STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers
Virtual Conference
September 28-29, 2021
A SNIA Event

multichannel / io_uring

Status Update within Samba

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2021-09-28

https://samba.org/~metze/presentations/2021/SDC/

Topics

- What is SMB3 Multichannel?
- Updates in Samba 4.15
- What is io-uring?
- io-uring for Samba
- Performance research, prototyping and ideas
- Questions? Feedback!

What is SMB3 Multichannel?

- Multiple transport connections are bound to one logical connection
  - This allows using more than one network link
    - Good for performance
    - Good for availability reasons
  - Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- All transport connections (channels) share the same CliendGUID
  - This is important for Samba
- An authenticated binding is done at the user session layer
  - SessionID, TreeID and FileID values are valid on all channels
- Available network interfaces are auto-negotiated
  - FSCTL_QUERY_NETWORK_INTERFACE_INFO interface list
  - IP (v4 or v6) addresses are returned together with:
    - Interface Index (which addresses belong to the same hardware)
    - Link speed
    - RSS and RDMA capabilities

What is SMB3 Multichannel?

- Multiple transport connections are bound to one logical connection
  - This allows using more than one network link
    - Good for performance
    - Good for availability reasons
  - Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- All transport connections (channels) share the same CliendGUID
  - This is important for Samba
- An authenticated binding is done at the user session layer
  - SessionID, TreeID and FileID values are valid on all channels
- Available network interfaces are auto-negotiated
  - FSCTL_QUERY_NETWORK_INTERFACE_INFO interface list
  - IP (v4 or v6) addresses are returned together with:
    - Interface Index (which addresses belong to the same hardware)
    - Link speed
    - RSS and RDMA capabilities

What is SMB3 Multichannel?

- Multiple transport connections are bound to one logical connection
  - This allows using more than one network link
    - Good for performance
    - Good for availability reasons
  - Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- All transport connections (channels) share the same CliendGUID
  - This is important for Samba
- An authenticated binding is done at the user session layer
  - SessionID, TreeID and FileID values are valid on all channels
- Available network interfaces are auto-negotiated
  - FSCTL_QUERY_NETWORK_INTERFACE_INFO interface list
  - IP (v4 or v6) addresses are returned together with:
    - Interface Index (which addresses belong to the same hardware)
    - Link speed
    - RSS and RDMA capabilities

What is SMB3 Multichannel?

- Multiple transport connections are bound to one logical connection
  - This allows using more than one network link
    - Good for performance
    - Good for availability reasons
  - Non TCP transports like RDMA (InfiniBand, RoCE, iWarp)
- All transport connections (channels) share the same CliendGUID
  - This is important for Samba
- An authenticated binding is done at the user session layer
  - SessionID, TreeID and FileID values are valid on all channels
- Available network interfaces are auto-negotiated
  - FSCTL_QUERY_NETWORK_INTERFACE_INFO interface list
  - IP (v4 or v6) addresses are returned together with:
    - Interface Index (which addresses belong to the same hardware)
    - Link speed
    - RSS and RDMA capabilities

Last Status Updates (SDC 2020 / SambaXP 2021)

- I gave a similar talk at the storage developer conference 2020:
  - See https://samba.org/~metze/presentations/2020/SDC/
  - It explains the milestones and design up to Samba 4.13 (in detail)
- I gave a similar talk at the SambaXP 2021:
  - See https://samba.org/~metze/presentations/2021/SambaXP/
  - It explains the milestones and updates up to Samba 4.15 (in detail)

Updates in Samba 4.15

- Automated regression tests are in place:
  - socket_wrapper got basic fd-passing support(Bug #11899)
  - We added a lot more multichannel related regression tests
- The last missing features/bugs are fixed (Bug #14524)
  - The connection passing is fire and forget (Bug #14433)
  - Pending async operations are canceled (Bug #14449)
- 4.15 finally has "server multi channel support = yes"
  - We require support for TIOCOUTQ (Linux) or FIONWRITE (FreeBSD)
  - We disable multichannel feature if the platform doesn't support this
    - See: Retries of Lease/Oplock Break Notifications (Bug #11898)

Updates in Samba 4.15

- Automated regression tests are in place:
  - socket_wrapper got basic fd-passing support(Bug #11899)
  - We added a lot more multichannel related regression tests
- The last missing features/bugs are fixed (Bug #14524)
  - The connection passing is fire and forget (Bug #14433)
  - Pending async operations are canceled (Bug #14449)
- 4.15 finally has "server multi channel support = yes"
  - We require support for TIOCOUTQ (Linux) or FIONWRITE (FreeBSD)
  - We disable multichannel feature if the platform doesn't support this
    - See: Retries of Lease/Oplock Break Notifications (Bug #11898)

Updates in Samba 4.15

- Automated regression tests are in place:
  - socket_wrapper got basic fd-passing support(Bug #11899)
  - We added a lot more multichannel related regression tests
- The last missing features/bugs are fixed (Bug #14524)
  - The connection passing is fire and forget (Bug #14433)
  - Pending async operations are canceled (Bug #14449)
- 4.15 finally has "server multi channel support = yes"
  - We require support for TIOCOUTQ (Linux) or FIONWRITE (FreeBSD)
  - We disable multichannel feature if the platform doesn't support this
    - See: Retries of Lease/Oplock Break Notifications (Bug #11898)

What is io-uring? (Part 1)

- Linux 5.1 introduced a new scalable AIO infrastructure
  - It's designed to avoid syscalls as much as possible
  - kernel and userspace share mmap'ed rings:
    - submission queue (SQ) ring buffer
    - completion queue (CQ) ring buffer
  - See "Ringing in a new asynchronous I/O API" on LWN.NET
- This can be nicely integrated with our async tevent model
  - It may delegate work to kernel threads
  - It seems to perform better compared to our userspace threadpool
  - It can also inline non-blocking operations

What is io-uring? (Part 1)

- Linux 5.1 introduced a new scalable AIO infrastructure
  - It's designed to avoid syscalls as much as possible
  - kernel and userspace share mmap'ed rings:
    - submission queue (SQ) ring buffer
    - completion queue (CQ) ring buffer
  - See "Ringing in a new asynchronous I/O API" on LWN.NET
- This can be nicely integrated with our async tevent model
  - It may delegate work to kernel threads
  - It seems to perform better compared to our userspace threadpool
  - It can also inline non-blocking operations

io-uring for Samba (Part 1)

- Between userspace and filesystem (available from 5.1):
  - IORING_OP_READV, IORING_OP_WRITEV and IORING_OP_FSYNC
  - Supports buffered and direct io
- Between userspace and socket (and also filesystem) (from 5.8)
  - IORING_OP_SENDMSG, IORING_OP_RECVMSG
  - Improved MSG_WAITALL support (5.12, backported to 5.11, 5.10)
  - IORING_OP_SPLICE, IORING_OP_TEE
  - Maybe using IORING_SETUP_SQPOLL or IOSQE_ASYNC
- Path based syscalls with async impersonation (from 5.6)
  - IORING_OP_OPENAT2, IORING_OP_STATX
  - Using IORING_REGISTER_PERSONALITY for impersonation
  - IORING_OP_UNLINKAT, IORING_OP_RENAMEAT (from 5.10)
  - IORING_OP_MKDIRAT, IORING_OP_SYMLINKAT, IORING_OP_LINKAT (from 5.15)

io-uring for Samba (Part 1)

- Between userspace and filesystem (available from 5.1):
  - IORING_OP_READV, IORING_OP_WRITEV and IORING_OP_FSYNC
  - Supports buffered and direct io
- Between userspace and socket (and also filesystem) (from 5.8)
  - IORING_OP_SENDMSG, IORING_OP_RECVMSG
  - Improved MSG_WAITALL support (5.12, backported to 5.11, 5.10)
  - IORING_OP_SPLICE, IORING_OP_TEE
  - Maybe using IORING_SETUP_SQPOLL or IOSQE_ASYNC
- Path based syscalls with async impersonation (from 5.6)
  - IORING_OP_OPENAT2, IORING_OP_STATX
  - Using IORING_REGISTER_PERSONALITY for impersonation
  - IORING_OP_UNLINKAT, IORING_OP_RENAMEAT (from 5.10)
  - IORING_OP_MKDIRAT, IORING_OP_SYMLINKAT, IORING_OP_LINKAT (from 5.15)

io-uring for Samba (Part 1)

- Between userspace and filesystem (available from 5.1):
  - IORING_OP_READV, IORING_OP_WRITEV and IORING_OP_FSYNC
  - Supports buffered and direct io
- Between userspace and socket (and also filesystem) (from 5.8)
  - IORING_OP_SENDMSG, IORING_OP_RECVMSG
  - Improved MSG_WAITALL support (5.12, backported to 5.11, 5.10)
  - IORING_OP_SPLICE, IORING_OP_TEE
  - Maybe using IORING_SETUP_SQPOLL or IOSQE_ASYNC
- Path based syscalls with async impersonation (from 5.6)
  - IORING_OP_OPENAT2, IORING_OP_STATX
  - Using IORING_REGISTER_PERSONALITY for impersonation
  - IORING_OP_UNLINKAT, IORING_OP_RENAMEAT (from 5.10)
  - IORING_OP_MKDIRAT, IORING_OP_SYMLINKAT, IORING_OP_LINKAT (from 5.15)

io-uring for Samba (Part 2)

IORING_FEAT_NATIVE_WORKERS (from 5.12)

- In the kernel...
  - The io-uring kernel threads are clone()'ed from the userspace thread
  - They just appear to be blocked in a syscall and never return
  - This makes the accounting in the kernel much saner
  - Allows a lot of restrictions to be relaxed in the kernel

- For admins and userspace developers...
  - They are no longer 'io_wqe_work' kernel threads
  - 'top' shows them as part of the userspace process ('H' shows them)
  - They are now visible in containers
  - 'pstree -a -t -p' is very useful to see them
  - They are shown as iou-wrk-1234, for a task with pid/tid 1234

IORING_FEAT_NATIVE_WORKERS (from 5.12)

- In the kernel...
  - The io-uring kernel threads are clone()'ed from the userspace thread
  - They just appear to be blocked in a syscall and never return
  - This makes the accounting in the kernel much saner
  - Allows a lot of restrictions to be relaxed in the kernel
- For admins and userspace developers...
  - They are no longer 'io_wqe_work' kernel threads
  - 'top' shows them as part of the userspace process ('H' shows them)
  - They are now visible in containers
  - 'pstree -a -t -p' is very useful to see them
  - They are shown as iou-wrk-1234, for a task with pid/tid 1234

vfs_io_uring in Samba 4.12 (2020)

- With Samba 4.12 we added "io_uring" vfs module
  - For now it only implements SMB_VFS_PREAD,PWRITE,FSYNC_SEND/RECV
  - It has less overhead than our pthreadpool default implementations
  - I was able to speed up a smbclient 'get largefile /dev/null'
    - Using against smbd on loopback
    - The speed changes from 2.2GBytes/s to 2.7GBytes/s
- The improvement only happens by avoiding context switches
  - But the data copying still happens:
    - From/to a userspace buffer to/from the filesystem/page cache
  - The data path between userspace and socket is completely unchanged
  - For both cases the cpu is mostly busy with memcpy

vfs_io_uring in Samba 4.12 (2020)

- With Samba 4.12 we added "io_uring" vfs module
  - For now it only implements SMB_VFS_PREAD,PWRITE,FSYNC_SEND/RECV
  - It has less overhead than our pthreadpool default implementations
  - I was able to speed up a smbclient 'get largefile /dev/null'
    - Using against smbd on loopback
    - The speed changes from 2.2GBytes/s to 2.7GBytes/s
- The improvement only happens by avoiding context switches
  - But the data copying still happens:
    - From/to a userspace buffer to/from the filesystem/page cache
  - The data path between userspace and socket is completely unchanged
  - For both cases the cpu is mostly busy with memcpy

Performance research (SMB2 Read)

- In October 2020 I was able to do some performance research
  - With 100GBit/s interfaces and two NUMA nodes per server.
- At that time I focussed on the SMB2 Read performance only
  - We had limited time on the given hardware
  - We mainly tested with fio.exe on a Windows client
  - Linux kernel 5.8.12 on the server
- More verbose details can be found here:
  - https://lists.samba.org/archive/samba-technical/2020-October/135856.html

Performance research (SMB2 Read)

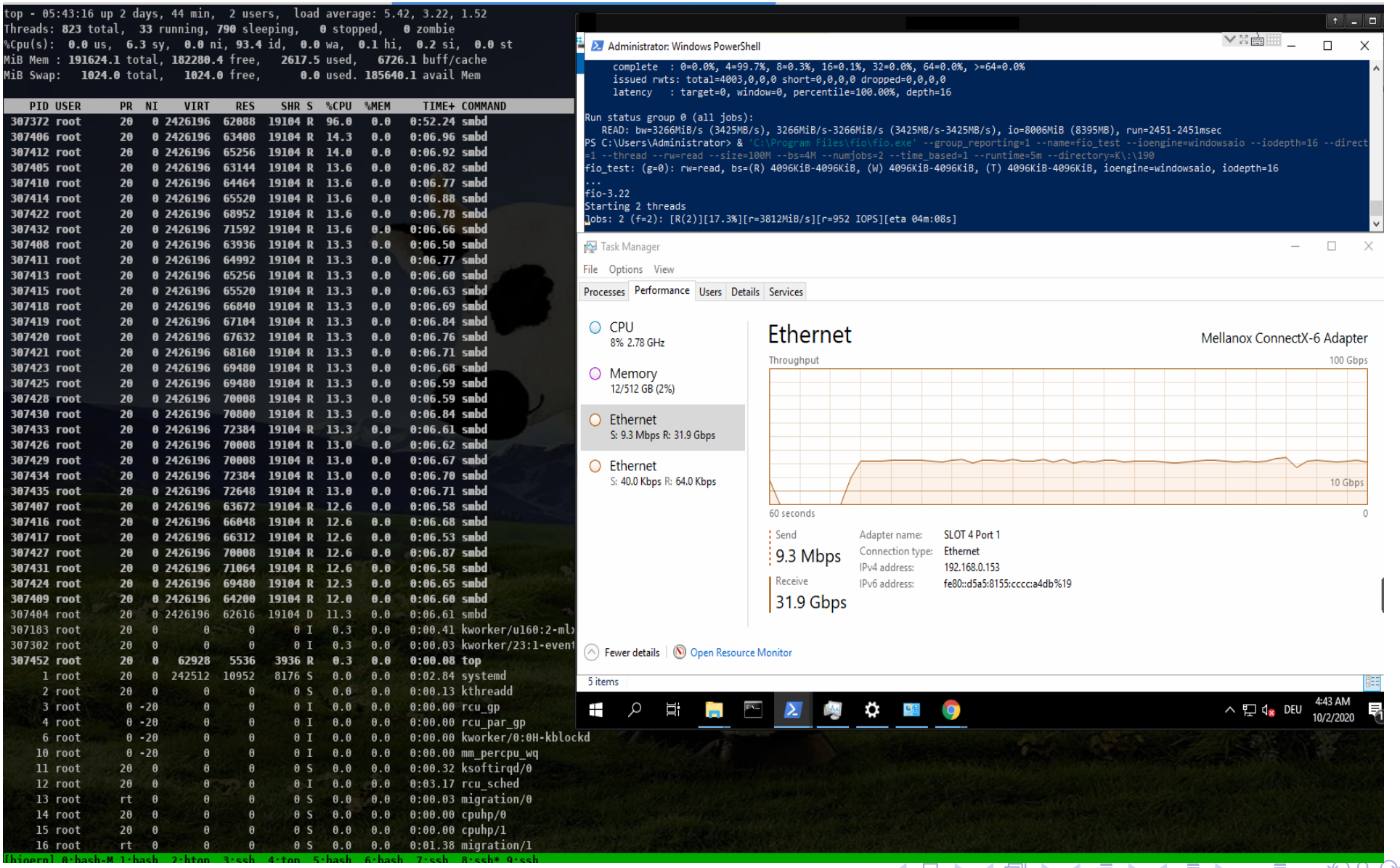
- In October 2020 I was able to do some performance research
  - With 100GBit/s interfaces and two NUMA nodes per server.
- At that time I focussed on the SMB2 Read performance only
  - We had limited time on the given hardware
  - We mainly tested with fio.exe on a Windows client
  - Linux kernel 5.8.12 on the server
- More verbose details can be found here:
  - https://lists.samba.org/archive/samba-technical/2020-October/135856.html

Performance research (SMB2 Read)

- In October 2020 I was able to do some performance research
  - With 100GBit/s interfaces and two NUMA nodes per server.
- At that time I focussed on the SMB2 Read performance only
  - We had limited time on the given hardware
  - We mainly tested with fio.exe on a Windows client
  - Linux kernel 5.8.12 on the server
- More verbose details can be found here:
  - https://lists.samba.org/archive/samba-technical/2020-October/135856.html

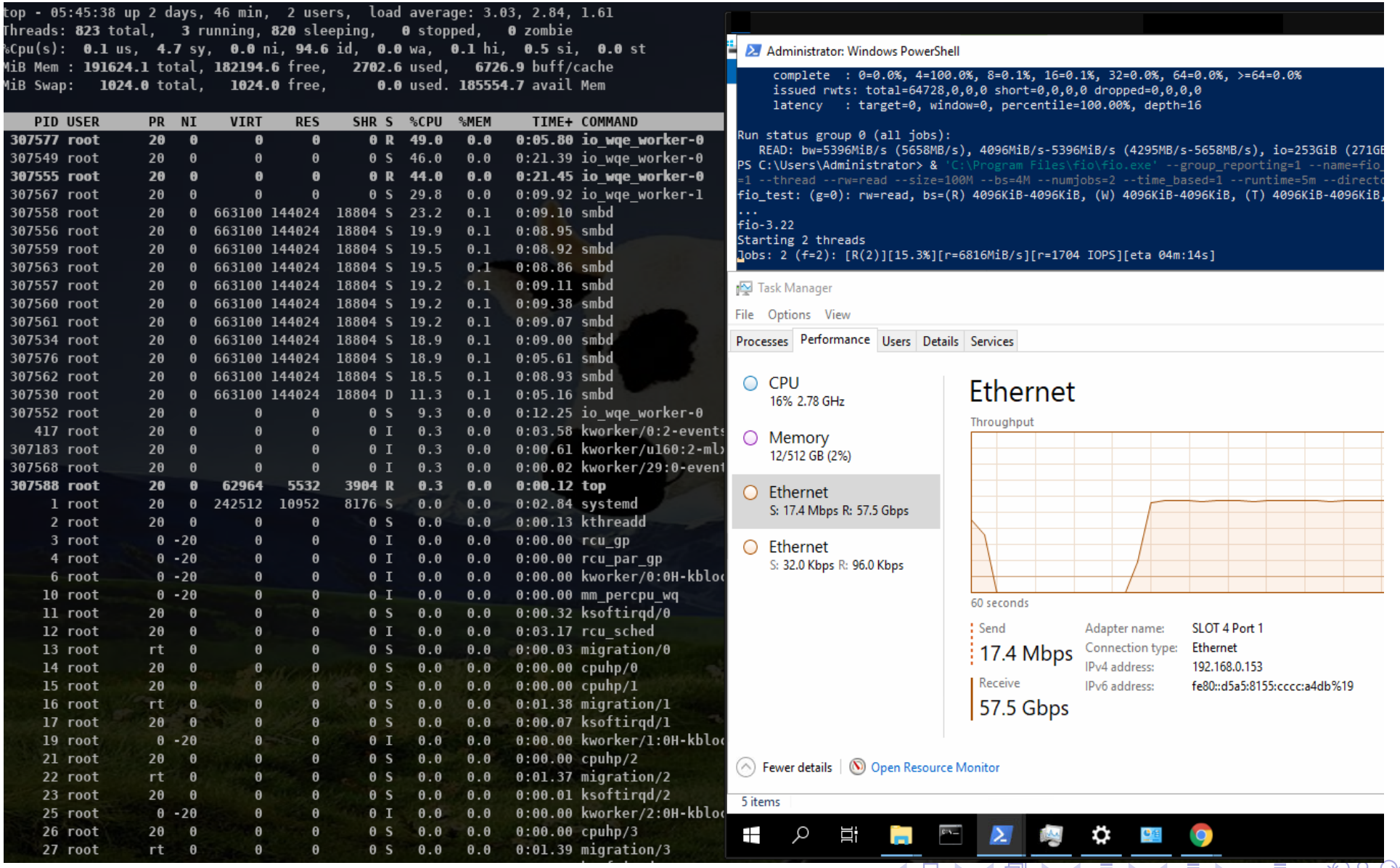
Performance with MultiChannel, sendmsg()

4 connections, ~3.8 GBytes/s, bound by >500% cpu in total, sendmsg() takes up to 0.5 msecs



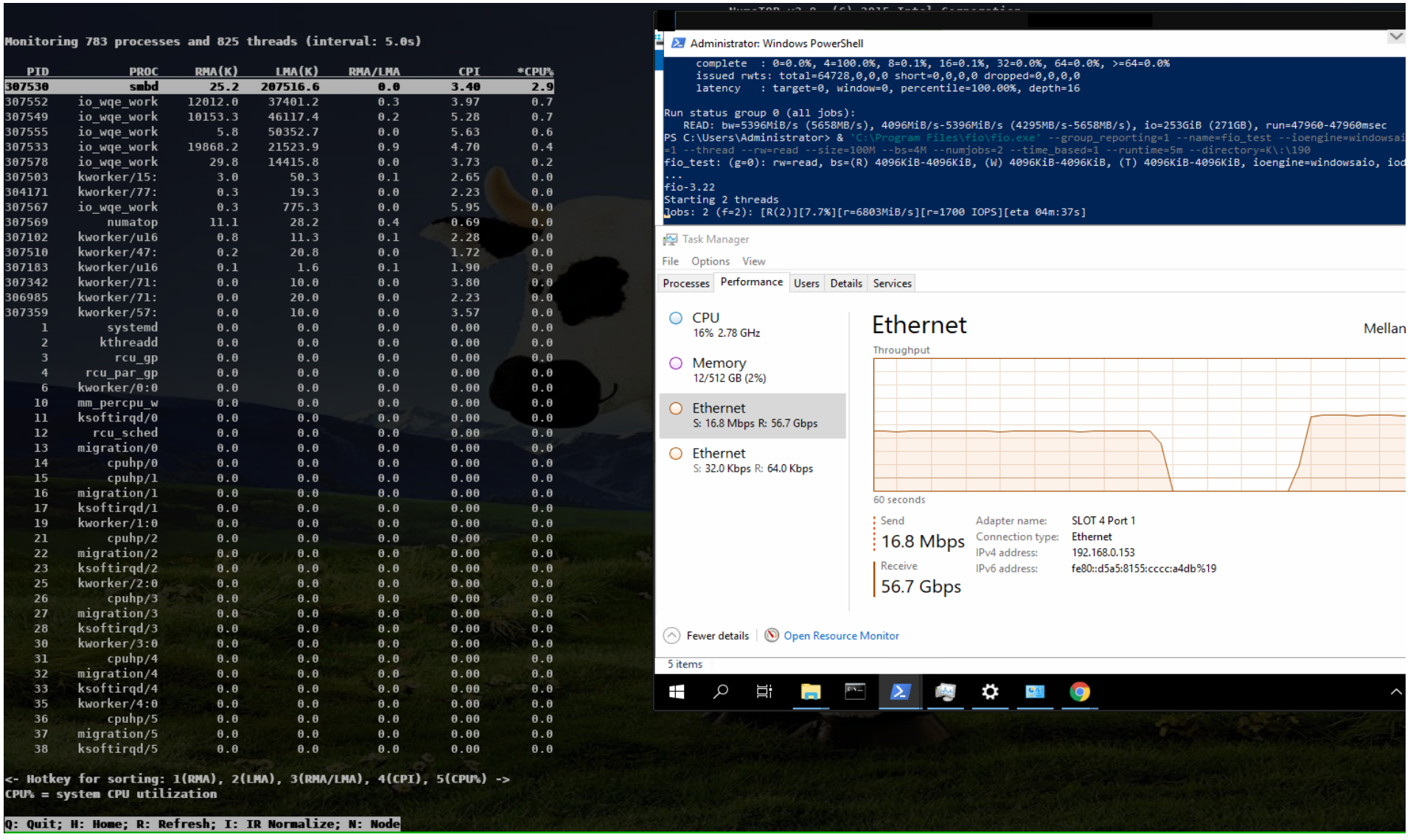
IORING_OP_SENDMSG (Part1)

4 connections, ~6.8 GBytes/s, smbd only uses ~11% cpu, (io_wqe_work ~50% cpu) per connection, we still use >300% cpu in total



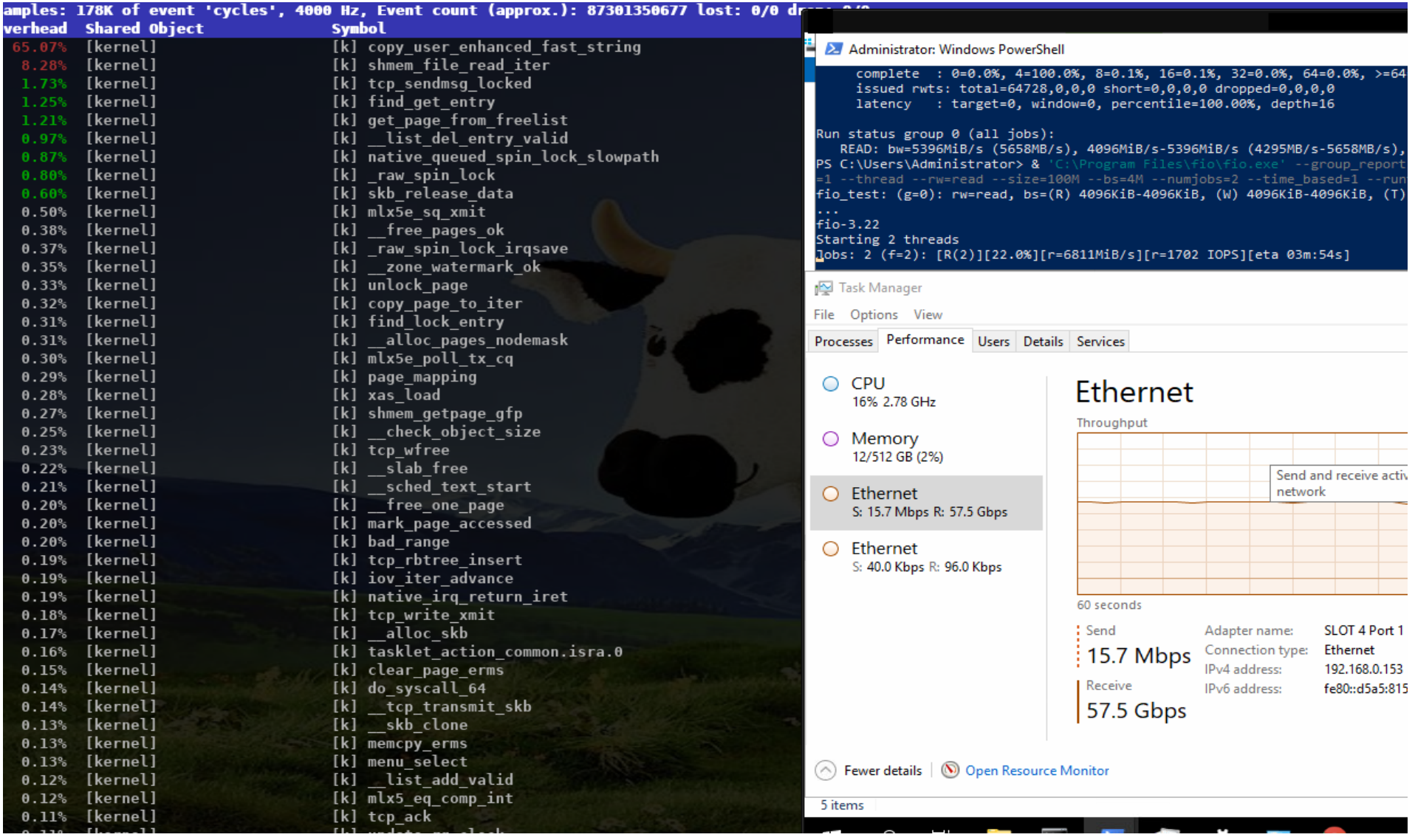
IORING_OP_SENDMSG (Part2)

The results vary havily depending on the NUMA bouncing, between 5.0 GBytes/s and 7.6 GBytes/s



IORING_OP_SENDMSG (Part3)

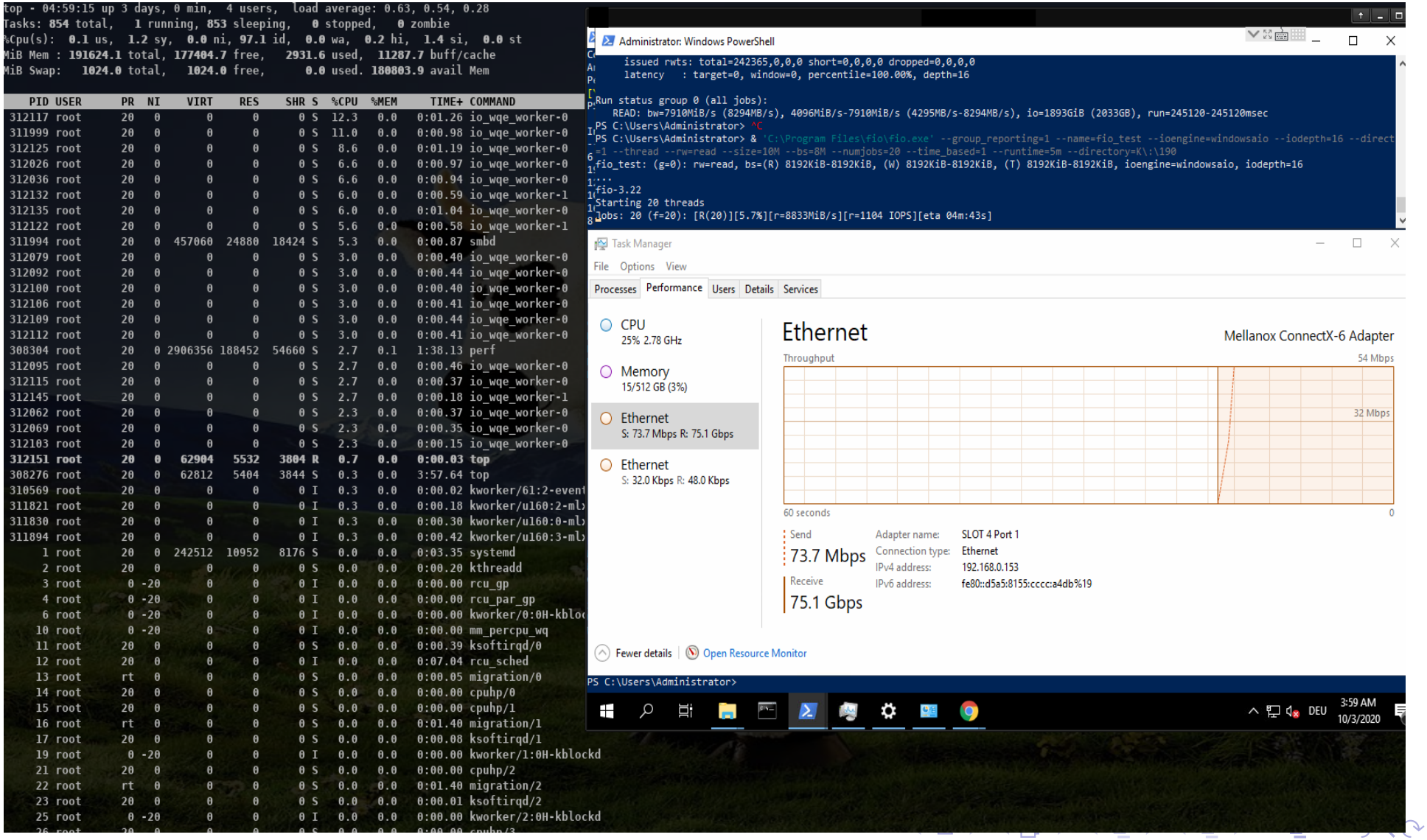
The major problem still exists, memory copy done by copy_user_enhanced_fast_string()



IORING_OP_SENDMSG + IORING_OP_SPLICE (Part1)

16 connections, ~8.9 GBytes/s, smbd ~5% cpu, (io_wqe_work 3%-12% cpu filesystem->pipe->socket), only ~100% cpu in total.

The Windows client was still the bottleneck with "Set-SmbClientConfiguration -ConnectionCountPerRssNetworkInterface 16"



smbclient IORING_OP_SENDMSG/SPLICE (network)

4 connections, ~11 GBytes/s, smbd 8.6% cpu, with 4 io_wqe_work threads (pipe to socket) at ~20% cpu each.

smbclient is the bottleneck here too

```
getting file \50G.dat of size 2097152000 as /dev/null (2771312.2 KiloBytes/sec) (average 2746784.9 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (3185069.5 KiloBytes/sec) (average 3223967.9 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (3180123.7 KiloBytes/sec) (average 3176906.8 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (2824827.2 KiloBytes/sec) (average 2828685.4 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (3255961.3 KiloBytes/sec) (average 3224002.5 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (2782608.3 KiloBytes/sec) (average 2746830.3 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (3230283.4 KiloBytes/sec) (average 3176965.0 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (3215070.2 KiloBytes/sec) (average 3223992.8 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (2790190.4 KiloBytes/sec) (average 2828636.8 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (3185069.5 KiloBytes/sec) (average 3176974.0 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (2797813.8 KiloBytes/sec) (average 2746894.5 KiloBytes/sec)
getting file \50G.dat of size 2097152000 as /dev/null (3250793.1 KiloBytes/sec) (average 3224021.8 KiloBytes/sec)
```

```
top - 02:41:58 up 17 days, 17:34,  1 user,  load average: 3.97, 4.22, 3.55
Tasks: 977 total,   5 running, 972 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  4.6 sy,  0.0 ni, 93.5 id,  0.0 wa,  0.0 hi,  1.7 si,  0.0 st
MiB Mem : 191888.7 total, 127133.7 free,   3813.5 used,  60941.4 buff/cache
MiB Swap:   1024.0 total,    737.0 free,    287.0 used. 131646.8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 740188 root      20   0  375608  35960  16852 R  99.3   0.0   9:35.55 smbclient
 740185 root      20   0  375664  36180  17016 R  99.0   0.0   9:30.87 smbclient
 740187 root      20   0  375692  35888  16696 R  88.1   0.0   8:44.88 smbclient
 740186 root      20   0  375652  35896  16740 R  86.4   0.0   8:49.20 smbclient
 108190 root      20   0   31540   7872   3412 S   2.0   0.0 180:03.15 htop
    238 root      20   0       0      0      0 S   1.3   0.0   5:56.39 ksoftirqd/45
 740176 root      20   0  249536   8076   5136 S   1.3   0.0   0:11.20 iftop
```

```
top - 02:41:57 up 3 days, 21:43,  5 users,  load average: 1.11, 0.89, 0.62
Tasks: 877 total,   1 running, 876 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  1.4 sy,  0.0 ni, 97.6 id,  0.0 wa,  0.1 hi,  0.9 si,  0.0 st
MiB Mem : 191624.1 total, 177248.5 free,   3055.5 used,  11320.1 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used. 180675.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 316136 root      20   0       0      0      0 S  21.3   0.0   0:52.01 io_wqe_worker-0
 316133 root      20   0       0      0      0 S  20.3   0.0   0:53.37 io_wqe_worker-0
 316139 root      20   0       0      0      0 S  17.9   0.0   0:40.39 io_wqe_worker-0
 316121 root      20   0       0      0      0 S  17.3   0.0   0:34.48 io_wqe_worker-0
 316116 root      20   0  458080  21264  17652 S   8.6   0.0   0:48.53 smbd
```

```
Samples: 70M of event 'cycles', 4000 Hz, Event count (approx.): 35340326236 lost: 0/0 drop: 0/32890
Overhead  Shared Object                Symbol
   7.85%  [kernel]                     [k] do_tcp_sendpages
   5.37%  [kernel]                     [k] _raw_spin_lock_bh
   4.08%  [kernel]                     [k] copy_page_to_iter
   3.75%  [kernel]                     [k] page_cache_pipe_buf_release
   3.25%  [kernel]                     [k] __x86_retpoline_rax
   3.20%  [kernel]                     [k] page_cache_pipe_buf_confirm
   2.87%  [kernel]                     [k] native_queued_spin_lock_slowpath
   2.84%  [kernel]                     [k] shmem_file_read_iter
   2.78%  [kernel]                     [k] inet_sendpage
   2.61%  [kernel]                     [k] tcp_sendpage
For a higher level overview, try: perf top --sort comm,dso
```

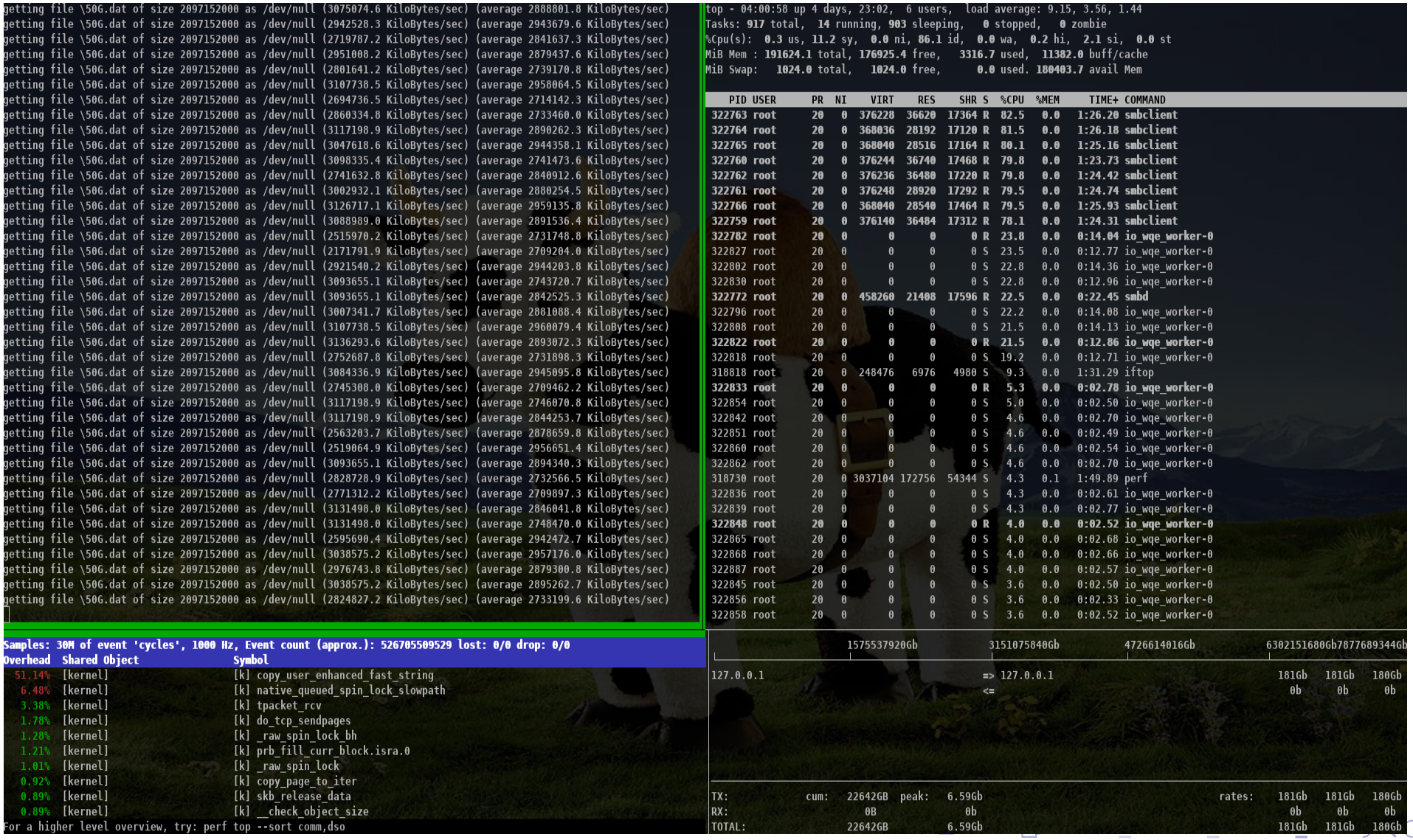
```
                                1546030464Gb          3092060928Gb          4638091264Gb          6184121856Gb7730152448Gb
192.168.10.191                   => 192.168.10.190                                                 91.7Gb  91.5Gb  89.7Gb
                                 <=                                                                18.3Mb  18.7Mb  19.6Mb
192.168.10.191                   => 192.168.0.153                                                     0b      0b    238b
                                 <=                                                                   0b      0b    210b

TX:             cum:   314GB   peak:     0b                                    rates:    91.7Gb  91.5Gb  89.7Gb
RX:                    68.7MB           22.1Mb                                          18.3Mb  18.7Mb  19.6Mb
TOTAL:                 314GB             0b                                            91.8Gb  91.5Gb  89.7Gb
```

smbclient IORING_OP_SENDMSG/SPLICE (loopback)

8 connections, ~22 GBytes/s, smbd 22% cpu, with 4 io_wqe_work threads (pipe to socket) at ~22% cpu each.

smbclient is the bottleneck here too, it triggers the memory copy done by copy_user_enhanced_fast_string()



More loopback testing on brand new hardware

- Recently I re-did the loopback read tests IORING_OP_SENDMSG/SPLICE (from /dev/shm/)
  - 1 connection, ~10-13 GBytes/s, smbd 7% cpu, with 4 iou-wrk threads at 7%-50% cpu.
  - 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 iou-wrk threads at 3%-35% cpu.
- I also implemented SMB2 writes with IORING_OP_RECVMSG/SPLICE (tested to /dev/null)
  - 1 connection, ~7-8 GBytes/s, smbd 5% cpu, with 3 io-wrk threads at 1%-20% cpu.
  - 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io-wrk threads at 1%-20% cpu.
- I tested with a Linux Kernel 5.13
  - In both cases the bottleneck is clearly on the smbclient side
  - We could apply similar changes to smbclient and add true multichannel support
  - It seems that the filesystem->pipe->socket path is much better optimized

More loopback testing on brand new hardware

- Recently I re-did the loopback read tests IORING_OP_SENDMSG/SPLICE (from /dev/shm/)
  - 1 connection, ~10-13 GBytes/s, smbd 7% cpu, with 4 iou-wrk threads at 7%-50% cpu.
  - 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 iou-wrk threads at 3%-35% cpu.
- I also implemented SMB2 writes with IORING_OP_RECVMSG/SPLICE (tested to /dev/null)
  - 1 connection, ~7-8 GBytes/s, smbd 5% cpu, with 3 io-wrk threads at 1%-20% cpu.
  - 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io-wrk threads at 1%-20% cpu.
- I tested with a Linux Kernel 5.13
  - In both cases the bottleneck is clearly on the smbclient side
  - We could apply similar changes to smbclient and add true multichannel support
  - It seems that the filesystem->pipe->socket path is much better optimized

More loopback testing on brand new hardware

- Recently I re-did the loopback read tests IORING_OP_SENDMSG/SPLICE (from /dev/shm/)
  - 1 connection, ~10-13 GBytes/s, smbd 7% cpu, with 4 iou-wrk threads at 7%-50% cpu.
  - 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 iou-wrk threads at 3%-35% cpu.
- I also implemented SMB2 writes with IORING_OP_RECVMSG/SPLICE (tested to /dev/null)
  - 1 connection, ~7-8 GBytes/s, smbd 5% cpu, with 3 io-wrk threads at 1%-20% cpu.
  - 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io-wrk threads at 1%-20% cpu.
- I tested with a Linux Kernel 5.13
  - In both cases the bottleneck is clearly on the smbclient side
  - We could apply similar changes to smbclient and add true multichannel support
  - It seems that the filesystem->pipe->socket path is much better optimized

Improvements for transfers with SMB3 signing

- Samba 4.15 has support for AES-128-GMAC signing:
  - This is also available in recent Windows versions
  - It's based on AES-128-GCM (but only with authentication data)
  - The gnutls library is able to provide:
    - ~6 GBytes/s for AES-128-GCM
    - ~10 GBytes/s for AES-128-GMAC
- For SMB3 signing/encryption we use:
  - IORING_OP_SPLICE from a file into a (splice)pipe
  - IORING_OP_TEE from the (splice)pipe to a 2nd (tee)pipe
  - IORING_OP_READ from the (tee)pipe into a userspace buffer
    - (vmsplice might work even better)
  - The userspace buffer is only used to calculate the signing signature
  - IORING_OP_SENDMSG and IORING_OP_SPLICE are used in order to avoid a copy back to the kernel
- For a SMB2 read test I removed the signing check in smbclient:
  - The performance changed from ~3 GBytes/s before
  - To ~5 GBytes/s using the IORING_OP_TEE trick
    - With smbclient still being the bootleneck at 100% cpu

Improvements for transfers with SMB3 signing

- Samba 4.15 has support for AES-128-GMAC signing:
  - This is also available in recent Windows versions
  - It's based on AES-128-GCM (but only with authentication data)
  - The gnutls library is able to provide:
    - ~6 GBytes/s for AES-128-GCM
    - ~10 GBytes/s for AES-128-GMAC
- For SMB3 signing/encryption we use:
  - IORING_OP_SPLICE from a file into a (splice)pipe
  - IORING_OP_TEE from the (splice)pipe to a 2nd (tee)pipe
  - IORING_OP_READ from the (tee)pipe into a userspace buffer
    - (vmsplice might work even better)
  - The userspace buffer is only used to calculate the signing signature
  - IORING_OP_SENDMSG and IORING_OP_SPLICE are used in order to avoid a copy back to the kernel
- For a SMB2 read test I removed the signing check in smbclient:
  - The performance changed from ~3 GBytes/s before
  - To ~5 GBytes/s using the IORING_OP_TEE trick
    - With smbclient still being the bootleneck at 100% cpu

Improvements for transfers with SMB3 signing

- Samba 4.15 has support for AES-128-GMAC signing:
  - This is also available in recent Windows versions
  - It's based on AES-128-GCM (but only with authentication data)
  - The gnutls library is able to provide:
    - ~6 GBytes/s for AES-128-GCM
    - ~10 GBytes/s for AES-128-GMAC
- For SMB3 signing/encryption we use:
  - IORING_OP_SPLICE from a file into a (splice)pipe
  - IORING_OP_TEE from the (splice)pipe to a 2nd (tee)pipe
  - IORING_OP_READ from the (tee)pipe into a userspace buffer
    - (vmsplice might work even better)
  - The userspace buffer is only used to calculate the signing signature
  - IORING_OP_SENDMSG and IORING_OP_SPLICE are used in order to avoid a copy back to the kernel
- For a SMB2 read test I removed the signing check in smbclient:
  - The performance changed from ~3 GBytes/s before
  - To ~5 GBytes/s using the IORING_OP_TEE trick
    - With smbclient still being the bootleneck at 100% cpu

Future Improvements

- recvmsg and splice deliver partial SMB packets to userspace
  - I tested with AF_KCM (Kernel Connection Multiplexor) and an eBPF helper
  - But MSG_WAITALL is the much simpler and faster solution
  - I also prototyped a SPLICE_F_WAITALL
  - eBPF support in io-uring would also be great for optimizations
- It also seems that socket->pipe->filesystem:
  - Does not implement zero copy for all cases
  - Maybe it's possible to optimize this in future
- In the end SMB-Direct will also be able to reduce overhead
  - My smbdirect driver is still work in progress...
  - With the IORING_FEAT_NATIVE_WORKERS feature it will be possible glue it to IORING_OP_SENDMSG

Future Improvements

- recvmsg and splice deliver partial SMB packets to userspace
  - I tested with AF_KCM (Kernel Connection Multiplexor) and an eBPF helper
  - But MSG_WAITALL is the much simpler and faster solution
  - I also prototyped a SPLICE_F_WAITALL
  - eBPF support in io-uring would also be great for optimizations
- It also seems that socket->pipe->filesystem:
  - Does not implement zero copy for all cases
  - Maybe it's possible to optimize this in future
- In the end SMB-Direct will also be able to reduce overhead
  - My smbdirect driver is still work in progress...
  - With the IORING_FEAT_NATIVE_WORKERS feature it will be possible glue it to IORING_OP_SENDMSG

Future Improvements

- recvmsg and splice deliver partial SMB packets to userspace
  - I tested with AF_KCM (Kernel Connection Multiplexor) and an eBPF helper
  - But MSG_WAITALL is the much simpler and faster solution
  - I also prototyped a SPLICE_F_WAITALL
  - eBPF support in io-uring would also be great for optimizations
- It also seems that socket->pipe->filesystem:
  - Does not implement zero copy for all cases
  - Maybe it's possible to optimize this in future
- In the end SMB-Direct will also be able to reduce overhead
  - My smbdirect driver is still work in progress...
  - With the IORING_FEAT_NATIVE_WORKERS feature it will be possible glue it to IORING_OP_SENDMSG

Questions? Feedback!

- Feedback regarding real world testing would be great!

- Stefan Metzmacher, `metze@samba.org`
- https://www.sernet.com
- https://samba.plus

Slides: https://samba.org/~metze/presentations/2021/SDC/