RDMA Contest Proposal

Ocean University of China

Oct.7, 2015

Index

1 Introduction

1.1 Background

With the rapid development of CPU, the performance of I / O system has become a bottleneck which restricts Server Performance. In order to make up for the deficiencies of PCI bus architecture, InfiniBand architecture came into being.

InfiniBand (IB) is a high-speed, low latency, low CPU overhead, highly efficient and scalable server and storage interconnect technology. One of the key capabilities of InfiniBand is its support for native Remote Direct Memory Access (RDMA). InfiniBand enables data transfer between servers and between server and storage without the involvement of the host CPU in the data path. InfiniBand provides various technology or solution speeds ranging from 10Gb/s (SDR) up to 56Gb/s (FDR) per port, using copper and optical fiber connections. InfiniBand efficiency and scalability have made it the optimal performance and cost/performance interconnect solution for the world's leading high-performance computing, cloud, Web 2.0, storage, database and financial data centers and applications. The development of InfiniBand network will become the main trend of network communication.

1.2 Development of network communication

1.2.1 Introduction of TCP/IP

The Internet protocol suite is the computer networking model and set of communications protocols used on the Internet and similar computer networks. It is commonly known as TCP/IP, because among many protocols, the Transmission Control Protocol (TCP) and the Internet Protocol (IP) is the accepted and most widely used protocol in Internet. Often also called the Internet model, it was originally also known as the DoD model, because the development of the networking model was funded by DARPA, an agency of the United States Department of Defense.[1]

1.2.2 Development of RDMA

In computing, remote direct memory access (RDMA) is a direct memory access from the memory of one computer into that of another without involving either one's operating system. This permits high-throughput, low-latency networking, which is especially useful in massively parallel computer clusters.

RDMA supports zero-copy networking by enabling the network adapter to transfer data directly to or from application memory, eliminating the need to copy data between application

memory and the data buffers in the operating system. Such transfers require no work to be done by CPUs, caches, or context switches, and transfers continue in parallel with other system operations. When an application performs an RDMA Read or Write request, the application data is delivered directly to the network, reducing latency and enabling fast message transfer.[2]

1.3 Overview of memcached

Memcached is a high performance distributed memory object caching system for dynamic Web applications to reduce the database load.It does this by caching data and objects in memory to reduce the frequency read from the database,thereby increasing the speed of the dynamic and database driven web sites.

Memcached is simple yet powerful. Its simple design promotes quick deployment, ease of development, and solves many problems facing large data caches. Its API is available for most popular languages.[3]

1.4 What we do

We try to modify the communication of memcached from SOCKET communication to RDMA communication. In chapter 2 and chapter 3, we give brief introduction of TCP/IP and RDMA. Chapter 4 and chapter 5 are the main parts of this report. Our original analysis is given in chapter 4. Details of our modification are given in chapter 5. At last, summarize and future work are given.

2 TCP/IP

2.1 Hierarchical structure of TCP/IP

TCP/IP provides end-to-end connectivity specifying how data should be packetized, addressed, transmitted, routed and received at the destination. This functionality is organized into four abstraction layers which are used to sort all related protocols according to the scope of networking involved.From lowest to highest, the layers are the link layer, containing communication technologies for a single network segment (link); the internet layer, connecting hosts across independent networks, thus establishing internet working; the transport layer handling host-to-host communication; and the application layer, which provides process-to-process application data exchange.Internet protocol suite(TCP/IP) structure is shown in the following Figure 1.[1]

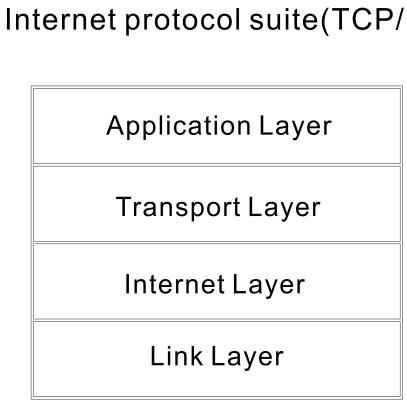

Figure 1 Hierarchical structure diagram of TCP/IP

2.2 Socket communication programming

A schematic diagram for socket communication programming is shown in Figure 2. Function-analysis is shown in Table 1. The server creates a socket and binds it to listen, and then calls accept command. Block the main thread and then wait for a connection till the socket created by the client give a link request. Thereby, the two sides establish a connection. After the link is established, communicate through via recv() and send() and other functions and then end the communication by calling close().

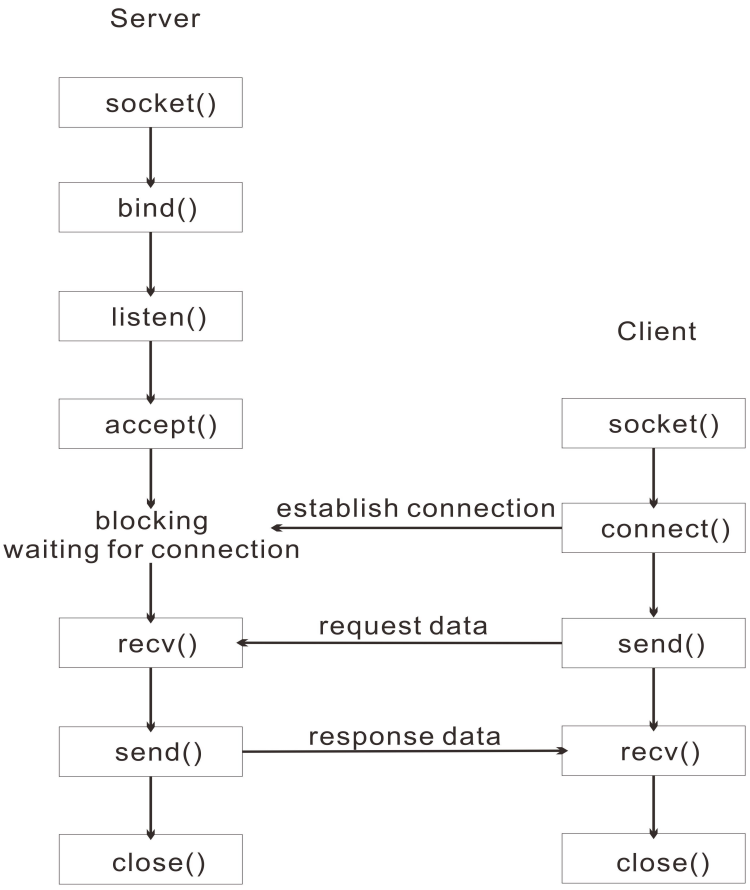

Figure 2 Socket communication flow chart

Socket()	Create new socket
Bind()	The address specified by addr assigned to the representatives of the socket with file descriptor
Listen()	Listening socket
Accept()	Blocking waiting for connection
Connect()	Establish connection
Recv()	Receive information through socket
Send()	Send information through socket
Close()	Close Session

Table 1 Function-analysis of socket communication

3 RDMA

3.1 Introduction

Direct memory access (DMA) is an ability of a device to access host memory directly, without the intervention of the CPU(s).

RDMA (Remote DMA) is the ability of accessing (i.e. reading from or writing to) memory on a remote machine without interrupting the processing of the CPU(s) on that system.

RDMA advantages:

(1) Direct user-level access to HW
  a.Zero-copy
  b.Kernel bypass in the fast path
  c.User buffers are accessed directly
  d.No CPU involvement

(2) Asynchronous communication
  a.Computation and communication overlap

(3) HW managed transport
  a.SW deals with buffers, not packets
  b.Per “socket” context maintained in HW

(4) No need for OS to multiplex HW
  a.Explicit memory management
  b.Improve small packet throughput

3.2 Comparison between TCP/IP and RDMA

RDMA (Remote Direct Memory Access) is passed directly through the network to a data storage area of a computer. It will quickly move data from one system to the remote system memory without any impact on the operating system, so you do not need to use too many processing functions of the computer. It eliminates the external memory copy operation and exchange text, which can free up bus space and CPU cycles to improve application performance.[4]

Ordinary card integrated the functions of supporting hardware checksum and it has improved the software, thereby reducing the amount of transmitted data copy, but you can not reduce the amount of received data copy which is part of the amount to be occupied by a large number of compute cycles of the processor. NIC ordinary working process shown as follows: first, received data packet cache to the system. After treatment, the corresponding data is assigned to a TCP connection. Next, the receiving system then unsolicited application with the appropriate link and the data from the system buffer is copied to the target memory address.

Ethernet has been able to meet the high-performance applications on the network throughput requirements with high throughput and cost advantages. In order to link with a high-performance network applications, the main problem is the application throughput. Under normal circumstances, the system need to take up CPU resources in the host CPU when continue to handle Ethernet communications. CPU speed will restrict network data rate; continuous processing such communication will lead to CPU performance downgrade; for multi-port Gigabit or 10 Gigabit Ethernet single-port, these problems become more serious.Working mechanism comparison chart of TCP / IP and RDMA is shown in Figure3.

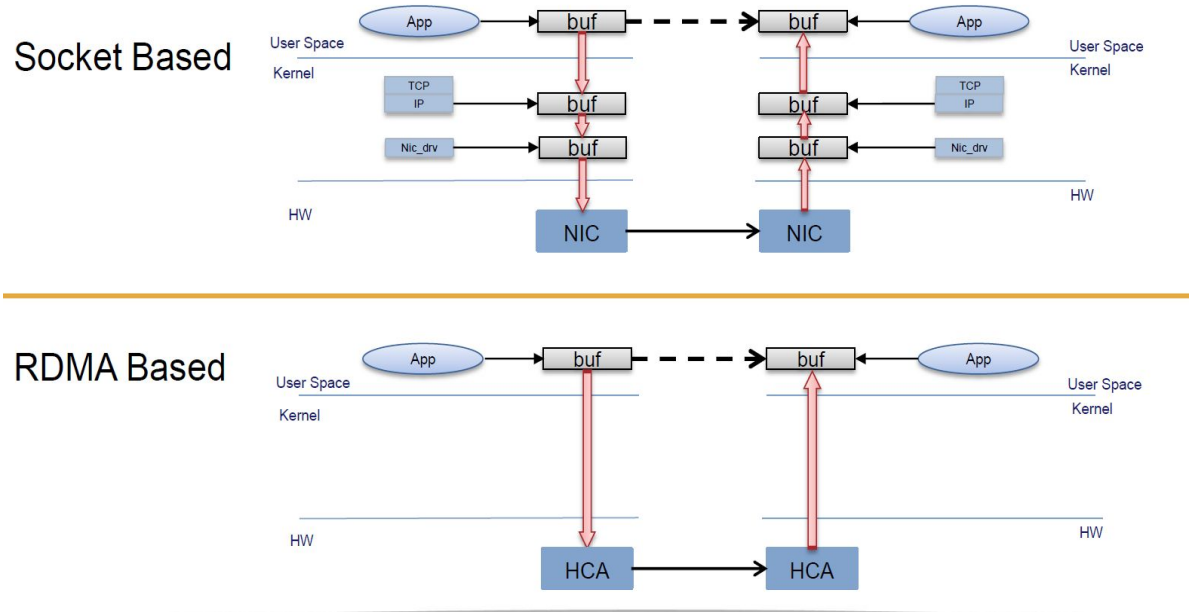

Figure 3 Working mechanism comparison chart of TCP / IP and RDMA

3.3 RDMA communication programming

InfiniBand connection establishment process as shown below:

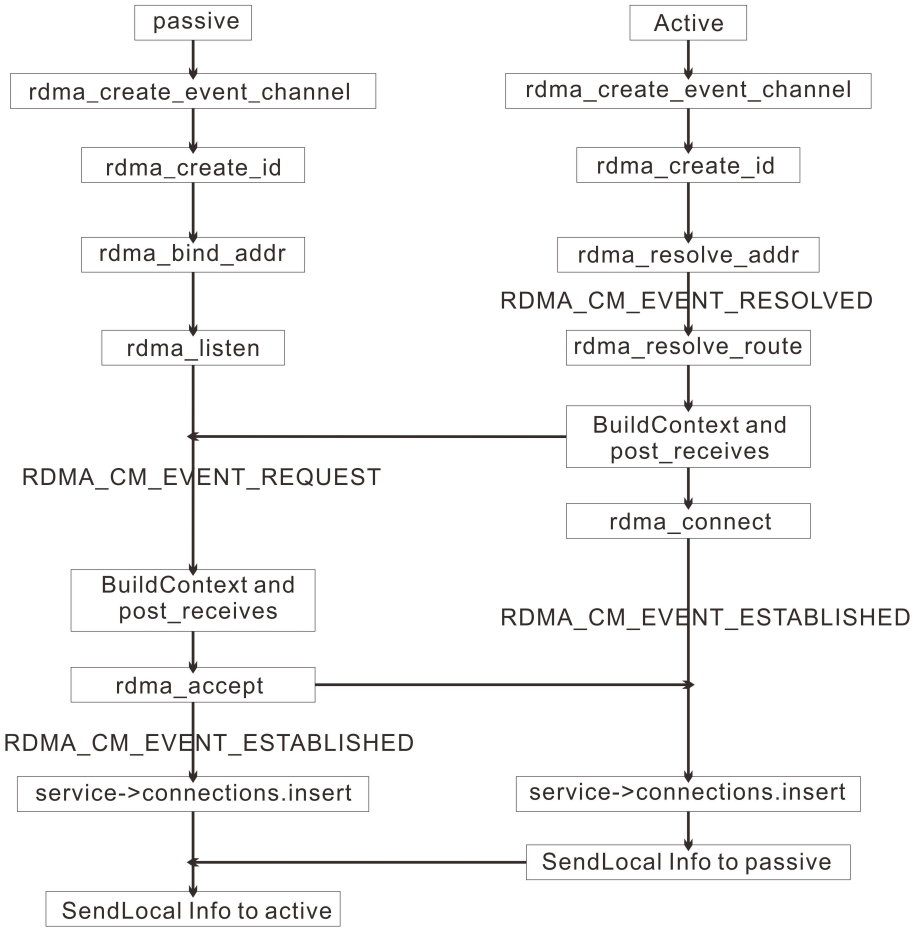

Figure 4 Communication function schematic of RDMA

After the connection is established, you can call ibv_post_recv and ibv_post_send send and receive data, and the send and receive requests have been placed in QP, the background need to call ibv_poll_cq to process the request one by one. During InfiniBand connection, if there is a failure of data sent or received , the send and receive of all the data after will fail. So, once detected WC status is not successful, we need to deal with the error immediately(at this point, you'd better disconnect).

4 Analysis of Memcached (main work)

In order to modify the communication part of memcached, we have to read the source code of memcached sincerely. Here we give our understanding of memcached in detail.

4.1 Flow chart

The main flow of Memcached is shown in Figure 5：

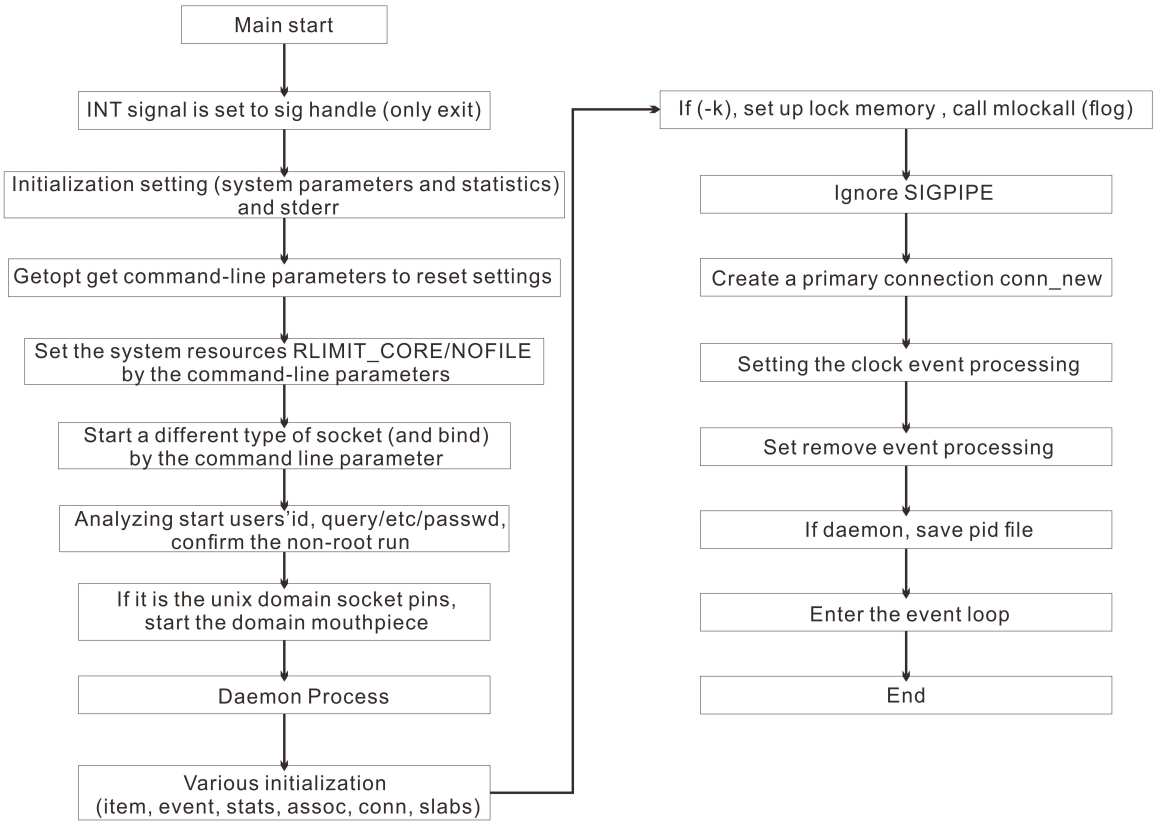

Figure 5 Flow chart of memcached

4.2 Function call

The function call involved in socket communication is shown in Figure 6, where the green represents the socket communication function and the blue represents the function to be modified.

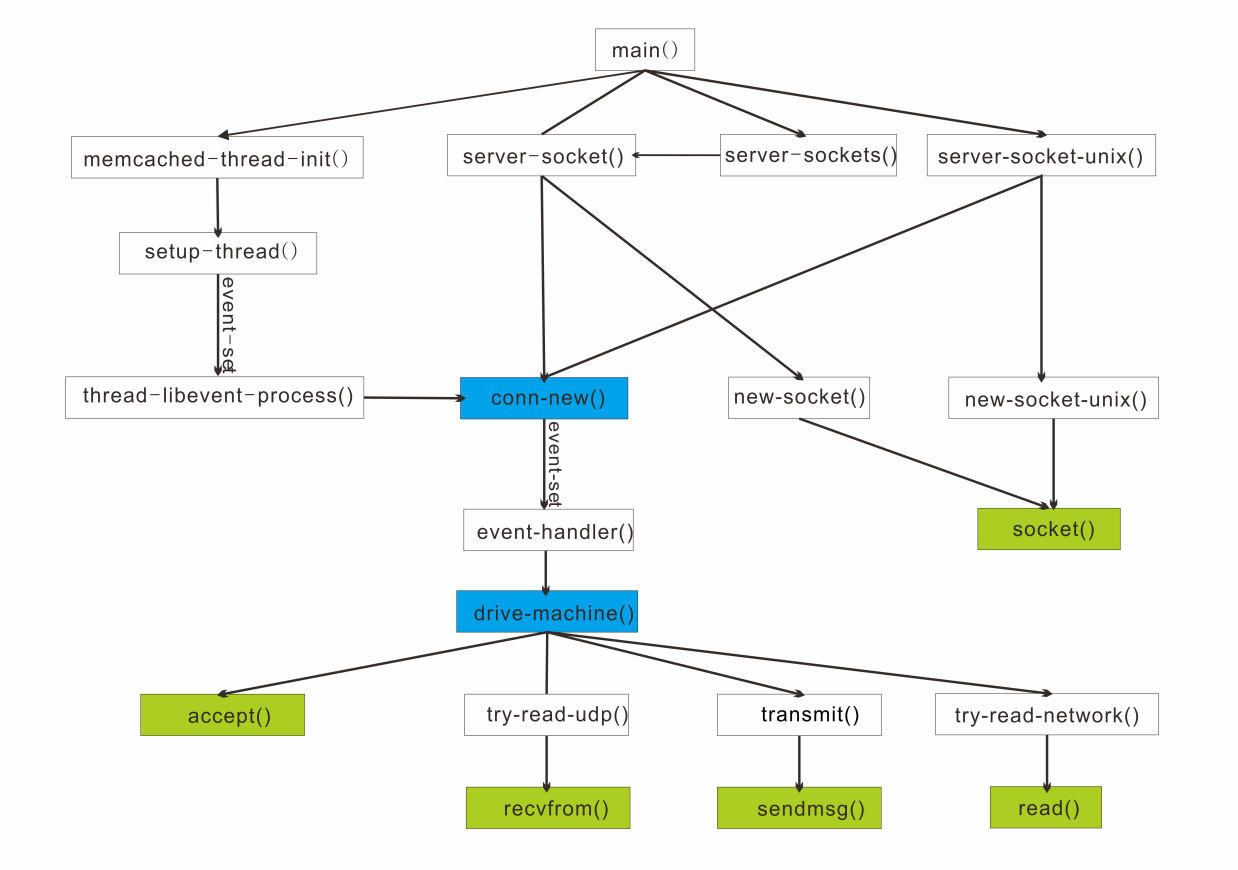

Figure 6 RDMA function sight

4.3 Analysis of function

Function analysis involved in Figure 6 is shown in Table 2:

Function name	Instruction
server_socket()	The main thread calls,create a listener link
conn_new()	Create conn, and call event_set (),Register a callback function event_handler ()
memcached_thread_init()	Initialize each thread
setup_thread()	call event_set (),Register a callback function thread_event_process()
thread_event_process()	Call when the main thread over the information, for creating new conn
event_handler()	Link corresponding callback function, thereby entering the drive machine

drive_machine()	State machine, all the time in the memcached will be handled. Its implementation is: There is a huge switch case in a while loop. Based on the connection object "conn" current connection status conn_state, enter different cases. Some cases let the program enter another case in the next cycle by changing the connection state of"conn" object. After several cycles, program will eventually enter into "no out-degree of a vertex" ,then end state machine.

Table 2 Function Analysis

4.4 Analysis of communication

Memcached communication diagram shown in Figure 7, the main thread creates listening_conn, and calls drive_machine function when calling into the monitor branch, and then blocks waiting for client connections. When a client connection request came, the two sides establish a connection and the main thread will package the socket received into CQ_ITEM and then push to the CQ queue of worker threads. After the worker receives, calling thread_libevent_process() function, creating a new link, and calls drive_machine. Since then, the worker thread has established good communication tasks for client connections and will have been responsible for the appropriate client. When work thread doing communication tasks, the change in state which new conn has in drive machine is shown in Figure 8.

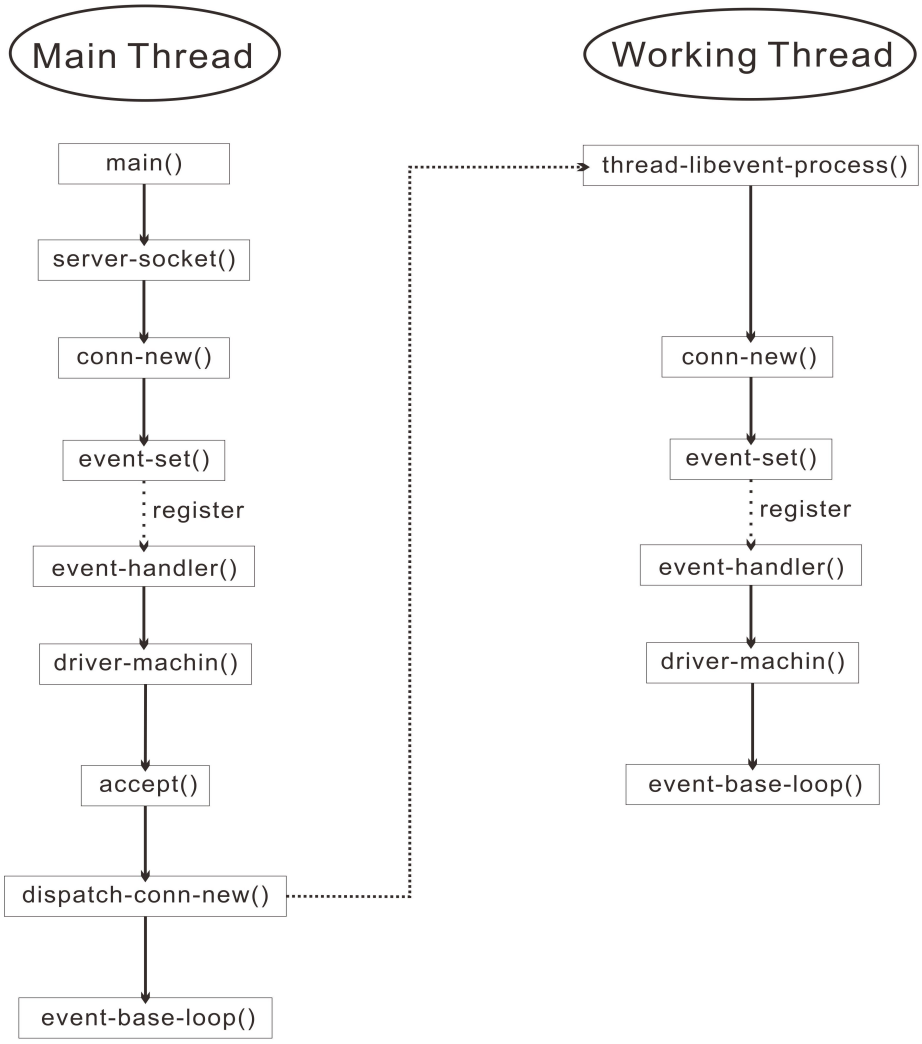

Figure 7 Memcached communication diagram

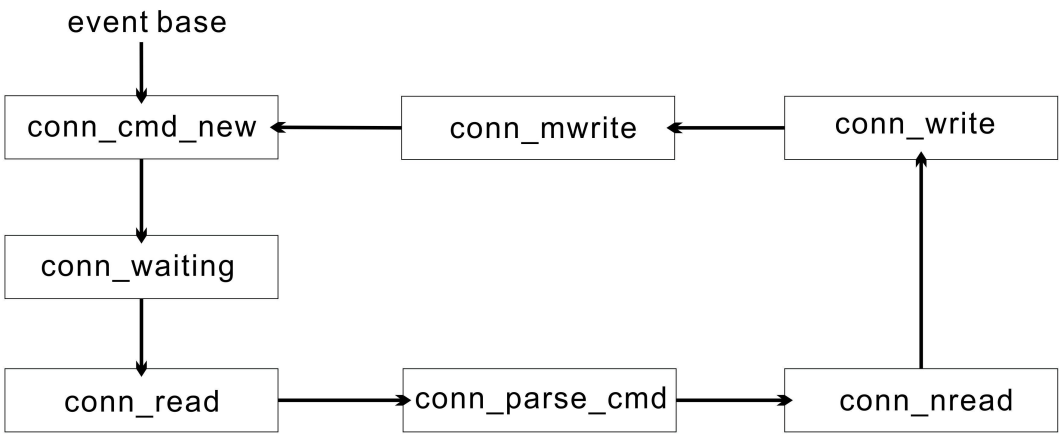

Figure 8 State transition diagram of drive_machine

5 Modify Memcached with RDMA (main work)

5.1 Modification strategy

We merge the Conn of memcached with the RES which is necessary for RDMA communication, and then modify the functions given in Figure 6 with blue color.

Two new program files are added:
(1) rdma_sock.h : the header file of "rdma_sock.c"
(2) rdma_sock.c : the function used in RDMA communication.

Two files are modified:
(1)Memcached.h : the header file of "Memcached.c"
(2)Memcached.c : two functions are modified
a) Conn_new() : Create conn structure, and register a callback function event_handler.
b) Drive_machine() : Handle all the connected event, mainly make uo by a while loop and switch.

5.2 Source code of modification

5.2.1 rdma_sock.h (new added)

```
#ifndef SOCK_H
#define SOCK_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdint.h>
#include <inttypes.h>
#include <endian.h>
#include <byteswap.h>
#include <getopt.h>
#include <sys/time.h>
#include <arpa/inet.h>
#include <infiniband/verbs.h>

#define MAX_POLL_CQ_TIMEOUT 2000

#if __BYTE_ORDER == __LITTLE_ENDIAN
static inline uint64_t htonll(uint64_t x) { return bswap_64(x); }
```

```
static inline uint64_t ntohll(uint64_t x) { return bswap_64(x); }
#elif __BYTE_ORDER == __BIG_ENDIAN
static inline uint64_t htonll(uint64_t x) { return x; }
static inline uint64_t ntohll(uint64_t x) { return x; }
#else
#error __BYTE_ORDER is neither __LITTLE_ENDIAN nor __BIG_ENDIAN
#endif

/* structure of test parameters */
struct config_t {
    const char          *dev_name; /* IB device name */
    char                *server_name;   /* daemon host name */
    u_int32_t           tcp_port;      /* daemon TCP port */
    int                 ib_port; /* local IB port to work with */
};

/* structure to exchange data which is needed to connect the QPs */
struct cm_con_data_t {
    uint64_t            addr;          /* Buffer address */
    uint32_t            rkey;          /* Remote key */
    uint32_t            qp_num;            /* QP number */
    uint16_t            lid;       /* LID of the IB port */
} __attribute__ ((packed));

/* structure of needed test resources */
struct resources {
    struct ibv_device_attr      device_attr;  /* Device attributes */
    struct ibv_port_attr        port_attr;     /* IB port attributes */
    struct ibv_device       **dev_list;   /* device list */
    struct ibv_context          *ib_ctx; /* device handle */
    struct ibv_pd           *pd;           /* PD handle */
    struct ibv_comp_channel       *comp_channel;  /* completion channel */
    struct ibv_cq           *cq;       /* CQ handle */
    struct ibv_qp           *qp;           /* QP handle */
    struct ibv_mr               *mr;           /* MR handle */
    char                *buf;          /* memory buffer pointer */
    int                 bsize;             /*size of memory buffer*/
    int                 sock;          /* TCP socket file descriptor */
};

int sock_daemon_connect(
    int port);

int sock_client_connect(
```

```
    const char *server_name,
    int port);

int sock_sync_data(
    int sock_fd,
    int is_daemon,
    size_t size,
    const void *out_buf,
    void *in_buf);

int sock_sync_ready(
    int sock_fd,
    int is_daemon);

static int poll_completion(
    struct resources *res);

static int post_send(
    struct resources *res);

static int post_receive(
    struct resources *res);

static void resources_init(
    struct resources *res);

static int resources_create(
    struct resources *res);

static int modify_qp_to_init(
    struct ibv_qp *qp);

static int modify_qp_to_rtr(
    struct ibv_qp *qp,
    uint32_t remote_qpn,
    uint16_t dlid);

static int modify_qp_to_rts(
    struct ibv_qp *qp);

static int connect_qp(
    struct resources *res);

static int resources_destroy(
```

```
    struct resources *res);

#endif
```

5.2.2 rdma_sock.c (new added)

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include "rdma_sock.h"
/****************************************
* Function: sock_send
****************************************/
static int sock_send(  int sock_fd,  size_t size,   const void *buf) {
    int rc;
    retry_after_signal:
    rc = send(sock_fd, buf, size, 0);
    if (rc != size) {
        fprintf(stderr, "send failed: %s, rc=%d\n", strerror(errno), rc);
        if ((errno == EINTR) && (rc != 0))
            goto retry_after_signal;       /* Interrupted system call */
        if (rc)
            return rc;
        else
            return -1;
    }
    return 0;
}

/****************************************
* Function: sock_sync_data
****************************************/
int sock_sync_data(int sock_fd,int is_daemon,size_t size,const void *out_buf, void *in_buf){
    int rc;
    if (is_daemon) {
        rc = sock_send(sock_fd, size, out_buf);
        if (rc)
```

```
                return rc;
        rc = sock_recv(sock_fd, size, in_buf);
        if (rc)
                return rc;
    } else {
        rc = sock_recv(sock_fd, size, in_buf);
        if (rc)
                return rc;
        rc = sock_send(sock_fd, size, out_buf);
        if (rc)
                return rc;
    }
    return 0;
}

/****************************************
* Function: sock_sync_ready
****************************************/
int sock_sync_ready( int sock_fd,  int is_daemon){
    char cm_buf = 'a';
    return sock_sync_data(sock_fd, is_daemon, sizeof(cm_buf), &cm_buf, &cm_buf);
}
struct config_t config = {
    "mthca0",               /* dev_name */
    NULL,                   /* server_name */
    19875,                  /* tcp_port */
    1                    /* ib_port */
};

/****************************************
* Function: poll_completion
****************************************/
static int poll_completion(struct resources *res){
    struct ibv_wc wc;
    void *ev_ctx;
    struct ibv_cq *ev_cq;
    int rc;

    fprintf(stdout, "waiting for completion event\n");
    /* Wait for the completion event */
    if (ibv_get_cq_event(res->comp_channel, &ev_cq, &ev_ctx)) {
        fprintf(stderr, "failed to get cq_event\n");
        return 1;
    }
```

```
	fprintf(stdout, "got completion event\n");
	/* Ack the event */
	ibv_ack_cq_events(ev_cq, 1);

	/* Request notification upon the next completion event */
	rc = ibv_req_notify_cq(ev_cq, 0);
	if (rc) {
		fprintf(stderr, "Couldn't request CQ notification\n");
		return 1;
	}
	/* in a real program, the user should empty the CQ before waiting for the next completion
event */
	/* poll the completion that causes thew event (if exists) */
	rc = ibv_poll_cq(res->cq, 1, &wc);
	if (rc < 0) {
		fprintf(stderr, "poll CQ failed\n");
		return 1;
	}
	/* check if the CQ is empty (there can be an event event when the CQ is empty, this can
happen
		when more than one completion(s) are being created. Here we create only one
completion
		so empty CQ means there is an error) */
	if (rc == 0) {
		fprintf(stderr, "completion wasn't found in the CQ after timeout\n");
		return 1;
	}

	fprintf(stdout, "completion was found in CQ with status 0x%x\n", wc.status);

	/* check the completion status (here we don't care about the completion opcode */
	if (wc.status != IBV_WC_SUCCESS) {
		fprintf(stderr, "got bad completion with status: 0x%x, vendor syndrome: 0x%x\n",
			wc.status, wc.vendor_err);
		return 1;
	}
	return 0;
}

/******************************************
* Function: post_send
******************************************/
static int post_send(struct resources *res){
```

```
    struct ibv_send_wr sr;
    struct ibv_sge sge;
    struct ibv_send_wr *bad_wr;
    int rc;


    /* prepare the scatter/gather entry */
    memset(&sge, 0, sizeof(sge));

    sge.addr = (uintptr_t)res->buf;
    sge.length = res->bsize;
    sge.lkey = res->mr->lkey;

    /* prepare the SR */
    memset(&sr, 0, sizeof(sr));

    sr.next             = NULL;
    sr.wr_id            = 0;
    sr.sg_list          = &sge;
    sr.num_sge          = 1;
    sr.opcode           = IBV_WR_RDMA_READ;
    sr.send_flags = IBV_SEND_SIGNALED;

    sr.wr.rdma.remote_addr = res->remote_props.addr;
    sr.wr.rdma.rkey                = res->remote_props.rkey;

    /* there is a Receive Request in the responder side, so we won't get any into RNR flow */
    rc = ibv_post_send(res->qp, &sr, &bad_wr);
    if (rc) {
        fprintf(stderr, "failed to post SR\n");
        return 1;
    }
    fprintf(stdout, "Send Request was posted\n");

    return 0;
}


/******************************************
* Function: resources_create
******************************************/
static int resources_create(struct resources *res){
    struct ibv_qp_init_attr qp_init_attr;
    struct ibv_device *ib_dev = NULL;
```

```
size_t size;
int i;
int mr_flags = 0;
int cq_size = 0;
int num_devices;
int rc;

fprintf(stdout, "searching for IB devices in host\n");

/* get device names in the system */
res->dev_list = ibv_get_device_list(&num_devices);
if (!res->dev_list) {
	fprintf(stderr, "failed to get IB devices list\n");
	return 1;
}

/* if there isn't any IB device in host */
if (!num_devices) {
	fprintf(stderr, "found %d device(s)\n", num_devices);
	return 1;
}

fprintf(stdout, "found %d device(s)\n", num_devices);

/* search for the specific device we want to work with */
for (i = 0; i < num_devices; i ++) {
	if (!strcmp(ibv_get_device_name(res->dev_list[i]), config.dev_name)) {
		ib_dev = res->dev_list[i];
		break;
	}
}

/* if the device wasn't found in host */
if (!ib_dev) {
	fprintf(stderr, "IB device %s wasn't found\n", config.dev_name);
	return 1;
}

/* get device handle */
res->ib_ctx = ibv_open_device(ib_dev);
if (!res->ib_ctx) {
	fprintf(stderr, "failed to open device %s\n", config.dev_name);
	return 1;
}
```

```
/* query port properties    */
if (ibv_query_port(res->ib_ctx, config.ib_port, &res->port_attr)) {
      fprintf(stderr, "ibv_query_port on port %u failed\n", config.ib_port);
      return 1;
}

/* allocate Protection Domain */
res->pd = ibv_alloc_pd(res->ib_ctx);
if (!res->pd) {
      fprintf(stderr, "ibv_alloc_pd failed\n");
      return 1;
}

res->comp_channel = ibv_create_comp_channel(res->ib_ctx);
if (!res->comp_channel) {
      fprintf(stderr, "ibv_create_comp_channel failed\n");
      return 1;
}

/* each side will send only one WR, so Completion Queue with 1 entry is enough */
cq_size = 1;
res->cq = ibv_create_cq(res->ib_ctx, cq_size, NULL, res->comp_channel, 0);
if (!res->cq) {
      fprintf(stderr, "failed to create CQ with %u entries\n", cq_size);
      return 1;
}

/* Arm the CQ before any completion is expected (to prevent races) */
rc = ibv_req_notify_cq(res->cq, 0);
if (rc) {
      fprintf(stderr, "failed to arm the CQ\n");
      return 1;
}
fprintf(stdout, "CQ was armed\n");


mr_flags =IBV_ACCESS_LOCAL_WRITE ;
res->mr = ibv_reg_mr(res->pd, res->buf, res->bsize, mr_flags);
if (!res->mr) {
      fprintf(stderr, "ibv_reg_mr failed with mr_flags=0x%x\n", mr_flags);
      return 1;
}
```

```
    fprintf(stdout, "MR was registered with addr=%p, lkey=0x%x, rkey=0x%x,
flags=0x%x\n", res->buf, res->mr->lkey, res->mr->rkey, mr_flags);


    /* create the Queue Pair */
    memset(&qp_init_attr, 0, sizeof(qp_init_attr));

    qp_init_attr.qp_type       = IBV_QPT_RC;
    qp_init_attr.sq_sig_all = 1;
    qp_init_attr.send_cq       = res->cq;
    qp_init_attr.recv_cq       = res->cq;
    qp_init_attr.cap.max_send_wr    = 1;
    qp_init_attr.cap.max_recv_wr    = 1;
    qp_init_attr.cap.max_send_sge = 1;
    qp_init_attr.cap.max_recv_sge = 1;

    res->qp = ibv_create_qp(res->pd, &qp_init_attr);
    if (!res->qp) {
        fprintf(stderr, "failed to create QP\n");
        return 1;
    }
    fprintf(stdout, "QP was created, QP number=0x%x\n", res->qp->qp_num);

    return 0;
}

/******************************************
* Function: modify_qp_to_init
******************************************/
static int modify_qp_to_init(struct ibv_qp *qp) {
    struct ibv_qp_attr attr;
    int flags;
    int rc;

    /* do the following QP transition: RESET -> INIT */
    memset(&attr, 0, sizeof(attr));

    attr.qp_state = IBV_QPS_INIT;
    attr.port_num = config.ib_port;
    attr.pkey_index = 0;
    /* we don't do any RDMA operation, so remote operation is not permitted */
    attr.qp_access_flags = 0;

    flags = IBV_QP_STATE | IBV_QP_PKEY_INDEX | IBV_QP_PORT |
```

```
IBV_QP_ACCESS_FLAGS;

    rc = ibv_modify_qp(qp, &attr, flags);
    if (rc) {
        fprintf(stderr, "failed to modify QP state to INIT\n");
        return rc;
    }

    return 0;
}

/****************************************
* Function: modify_qp_to_rtr
****************************************/
static int modify_qp_to_rtr(struct ibv_qp *qp,uint32_t remote_qpn,uint16_t dlid){
    struct ibv_qp_attr attr;
    int flags;
    int rc;

    /* do the following QP transition: INIT -> RTR */
    memset(&attr, 0, sizeof(attr));

    attr.qp_state = IBV_QPS_RTR;
    attr.path_mtu = IBV_MTU_256;
    attr.dest_qp_num = remote_qpn;
    attr.rq_psn = 0;
    attr.max_dest_rd_atomic = 0;
    attr.min_rnr_timer = 0x12;
    attr.ah_attr.is_global = 0;
    attr.ah_attr.dlid = dlid;
    attr.ah_attr.sl = 0;
    attr.ah_attr.src_path_bits = 0;
    attr.ah_attr.port_num = config.ib_port;

    flags = IBV_QP_STATE | IBV_QP_AV | IBV_QP_PATH_MTU | IBV_QP_DEST_QPN
| IBV_QP_RQ_PSN | IBV_QP_MAX_DEST_RD_ATOMIC | IBV_QP_MIN_RNR_TIMER;

    rc = ibv_modify_qp(qp, &attr, flags);
    if (rc) {
        fprintf(stderr, "failed to modify QP state to RTR\n");
        return rc;
    }
    return 0;
}
```

```
/******************************************
* Function: modify_qp_to_rts
******************************************/
static int modify_qp_to_rts(struct ibv_qp *qp){
    struct ibv_qp_attr attr;
    int flags;
    int rc;
    /* do the following QP transition: RTR -> RTS */
    memset(&attr, 0, sizeof(attr));

    attr.qp_state = IBV_QPS_RTS;
    attr.timeout = 0x12;
    attr.retry_cnt = 6;
    attr.rnr_retry = 0;
    attr.sq_psn = 0;
    attr.max_rd_atomic = 0;

    flags = IBV_QP_STATE | IBV_QP_TIMEOUT | IBV_QP_RETRY_CNT |
        IBV_QP_RNR_RETRY | IBV_QP_SQ_PSN | IBV_QP_MAX_QP_RD_ATOMIC;

    rc = ibv_modify_qp(qp, &attr, flags);
    if (rc) {
        fprintf(stderr, "failed to modify QP state to RTS\n");
        return rc;
    }

    return 0;
}

/******************************************
* Function: connect_qp
******************************************/
static int connect_qp( struct resources *res){
    struct cm_con_data_t local_con_data, remote_con_data, tmp_con_data;
    int rc;
    /* modify the QP to init */
    rc = modify_qp_to_init(res->qp);
    if (rc) {
        fprintf(stderr, "change QP state to INIT failed\n");
        return rc;
    }
    /* exchange using TCP sockets info required to connect QPs */
    local_con_data.addr      = htonll((uintptr_t)res->buf);
```

```
    local_con_data.rkey      = htonl(res->mr->rkey);
    local_con_data.qp_num = htonl(res->qp->qp_num);
    local_con_data.lid       = htons(res->port_attr.lid);
    fprintf(stdout, "\nLocal LID              = 0x%x\n", res->port_attr.lid);

    if (sock_sync_data(res->sock, !config.server_name, sizeof(struct cm_con_data_t),
&local_con_data, &tmp_con_data) < 0) {
        fprintf(stderr, "failed to exchange connection data between sides\n");
        return 1;
    }
    remote_con_data.addr      = ntohll(tmp_con_data.addr);
    remote_con_data.rkey      = ntohl(tmp_con_data.rkey);
    remote_con_data.qp_num = ntohl(tmp_con_data.qp_num);
    remote_con_data.lid       = ntohs(tmp_con_data.lid);

    /* save the remote side attributes, we will need it for the post SR */
    res->remote_props = remote_con_data;

    /* modify the QP to RTR */
    rc = modify_qp_to_rtr(res->qp, remote_con_data.qp_num, remote_con_data.lid);
    if (rc) {
        fprintf(stderr, "failed to modify QP state from RESET to RTS\n");
        return rc;
    }
    /* only the daemon post SR, so only he should be in RTS
       (the client can be moved to RTS as well)
     */
    rc = modify_qp_to_rts(res->qp);
    if (rc) {
        fprintf(stderr, "failed to modify QP state from RESET to RTS\n");
        return rc;
    }

        fprintf(stdout, "QP state was change to RTS\n");
    }

    /* sync to make sure that both sides are in states that they can connect to prevent packet
loose */
    if (sock_sync_ready(res->sock, !config.server_name)) {
        fprintf(stderr, "sync after QPs are were moved to RTS\n");
        return 1;
    }
    return 0;
}
```

```
/****************************************
* Function: resources_destroy
****************************************/
static int resources_destroy(struct resources *res){
    int test_result = 0;
    if (res->qp) {
        if (ibv_destroy_qp(res->qp)) {
            fprintf(stderr, "failed to destroy QP\n");
            test_result = 1;
        }
    }

    if (res->mr) {
        if (ibv_dereg_mr(res->mr)) {
            fprintf(stderr, "failed to deregister MR\n");
            test_result = 1;
        }
    }

    if (res->cq) {
        if (ibv_destroy_cq(res->cq)) {
            fprintf(stderr, "failed to destroy CQ\n");
            test_result = 1;
        }
    }

    if (res->comp_channel) {
        if (ibv_destroy_comp_channel(res->comp_channel)) {
            fprintf(stderr, "failed to destroy completion channel\n");
            test_result = 1;
        }
    }

    if (res->pd) {
        if (ibv_dealloc_pd(res->pd)) {
            fprintf(stderr, "failed to deallocate PD\n");
            test_result = 1;
        }
    }

    if (res->ib_ctx) {
        if (ibv_close_device(res->ib_ctx)) {
            fprintf(stderr, "failed to close device context\n");
```

```
                test_result = 1;
            }
        }

        if (res->dev_list)
            ibv_free_device_list(res->dev_list);

        }
        return test_result;
}
```

5.2.3 memcached.h (modified)

Only one #include command is inserted.

```
. . .
#include "rdma_sock.h"

. . .
```

5.2.4 memcached.c (modified)

There are three modifications denoted as following. Note that the deleted code is represented by strike-through line (like ~~abc~~) and the additional code is represented by shadow (like abc).

. . .

(1) the first modification

```
// Configuration information, used in the communication of RDMA
struct config_t config = {
    "mthca0",                /* dev_name */
    NULL,                    /* server_name */
    19875,                   /* tcp_port */
    1                     /* ib_port */
};
```

. . .

(2) the second modification

```
conn *conn_new(const int sfd, enum conn_states init_state, const int event_flags,
   const int read_buffer_size, enum network_transport transport, struct event_base *base) {
```

```
• • •
    //Initialize rdma_res, assign its socket to a socket descriptor received by the main thread before.
    resources_init(&c->rdma_res);
    c->rdma_res.sock=c->sfd;
    return c;
}
```

(3) the third modification

```
static void drive_machine(conn *c) {

• • •

        case conn_nread:

            • • •
            /*    now try reading from the socket */
             //res = read(c->sfd, c->ritem, c->rlbytes);

            //This is socket communication, change it to RDMA communication
            res=0;
            //reset config
            config.tcp_port=settings.port;
            c->rdma_res.buf=c->ritem;
            c->rdma_res.bsize=c->rlbytes;
            /* create resources before using them */
            if (resources_create(&c->rdma_res)) {
                fprintf(stderr, "failed to create resources\n");
                res=-1;
                goto cleanup;
            }

            /* connect the QPs */
            if (connect_qp(&c->rdma_res)) {
                fprintf(stderr, "failed to connect QPs\n");
                res=-1;
                goto cleanup;
            }

            if (post_send(&c->rdma_res)) {
                fprintf(stderr, "poll completion failed\n");
                res=-1;
```

```
                goto cleanup;
            }

            /* we expect to get a completion */
            if (poll_completion(&c->rdma_res)) {
                fprintf(stderr, "poll completion failed\n");
                res=-1;
                goto cleanup;
            }
            cleanup:

            if (resources_destroy(&c->rdma_res)) {
                fprintf(stderr, "failed to destroy resources\n");
            }



            /*if (res > 0) {
                pthread_mutex_lock(&c->thread->stats.mutex);

                c->thread->stats.bytes_read += res;

                pthread_mutex_unlock(&c->thread->stats.mutex);
                if (c->rcurr == c->ritem) {
                    c->rcurr += res;
                }
                c->ritem += res;

                c->rlbytes -= res;
                break;
            }*/

              if (res == 0) {
                 //conn_set_state(c, conn_closing);
                  c->rlbytes    = 0;
                  break;
              }

• • •
```

6 Summary & Future work

The main work of this report is:
(a) give a detailed analysis about memcached in our own point of view.
(b) modify the whole communication of memcached with RDMA.

Due to the time and hardware limitation, our modified source code has not been carried out successfully. If there is an opportunity in the future, we hope we can have a more in-depth study.

Reference

【1】https://en.wikipedia.org/wiki/Internet_protocol_suite , 2015.10.07.
【2】https://en.wikipedia.org/wiki/Remote_direct_memory_access , 2015.10.07.
【3】http://memcached.org/ , 2015.10.07.
【4】http://baike.baidu.com/view/2194201.htm, 2015.10.07.

Appendix:Team Introduction

Team members

Name	Sex	Education	University	Grade & Specialty	Personal hobby
Qinyu Wang	Male	Undergraduate	School of Mathematical Sciences, Ocean University of China	Senior / Information & Computational Sciences	System & software

Name	Sex	Education	University	Grade & Specialty	Personal hobby
Chentao Wang	Male	Undergraduate	School of Mathematical Sciences, Ocean University of China	Junior / Information & Computational Sciences	programming

Name	Sex	Education	University	Grade & Specialty	Personal hobby

Yingqi Zhao	Female	Undergraduate	School of Mathematical Sciences, Ocean University of China	Junior / Information & Computational Sciences	operating system

Instructor

Name	Linjie Zhang	Sex	Female	Academic title	Associate Professor
Department	School of Mathematical Sciences, Ocean University of China		Education / Degree	Ph.D. of Science in Engineering	