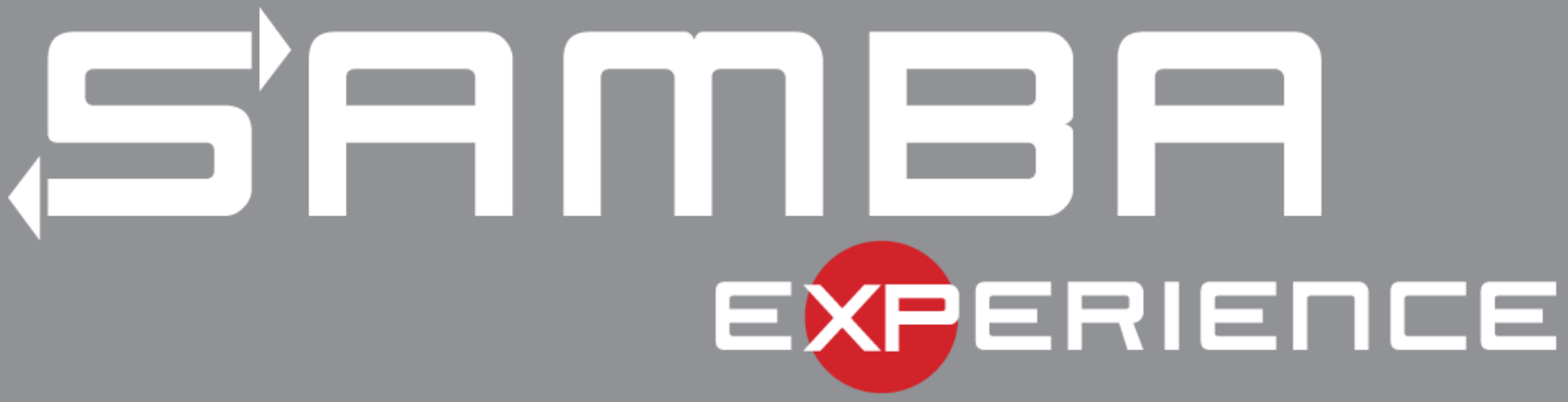


New Transports in Samba

quic + highspeed with io_uring and smbdirect ...

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2026-04-20

https://samba.org/~metze/presentations/2026/SambaXP/

Topics

- Existing Transports: NBT, TCP
- New Transports: SMBDIRECT, QUIC
- QUIC support using quic.ko on Linux
- How QUIC is tested without kernel support
- SMBDIRECT support for Linux and Samba
- How we test without kernel support
- Generic improvements using io_uring
- Performance boost with io_uring
- smb2.bench.read performance examples
- Samba is back in the game
- Questions? Feedback!

Last Status Updates for io_uring and smbdirect

- smbdirect support within Samba and Linux (SDC 2018):
  - See https://samba.org/~metze/presentations/2018/SDC/
  - It explains the initial research and prototype phase (in detail)
- io_uring status update within Samba (SDC 2020):
  - See https://samba.org/~metze/presentations/2020/SDC/
  - It explains the milestones and design up to Samba 4.13 (in detail)
- io_uring status update within Samba (SDC 2021):
  - See https://samba.org/~metze/presentations/2021/SDC/
  - It explains the milestones and updates up to Samba 4.15 (in detail)
- io_uring status update within Samba (SambaXP 2023):
  - See https://samba.org/~metze/presentations/2023/SambaXP/
  - It explains the milestones and updates up to Samba 4.19 (in detail)
- new transports in Samba, quic and smbdirect support (SDC 2025):
  - See https://samba.org/~metze/presentations/2025/SDC/
  - It explains the milestones and updates up to Samba 4.23 (in detail)

Existing Transports: NBT, TCP

- SMB Versions 1, 2 and 3 operate on message based transports
  - Each transport message can transport one or more SMB PDUs
- The oldest transport supported by Samba is NBT over TCP
  - This uses tcp port 139
  - Message header: 8 bits type, 7 bits reserved, 17 bits length
  - Simple exchange of netbios names at the start of a connection
  - Payload header (type=0): 4 bytes big endian, max ~128KiB (0x1FFFF)
  - Keepalive messages (type=85)
- Direct TCP is also (almost forever) available in Samba
  - This uses tcp port 445
  - Message header: 8 bits zero, 24 bits length
  - Just payload messages: 4 bytes big endian, max ~16MiB (0xFFFFFF)
- Both operate just on BSD stream sockets for TCP
  - They are identical to implement for the payload messages

New Transports: SMBDIRECT

- SMBDIRECT (a tiny layer on top of RDMA/IB Verbs APIs)
  - Infiniband, RoCE (v1 and v2) and iWarp are supported protocols
  - A message based transport is implement using SEND[_INV] verbs
  - SEND verbs are also message based but only allow small messages
  - Payload messages are reassembled, typically up to 1MiB (0x100000)
  - RDMA Read/Write verbs are used as out of band memory copy
- Windows 2012 introduced it to be used via SMB3 multi-channel
  - Clients start with TCP and ask for server interface IP addresses
  - The client connects via SMBDIRECT is the interface is RDMA capable
  - But the connection only uses IP addresses, so Infinibad requires IPoIB
- SMB2/3 Read/Write can offload data transfer to RDMA Read/Write
  - The client tells the server what client memory the server can read/write
  - The server is the initiator for RDMA Read/Write
  - SMB2 Read results in RDMA Write from server to client
  - SMB2 Write results in RDMA Read from client to server

New Transports: QUIC

- QUIC a rich stream protocol based on TLS 1.3 and UDP
  - Designed for HTTP/3 with security, performance and robustness
  - Used UDP port 443 and should go through routers and firewalls
  - Supports Application-Layer Protocol Negotiation (ALPN)
  - ALPN: "h3" for HTTP/3, "smb" for SMB3
  - Multiple logical stream over a single QUIC connection
- Windows 2022 introduced SMB3 over QUIC
  - Clients try TCP first and fallback to QUIC
  - Only a single logical stream is used per QUIC connection
  - Message header: 8 bits zero, 24 bits length (Same as Direct TCP!)
  - Clients need to access the server via a name
  - Via ip address the cerfiticate would not match
  - SMB3 encryption is done in addition to the QUIC encryption
- Windows 2025 introduced new features
  - The double encryption of QUIC and SMB3 layer is optional
  - SMB2_ACCEPT_TRANSPORT_LEVEL_SECURITY can be negotiated
  - Server can required client certificates

QUIC support using quic.ko on Linux

- quic.ko for Linux provides a BSD stream socket
  - Standalone: https://github.com/lxin/quic
  - It uses socket(AF_INET[6], SOCK_STREAM, IPPROTO_QUIC);
  - Userspace needs to call quic_handshake() until connection is ready
  - From there it is just a stream socket similar to TCP
- The changes to Samba are not very complex
  - We just need to replace IPPROTO_TCP with IPPROTO_QUIC
  - Then setup the gnutls bases tls structures
  - And call quic_handshake()
  - The message processing is not changed at all
  - Only smbstatus –json output was asjusted to show the transport
  - New options "client smb transports" and "server smb transport"
- Linux quic.ko will hopefully mainlined soon
  - Review for v11 of the patches:
  - https://lore.kernel.org/quic/cover.1774410440.git.lucien.xin@gmail.com/

QUIC support with only UDP sockets from the kernel

- On systems without IPPROTO_QUIC support we use libngtcp2
  - It provides userspace QUIC support on top of UDP sockets
  - It is relatively easy to use with gnutls
  - We can use the same handshake code as with quic.ko
  - It can operate non-blocking and has callback apis
- It is easy to use in Samba on the client side
  - It plugs nicely into our tevent and tstream abstractions
  - We just use it as fallback if quic.ko is not available
  - Should also work on non Linux systems
- Server support is not possible
  - We rely stream sockets and fd-passing for multichannel
  - Sharing a single UDP socket would be very complex

SMBDIRECT support for Linux and Samba

- The Linux kernel already has some support for SMBDIRECT
  - The client (cifs.ko) has an "rdma" mount option
  - The server (ksmbd.ko) listens on any available rdma interface
  - Both had their own implementation (partly copied)
- The recent 7.1 merge windows included a new smbdirect.ko
  - After a long journey and ~420 small patches
  - cifs.ko and ksmbd.ko use a common smbdirect.ko
- There are IPPROTO_SMBDIRECT patches for in kernel use
  - cifs.ko and ksmbd.ko use the generic socket layer where possible
  - https://lore.kernel.org/linux-cifs/cover.1775571957.git.metze@samba.org/
- IPPROTO_SMBDIRECT has a working prototype for userspace
  - https://git.samba.org/?p=metze/linux/wip.git;a=shortlog;h=refs/heads/master-ipproto-smbdirect
  - It basically works and but still needs a bit cleanup for upstream
- The Samba part is basically finished
  - https://gitlab.com/samba-team/samba/-/merge_requests/4343
  - It works and shows very good results (examples follow :-)

How we test without kernel support

- We use socket_wrapper in Samba testing
  - It emulates TCP and UDP sockets using unix domain sockets
  - It uses LD_PRELOAD in order to inject itself into syscalls
  - socket_wrapper_ipproto_quic_socket() uses SOCK_SEQPACKET
  - socket_wrapper_ipproto_smbdirect_socket() uses SOCK_STREAM
- quic_ko_wrapper was created in order to emulate quic.ko
  - It works on top of socket_wrapper_ipproto_quic_socket()
  - SOCK_SEQPACKET on unix sockets provides the connection semantic
  - The QUIC crypto is implemented using libngtcp2 too
- smbdirect_ko_wrapper was created in order to emulate smbdirect.ko
  - It works on top of socket_wrapper_ipproto_smbdirect_socket()
  - process_vm_readv()/process_vm_writev() for RDMA Read/Write
  - Thanks, Volker!
- They are good enough for regression testing in Samba
  - We run a few tests to explore the related code paths

samba_io_uring abstraction (Part 1)

API glue to tevent:

```
void samba_io_uring_ev_register(void);

const struct samba_io_uring_features *samba_io_uring_system_features(void);

struct samba_io_uring *samba_io_uring_ev_context_get_ring(struct tevent_context *ev);

const struct samba_io_uring_features *samba_io_uring_get_features(
                                        const struct samba_io_uring *ring);

ev = tevent_context_init_byname(mem_ctx, "samba_io_uring_ev");
```

- samba_io_uring abstraction factored out of vfs_io_uring:
  - samba_io_uring_ev_hybrid tevent backend (glued on epoll backend)
  - It means every layer getting the tevent_context can use io_uring
  - No #ifdef's just checking if the required features are available

samba_io_uring abstraction (Part 2)

generic submission/completion api:

```
void samba_io_uring_completion_prepare(struct samba_io_uring_completion *completion,
            void (*completion_fn)(struct samba_io_uring_completion *completion,
                                  void *completion_private,
                                  const struct io_uring_cqe *cqe),
            void *completion_private);

void samba_io_uring_submission_prepare(struct samba_io_uring_submission *submission,
            void (*submission_fn)(struct samba_io_uring *ring,
                                  struct samba_io_uring_submission *submission,
                                  void *submission_private),
            void *submission_private,
            struct samba_io_uring_completion *completion);

struct io_uring_sqe *samba_io_uring_submission_sqe(struct samba_io_uring_submission *
     submission);

size_t samba_io_uring_queue_submissions(struct samba_io_uring *ring,
                                        struct samba_io_uring_submission *submission);
int samba_io_uring_submit_sqes(struct samba_io_uring *ring);
```

- Using it ...
  - convert vfs_io_uring
  - use it in smb2_server.c
  - In future use it in other performance critical places too.

performance boost with io_uring (Part 1)

- sendmsg() and recvmsg() are performance killers
  - They block the main process burning cpu doing memcpy
  - Even on highspeed networks only a few GBytes/s are possible
- Refactoring of smb2_server.c
  - add optional IORING_OP_SENDMSG, IORING_OP_RECVMSG support
  - using IOSQE_ASYNC in order to memcpy in helper threads
  - improves multichannel between 30% and 100% depending on the setup
- With IORING_OP_SENDMSG_ZC only 1 one copy is used:
  - It is able to avoid copying to the socket
  - We get an extra completion once the buffers are not needed anymore
  - Only with real hardware, not on loopback in an upstream kernel
  - improves again between 30% and 60% depending on the setup
  - reduces the server cpu usage by 100% and more
  - currently tcp only, but smbdirect will follow

performance boost with io_uring (Part 2)

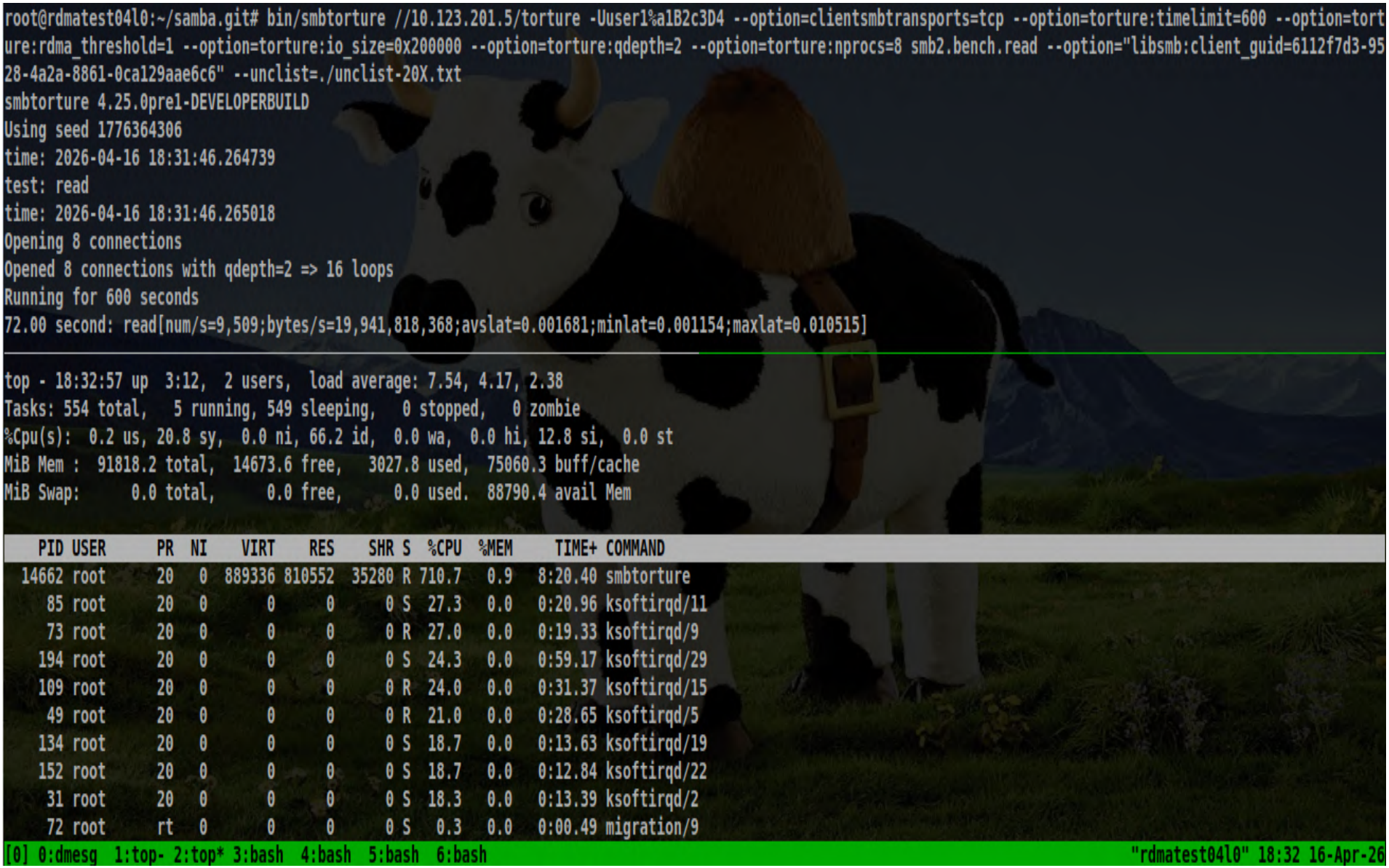
- Usage of IORING_OP_SPLICE socket ->pipe ->filesystem
  - IORING_OP_RECVMSG only for the header
  - IORING_OP_SPLICE from socket to a pipe (zero-copy)
  - IORING_OP_SPLICE from pipe to file (copy)
  - currently tcp only, but would work without problems
  - rdma uses RDMA READ offload instead
- Usage of IORING_OP_SPLICE filesystem ->pipe ->socket
  - IORING_OP_SPLICE from file to pipe (zero-copy from page cache)
  - IORING_OP_SENDMSG[_ZC] only for the header
  - IORING_OP_SPLICE from pipe to socket (for tcp)
  - SMBDIRECT_URING_CMD_BUFFER_DO_REMOTE_SPLICE_WRITE
  - (pipe to RDMA WRITE offload)
- There are structural problems with splice from a file
  - I had a discussion with the Linux developers about it:
  - The page content from the page cache may change unexpectetly
  - https://lists.samba.org/archive/samba-technical/2023-February/thread.html#137945
  - So this may not be the default, but maybe there will tricks to handle it

performance testsetup

- 2 Servers with a AMD Ryzen 9 9950X3D 16-Core Processor
  - 96 GB DRAM (2 x 48 GB DDR5 5600 MT/s)
  - PCIe 5.0
  - 2x SAMSUNG MZ7L3960 SATA-SSD (Windows2025/Ubuntu26.04)
  - Ubuntu 26.04 used a custom 7.0 kernel and custom samba
  - 2x Samsung NVMe SSD 9100 PRO (DATA: Windows/Linux)
  - Mallanox ConnectX-7 (RoCEv2) with 2x 200GBit/s ports
  - The 200GBit/s links are connected via direct cables
- Additional RDMA cards not always connected
  - For additional correctness testing we can add them
  - Mallanox ConnectX-7 also supports InfiniBand
  - Intel-E810-CQDA2 2x 100GBit/s (iWarp and RoCEv2)
  - Chelsio T520-BT 2x 10GBit/s (iWarp)

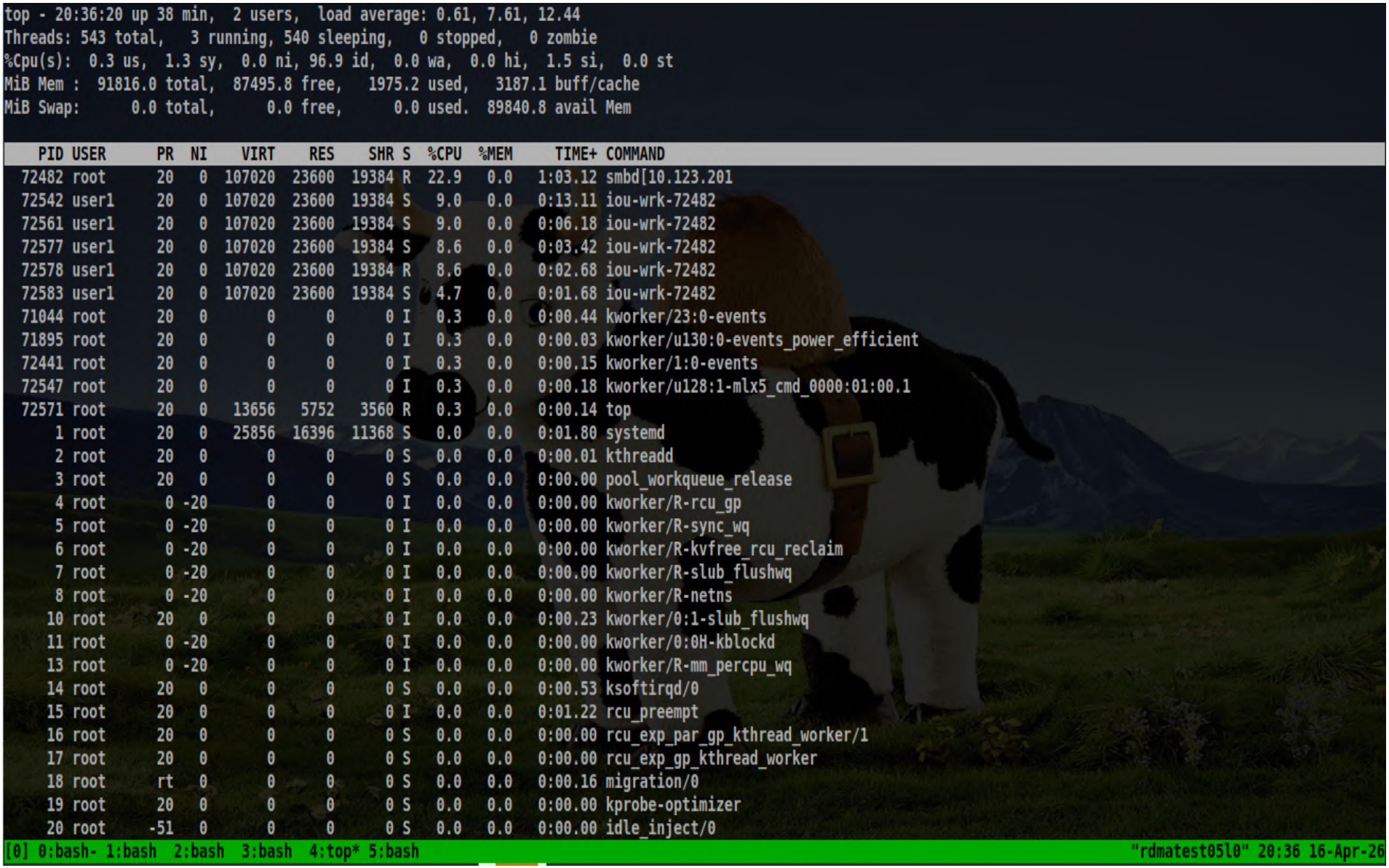
smb2.bench.read io_size=2MB tcp splice (Samba)

single client, 8 connections, qdepth=2, ~20 GBytes/s, ~700% client cpu, 2x200GBit/s interfaces



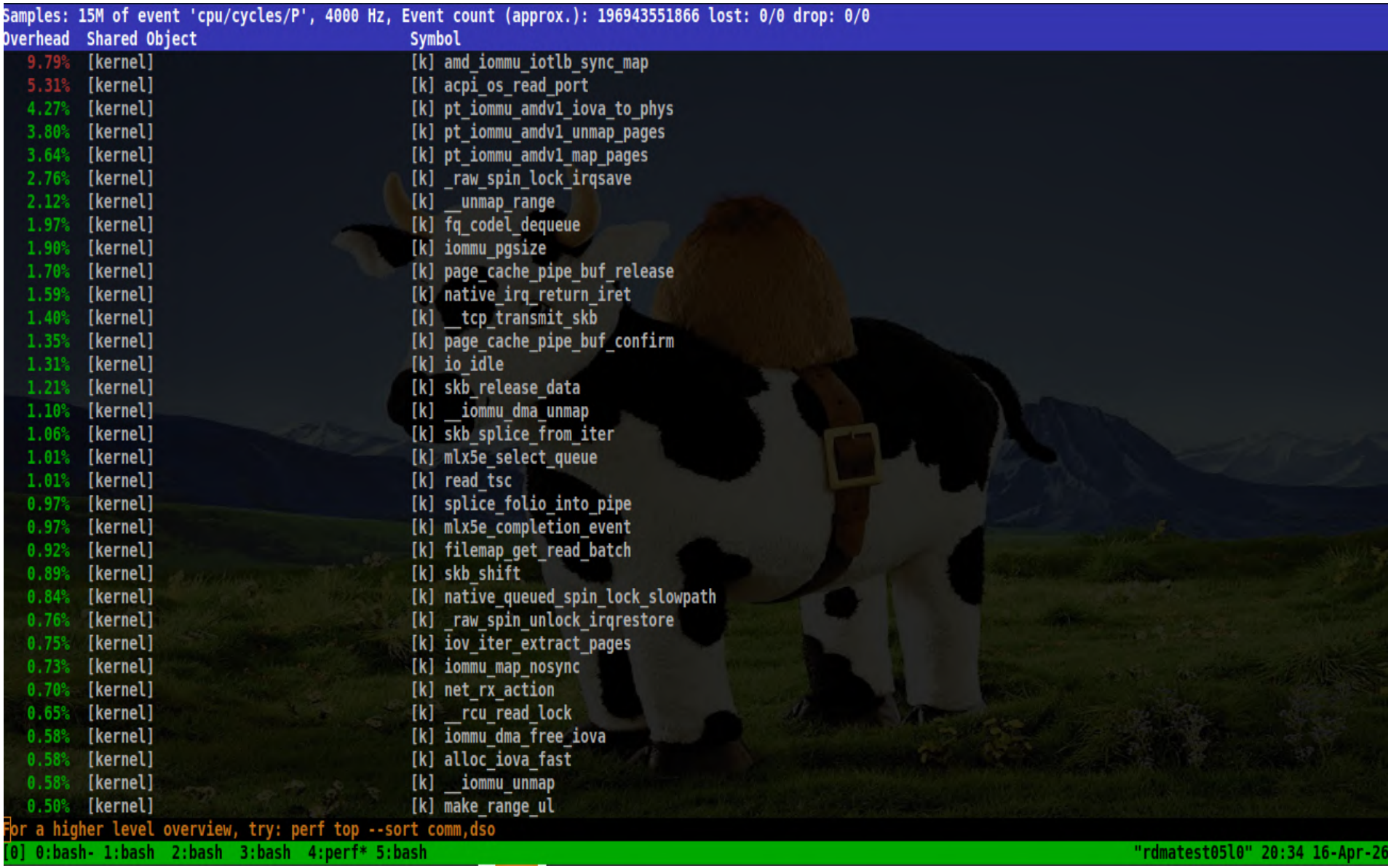
smb2.bench.read io_size=2MB tcp splice (Samba top)

single smbd + iou-wrk threads, 8 connections, qdepth=2, ~20 GBytes/s, ~70% server cpu, 2x200GBit/s interfaces



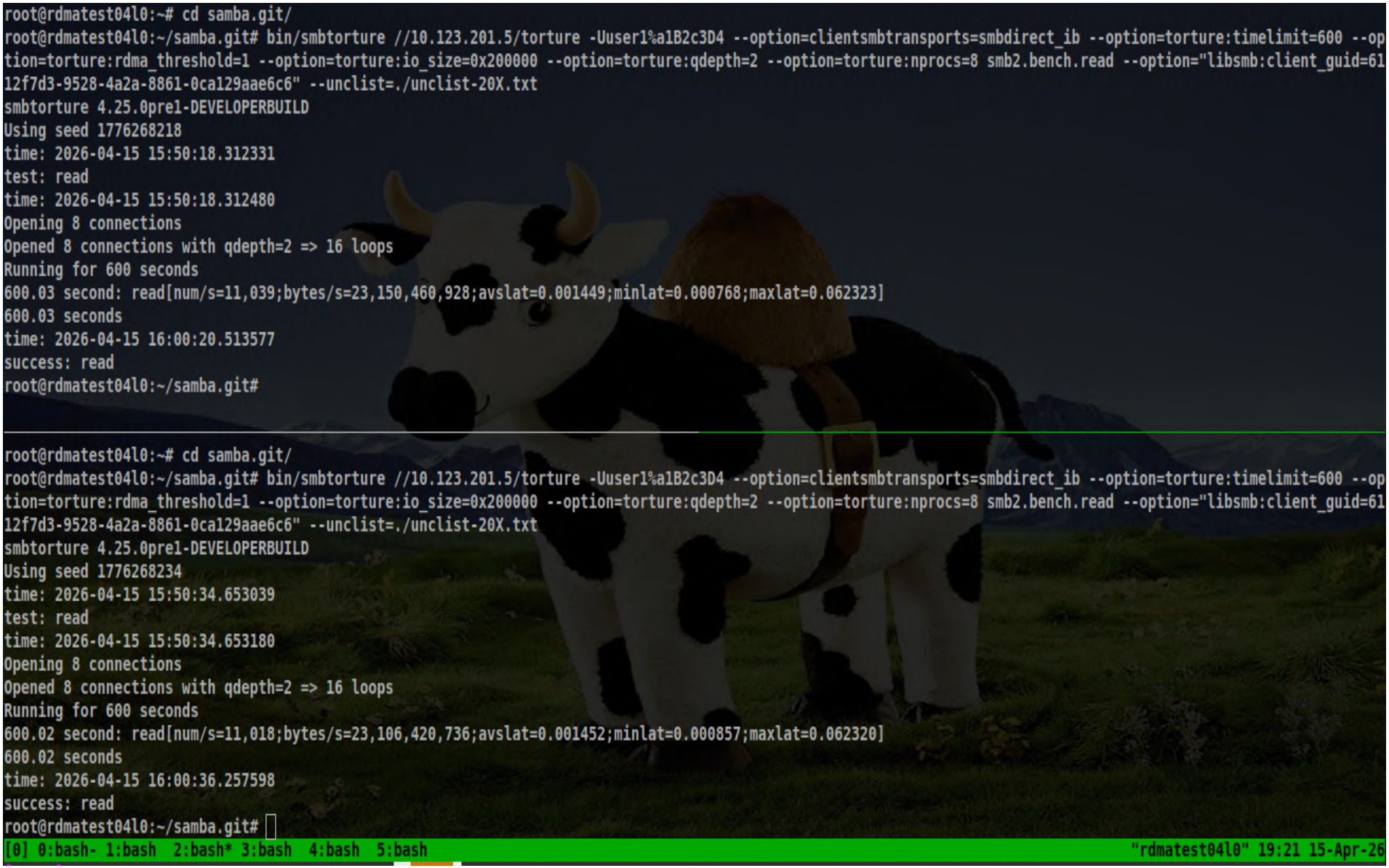
smb2.bench.read io_size=2MB tcp splice (Samba perf top)

single smbd + iou-wrk threads, 8 connections, qdepth=2, ~20 GBytes/s, no copy_{to,from}_iter, 2x200GBit/s interfaces



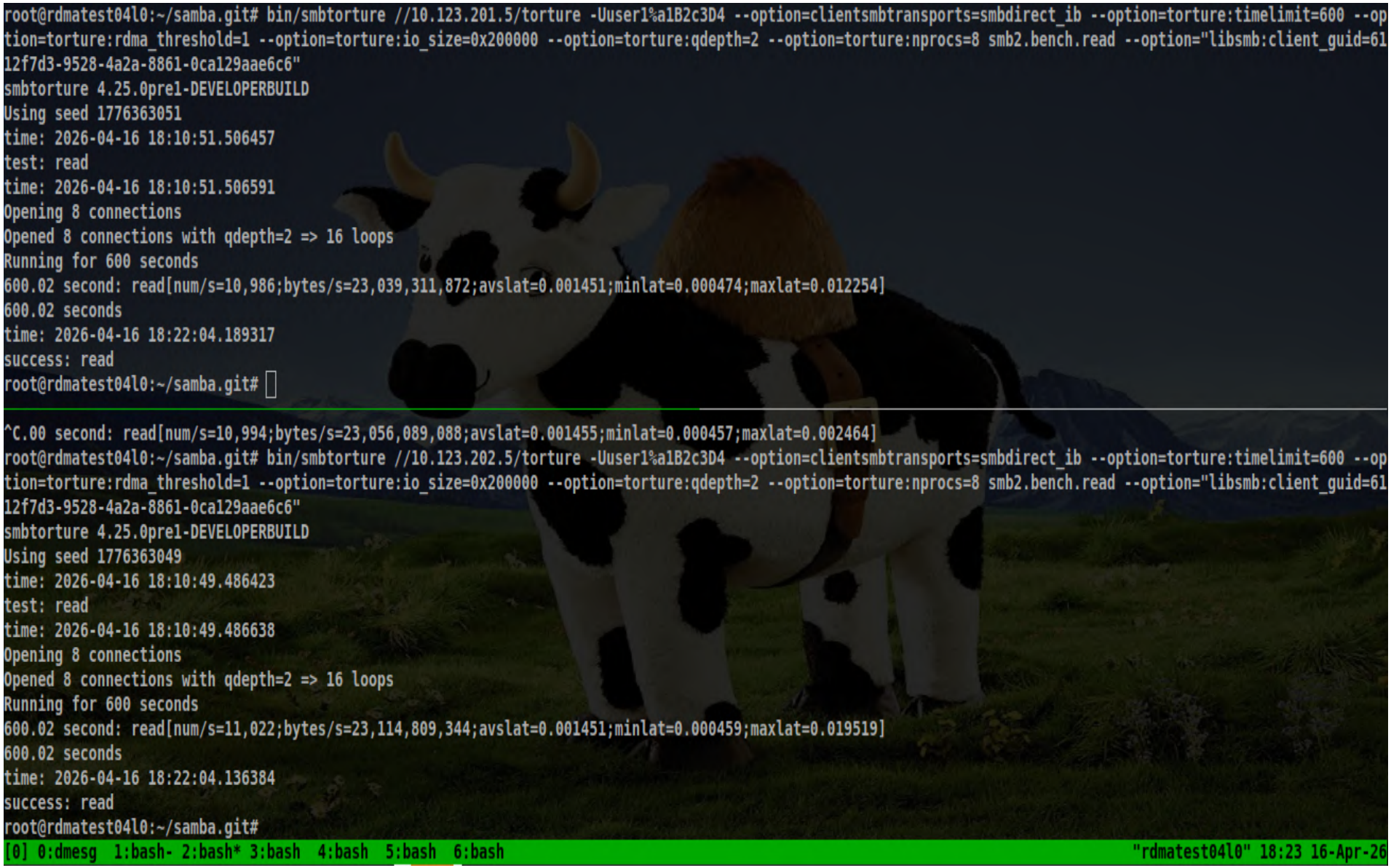
smb2.bench.read io_size=2MB rdma offload (Windows)

single client, 2x8 connections, qdepth=2, ~46 GBytes/s, 2x200GBit/s interfaces, RoCEv2



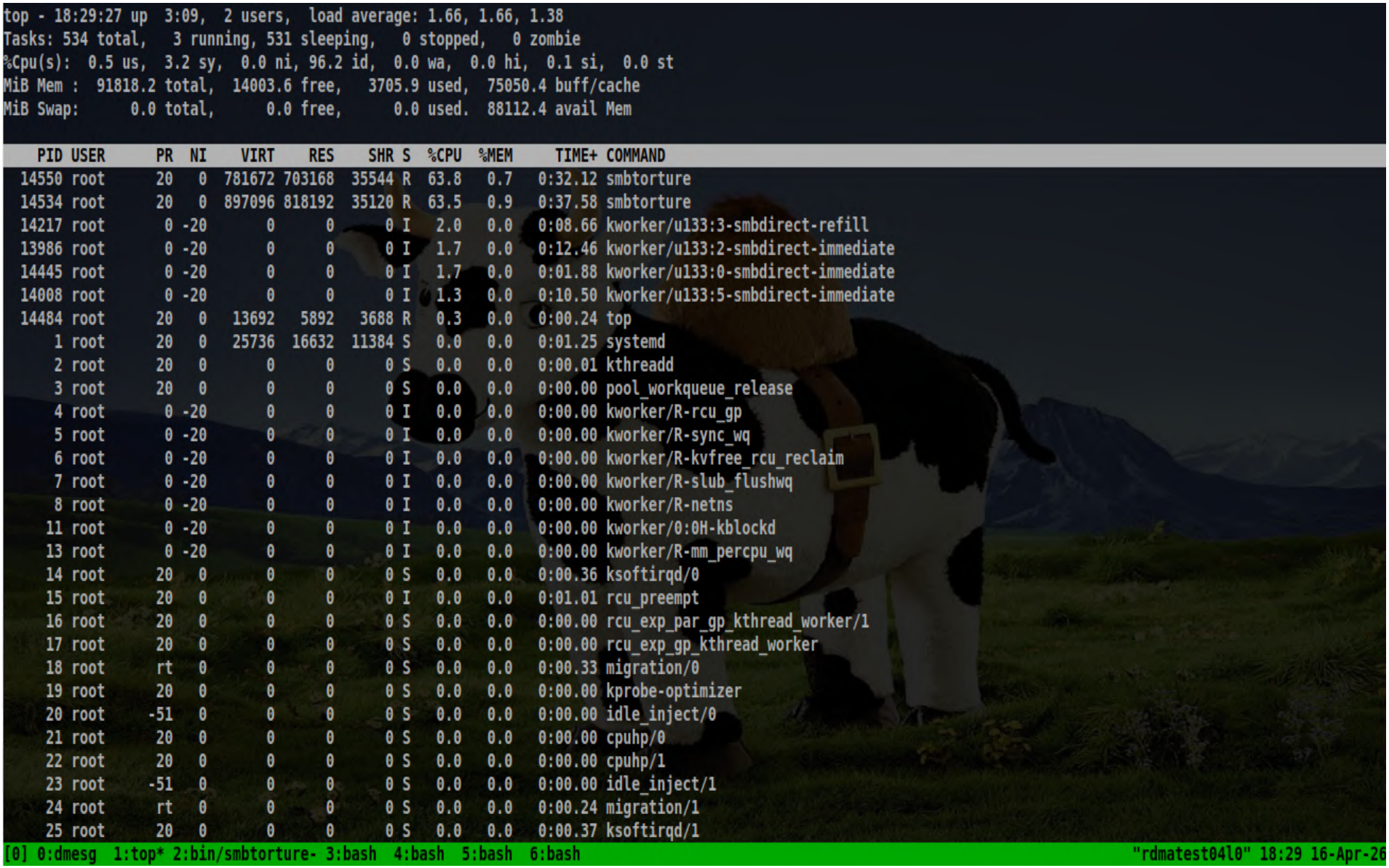
smb2.bench.read io_size=2MB rdma splice (Samba)

single client, 2x8 connections, qdepth=2, ~46 GBytes/s, 2x200GBit/s interfaces, RoCEv2



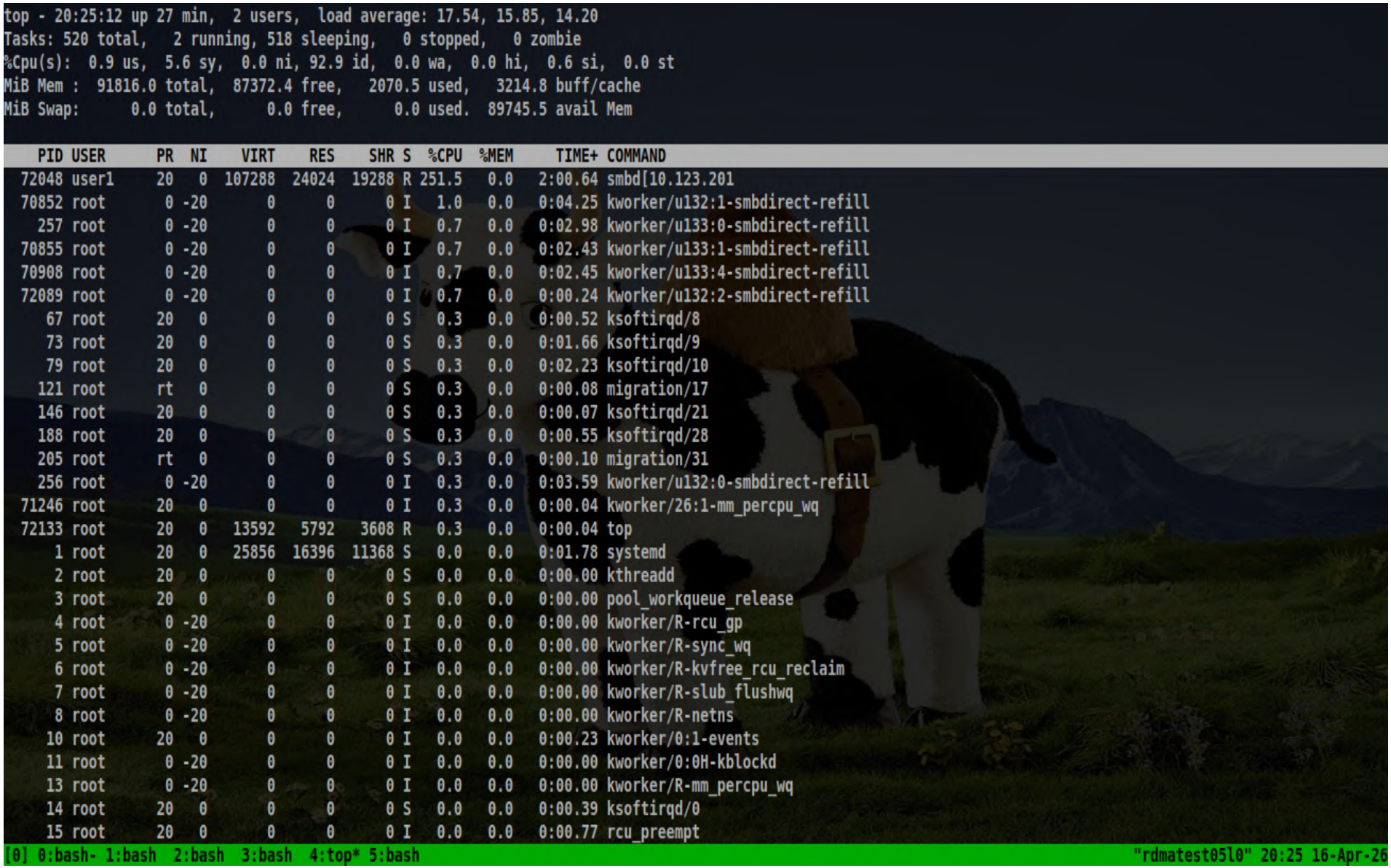
smb2.bench.read io_size=2MB rdma splice (client top)

single client, 2x8 connections, qdepth=2, ~46 GBytes/s, 2x ~63% client cpu, 2x200GBit/s interfaces, RoCEv2



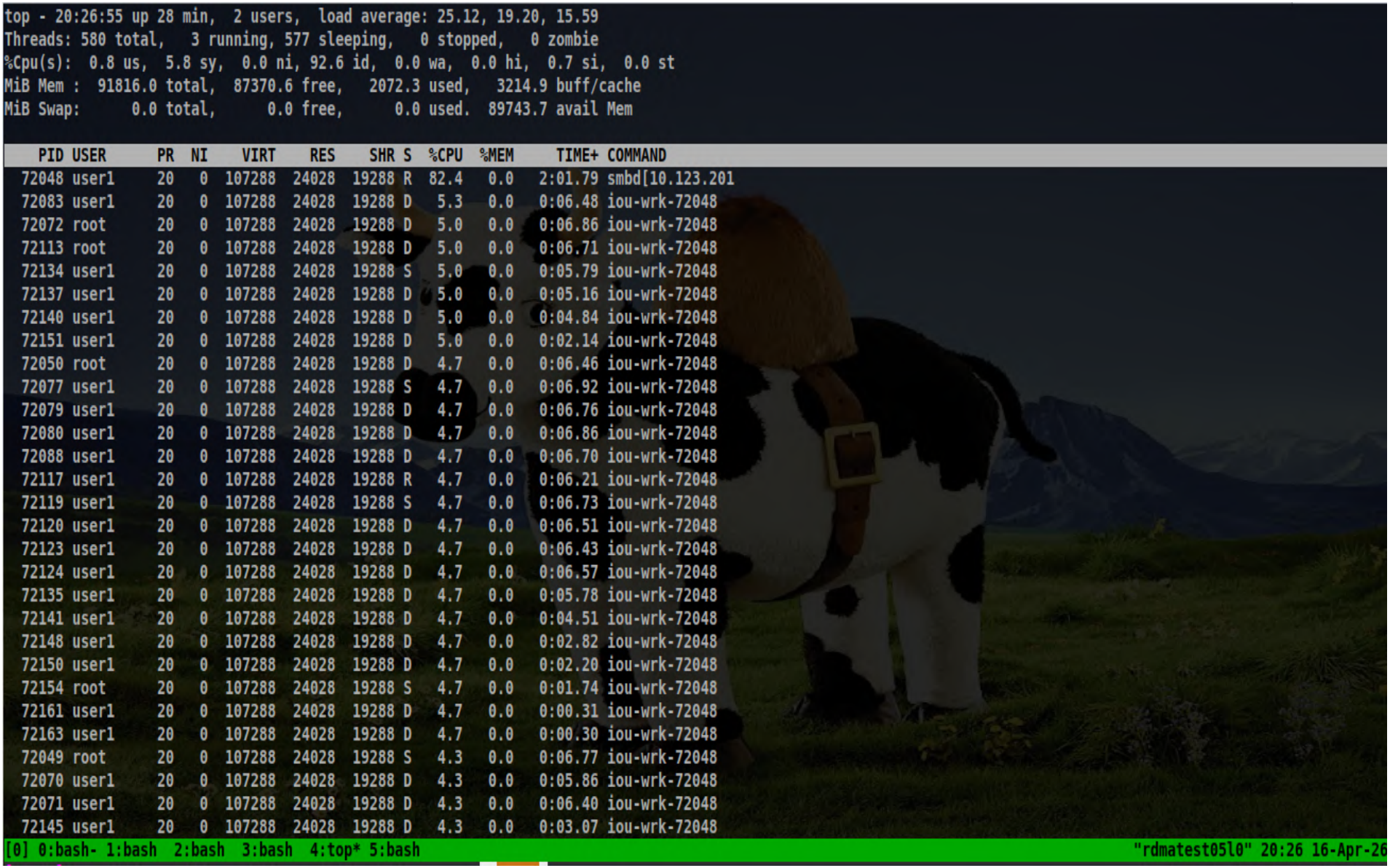
smb2.bench.read io_size=2MB rdma splice (server top)

single smbd + iou-wrk threads, 2x8 connections, qdepth=2, ~46 GBytes/s, ~250% server cpu, 2x200GBit/s interfaces, RoCEv2



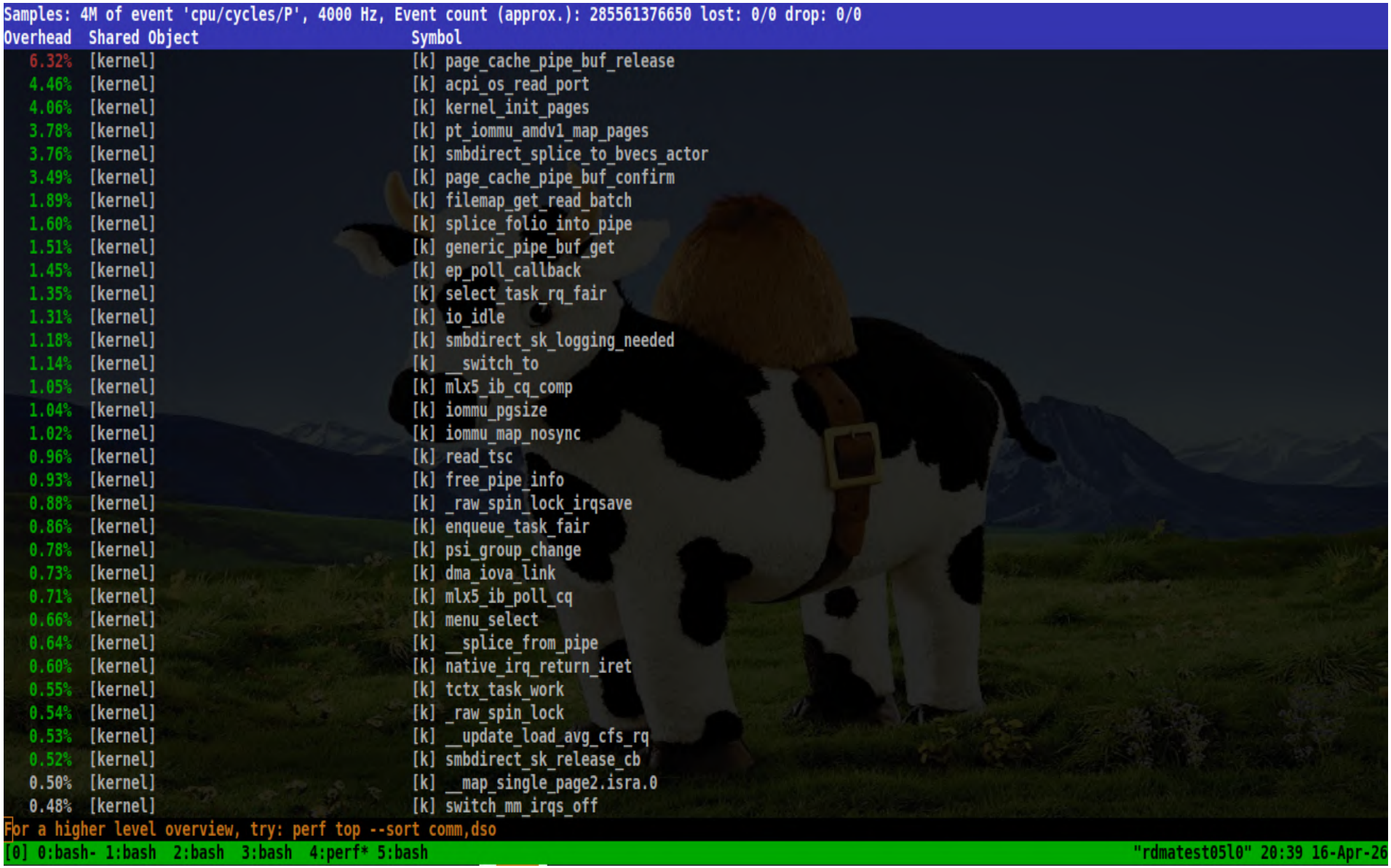
smb2.bench.read io_size=2MB rdma splice (server topH)

single smbd + iou-wrk threads, 2x8 connections, qdepth=2, ~46 GBytes/s, ~82% server cpu smbd, 2x200GBit/s interfaces, RoCEv2



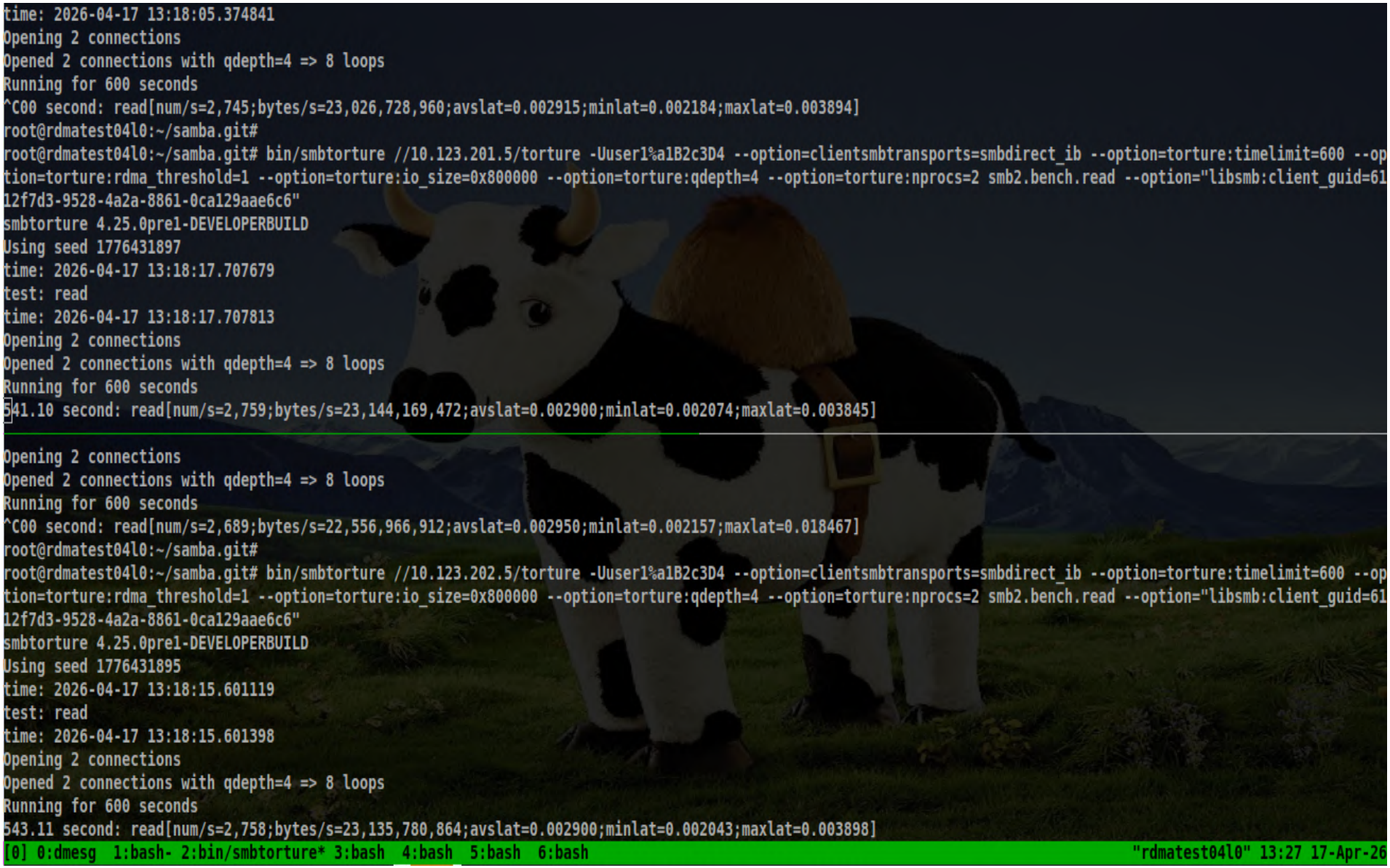
smb2.bench.read io_size=2MB rdma splice (perf top)

single smbd + iou-wrk threads, 2x8 connections, qdepth=2, ~46 GBytes/s, no copy_{to,from}_iter, 2x200GBit/s interfaces, RoCEv2



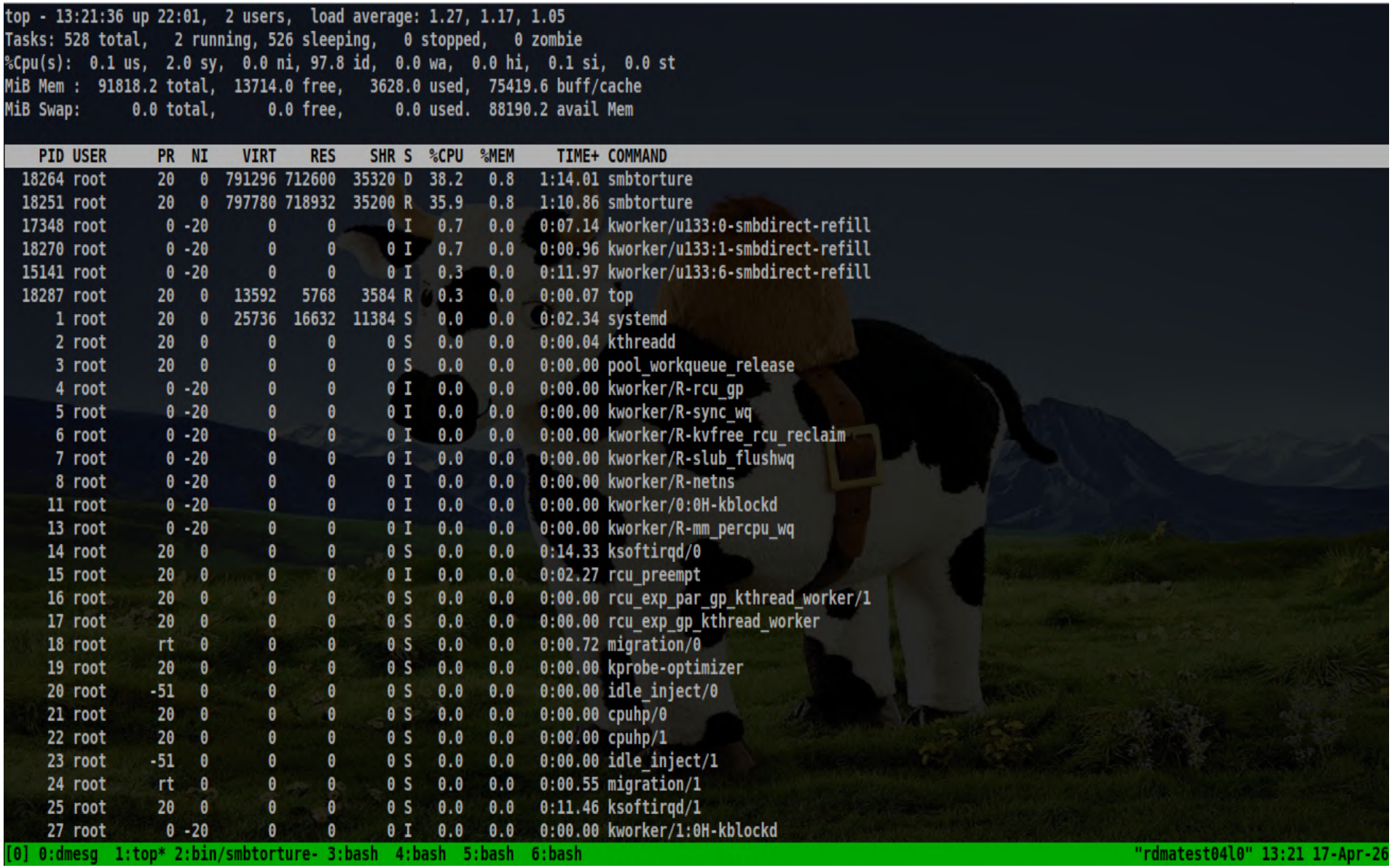
smb2.bench.read io_size=8MB rdma splice (Samba)

single client, 2x2 connections, qdepth=4, ~46 GBytes/s, 2x200GBit/s interfaces, RoCEv2



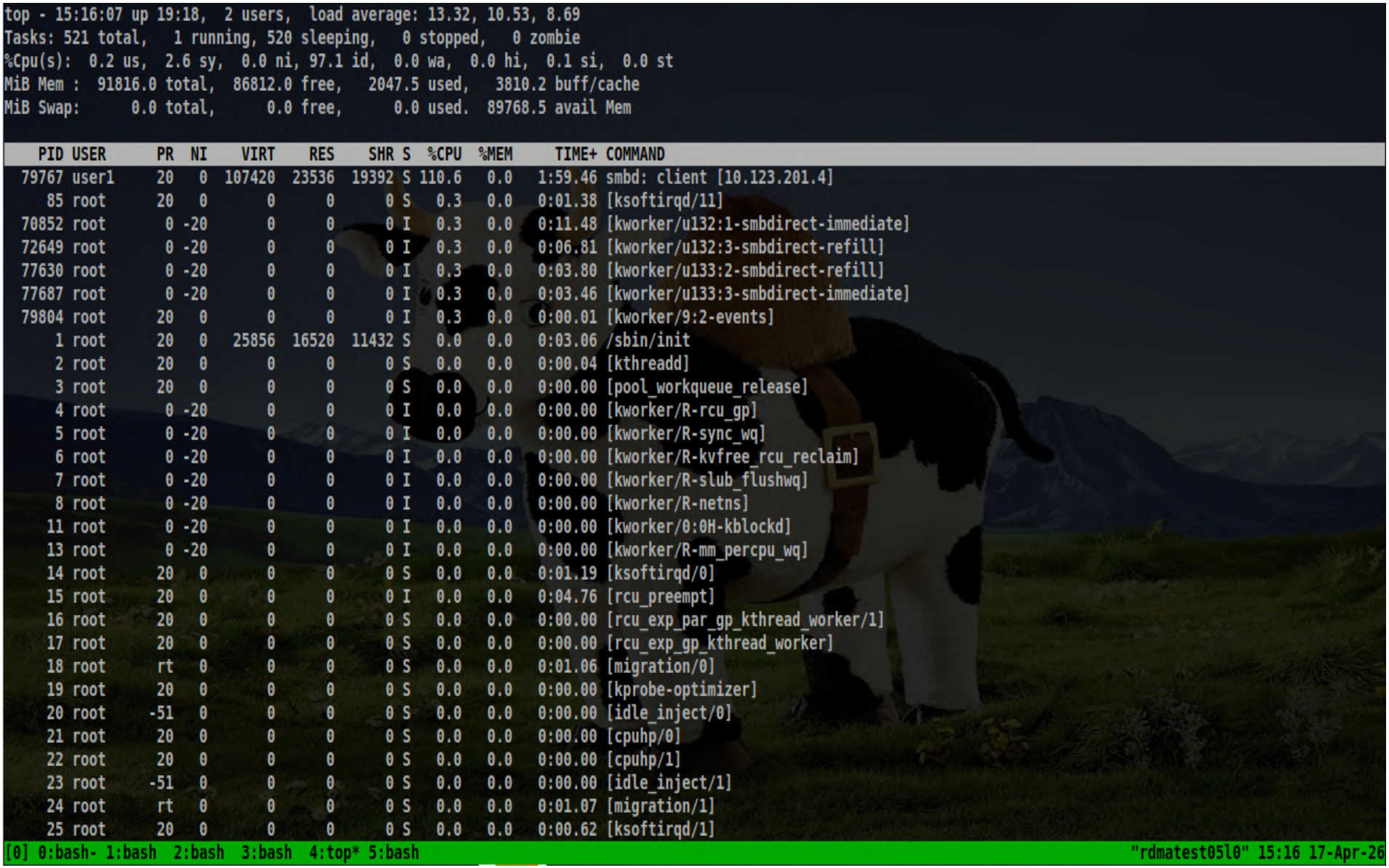
smb2.bench.read io_size=8MB rdma splice (client top)

single client, 2x2 connections, qdepth=4, ~46 GBytes/s, 2x ~38% client cpu, 2x200GBit/s interfaces, RoCEv2



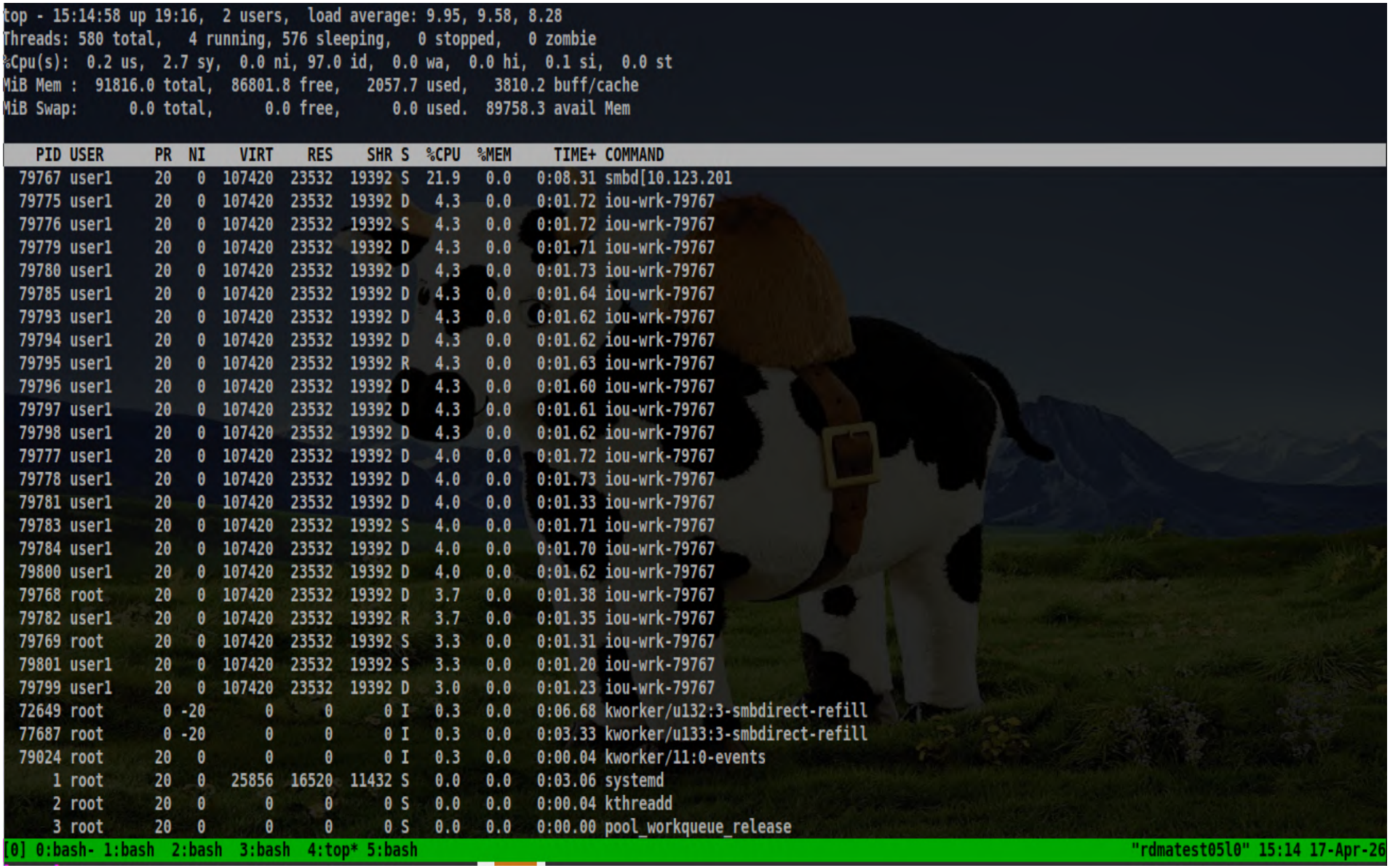
smb2.bench.read io_size=8MB rdma splice (server top)

single smbd + iou-wrk threads, 2x2 connections, qdepth=4, ~46 GBytes/s, ~110% server cpu, 2x200GBit/s interfaces, RoCEv2



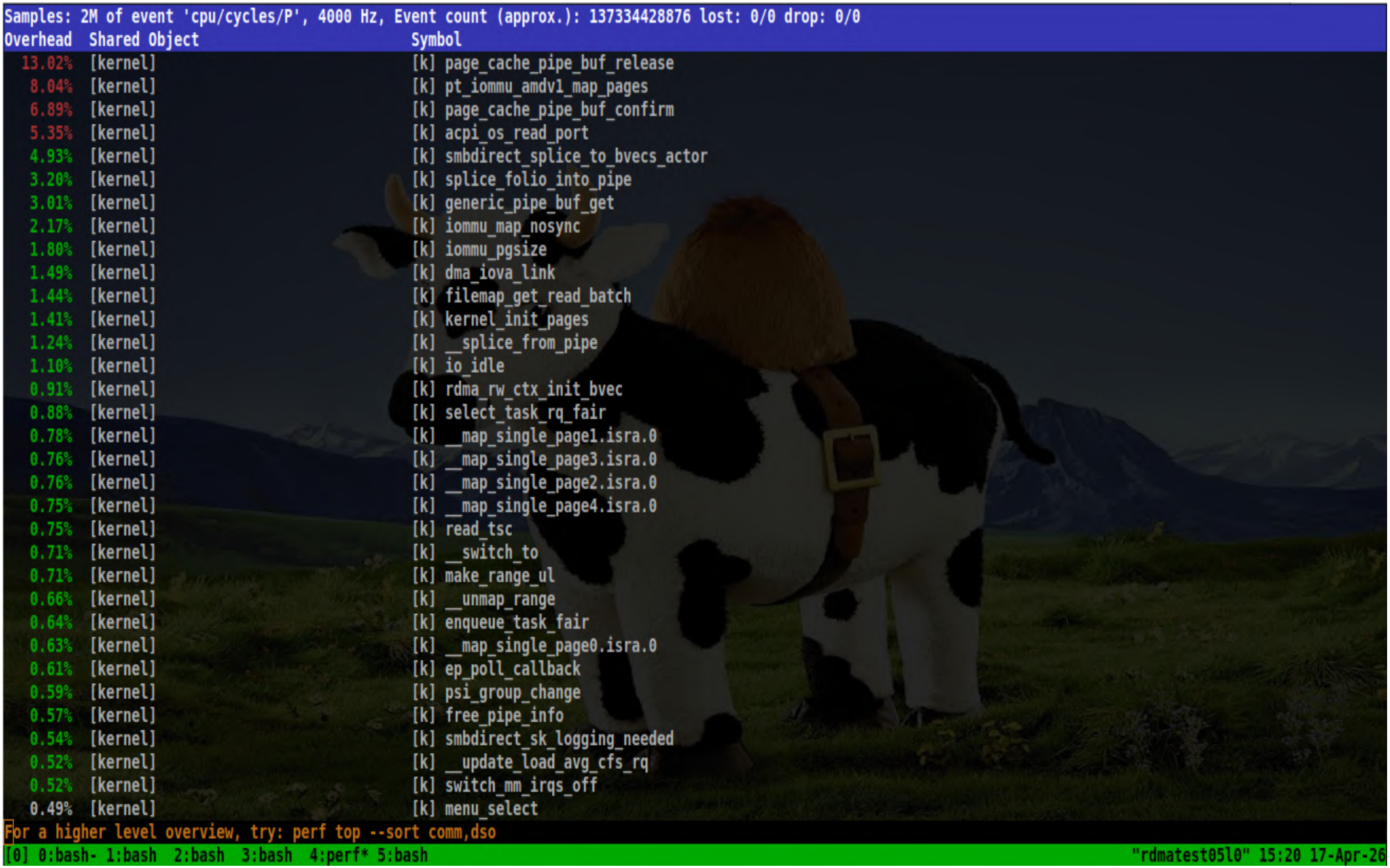
smb2.bench.read io_size=8MB rdma splice (server topH)

single smbd + iou-wrk threads, 2x2 connections, qdepth=4, ~46 GBytes/s, ~21% server cpu smbd, 2x200GBit/s interfaces, RoCEv2



smb2.bench.read io_size=8MB rdma splice (perf top)

single smbd + iou-wrk threads, 2x2 connections, qdepth=4, ~46 GBytes/s, no copy_{to,from}_iter, 2x200GBit/s interfaces, RoCEv2



Samba is back in the game

- This means Samba is able to handle highspeed networks
  - There's no longer the need to use proprietary smb stacks on Linux!
- Most if not all should be ready for the next Samba version
  - This will be released in September together with
  - Support for directory leases
  - Support for persistent handles

Questions? Feedback!

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

Slides: https://samba.org/~metze/presentations/2026/SambaXP/