SMB Direct Support
within Samba and Linux

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2018-09-25

https://samba.org/~metze/presentations/2018/SDC/
Topics

- What is SMB-Direct? What is RDMA?
- RDMA Verbs Specification/Protocols
- SMB-Direct Transport
- SMB3 Multichannel
- Support on Windows
- RDMA Stack on Linux (Kernel/Userspace)
- The first SMB-Direct experiments in Samba
- SMB-Direct Userspace Dispatcher for Samba
- SMB-Direct Kernel Implementation
- Recent Progress
- Future Optimizations
- The way to upstream
- Thanks!
- Questions?
What is SMB-Direct?

- SMB-Direct [MS-SMBD] is a simple transport layer
  - Similar to TCP or Netbios
  - Designed to serve SMB3 on top
  - Provides additional out-of-band features
  - I use "SMB-Direct" as "smbd" is the file server of Samba

- SMB-Direct focuses on performance
  - Low latency and high throughput
  - Minimal CPU utilization for I/O processing

- SMB-Direct requires RDMA (Remote Direct Memory Access)
  - Supports Infiniband, RoCE and iWarp
  - Typically implemented in hardware

- SMB-Direct is negotiated transparently
  - SMB3 MultiChannel is used for the negotiation
  - The strategy is up to the client, it can even skip an initial TCP connection
What is RDMA?

- **Direct Memory Access (DMA)**
  - Is available on all modern computer architectures
  - Allows RAM to be accessed directly by attached devices
  - Typically via the PCI(Express) BUS without active CPU interaction

- **Remote Direct Memory Access (RDMA)**
  - Makes DMA possible over networks to remote peers
  - RDMA-capable NICs are called R-NICs
  - Allows direct data transfers between application buffers
  - Doesn’t require any CPU interaction in order to do the transfer
  - Bypasses the operating system and its protocol stack
RDMA Verbs Specification (Part1)

The Specification defines various operations called "Verbs":
- They form Work Requests (WRs) which are "posted" via a Queue Pair (QP)
- The QP defines a bi-directional connection and interacts with the hardware
- They expect Work Completions (WCs) to be signaled by the hardware
- WCs arrive though Completion Queues (CQs)
- Usage of RDMA requires Memory Registrations (MRs)
- The application needs to keep resources available between "post" and arrival of WC

Available Verbs:
- SEND, SEND_WITH_IMM, SEND_WITH_INV
- REG_MR, LOCAL_INV
- RDMA_WRITE, RDMA_WRITE_WITH_IMM
- RDMA_READ, RDMA_READ_WITH_INV
- ATOMIC_FETCH_AND_ADD, ATOMIC_CMP_AND_SWAP
- ...

Stefan Metzmacher SMB Direct Support (5/31)
The passive side needs to prepare in advance

- Posts fixed size RECVs to the R-NIC, in order to allow SENDs from the peer to arrive
- Registers (REG_M) DMA regions with the hardware for RDMA_READ/WRITEs
- Invalidates (LOCAL_INV) the region again once the RDMA operation completed

The active side triggers operations.

- Posts SENDs to the R-NIC in order to deliver a application message to the peer
- It issues RDMA_READ/WRITEs to the R-NIC specifying local buffers and remote buffer descriptors
RDMA Protocols

- There are multiple protocols proving comparable functionality
- InfiniBand (IB) was the first of these protocols
  - Started around 2000 as cluster node interconnect
  - It provides very low latency and very high throughput
  - But it requires special network cards and switches
- Internet Wide-area RDMA Protocol (iWarp)
  - Started in 2007 with MPA rev1
  - Implemented on top of TCP
  - The current revision is MPA rev2 (defined in 2014)
  - It provides low latency and high throughput
  - Work on any IP based network infrastructure
- RDMA over Converged Ethernet (RoCE)
  - Started around 2010 with RoCE (v1) on raw ethernet
  - RoCE v2 (from 2014) is implemented on top of UDP
  - It provides low latency and high throughput
  - Requires special configurations in network switches
SMB-Direct Transport

- Uses only a few RDMA verbs supported by all protocols
  - SEND or SEND_WTH_INV(alidate) for datagram messages
  - RDMA_READ, RDMA_WRITE for offloads

- It provides a 2-way full duplex transport
  - Datagram style send/receive (similar to SOCK_SEQPACKET)
  - Large messages are send as multiple fragments

- Negotiation Request and Response figure out:
  - Initial credits
  - Max (fragmented) send and receive sizes
  - Max read write sizes

- Data Transfer Messages handles the rest
  - The payload contains from 0 up to max_send_size bytes
  - It indicates the remaining length of following related fragments
  - Sending a message requires having at least one credit
  - The sender can ask for an immediate response
  - For keepalive and credit refunding
How it looks like on the wire (Part1)

- The negotiation exchange

**SMB-Direct (SMB RDMA Transport)**
- **NegotiateRequest**
  - MinVersion: 0x0100
  - MaxVersion: 0x0100
  - CreditsRequested: 255
  - PreferredSendSize: 1364
  - MaxReceiveSize: 8192
  - MaxFragmentedSize: 1048576

**SMB-Direct (SMB RDMA Transport)**
- **NegotiateResponse**
  - MinVersion: 0x0100
  - MaxVersion: 0x0100
  - NegotiatedVersion: 0x0100
  - CreditsRequested: 255
  - CreditsGranted: 15
  - Status: STATUS_SUCCESS (0x00000000)
  - MaxReadWriteSize: 8388608
  - PreferredSendSize: 1364
  - MaxReceiveSize: 1364
  - MaxFragmentedSize: 1048576
How it looks like on the wire (Part2)

- SMB over a Data Transfer Message

  - Ethernet II, Src: 00:00:00_09:01:66 (00:00:00:09:01:66), Dst: 00:00:
  - Internet Protocol Version 4, Src: 172.31.9.166, Dst: 172.31.9.1
  - Transmission Control Protocol, Src Port: 49520, Dst Port: 5445, Seq:
  - iWARP Marker Protocol data unit Aligned framing
  - iWARP Direct Data Placement and Remote Direct Memory Access Protocol
  - SMB-Direct (SMB RDMA Transport)
    - DataMessage
      - CreditsRequested: 255
      - CreditsGranted: 1
      - Flags: 0x0000
      - RemainingLength: 0
      - DataOffset: 24
      - DataLength: 128
  - SMB2 (Server Message Block Protocol version 2)
How it looks like on the wire (Part3)

- SMB3 Write with a RDMA Buffer Descriptor

- SMB2 (Server Message Block Protocol version 2)
  - SMB2 Header
  - Write Request (0x09)
    - StructureSize: 0x0031
    - Data Offset: 0x0000
    - Write Length: 0
    - File Offset: 0
    - GUID handle File: hello.txt
    - Channel: RDMA V1_INVALIDATE (0x00000002)
    - Remaining Bytes: 6
    - Write Flags: 0x00000000
    - Blob Offset: 0x00000070
    - Blob Length: 16
  - Channel Info Blob: SMBDirect Buffer Descriptor V1:
    - RDMA V1
      - Offset: 18446637411657875568
      - Token: 0x81424001
      - Length: 3984
    - Write Data: <MISSING>
- The message flow of an SMB3 Write using RDMA READ

<table>
<thead>
<tr>
<th>Layer</th>
<th>Message</th>
<th>Sequence</th>
<th>Acknowledgment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMB2</td>
<td>Write Request Len:0 Off:0 File: hello.txt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>5445 → 49520 [ACK] Seq=3864353704 Ack=2101125016</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDP/RDMA</td>
<td>5445 &gt; 49520 Read Request [last DDP segment]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDP/RDMA</td>
<td>49520 &gt; 5445 Read Response [last DDP segment]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>5445 → 49520 [ACK] Seq=3864353756 Ack=2101125044</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMB2</td>
<td>Write Response</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SMB3 MultiChannel

- SMB3 introduced the multi channel feature
  - The client can enumerate the servers network interfaces
  - The server returns IPv4/v6 addresses including an interface index, capabilities and the link speed.
  - The server can announce interfaces as RDMA-capable

- The client decides how to connect
  - Typically it opens multiple connections and binds them together
  - RDMA and higher link speeds are prefered for I/O

- SMB-Direct is just an additional transport
  - Clients can also use it directly without multi channel
  - Even SMB1 is possible over SMB-Direct
Windows first announced SMB-Direct with SMB 2.2.2 in 2011
  - The initial version already showed really good results

Windows Server 2012 was the first production release
  - It was released SMB 2.2.2 rebranded as SMB 3.0.0
  - It supports SMB-Direct out of the box
  - The results were even more impressive

In 2013 Windows Server 2012R2 shipped SMB 3.0.2
  - SMB2_CHANNEL_RDMA_V1_INVALIDATE was implemented with SEND_WITH_INV
  - The server remotely invalidates the MR of the client
  - This reduced the I/O latency in the client stack dramatically
  - It saved the LOCAL_INV roundtrip to the hardware
RDMA Stack on Linux (Kernel/Userspace) (Part1)

- RDMA APIs related to SMB-Direct:
  - rdma/rdma_cma.h and infiniband/verbs.h in userspace
  - rdma/rdma_cm.h and rdma/ib_verbs.h in the kernel

- The core implementation lives in the Linux Kernel
  - Device drivers are implemented as kernel modules
  - It includes a verbs API for in kernel consumers
  - It provides for userspace access to the hardware

- The userspace libraries and providers were consolidated in 2016
  - Before they were spread across multiple git repositories
  - It was hard to find a system with a working RDMA stack.
  - Now everything is available in the rdma-core git repository

- Recent distributions come with a usable RDMA stack
  - Linux v4.10 together with the rdma-core related packages
The userspace libraries require providers/drivers
  - The provider needs to match the corresponding kernel driver
  - Provider and kernel driver interact during the setup phase
  - The userspace provider takes over the communication with the device
  - The kernel is bypassed for most operations

Linux supports RoCE and iWarp in pure software
  - Extremely useful for testing! It’s easy to take network captures
  - rdma_rxe (upstream since v4.7) provides RoCEv2
  - siw (SoftiWARP) provides iWarp as out of tree module
  - https://github.com/zrlio/softiwarpl dev-siw.mem_ext works with v4.15
librdmacm and libibverbs do not support a fork process model
  ▶ There are some fork related feature, but they are not useable for us
  ▶ Samba’s one process per client model is not supported
  ▶ Samba’s multi channel design with fd-passing to another process is also not supported
The first SMB-Direct experiments in Samba

- SMB-Direct became my annual Microsoft interop lab hobby
  - At the SDC 2012 I got a few iWarp cards from Chelsio
  - I took network captures of the communication between Windows Servers
  - Then I wrote a wireshark dissector for SMB-Direct
  - This way I got an understanding to understand the protocol

- The first experiments with the APIs and drivers
  - I mainly used the SoftiWarp driver on my laptop
  - I did some experiments with modifying rping to send packets

- SMB1 over SMB-Direct...
  - One week later I a prototype for SMB-Direct in smbclient
  - It only supported SMB1 at that time...
  - But it was very useful to get an understanding about the protocol
After a few years pausing I continued in 2016
- Ralph Böhme and I developed userspace SMB-Direct deamon
- It took care of all SMB-Direct logic
- It provided unix domain sockets to smbclient and smbd
- The prototype worked protocol-wise
- But it was way to slow in order to be useful beside research

In 2017 I finally started to work on a kernel driver
- There were some unsuccessful attempts before
- But I gathered enough knowledge about the protocol
- I was very confident that something useful could be created
Reasons for an SMB-Direct Kernel Implementation

- It should be as simple as possible
  - SMB-Direct is just another transport
  - A stream socket with just sendmsg/recvmsg is all we need

- Should be usable just like a TCP socket
  - Port 445 uses messages prefixed with a 4 byte length header
  - The driver should detect the messages based on the 4 byte header
  - The message needs to fit into the max_fragmented_send_size
  - The message is split into SMB-Direct DataTransferMessage pdus

- Minimize the required changes to Samba
  - The SMB layer just needs to replace its socket() call
  - For now we have smbdirect_socket()

- Sometimes smbd blocks in syscalls
  - close() or unlink() are not yet async
  - They can be take up to minutes in cluster environments
  - The kernel takes care of all keepalive handling
  - And the connection would still be available
Working (unoptimized) prototype (smbdirect.ko)

The diffstat of the smbdirect.ko (compiles against v4.10 up to master):

<table>
<thead>
<tr>
<th>File</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>smbdirect.h</td>
<td>541 ++++</td>
</tr>
<tr>
<td>smbdirect_accept.c</td>
<td>676 ++++</td>
</tr>
<tr>
<td>smbdirect_connect.c</td>
<td>751 +++++</td>
</tr>
<tr>
<td>smbdirect_connection.c</td>
<td>1532 +++++++</td>
</tr>
<tr>
<td>smbdirect_device.c</td>
<td>232 ++</td>
</tr>
<tr>
<td>smbdirect_main.c</td>
<td>132 +-</td>
</tr>
<tr>
<td>smbdirect_private.h</td>
<td>779 +++++</td>
</tr>
<tr>
<td>smbdirect_proc.c</td>
<td>206 ++</td>
</tr>
<tr>
<td>smbdirect_socket.c</td>
<td>2688 ++++++++++++++++++++</td>
</tr>
</tbody>
</table>

9 files changed, 7535 insertions (+), 2 deletions (-)

Userspace API for smbdirect (without optimizations):

```c
int smbdirect_socket(int family, int type, int protocol);
int smbdirect_connection_get_parameters(int sockfd,
                                           struct smbdirect_connection_parameters *params);
ssize_t smbdirect_rdma_v1_register(int sockfd,
                                     struct smbdirect_buffer_descriptors_v1 *local,
                                     int iovcnt, const struct iovec *iov);
ssize_t smbdirect_rdma_v1_deregister(int sockfd,
                                      const struct smbdirect_buffer_descriptors_v1 *local);
ssize_t smbdirect_rdma_v1_writev(int sockfd,
                                   const struct smbdirect_buffer_descriptors_v1 *remote,
                                   int iovcnt, const struct iovec *iov);
ssize_t smbdirect_rdma_v1_readv(int sockfd,
                                 const struct smbdirect_buffer_descriptors_v1 *remote,
                                 int iovcnt, const struct iovec *iov);
```
Working (unoptimized) prototype (smbclient/smbd)

The diffstat of the client side changes:

<table>
<thead>
<tr>
<th>File Path</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>libcli/smb/smb2_constants.h</td>
<td>6 +</td>
</tr>
<tr>
<td>libcli/smb/smb2cli_read.c</td>
<td>71 +++-</td>
</tr>
<tr>
<td>libcli/smb/smb2cli_write.c</td>
<td>81 +++-</td>
</tr>
<tr>
<td>libcli/smb/smbXcli_base.c</td>
<td>313 ++++++</td>
</tr>
<tr>
<td>libcli/smb/smbXcli_base.h</td>
<td>32 ++</td>
</tr>
<tr>
<td>source3/lib/util_sock.c</td>
<td>20 +-</td>
</tr>
</tbody>
</table>

6 files changed, 496 insertions (+), 27 deletions (-)

The diffstat of the server side changes:

<table>
<thead>
<tr>
<th>File Path</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>source3/smbd/globals.h</td>
<td>24 +++</td>
</tr>
<tr>
<td>source3/smbd/process.c</td>
<td>17 ++</td>
</tr>
<tr>
<td>source3/smbd/smb2_negprot.c</td>
<td>5 +</td>
</tr>
<tr>
<td>source3/smbd/smb2_read.c</td>
<td>148 ++++++</td>
</tr>
<tr>
<td>source3/smbd/smb2_server.c</td>
<td>199 ++++++++</td>
</tr>
<tr>
<td>source3/smbd/smb2_tcon.c</td>
<td>10 +</td>
</tr>
<tr>
<td>source3/smbd/smb2_write.c</td>
<td>119 +++++++</td>
</tr>
</tbody>
</table>

7 files changed, 516 insertions (+), 6 deletions (-)
The function call to create an SMB Direct socket (in kernel):

```c
int smbdirect_sock_create_kern(struct net *net,
    int family, int type, int protocol,
    struct socket **res);

int smbdirect_kern_connection_get_parameters(struct socket *sock,
    struct smbdirect_connection_parameters *params);

ssize_t smbdirect_kern_rdma_v1_register_pages(struct socket *sock,
    struct smbdirect_buffer_descriptors_v1 *local,
    struct page *pages[], int num_pages,
    int pagesz, int fp_ofs, int lp_len);

ssize_t smbdirect_kern_rdma_v1_deregister(struct socket *sock,
    struct smbdirect_buffer_descriptors_v1 *local);
```
Recent Progress...

- I made very good progress last week at Microsoft
  - I have a first functional prototype
  - It still has memory leaks and misses some error checks
  - But smbclient works against Windows and smbd using RDMA
  - smbclient fills a 10GBit/s Link with TCP and iWarp

- Reduced CPU usage in the client using smbdirect:
  - userspace CPU/time by 25%, system CPU/time by 30%
  - Just in the first test run, without further optimization

- A lot of hardware/driver problems disrupted my work
  - The same test with exactly the same software drop by 80%
  - This happens for both TCP (also over the R-NIC) and iWarp/RoCE
  - The Microsoft SMB-Direct testsuite gets just a TCP reset
  - While smbclient can connect without problems
Future Optimizations… (Part1)

▶ There are a lot of ways to further improve
  ▶ The key is to avoid latency and processing overhead
  ▶ We likely need to add NUMA awareness

▶ Towards the upper layer
  ▶ We can avoid syscalls by letting it prepare the memory descriptors
  ▶ Memory registrations can be hooked into msg_control on sendmsg()
  ▶ Deregistrations can be made async
  ▶ Or even be removed with SMB $\geq$ 3.02 using SEND_WITH_INV

▶ Towards the RDMA layer
  ▶ We should reduce the roundtrips between CPU and R-NIC as much as possible
  ▶ We can batch WRs by passing a list to ib_post_send/recv()
  ▶ For related operations can only request to be signaled on the last operation
  ▶ The correct order is garanteed for posts and completions
Future Optimizations... (Part2)

- Typically smbd serves files from a kernel filesystem
  - Bytes are copied via the filesystem into a userspace buffer
  - The userspace buffer is then handed to the smbdirect socket
  - This happens for SMB3 Read
  - In the reversed direction for SMB3 Write

Possible functions to avoid data copy on the server:

```c
ssize_t smbdirect_rdma_v1_write_from_file(int sockfd,
                                            const struct smbdirect_buffer_descriptors_v1 *remote,
                                            int source_fd, size_t source_length, off_t source_offset);

ssize_t smbdirect_rdma_v1_read_to_file(int sockfd,
                                        const struct smbdirect_buffer_descriptors_v1 *remote,
                                        int source_fd, size_t source_length, off_t source_offset);
```

- These could be further optimized
  - "rdma write from file" could use msg_control of sendmsg()
  - Both can be made async with some epoll based completion
  - The completion could be batched with msg_control on recvmsg()
Future Optimizations... (Part3)

- It’s not unlikely that we hit generic performance bottlenecks
  - Samba’s smbd runs in usermode
  - It uses a single process (wit helper threads) per client

- RDMA Push Mode for SMB3
  - Microsoft is researching a full I/O offload between client and server
  - The client memory maps the file
  - The server creates MRs for file ranges on persistent memory
  - The client uses direct RDMA operations without SMB3 READ/WRITE
  - Requires new RDMA Verbs to be implemented

- Push mode will remove the usermode restrictions
  - smbd just needs to perform an mmap and create MRs
  - All the rest happens outside of smbd
The way to upstream (Part 1)

- This is currently a hobby project
  - I have like 2-3 weeks a year to work on it
  - Only about 2-3 month since the first experiments in 2012
  - At that level it will take a few additional years to get production ready
  - Sponsors are most welcome!

- Items of step 1 (the smbdirect driver):
  - The code quality needs to be cleaned up
  - We need to handle all possible errors
  - ftrace based trace points would make debugging much easier
  - We need a standalone testsuite that runs without Samba
  - Then we can optimize further

- Items of step 2 (multi channel support in Samba):
  - We need to make multi channel production ready (with tests)
  - We need to plugin SMB-Direct to the multi channel layer
  - We need to think about ways to automatically test the SMB-Direct code path
The way to upstream (Part 2)

- We need to coordinate with the Linux Kernel Developers:
  - What will be way to expose the UAPI
  - Could we expose it as IPPROTO_SMBDIRECT (with a number > 255)
  - Is it ok to use ioctl()’s for the extended operations?
  - Do we need to implement more of the struct sock/socket function pointers?
  - In what directory could it be placed in the kernel, net/smbdirect/?

- It could be used just internally by cifs.ko first
  - We could defer exposing a UAPI until everything is stable
  - Once it provides the same quality as the current smbdirect implementation, we could switch

- When can we add it to upstream Samba?
  - Would it be ok to have as an optional feature?
  - While it still relies on an external kernel module?
  - Can we add some magic to socket wrapper for autobuild?
Thanks!

I’d like to thank:

→ Chelsio for giving me iWarp NICs to test with!

→ Tom Talpey and others from Microsoft for the great help and support!

→ elements.tv for the access to RoCE test hardware
Questions?

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

→ **SerNet / SAMBA+** sponsor booth

Work in Progress (smbdirect.ko):
https://git.samba.org/?p=metze/linux/smbdirect.git;a=summary

Work in Progress (Samba):
https://git.samba.org/?p=metze/samba/wip.git;a=shortlog;h=refs/heads/master3-smbdirect

Slides:
https://samba.org/~metze/presentations/2018/SDC/