Page is owned by foreign allocator. */
#endif
…
static inline void SetPageForeign(struct page *page,
                  void (*dtor)(struct page *, unsigned int)) {
    set_bit(PG_foreign, &page->flags);
    page->index = (long)dtor;
}

static inline void PageForeignDestructor(struct page *page,…) {
    ((void (*)(struct page *, unsigned int))page->index)(page,…);
}
```

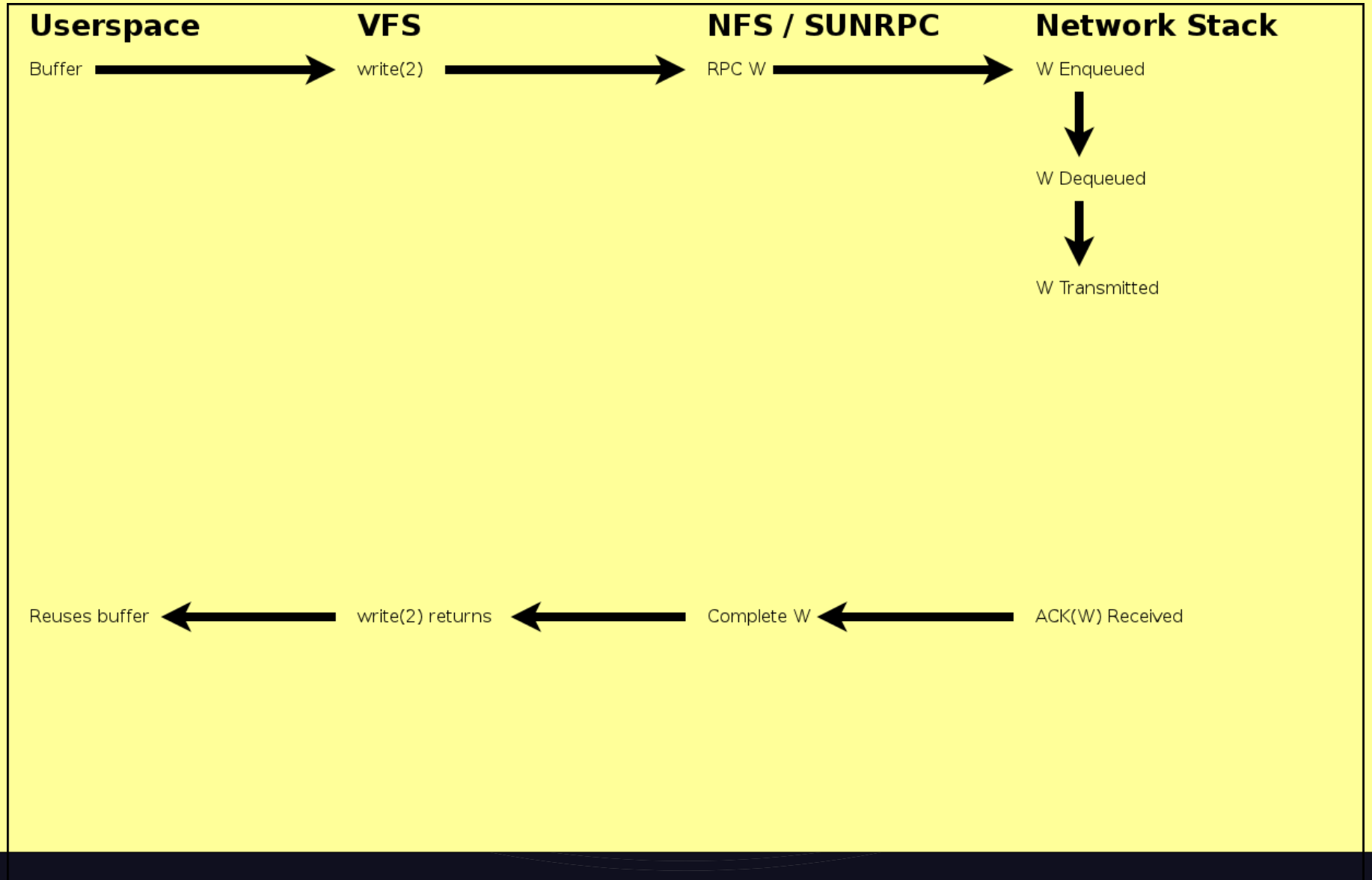
- **mm/page_alloc.c:**

```
static void free_hot_cold_page(struct page *page, int cold) {
…
#ifdef CONFIG_XEN
    if (PageForeign(page)) {
            PageForeignDestructor(page, 0);
            return;
    }
```

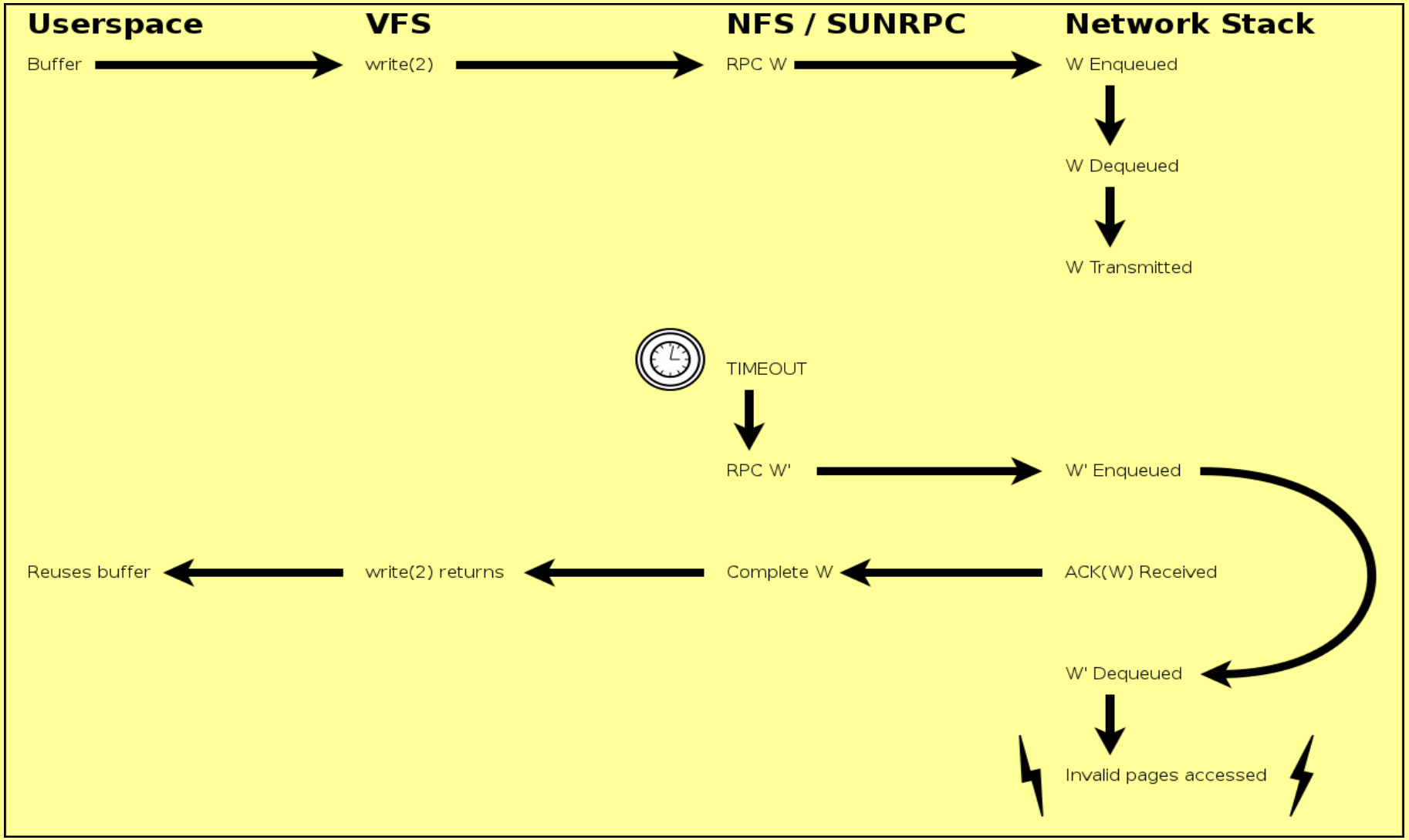
Netback in mainline

- PageForeign approach unsuitable for mainline
- netback upstreamed with copy instead of map
- Better than nothing...
- … but obviously we would like zero-copy again

NFS Write


Userspace
VFS
NFS / SUNRPC
Network Stack
Buffer
write(2)
RPC W
W Enqueued
W Dequeued
W Transmitted
Reuses buffer
write(2) returns
Complete W
ACK(W) Received

NFS Write Retransmit



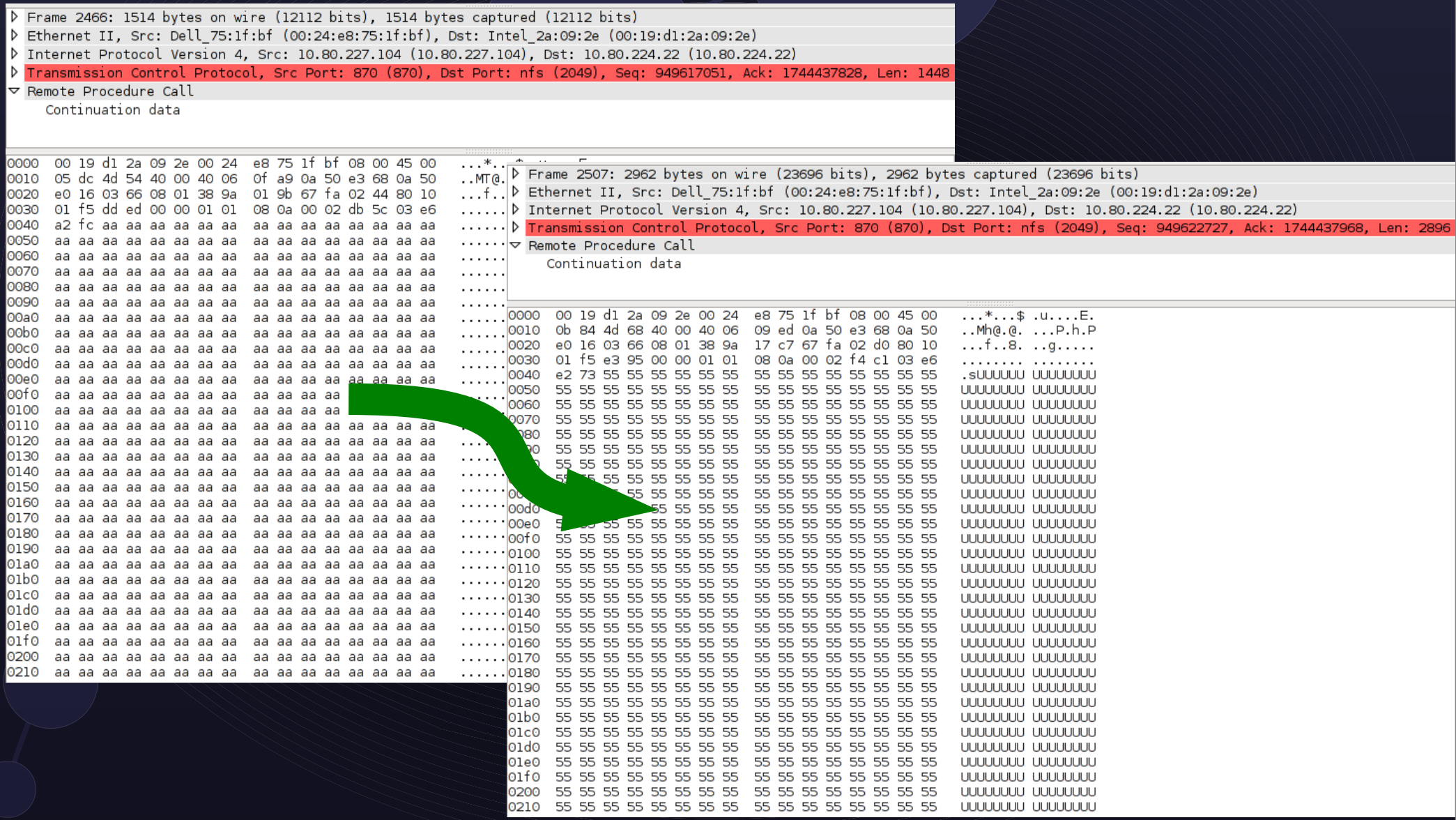
Reproduction

- `iptables -I INPUT --source $SERVER -j QUEUE`
- `netoutage.c`:
  - delay 1/N packets for 60s using libnetfilter_queue
- `test.c`:

```
{
        …
        memset(buf, SIZE, 0xAA);
        write(fd, buf, SIZE);
        memset(buf, SIZE, 0x55);
        …
}
```

- `tcpdump…'host $SERVER and ip[184:4] == 0x55555555'`

Captured Result



Requirements for a General Solution

- Callers to be able to give a page to the network stack while retaining ownership
- Not sufficient to just know when page is freed
  - There may be other holders of core kernel references on a page
- Need to track all references with the stack across skb_clone, skb_pull_up, segmentation etc
  - Existing skb destructor not sufficient

SKB Paged Fragment Usage

```
struct skb_frag_struct {
     struct page *page;
     …
};
…
frag->page = page;
…
get_page(skb_shinfo(skb)->frags[i].page)
…
dma_map_page(…, frag->page, …)
…
put_page(skb_shinfo(skb)->frags[i].page)
```

SKB Paged Fragment API

```
static inline struct page *skb_frag_page(const skb_frag_t *frag)
…
static inline void skb_frag_ref(struct sk_buff *skb, int f)
…
static inline void skb_frag_unref(struct sk_buff *skb, int f)
…
static inline dma_addr_t skb_frag_dma_map(
                                    struct device *dev,
                                    const skb_frag_t *frag,
                                    size_t offset, size_t size,
                                    enum dma_data_direction dir)
…
```

SKB Paged Fragment API Patches

- Touches a lot of code

```
87 files changed, 513 insertions(+), 356 deletions(-)
```

- In particular touches most network drivers
- Coccinelle (spatch) was very useful in the early stages of the conversion
- API itself is in net-next
- Driver conversions are under way as well
  - ~ half complete.

SKB Paged Fragment Destructors

```
struct skb_frag_destructor {
        atomic_t ref;
        int (*destroy)(void *data);
        void *data;
};

struct skb_frag_struct {
        struct {
                struct page *p;
                struct skb_frag_destructor *destr…;
        } page;
        …
};
```

SKB Paged Fragment Destructors

- Builds on the Fragment API, will come after
- Far less intrusive

```
include/linux/skbuff.h |   54 ++++++++++++[...]+++++++++++++++---
net/core/skbuff.c      |   26 ++++++++++++++++++++-
2 files changed, 74 insertions(+), 6 deletions(-)
```

- Additional patch uses it in the RPC layer

```
include/linux/sunrpc/xdr.h  |    2 ++
include/linux/sunrpc/xprt.h |    5 ++++-
net/sunrpc/clnt.c           |   27 ++++++++++++++++++++++-----
net/sunrpc/svcsock.c        |    3 ++-
net/sunrpc/xprt.c           |   13 +++++++++++
net/sunrpc/xprtsock.c       |    3 ++-
6 files changed, 45 insertions(+), 8 deletions(-)
```

Summary

- Fragment API trickling into net-next
- Fragment destructor patches ready
- NFS / SUNRPC fix ready
- Next steps:
  - Zero-copy in netback
  - iSCSI?
  - Packet mmap?

Questions?