DLD for Patchless Zero Copy Socklnd

Liang Zhen

Nov 27, 2006

Contents

1 Introduction

This DLD describes the details of patchless ZC(zero copy) socklnd.

2 Functional Specification

2.1 Data structures

2.1.1 Handshake data structure

Add a new data structure for handshake:

```
typedef struct {
    __u32 kshm_magic; /* magic number of socklnd message */
    __u16 kshm_version_major; /* major version of socklnd message */
    __u16 kshm_version_minor; /* minor version of socklnd message */
    lnet_nid_t kshm_src_nid; /* sender's nid */
    lnet_nid_t kshm_dst_nid; /* destination nid */
    lnet_pid_t kshm_src_pid; /* sender's pid */
    lnet_pid_t kshm_dst_pid; /* destination pid */
    __u64 kshm_src_incarnation; /* sender's incarnation */
    __u64 kshm_dst_incarnation; /* destination's incarnation */
    __u32 kshm_ctype; /* connection type */
    __u32 kshm_nips; /* # IP addrs */
    __u32 kshm_ips[0]; /* IP addrs */
} WIRE_ATTR ksock_hello_msg_t;
```

2.1.2 Send/Receive data structure

These structures are used for normal send/receive message between two peers

```
typedef struct {
    lnet_hdr_t              ksnm_hdr;       /* lnet hdr */
    char                    ksnm_payload[0];/* lnet payload */
} WIRE_ATTR ksock_lnet_msg_t;
typedef struct {
    __u32                   ksm_type;       /* type of socklnd message */
    __u32                   ksm_cksum;      /* checksum if != 0 */
    __u64                   ksm_req_cookie; /* ack required if != 0 */
    __u64                   ksm_ack_cookie; /* ack if != 0 */
    union {
        ksock_lnet_msg_t    normal;
    } WIRE_ATTR ksm_u;
} WIRE_ATTR ksock_msg_t;
#define KSOCK_MSG_NOOP    0xC0
#define KSOCK_MSG_LNET    0xC1
```

2.1.3 Protocol data structure

This structure includes a function table should be defined for each protocol version

```
typedef struct {
    int             (*pro_send_hello)(lksock_conn_t *, ksock_hello_msg_t *);
    int             (*pro_recv_hello)(ksock_conn_t *, ksock_hello_msg_t *, i
    void            (*pro_pack)(ksock_tx_t *);        /* message pack */
```

```
    void            (*pro_unpack)(ksock_msg_t *);     /* message unpack */
} ksock_protocol_t;
#define KSOCK_PROTO_V1_MAJOR      LNET_PROTO_TCP_VERSION_MAJOR
#define KSOCK_PROTO_V1_MINOR      LNET_PROTO_TCP_VERSION_MINOR
#define KSOCK_PROTO_V2_MAJOR      2
#define KSOCK_PROTO_V2_MINOR      0
```

2.2 Interfaces

2.2.1 ksock_tx_t *ksocknal_alloc_tx(int size)

Parameters: `size`: descriptor size

Return Value: Pointer to tx on success, NULL on failure.

Description: Allocate a tx, or take a tx from idle list if required size is equal to sizeof NOOP tx. reference count of tx is initialized to 1 in allocating.

2.2.2 void ksocknal_free_tx(ksock_tx_t *tx)

Parameters: tx: Pointer to tx

Return Value: N/A

Description: Free the tx, or put the tx to idle list if it's a NOOP tx.

2.2.3 int ksocknal_piggyback_zcack_locked(__u64 cookie, ksock_conn_t *conn)

Parameters: `cookie: ACK cookie needs to be piggybacked`
conn: Connection for sending out ACK

Return Value: Return 1 if the ACK cookie is piggybacked, 0 if it's not.

Description: Try to piggyback ACK cookie to a normal packet in outgoing list of the connection, global lock should be held while calling it.

2.2.4 int ksocknal_handle_zc_req(ksock_peer_t *peer, __u64 cookie)

Parameters: `peer: Sender of the ZC packet.`
cookie: ZC cookie of sender

Return Value: 0 on success, errno on failure.

Description: Enqueue a ACK message(ACK cookie on NOOP message or a normal message) for the ZC request.

2.2.5 int ksocknal_handle_zc_ack(ksock_peer_t *peer, __u64 cookie)

Parameters: `peer:` Receiver of the ZC packet (sender of the ZC ACK);
`cookie: ZC ACK cookie`

Return Value: 0 on success, -EPROTO on failure (Can't find matching cookie)

Description: Free the requesting tx if there is a ZC request can match the ACK cookie.

2.2.6 int ksocknal_lib_zc_capable(ksock_tx_t *tx)

Parameters: `tx: packet needs to be sent`

Return Value: 1 if it can be sent by zero copy, 0 if it can't.

Description: Decide whether the tx can be sent by zero copy or not(packet for V1.x connection always get 0).

2.2.7 ksock_protocol_t * ksocknal_compat_protocol(ksock_hello_msg_t *hello)

Parameters: hello: Pointer to a received hello message

Return Value: Compatible protocol table for the input version on success, NULL on failure.

Description: Try to find protocol table that can be compatible with version(@major, @minor) and return it.

2.2.8 int ksocknal_send_hello_v1/2 (ksock_conn_t *conn, ksock_hello_msg_t *msg)

Parameters: conn: Connection to send out hello
msg: Hello message to be sent

Return Value: Zero on success, errno on failure.

Description: Send out hello message to peer in format of version 1.x/2.x

2.2.9 int ksocknal_recv_hello_v1/2 (ksock_conn_t *conn, ksock_hello_msg_t *msg, int timeout)

Parameters: conn: Connection to receive hello
msg: buffer to save hello message
timeout: receiving timeout

Return Value: Zero on success, errno on failure.

Description: Receive hello message from V1.x/V2.x peer, and save the hello message to @msg.

2.2.10 void ksocknal_pack_msg_v1/2(ksock_tx_t *tx)

Parameters: tx: packet to be launched

Return Value: N/A

Description: pack message header and first frag in format of V1.x/2.x

2.2.11 void ksocknal_unpack_msg_v2(ksock_msg_t *msg)

Parameters: msg: buffer to save the message

Return Value: N/A

Description: Unpack message header to @msg.

3 Use Cases

3.1 Send out ZC request

```
void ksocknal_queue_tx_locked (ksock_tx_t *tx, ksock_conn_t *conn)
{
    ......
    if (ksocknal_lib_zc_capable(tx)) {
        ksocknal_tx_addref(tx);
        ksocknal_conn_addref(conn);

        spin_lock(&net->ksnn_lock);
        msg->ksm_req_cookie = net->ksnn_next_cookie++;
        list_add_tail(&tx->tx_zc_list, &peer->ksnp_zc_req_list);
        spin_unlock(&net->ksnn_lock);
    }
    ......
}
```

3.2 Receive ZC REQ/ACK

```
int ksocknal_process_receive (ksock_conn_t *conn)
{
    ......
    switch (conn->ksnc_rx_state) {
    case SOCKNAL_RX_KSM_HEADER:
        if (conn->ksnc_msg.ksm_ack_cookie != 0)
            rc = ksocknal_handle_zc_ack(conn->ksnc_peer, conn->ksnc_msg.ksm_
    ......
    case SOCKNAL_RX_LNET_PAYLOAD:
        if (conn->ksnc_msg.ksm_req_cookie != 0)
            rc = ksocknal_handle_zc_req(conn->ksnc_peer, conn->ksnc_msg.ksm_
    ......
}
```

3.3 Create connection

```
ksocknal_create_conn (lnet_ni_t *ni, ksock_route_t *route, cfs_socket_t *soc
{
    ......
    if (route != NULL) {
        /* Active */
        ......
```

```
            conn->ksnc_version_major = route->ksnr_peer->ksnp_version_major;
            conn->ksnc_version_minor = route->ksnr_peer->ksnp_version_minor;
            conn->ksnc_proto = ksocknal_compat_protocol(conn->ksnc_version_major
            if (conn->ksnc_proto == NULL) {
                rc = -EPROTO;
                goto failed_1;
            }
            rc = ksocknal_send_hello (ni, conn, peerid.nid, ipaddrs, nipaddrs);
            if (rc != 0)
                goto failed_1;
        } else {
            ......
            /* Passive, use default protocol */
           conn->ksnc_version_major = KSOCK_PROTO_V2_MAJOR;
           conn->ksnc_version_minor = KSOCK_PROTO_V2_MINOR;
           conn->ksnc_proto = ksocknal_compat_protocol(conn->ksnc_version_major,
           if (conn->ksnc_proto == NULL) {
               rc = -EPROTO;
               goto failed_1;
           }
        }
        ...... /* Receive hello, etc.... */
    }
```

3.4 Send hello

```
int ksocknal_send_hello (lnet_ni_t *ni, ksock_conn_t *conn, lnet_nid_t peer_
                         __u32 *ipaddrs, int nipaddrs)
{
    ksock_hello_msg_t  hello;

    /* just initialize ksock_hello_msg_t, pro_send_hello() will convert it t
    hello.kshm_magic         = cpu_to_le32 (LNET_PROTO_TCP_MAGIC);
    hello.kshm_version_major = cpu_to_le16 (ksocknal_protocol[version].pro_v
    hello.kshm_version_minor = cpu_to_le16 (ksocknal_protocol[version].pro_v
    if (the_lnet.ln_testprotocompat != 0) {
        ......
    }
    hello.kshm_src_nid         = cpu_to_le64 (srcnid);
    hello.kshm_dst_nid         = cpu_to_le64 (peer_nid);
    ......
    rc = conn->ksnc_proto->pro_send_hello(conn, &hello);
    ......
    rc = libcfs_sock_write(sock, ipaddrs, nipaddrs * sizeof(*ipaddrs),
```

```
                                lnet_acceptor_timeout());
    ......
}
```

3.5 Receive Hello

```
int ksocknal_recv_hello (lnet_ni_t *ni, ksock_conn_t *conn, lnet_process_id_
                         __u64 *incarnation, __u32 *ipaddrs)
{
    ......
    rc = libcfs_sock_read(sock, &hmv->magic + 1,
                          sizeof(*hmv) - sizeof(hmv->magic), timeout);
    ......
    proto = ksocknal_compat_protocol(hmv->version_major, hmv->version_minor)
    if (proto == NULL || (active && proto != conn->ksnc_proto)) {
        if (!active)
            ksocknal_send_hello(ni, conn, ni->ni_nid, NULL, 0);
        else
            CERROR(...);
        return -EPROTO;
    }
    if (!active && conn->ksnc_proto != proto)
        /* Correct my protocol */
        conn->ksnc_proto = proto;

    rc = conn->ksnc_proto->pro_recv_hello(conn, &hello, timeout);
    ......
    /* We can share all code now, because handshake message is saved in ksoc
    if (hello.kshm_src_nid == LNET_NID_ANY) {
        CERROR("Expecting a HELLO message with a NID, but got LNET_NID_ANY"
               "from %u.%u.%u.%u\n", HIPQUAD(conn->ksnc_ipaddr));
        return (-EPROTO);
    }
    ......
}
```

3.6 Send message

```
ksocknal_queue_tx_locked (ksock_tx_t *tx, ksock_conn_t *conn)
{
    ......
    tx->tx_conn = conn;
    conn->ksnc_proto->pro_pack(tx);
```

```
    ......
}
```

3.7 Receive message

```
int ksocknal_process_receive (ksock_conn_t *conn)
{
    switch (conn->ksnc_rx_state) {
    ......
    case SOCKNAL_RX_LNET_HEADER:
        conn->ksnc_proto->pro_unpack(&conn->ksnc_msg);
    .....
}
```

3.8 Active connect

```
void ksocknal_connect(ksock_route_t *route)
{

    ....
    for (;;) {
        ......
        rc = lnet_connect (...)
        if (rc < 0) {
            goto failed;
        }
        rc = ksocknal_create_conn(...);
        if ((rc == -EPROTO || rc == -ECONNRESET) &&
             peer->ksnp_version_major == KSOCK_PROTO_V2_MAJOR) {
            /* Can't establish connection by V2.x protocol,
             * Let's try 1.x later */
            peer->ksnp_version_major = KSOCK_PROTO_V1_MAJOR;
            peer->ksnp_version_minor = KSOCK_PROTO_V2_MINOR;
            rc = EPROTO;
        } else if (rc < 0) {
            goto failed;
        }
        retry_later = rc != 0;
        write_lock_bh(glock);
    }
}
```

4 Logic Specification

4.1 ksocknal_alloc_tx

```
#define KSOCK_NOOP_MSG_SIZE     offsetof(ksock_tx_t, tx_frags.paged.kiov[0])
ksock_tx_t * ksocknal_alloc_tx (int size)
{
    ksock_tx_t *tx;
    tx = NULL;
    if (size == KSOCK_NOOP_MSG_SIZE) {
        /* allocate a noop tx */
        spin_lock(&ksocknal_data.ksnd_tx_lock);
        if (!list_empty(&ksocknal_data.ksnd_noop_txs)) {
            tx = list_entry(ksocknal_data.ksnd_noop_txs.next, ksock_tx_t, tx
            LASSERT(tx->tx_desc_size == size);
            list_del(&tx->tx_list);
        }
        spin_unlock(&ksocknal_data.ksnd_tx_lock);
    }
    if (tx == NULL)
        LIBCFS_ALLOC(tx, size);
    if (tx == NULL)
        return NULL;
    atomic_set(&tx->tx_refcount, 1);
    tx->tx_desc_size = size;
    atomic_inc(&ksocknal_data.ksnd_nactive_txs);
    return tx;
}
```

4.2 ksocknal_free_tx

```
void ksocknal_free_tx (ksock_tx_t *tx)
{
    atomic_dec(&ksocknal_data.ksnd_nactive_txs);
    if (tx->tx_desc_size == KSOCK_NOOP_MSG_SIZE) {
        /* it's a noop tx */
        spin_lock(&ksocknal_data.ksnd_tx_lock);
        list_add(&tx->tx_list, &ksocknal_data.ksnd_idle_noop_txs);
        spin_unlock(&ksocknal_data.ksnd_tx_lock);
    } else
        LIBCFS_FREE(tx, tx->tx_desc_size);
}
```

4.3 ksocknal_queue_tx_locked

This function is changed to have complex logic, and with a lot of interesting operations.

```
void ksocknal_queue_tx_locked (ksock_tx_t *tx, ksock_conn_t *conn)
{
    ksocknal_protocol[conn->ksnc_version].pro_pack(conn, tx);
    if (ksocknal_lib_zc_capable(conn, tx)) {
        /* insert tx to peer->ksnp_zc_req_list etc...*/
        ......
    }
    ......
    spin_lock_bh (&sched->kss_lock);
    ......
    if (msg->ksm_type == KSOCK_MSG_NOOP) {
        /* The packet is only for ZC ACK, piggyback the ack_cookie to a norm
        ltx = conn->ksnc_tx_piggyback;

        if (ltx != NULL) {
            ltx->tx_msg.ksm_ack_cookie = msg->ksm_ack_cookie;
            atomic_sub (tx->tx_nob, &conn->ksnc_tx_nob);
            if (ltx->tx_list.next != &conn->ksnc_tx_queue) {
                ltx = list_entry(ltx->tx_list.next, ksock_tx_t, tx_list);
                LASSERT(ltx->tx_msg.ksm_ack_cookie == 0);  /* It's impossibl
                conn->ksnc_tx_piggyback = ltx;
            } else { /* no more packet in tx_queue */
                conn->ksnc_tx_piggyback = NULL;
            }
            ltx = tx;     /* It's going to be added to free list */

        } else {
            /* all packets have ACK or no packet in tx_queue */
            list_add_tail (&tx->tx_list, &conn->ksnc_tx_queue);
            ltx = NULL;
        }
    } else {
        if (!list_empty(&conn->ksnc_tx_queue)) {
            /* don't need to search through the list, NOOP packets are
             * always continuous in the tx_queue, NOOP packet is at the
             * end of tx_queue only if there is no packet can be
             * piggybacked, if the last packet is not NOOP, there is NO
             * NOOP in the tx_queue and we don't need to search backward */

            ltx = list_entry(conn->ksnc_tx_queue.prev, ksock_tx_t, tx_list);
```

```
                if (ltx->tx_msg.ksm_type == KSOCK_MSG_NOOP &&
                    ltx->tx_resid == ltx->tx_nob) {
                    /* It's a NOOP packet */
                    LASSERT(ltx->tx_msg.ksm_ack_cookie != 0);
                    msg->ksm_ack_cookie = ltx->tx_msg.ksm_ack_cookie;

                    atomic_sub (ltx->tx_nob, &conn->ksnc_tx_nob);
                    list_del(&ltx->tx_list);
                } else
                    ltx = NULL;
            }
            list_add_tail (&tx->tx_list, &conn->ksnc_tx_queue);
            if (msg->ksm_ack_cookie == 0 && conn->ksnc_tx_piggyback == NULL)
                conn->ksnc_tx_piggyback = tx;
        }
        ...... /* enqueue tx and wakeup etc ... */
        spin_unlock_bh (&sched->kss_lock);
        if (ltx != NULL) {
            /* it's safe to call it at here,  ksocknal_free_tx()
             * is non-block for NOOP packet */
            ksocknal_free_tx(ltx);
        }
    }
```

4.4 ksocknal_scheduler

Just a few lines need to be added in ksocknal_schedule()

```
    int ksocknal_scheduler (void *arg)
    {
        ......
        if (!list_empty (&sched->kss_tx_conns)) {
            conn = list_entry(sched->kss_tx_conns.next,
                              ksock_conn_t, ksnc_tx_list);
            ......
            if (conn->ksnc_tx_piggyback == tx) {
                ksock_tx_t      *ltx;
                LASSERT(tx->tx_msg.ksm_ack_cookie == 0);
                conn->ksnc_tx_piggyback = NULL;
                if (!list_empty(&conn->ksnc_tx_queue)) {
                    ltx = list_entry(conn->ksnc_tx_queue.next, ksock_tx_t, tx_li
                    if (ltx->tx_msg.ksm_ack_cookie == 0)
                        conn->ksnc_tx_piggyback = ltx;
                }
            }
```

```
    ......
}
```

4.5 ksocknal_compat_protocol

```
ksock_protocol_t *ksocknal_convert_version(__u16 major, __u16 minor)
{
    /* We can move checking of minor version to pro_receive() in the future
    if (major == le16_to_cpu(KSOCK_PROTO_V1_MAJOR) && minor == le16_to_cpu(K
        return &ksocknal_protocol_v1x;
    else if (major == le16_to_cpu(KSOCK_PROTO_V2_MINOR && minor == le16_to_c
        return &ksocknal_protocol_v2x;
    else
        return NULL;
}
```

4.6 ksocknal_send_hello_v1

```
int ksocknal_send_hello_v1 (ksock_conn_t *conn, ksock_hello_msg_t *hello)
{
    cfs_socket_t        *sock = conn->ksnc_sock;
    lnet_hdr_t           hdr;
    lnet_magicversion_t *hmv = (lnet_magicversion_t *)&hdr.dest_nid;
    int                  rc;
    /* Re-organize V2.x message header to V1.x (lnet_dhr_t) header and send
    hmv->magic         = hello->kshm_magic;
    hmv->version_major = hello->kshm_version_major;
    hmv->version_minor = hello->kshm_version_minor;
    hdr.src_nid        = hello->kshm_src_nid;
    hdr.src_pid        = hello->kshm_src_pid;
    hdr.type           = cpu_to_le32 (LNET_MSG_HELLO);
    hdr.payload_length = cpu_to_le32 (le32_to_cpu(hello->kshm_nips) * sizeof
    hdr.msg.hello.type = hello->kshm_ctype;
    hdr.msg.hello.incarnation = hello->kshm_src_incarnation;
    rc = libcfs_sock_write(sock, &hdr, sizeof(hdr), lnet_acceptor_timeout())

    return rc;
}
```

4.7 ksocknal_send_hello_v2

```
int ksocknal_send_hello_v2 (ksock_conn_t *conn, ksock_hello_msg_t *msg)
```

```
{
    cfs_socket_t   *sock = conn->ksnc_sock;
    int             rc;
    rc = libcfs_sock_write(sock, msg, offsetof(ksock_msg_t, ksm_hello.kshm_i
    return (rc);
}
```

4.8 ksocknal_recv_hello_v1

```
int ksocknal_recv_hello_v1 (ksock_conn_t *conn, ksock_hello_msg_t *hello, in
{
    cfs_socket_t        *sock = conn->ksnc_sock;
    int                  rc;
    lnet_hdr_t           hdr
    rc = libcfs_sock_read(sock, &hdr.src_nid, sizeof (hdr) - offsetof (lnet_
    if (rc != 0) {
        CERROR("......");
        return (rc);
    }
    if (hdr->type != cpu_to_le32 (LNET_MSG_HELLO)) {
        CERROR("......");
        return (-EPROTO);
    }
    hello->kshm_src_nid         = le64_to_cpu (hdr.src_nid);
    hello->kshm_src_pid         = le32_to_cpu (hdr.src_pid);
    hello->kshm_ctype           = le32_to_cpu (hdr.msg.hello.type);
    hello->kshm_src_incarnation = le64_to_cpu (hdr.msg.hello.incarnation);
    hello->kshm_nips            = le32_to_cpu (hdr.payload_length) / sizeof
    return 0;
}
```

4.9 ksocknal_recv_hello_v2

```
int ksocknal_recv_hello_v2 (ksock_conn_t *conn, ksock_hello_msg_t *hello, in
{
    cfs_socket_t      *sock = conn->ksnc_sock;
    int                rc;
    rc = libcfs_sock_read(sock, &hello->kshm_src_nid,
                          offsetof(ksock_hello_msg_t, kshm_ips) - offsetof(ks
                          timeout);
    if (rc != 0) {
        CERROR("......");
        return (rc);
```

```
        }
        if (hello->kshm_magic != LNET_PROTO_TCP_MAGIC) {
            __swab32s(&hello->kshm_nips);
            __swab32s(&hello->kshm_ctype);
            __swab32s(&hello->kshm_src_pid);
            __swab64s(&hello->kshm_src_nid);
            __swab32s(&hello->kshm_dst_pid);
            __swab64s(&hello->kshm_dst_nid);
            __swab64s(&hello->kshm_src_incarnation);
            __swab64s(&hello->kshm_dst_incarnation);
        }
        return 0;
    }
```

4.10 ksocknal_pack_msg_v1

```
    static void ksocknal_pack_msg_v1(ksock_tx_t *tx)
    {
        lnet_hdr_t       *hdr = &tx->tx_lnetmsg->msg_hdr;
        LASSERT(tx->tx_msg.ksm_type != KSOCK_NOOP_MSG);
        LASSERT(tx->tx_lnetmsg != NULL);
        tx->tx_iov[0].iov_base = (void *)&tx->tx_lnetmsg->msg_hdr;
        tx->tx_iov[0].iov_len  = sizeof(lnet_hdr_t);
        tx->tx_resid = tx->tx_nob = tx->tx_lnetmsg->msg_len + sizeof(lnet_hdr_t)
    }
```

4.11 ksocknal_pack_msg_v2

```
    static void ksocknal_pack_msg_v2(ksock_tx_t *tx)
    {
        tx->tx_iov[0].iov_base = (void *)&tx->tx_msg;
        if (tx->tx_lnetmsg != NULL) {
            LASSERT(tx->tx_msg.ksm_type != KSOCK_MSG_NOOP);
            tx->tx_msg.ksm_u.normal.ksnm_hdr = tx->tx_lnetmsg->msg_hdr;
            tx->tx_iov[0].iov_len = offsetof(ksock_msg_t, ksm_u.normal.ksnm_payl
            tx->tx_resid = tx->tx_nob = offsetof(ksock_msg_t,  ksm_u.normal.ksnm
        } else {
            LASSERT(tx->tx_msg.ksm_type == KSOCK_MSG_NOOP);
            tx->tx_iov[0].iov_len = offsetof(ksock_msg_t, ksm_u.normal.ksnm_hdr)
            tx->tx_resid = tx->tx_nob = offsetof(ksock_msg_t,  ksm_u.normal.ksnm
        }
    }
```

4.12 ksocknal_unpack_msg_v1

```
static void ksocknal_unpack_msg_v1(ksock_msg_t *msg)
{
    msg->ksm_type = KSOCK_MSG_LNET;
    msg->ksm_req_cookie = msg->ksm_ack_cookie = 0;
}
```

4.13 ksocknal_unpack_msg_v2

```
static void ksocknal_unpack_msg_v2(ksock_msg_t *msg)
{
    return;  /* Do nothing */
}
```

5 State Machine

We have described state machine in HLD with detail information.