

New sendfile(2)

Gleb Smirnov
glebius@FreeBSD.org

FreeBSD Storage Summit
Netflix

20 February 2015



Miserable life w/o sendfile(2)

```
while ((cnt = read(filefd, buf, (u_int)blksize))  
       write(netfd, buf, cnt) == cnt)  
    byte_count += cnt;
```

send_data() в src/libexec/ftpd/ftpd.c,
FreeBSD 1.0, 1993



sendfile(2) introduced

```
int  
sendfile(int fd, int s, off_t offset, size_t nbytes, .. );
```

- 1997: HP-UX 11.00
- 1998: FreeBSD 3.0 and Linux 2.2



sendfile(2) in FreeBSD

- First implementation - mapping userland cycle to the kernel:
 - read(filefd) → VOP_READ(vnode)
 - write(netfd) → sosend(socket)
 - blksize → PAGE_SIZE



sendfile(2) in FreeBSD

- First implementation - mapping userland cycle to the kernel:
 - `read(filefd) → VOP_READ(vnode)`
 - `write(netfd) → sosend(socket)`
 - `blksize → PAGE_SIZE`
- Further optimisations:
 - 2004: `SF_NODISKIO` flag



sendfile(2) in FreeBSD

- First implementation - mapping userland cycle to the kernel:
 - `read(filefd) → VOP_READ(vnode)`
 - `write(netfd) → sosend(socket)`
 - `blksize → PAGE_SIZE`
- Further optimisations:
 - 2004: `SF_NODISKIO` flag
 - 2006: inner cycle, working on `sbspace()` bytes



sendfile(2) in FreeBSD

- First implementation - mapping userland cycle to the kernel:
 - `read(filefd) → VOP_READ(vnode)`
 - `write(netfd) → sosend(socket)`
 - `blksize → PAGE_SIZE`
- Further optimisations:
 - 2004: `SF_NODISKIO` flag
 - 2006: inner cycle, working on `sbspace()` bytes
 - 2013: sending a shared memory descriptor data



Problem #1: blocking on I/O

Algorithm of a modern HTTP-server:

- 1 Take yet another descriptor from kevent(2)
- 2 Do write(2)/read(2)/sendfile(2) on it
- 3 Go to 1



Problem #1: blocking on I/O

Algorithm of a modern HTTP-server:

- 1 Take yet another descriptor from kevent(2)
- 2 Do write(2)/read(2)/sendfile(2) on it
- 3 Go to 1

Bottleneck: any syscall time.



Attempts to solve problem #1

- Separate I/O contexts: processes, threads
 - Apache
 - nginx 2



Attempts to solve problem #1

- Separate I/O contexts: processes, threads
 - Apache
 - nginx 2
- SF_NODISKIO + aio_read(2)
 - nginx
 - Varnish



More attempts . . .

- `aio_mlock(2)` instead of `aio_read(2)`
- `aio_sendfile(2)` ???



Problem #2: control over VM

- VOP_READ() leaves pages in VM cache
- VOP_READ() [for UFS] does readahead



Problem #2: control over VM

- VOP_READ() leaves pages in VM cache
- VOP_READ() [for UFS] does readahead
- Not easy to prevent it doing that!



waht if VOP_GETPAGES()?

~~VOP_READ()~~ → VOP_GETPAGES()

- Pros:
 - sendfile() already works on pages
 - implementations for vnode and shmем converge
 - control over VM is now easier task



waht if VOP_GETPAGES()?

~~VOP_READ()~~ → VOP_GETPAGES()

- Pros:
 - sendfile() already works on pages
 - implementations for vnode and shmem converge
 - control over VM is now easier task
- Cons
 - Losing readahead heuristics ☹️



waht if VOP_GETPAGES()?

~~VOP_READ()~~ → VOP_GETPAGES()

- Pros:
 - sendfile() already works on pages
 - implementations for vnode and shmem converge
 - control over VM is now easier task
- Cons
 - Losing readahead heuristics ☹️
 - But no one used them! 😊



VOP_GETPAGES_ASYNC()

```
int  
VOP_GETPAGES(struct vnode *vp, vm_page_t *ma,  
int count, int reqpage);
```

- 1 Initialize buf(9)
- 2 buf->b_iodone = bdone;
- 3 bstrategy(buf);
- 4 bwait(buf); /* sleeps until I/O completes */
- 5 return;



VOP_GETPAGES_ASYNC()

```
int  
VOP_GETPAGES_ASYNC(struct vnode *vp,  
vm_page_t *ma, int count, int reqpage,  
vop_getpages_iodone_t *iodone, void *arg);
```

- 1 Initialize buf(9)
- 2 buf->b_iodone = vnode_pager_async_iodone;
- 3 bstrategy(buf);
- 4 return;

vnode_pager_async_iodone calls `iodone()` .



naive non-blocking sendfile(2)

In kern_sendfile():

- 1 nios++;
- 2 VOP_GETPAGES_ASYNC(sendfile_iodone);

In sendfile_iodone():

- 1 nios--;
- 2 if (nios) return;
- 3 sosend();

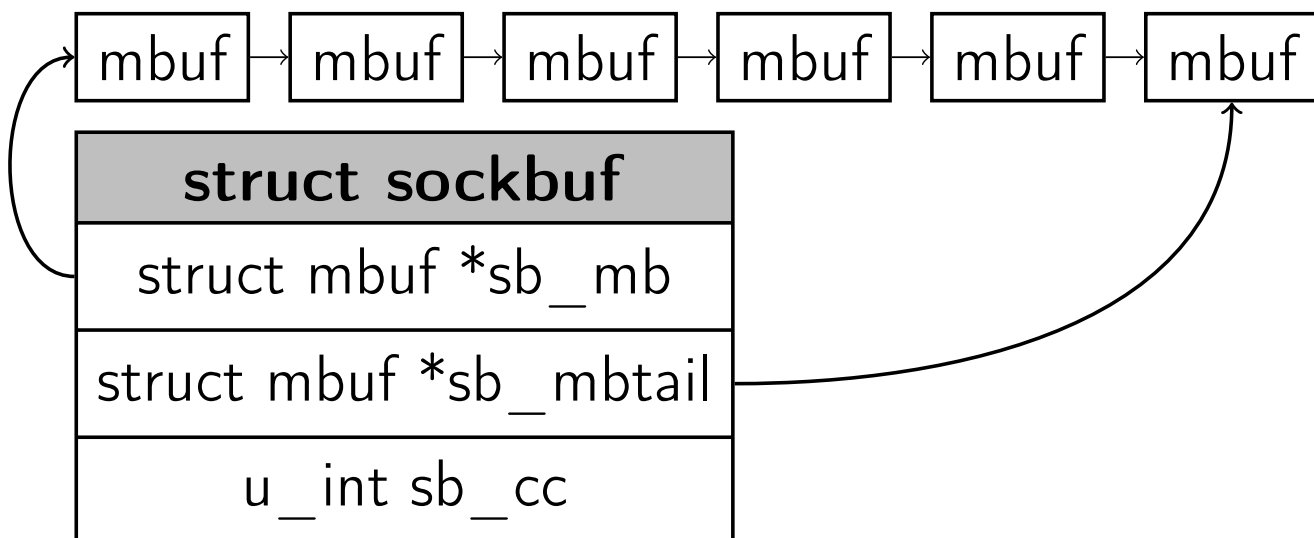


the problem of naive implementation

```
sendfile(filefd, sockfd, ..);  
write(sockfd, ..);
```

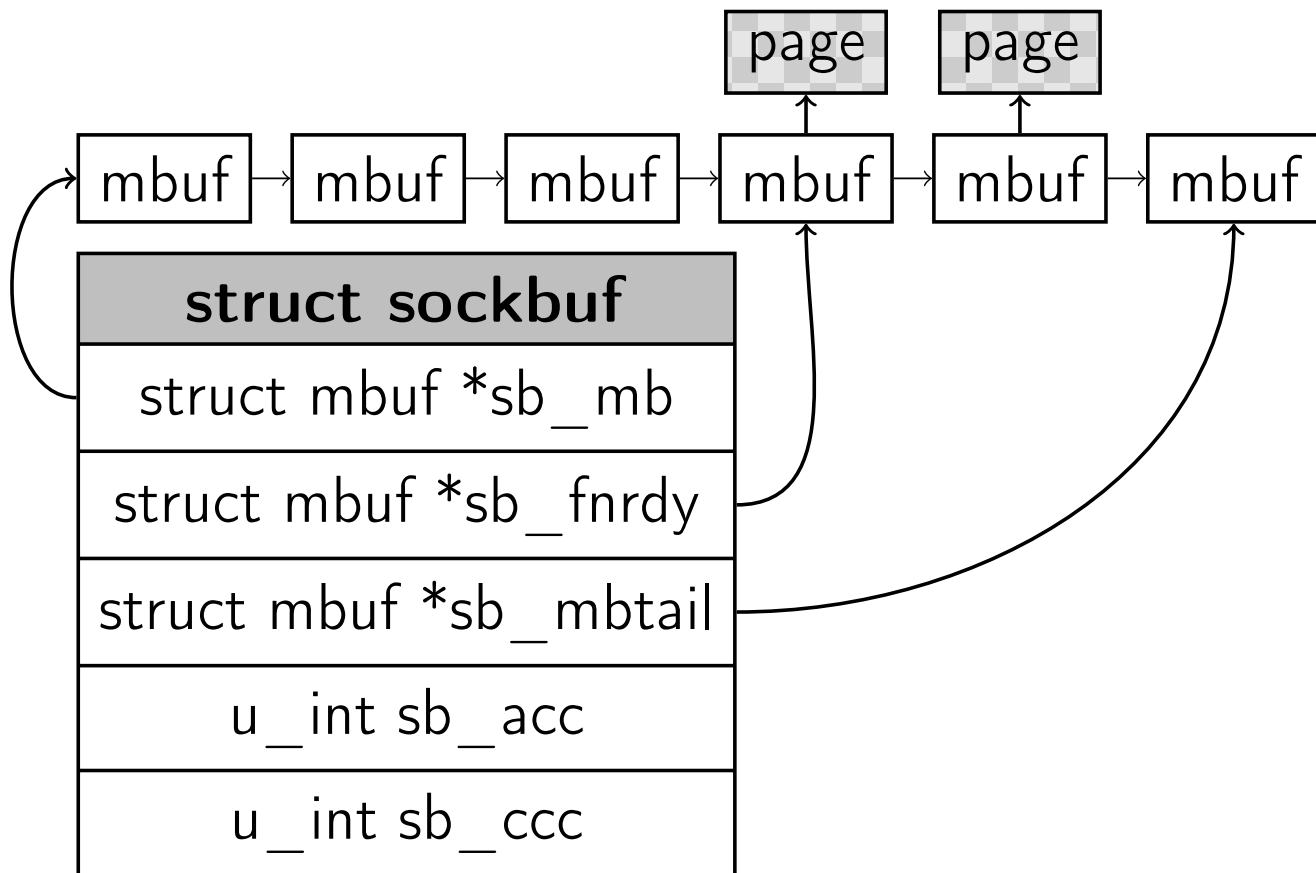


socket buffer





socket buffer with "not ready" data





non-blocking sendfile(2)

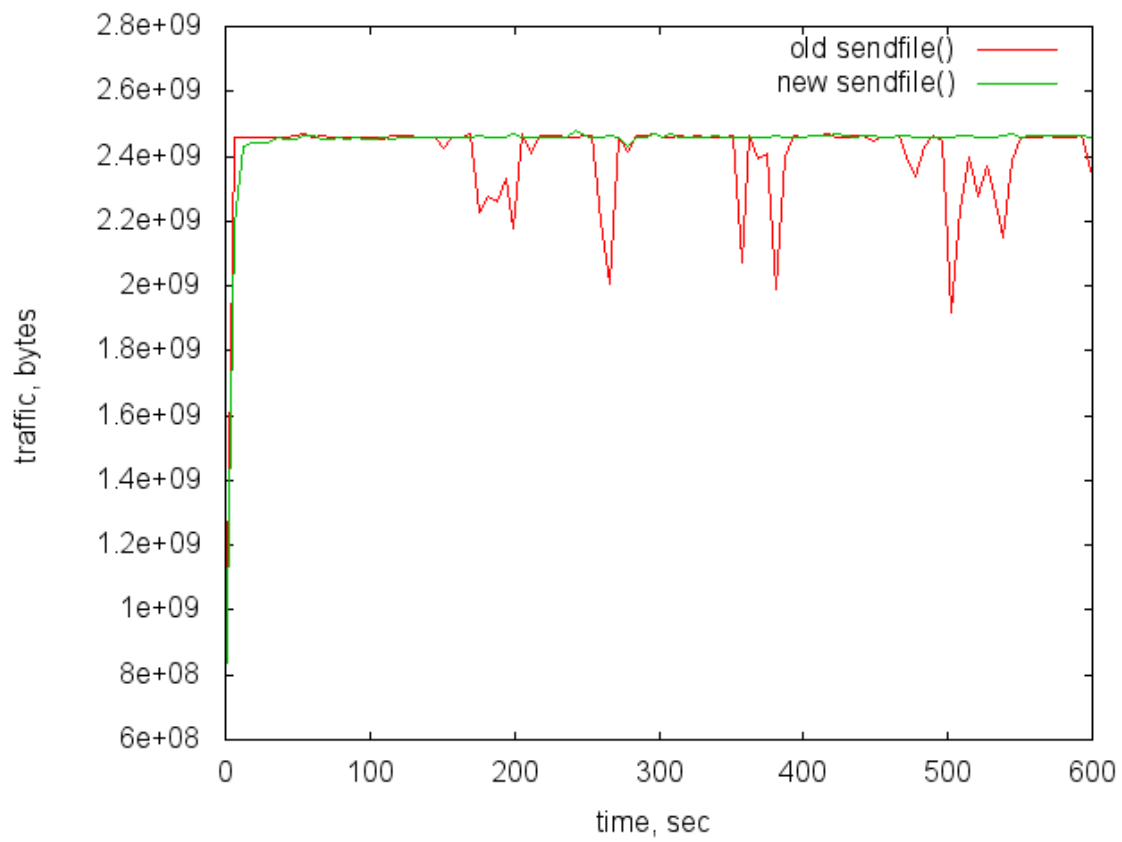
In kern_sendfile():

- 1 nios++;
- 2 VOP_GETPAGES_ASYNC(sendfile_iodone);
- 3 sosend(NOT_READY);

In sendfile_iodone():

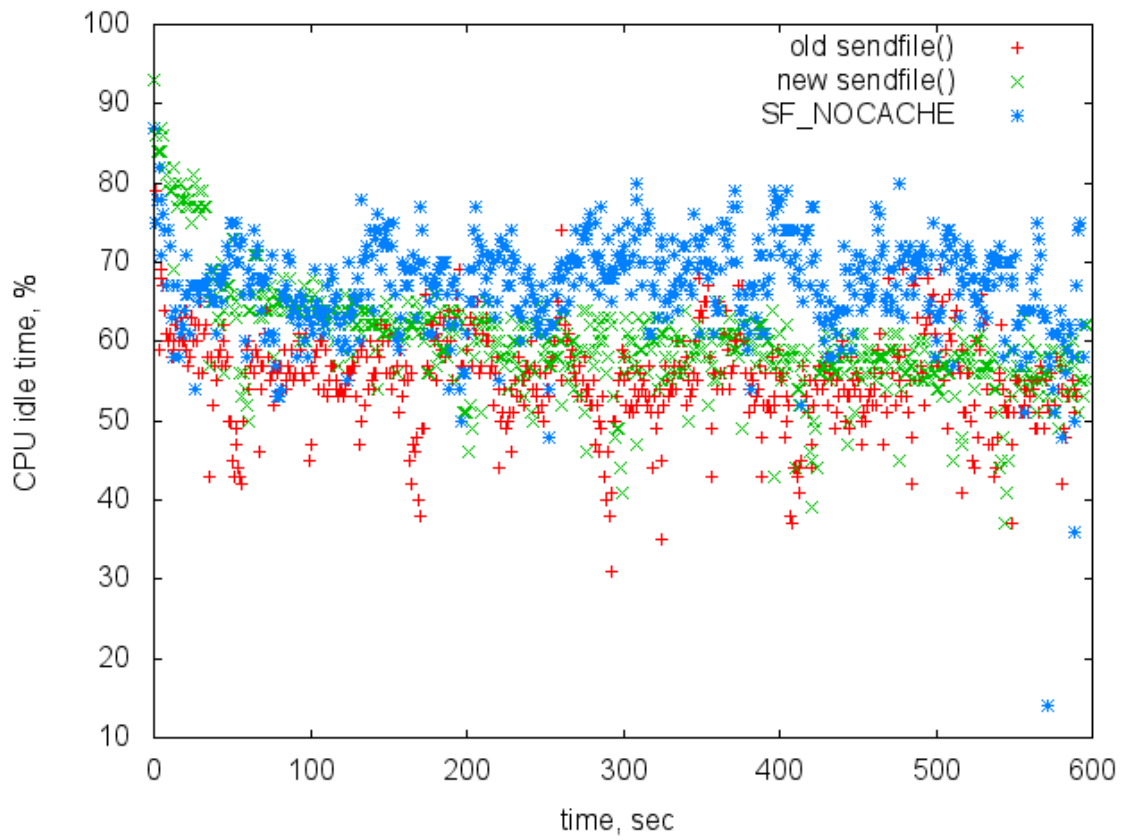
- 1 nios--;
- 2 if (nios) return;
- 3 soready();

traffic



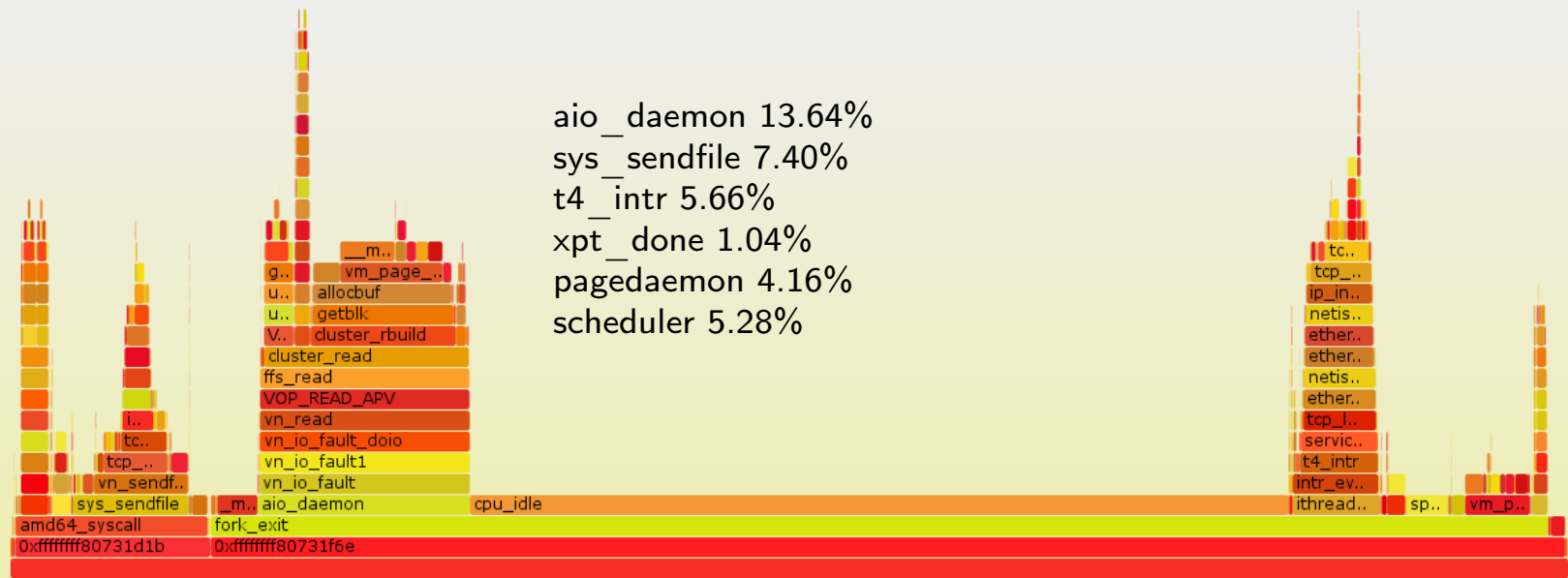


CPU idle





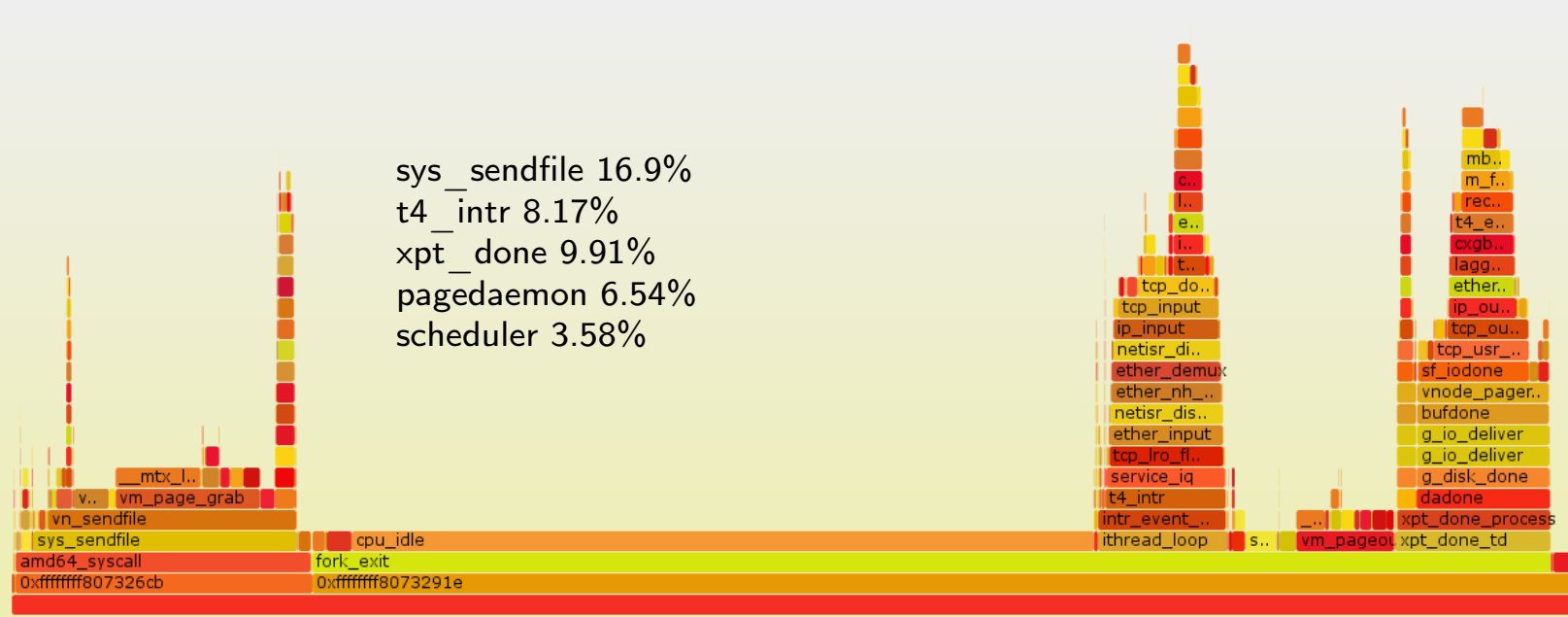
profiling sendfile(2) in head





profiling new sendfile(2)

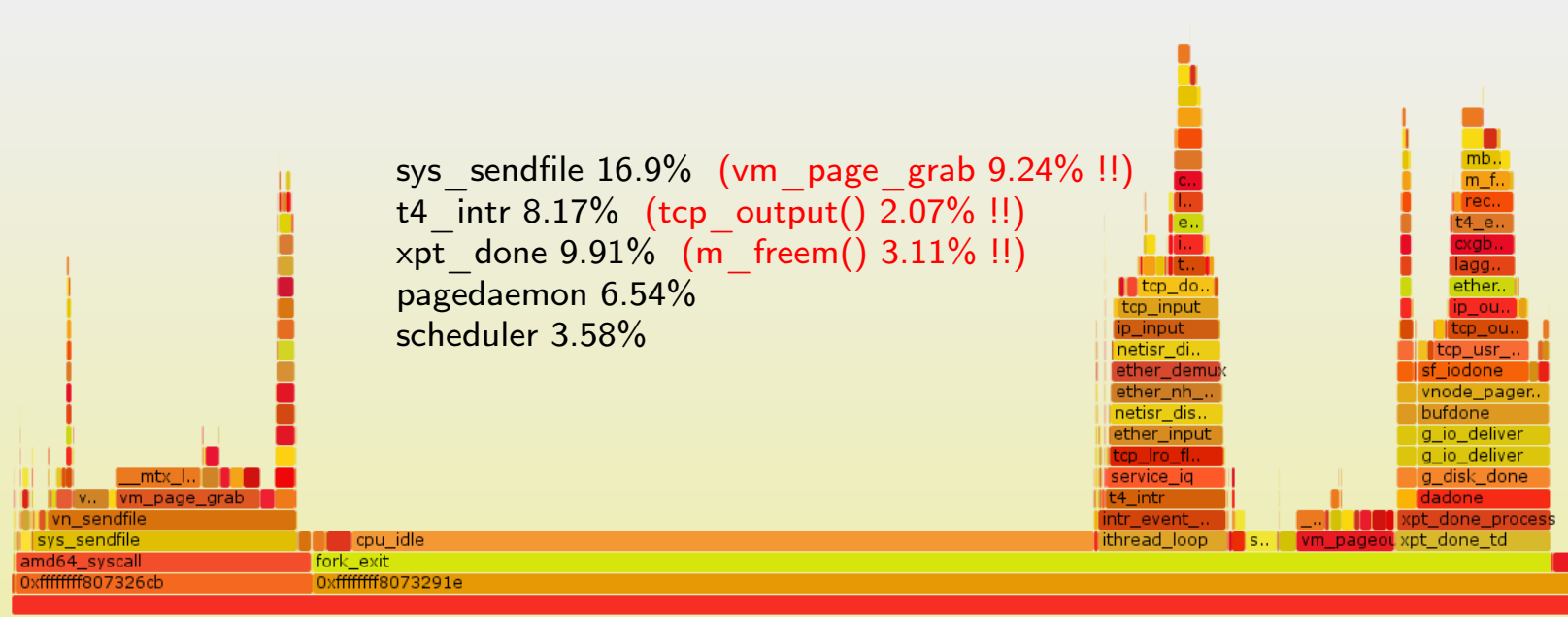
sys_sendfile 16.9%
t4_intr 8.17%
xpt_done 9.91%
pagedaemon 6.54%
scheduler 3.58%





profiling new sendfile(2)

sys_sendfile 16.9% (vm_page_grab 9.24% !!)
t4_intr 8.17% (tcp_output() 2.07% !!)
xpt_done 9.91% (m_freem() 3.11% !!)
pagedaemon 6.54%
scheduler 3.58%





what did change?

- New code always sends full socket buffer
 - Which is good for TCP (as protocol)
 - Which hurts VM, mbuf allocator, and unexpectedly TCP stack

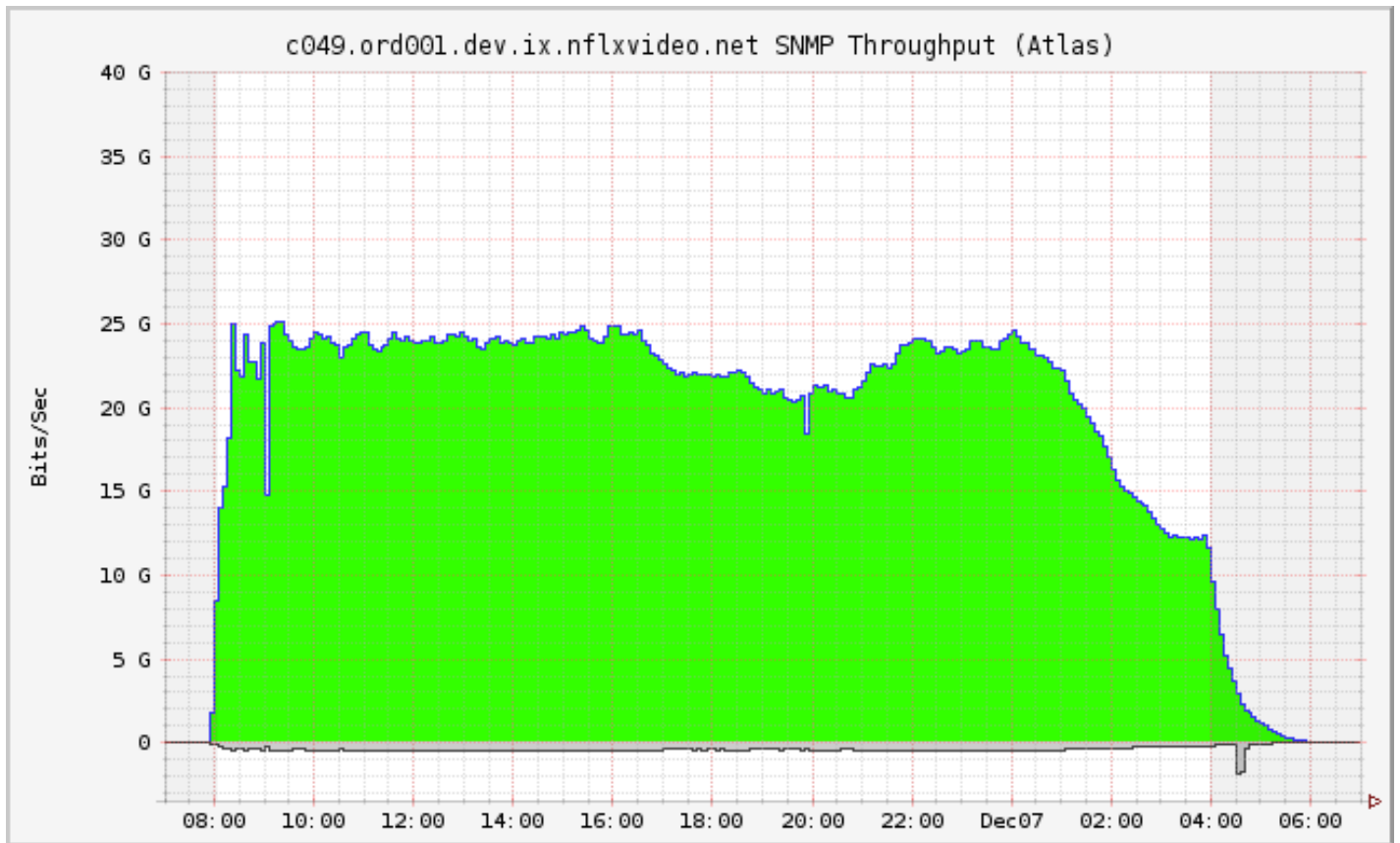


what did change?

- New code always sends full socket buffer
 - Which is good for TCP (as protocol)
 - Which hurts VM, mbuf allocator, and unexpectedly TCP stack
- Will fix that!

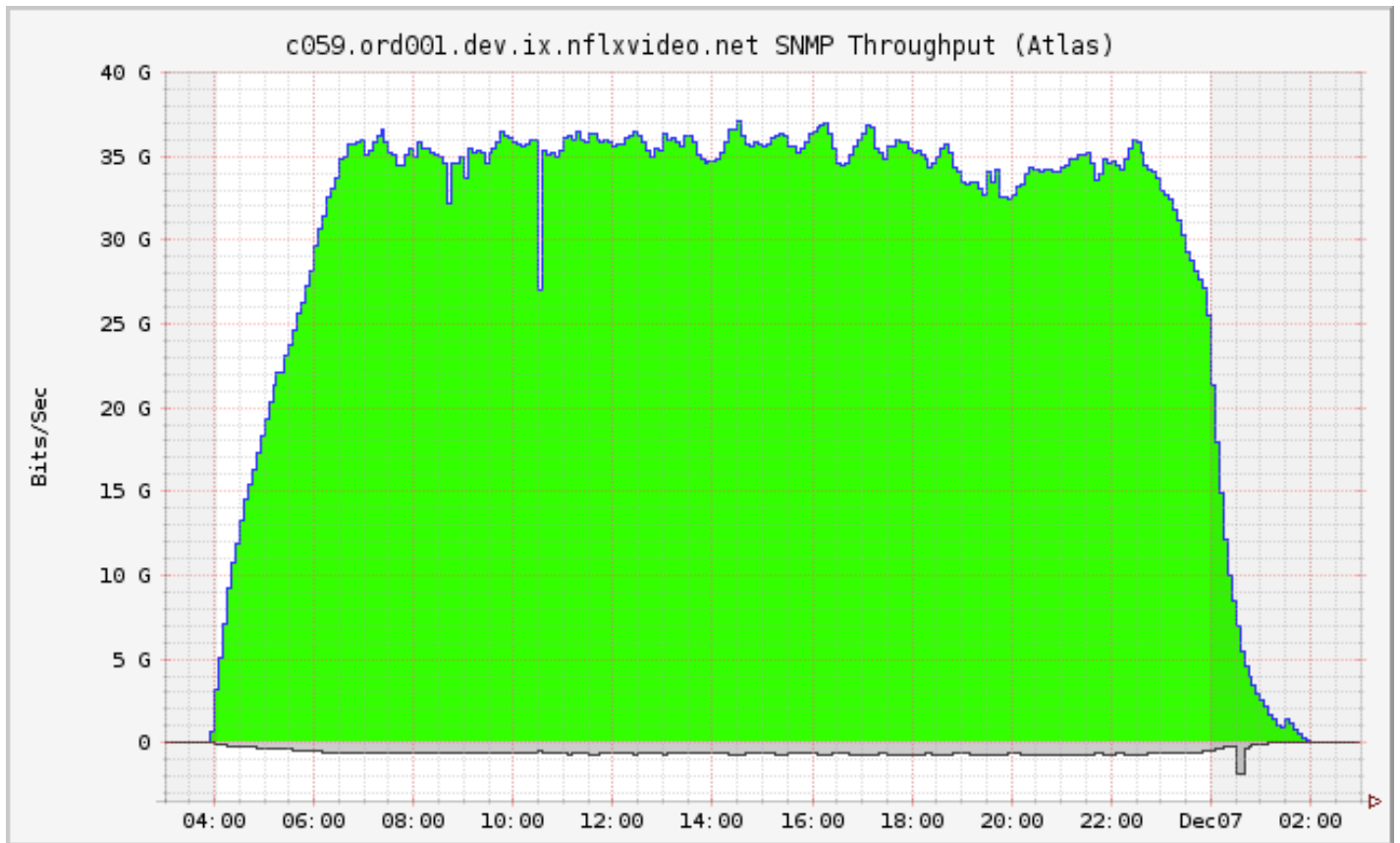


old sendfile(2) @ Netflix





new sendfile(2) @ Netflix





TODO list

Problems:

- VM & I/O overcommit
- ZFS
- SCTP



TODO list

Problems:

- VM & I/O overcommit
- ZFS
- SCTP

Future plans:

- sendfile(2) doing TLS

Questions?