TRANSPORT LAYER CSP POSIX-POSIX (WIRED)

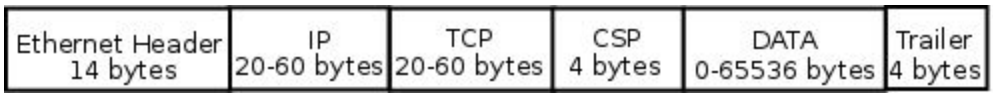
AALTO-1 SATELLITE
FIRST DELIVERY

Borja Tarraso Hueso

Helsinki, 22nd March 2014

1. CSP OVER TCP (STREAM SOCKETS)

1.1. PROTOCOL



1.2. REQUIREMENTS AND USAGE

```
$ git clone https://github.com/borjatarraso/libcsp.git
$ cd libcsp
$ git checkout aalto1
$ cd libcsp
$ ./waf configure build --enable-examples
$ cd build
$ ./csptcp server
$ ./csptcp client 127.0.0.1
```

1.3. IMPLEMENTATION

```
/*
csp_over_tcp.c - Example of CSP protocol running over stream sockets
Copyright (C) 2014 Borja Tarraso Hueso <borja.tarraso@member.fsf.org>
This file is a part of Aalto1, Aalto University, Department of Radio Science
and Engineering

This source is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This source is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
```

```
*/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <pthread.h>
#include <stdint.h>

#include <csp/csp.h>
#include <csp/csp_interface.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

#define TYPE_SERVER 1
#define TYPE_CLIENT 2
#define PORT        10
#define BUF_SIZE    250

#define SERV_PORT 10000
#define MAXLINE 4096
#define LISTENQ 1024

pthread_t rx_thread;
int server_socket, tx_channel;

int csp_socket_tx(csp_packet_t *packet, uint32_t timeout);

csp_iface_t csp_if_socket = {
    .name = "socket",
    .nexthop = csp_socket_tx,
    .mtu = BUF_SIZE,
};

int csp_socket_tx(csp_packet_t *packet, uint32_t timeout) {
    if (send(tx_channel, &packet->length, packet->length + sizeof(uint32_t) +
sizeof(uint16_t), 0) < 0)
        perror("send");
```

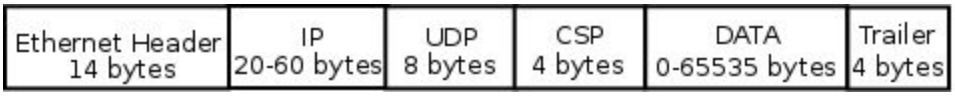
```
    csp_buffer_free(packet);
    return 1;
}

void* socket_rx(void * parameters) {
    csp_packet_t *buf = csp_buffer_get(BUF_SIZE);

    struct sockaddr_in sa;
    socklen_t length;

    int newfd = accept(server_socket, (struct sockaddr *) &sa, &length);

    while (recv(newfd, &buf->length, BUF_SIZE, 0) > 0) {
        csp_new_packet(buf, &csp_if_socket, NULL);
        buf = csp_buffer_get(BUF_SIZE);
    }

    return NULL;
}

int main(int argc, char **argv) {
    struct sockaddr_in servaddr;

    int me, other, type;
    char *message = "Testing CSP";
    csp_socket_t *sock;
    csp_conn_t *conn;
    csp_packet_t *packet;

    /* Run as either server or client */
    if (argc < 2) {
        printf("usage: %s <server/client> [ip]\n", argv[0]);
        return -1;
    } else if (strcmp(argv[1],"client") == 0 && argc != 3) {
        printf("Unspecified destination IP address\n");
        return -1;
    }

    /* Set type */
    if (strcmp(argv[1], "server") == 0) {
        me = 1;
        other = 2;

        if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
            perror("socket");
            exit(1);
        }
```

```
        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(SERV_PORT);

        if ((bind(server_socket, (struct sockaddr *) &servaddr,
sizeof(servaddr))) == -1) {
            perror("bind");
            exit(1);
        }

        if (listen(server_socket, LISTENQ) == -1) {
            perror("listen");
            exit(1);
        }

        type = TYPE_SERVER;
    } else if (strcmp(argv[1], "client") == 0) {
        me = 2;
        other = 1;

        tx_channel = socket(AF_INET, SOCK_STREAM, 0);

        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(SERV_PORT);
        inet_pton(AF_INET, argv[2], &servaddr.sin_addr);

        if (connect(tx_channel, (struct sockaddr *) &servaddr,
sizeof(servaddr)) == -1) {
            perror("connect");
            exit(1);
        }

        type = TYPE_CLIENT;
    } else {
        printf("Invalid type. Must be either 'server' or 'client'\r\n");
        return -1;
    }

    /* Init CSP and CSP buffer system */
    if (csp_init(me) != CSP_ERR_NONE || csp_buffer_init(10, 300) !=
CSP_ERR_NONE) {
        printf("Failed to init CSP\r\n");
        return -1;
    }

    /* Start socket RX task */
```

```
    pthread_create(&rx_thread, NULL, socket_rx, NULL);

    /* Set default route and start router */
    csp_route_set(CSP_DEFAULT_ROUTE, &csp_if_socket, CSP_NODE_MAC);
    csp_route_start_task(0, 0);

    /* Create socket and listen for incoming connections */
    if (type == TYPE_SERVER) {
        sock = csp_socket(CSP_SO_NONE);
        csp_bind(sock, PORT);
        csp_listen(sock, 5);
    }

    /* Super loop */
    while (1) {
        if (type == TYPE_SERVER) {
            /* Process incoming packet */
            conn = csp_accept(sock, 1000);
            if (conn) {
                packet = csp_read(conn, 0);
                if (packet)
                    printf("Received: %s\r\n", packet->data);
                csp_buffer_free(packet);
                csp_close(conn);
            }
        } else {
            /* Send a new packet */
            packet = csp_buffer_get(strlen(message));
            if (packet) {
                strcpy((char *) packet->data, message);
                packet->length = strlen(message);

                conn = csp_connect(CSP_PRIO_NORM, other, PORT, 1000,
CSP_O_NONE);
                printf("Sending: %s\r\n", message);
                if (!conn || !csp_send(conn, packet, 1000))
                    return -1;
                csp_close(conn);
            }
            sleep(1);
        }
    }

    close(server_socket);
    close(tx_channel);

    return 0;
}
```

2. CSP OVER UDP (DATAGRAM SOCKETS)

2.1. PROTOCOL



2.2. REQUIREMENTS AND USAGE

```
$ git clone https://github.com/borjatarraso/libcsp.git
$ cd libcsp
$ git checkout aalto1
$ cd libcsp
$ ./waf configure build --enable-examples
$ cd build
$ ./cspudp server
$ ./cspudp client 127.0.0.1
```

2.3. IMPLEMENTATION

```
/*
csp_over_udp.c - Example of CSP protocol running over datagram sockets.
Copyright (C) 2014 Borja Tarraso Hueso <borja.tarraso@member.fsf.org>
This file is a part of Aalto1, Aalto University, Department of Radio Science
and Engineering

This source is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This source is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.
```

```
You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
*/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <pthread.h>
#include <stdint.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

#include <csp/csp.h>
#include <csp/csp_interface.h>

#define TYPE_SERVER 1
#define TYPE_CLIENT 2
#define PORT        10
#define BUF_SIZE    250

#define SERV_PORT 10000
#define MAXLINE 4096

#define CSP_PROTO 213

pthread_t rx_thread;
int server_socket, tx_channel;
struct sockaddr_in servaddr;

int csp_socket_tx(csp_packet_t *packet, uint32_t timeout);

csp_iface_t csp_if_socket = {
    .name = "socket",
    .nexthop = csp_socket_tx,
    .mtu = BUF_SIZE,
};
```

```
int csp_socket_tx(csp_packet_t *packet, uint32_t timeout) {
    printf("Debug tx: %i\n", packet->length);

    if (sendto(tx_channel, &packet->length, packet->length + sizeof(uint32_t) +
sizeof(uint16_t), 0, (struct sockaddr *) &servaddr , sizeof(servaddr)) < 0)
        perror("sendto");

    csp_buffer_free(packet);
    return 1;
}

void * socket_rx(void * parameters) {
    csp_packet_t *buf;
    struct sockaddr_in sa;
    socklen_t length;
    int numbytes;

    /* To check origin/destination of the packet */
    char ipbuf[INET_ADDRSTRLEN];

    buf = csp_buffer_get(BUF_SIZE);

    while ((numbytes = recvfrom(server_socket, &buf->length, BUF_SIZE, 0,
(struct sockaddr *) &sa , &length)) > 0) {
        printf("Debug rx: Got packet from IP: %s with length %i payload %i\n",
inet_ntop(AF_INET, &sa.sin_addr, ipbuf, sizeof(ipbuf)), numbytes, buf->length);
        csp_new_packet(buf, &csp_if_socket, NULL);
        buf = csp_buffer_get(BUF_SIZE);
    }

    return NULL;
}

int main(int argc, char **argv) {

    int me, other, type;
    char *message = "Testing CSP";
    csp_socket_t *sock;
    csp_conn_t *conn;
    csp_packet_t *packet;

    /* Run as either server or client */
    if (argc < 2) {
        printf("usage: %s <server/client> [ip]\n", argv[0]);
        return -1;
    } else if (strcmp(argv[1],"client") == 0 && argc != 3) {
        printf("Unspecified destination IP address\n\n");
        return -1;
```

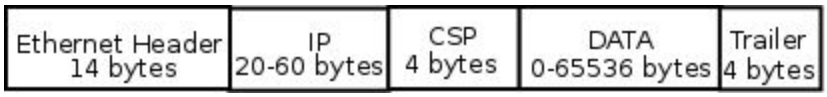
```
    }

    /* Set type */
    if (strcmp(argv[1], "server") == 0) {
        me = 1;
        other = 2;

        if ((server_socket = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
            perror("socket");
        }

        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(SERV_PORT);

        if (bind(server_socket, (struct sockaddr *) &servaddr,
sizeof(servaddr)) == -1) {
            perror("bind");
        }

        type = TYPE_SERVER;
    } else if (strcmp(argv[1], "client") == 0) {
        me = 2;
        other = 1;

        if ((tx_channel = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
            perror("socket");
        }

        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(SERV_PORT);
        inet_pton(AF_INET, argv[2], &servaddr.sin_addr);

        type = TYPE_CLIENT;
    } else {
        printf("Invalid type. Must be either 'server' or 'client'\r\n");
        return -1;
    }

    /* Init CSP and CSP buffer system */
    if (csp_init(me) != CSP_ERR_NONE || csp_buffer_init(10, 300) !=
CSP_ERR_NONE) {
        printf("Failed to init CSP\r\n");
        return -1;
    }
```

```
    /* Start socket RX task */
    pthread_create(&rx_thread, NULL, socket_rx, NULL);

    /* Set default route and start router */
    csp_route_set(CSP_DEFAULT_ROUTE, &csp_if_socket, CSP_NODE_MAC);
    csp_route_start_task(0, 0);

    /* Create socket and listen for incoming connections */
    if (type == TYPE_SERVER) {
        sock = csp_socket(CSP_SO_NONE);
        csp_bind(sock, PORT);
        csp_listen(sock, 5);
    }

    /* Super loop */
    while (1) {
        if (type == TYPE_SERVER) {
            /* Process incoming packet */
            conn = csp_accept(sock, 1000);
            if (conn) {
                packet = csp_read(conn, 0);
                if (packet)
                        printf("Received: %s\r\n", packet->data);
                csp_buffer_free(packet);
                csp_close(conn);
            }
        } else {
            /* Send a new packet */
            packet = csp_buffer_get(strlen(message));
            if (packet) {
                strcpy((char *) packet->data, message);
                packet->length = strlen(message);

                conn = csp_connect(CSP_PRIO_NORM, other, PORT, 1000,
CSP_O_NONE);
                printf("Sending: %s\r\n", message);
                if (!conn || !csp_send(conn, packet, 1000))
                    return -1;
                csp_close(conn);
            }
            sleep(1);
        }
    }

    close(server_socket);
    close(tx_channel);

    return 0;
```

}

3. CSP OVER IP (RAW SOCKETS)

3.1. PROTOCOL



3.2. REQUIREMENTS AND USAGE

```
$ git clone https://github.com/borjatarraso/libcsp.git
$ cd libcsp
$ git checkout aalto1
$ cd libcsp
$ ./waf configure build --enable-examples
$ cd build
$ su
# ./cspip server
# ./cspip client 127.0.0.1
```

3.3. IMPLEMENTATION

```
/*
csp_over_ip.c - Example of CSP protocol running over ip raw sockets.
Copyright (C) 2014 Borja Tarraso Hueso <borja.tarraso@member.fsf.org>
This file is a part of Aalto1, Aalto University, Department of Radio Science
and Engineering

This source is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This source is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
```

```
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
*/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <pthread.h>
#include <stdint.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <netinet/ip.h>

#include <csp/csp.h>
#include <csp/csp_interface.h>

#define TYPE_SERVER 1
#define TYPE_CLIENT 2
#define PORT        10
#define BUF_SIZE    250

#define SERV_PORT 10000
#define MAXLINE 4096

#define CSP_PROTO 213

pthread_t rx_thread;
int server_socket, tx_channel;
struct sockaddr_in servaddr;

int csp_socket_tx(csp_packet_t *packet, uint32_t timeout);

csp_iface_t csp_if_socket = {
    .name = "socket",
    .nexthop = csp_socket_tx,
    .mtu = BUF_SIZE,
};
```

```
int csp_socket_tx(csp_packet_t *packet, uint32_t timeout) {
    printf("Debug tx: %i\n", packet->length);

    if (sendto(tx_channel, &packet->length, packet->length + sizeof(uint32_t) +
sizeof(uint16_t), 0, (struct sockaddr *) &servaddr , sizeof(servaddr)) < 0)
        perror("sendto");

    csp_buffer_free(packet);
    return 1;
}

void * socket_rx(void * parameters) {
    csp_packet_t *buf;
    struct sockaddr_in sa;
    socklen_t length;
    struct iphdr *ip;
    int hlen;
    int numbytes;
    uint8_t buffer[BUF_SIZE];

    /* To check origin/destination of the packet */
    char ipbuf[INET_ADDRSTRLEN];

    while ((numbytes = recvfrom(server_socket, buffer, BUF_SIZE, 0, (struct
sockaddr *) &sa , &length)) > 0) {
        ip = (struct iphdr *) buffer;
        hlen = ip->ihl << 2; /* length of IP header */

        buf = csp_buffer_get(BUF_SIZE);

        memcpy(&buf->length, buffer+hlen, numbytes-hlen );

        printf("Debug rx: Got packet from IP: %s with raw length %i payload
%i\n", inet_ntop(AF_INET, &ip->saddr, ipbuf, sizeof(ipbuf)), numbytes-hlen,
buf->length);
        csp_new_packet(buf, &csp_if_socket, NULL);
    }

    return NULL;
}

int main(int argc, char **argv) {

    int me, other, type;
    char *message = "Testing CSP";
    csp_socket_t *sock;
    csp_conn_t *conn;
    csp_packet_t *packet;
```

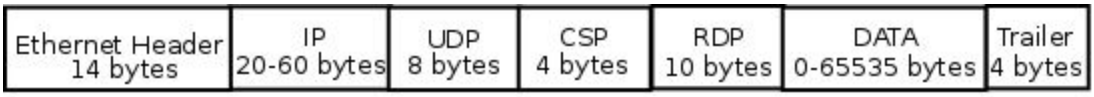
```
/* Required root to create a RAW socket */
if(geteuid() != 0) {
    printf("Required special privileges for create ip raw sockets\n\n");
    printf("You must be root or set CAP_NET_RAW capability\n");
    return -1;
}

/* Run as either server or client */
if (argc < 2) {
    printf("usage: %s <server/client> [ip]\n", argv[0]);
    return -1;
} else if (strcmp(argv[1],"client") == 0 && argc != 3) {
    printf("Unspecified destination IP address\n\n");
    printf("usage: server <server/client> [ip]\n");
    return -1;
}

/* Set type */
if (strcmp(argv[1], "server") == 0) {
    me = 1;
    other = 2;

    if ((server_socket = socket(AF_INET, SOCK_RAW, CSP_PROTO)) == -1) {
        perror("socket");
        exit(1);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

    type = TYPE_SERVER;
} else if (strcmp(argv[1], "client") == 0) {
    me = 2;
    other = 1;

    if ((tx_channel = socket(AF_INET, SOCK_RAW, CSP_PROTO)) == -1) {
        perror("socket");
        exit(1);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    inet_pton(AF_INET, argv[2], &servaddr.sin_addr);

    type = TYPE_CLIENT;
} else {
```

```
        printf("Invalid type. Must be either 'server' or 'client'\r\n");
        return -1;
    }

    /* Init CSP and CSP buffer system */
    if (csp_init(me) != CSP_ERR_NONE || csp_buffer_init(10, 300) !=
CSP_ERR_NONE) {
        printf("Failed to init CSP\r\n");
        return -1;
    }

    /* Start socket RX task */
    pthread_create(&rx_thread, NULL, socket_rx, NULL);

    /* Set default route and start router */
    csp_route_set(CSP_DEFAULT_ROUTE, &csp_if_socket, CSP_NODE_MAC);
    csp_route_start_task(0, 0);

    /* Create socket and listen for incoming connections */
    if (type == TYPE_SERVER) {
        sock = csp_socket(CSP_SO_NONE);
        csp_bind(sock, PORT);
        csp_listen(sock, 5);
    }

    /* Super loop */
    while (1) {
        if (type == TYPE_SERVER) {
            /* Process incoming packet */
            conn = csp_accept(sock, 1000);
            if (conn) {
                packet = csp_read(conn, 0);
                if (packet)
                        printf("Received: %s\r\n", packet->data);
                csp_buffer_free(packet);
                csp_close(conn);
            }
        } else {
            /* Send a new packet */
            packet = csp_buffer_get(strlen(message));
            if (packet) {
                strcpy((char *) packet->data, message);
                packet->length = strlen(message);

                conn = csp_connect(CSP_PRIO_NORM, other, PORT, 1000,
CSP_O_NONE);
                printf("Sending: %s\r\n", message);
                if (!conn || !csp_send(conn, packet, 1000))
```

```
                    return -1;
                csp_close(conn);
            }
            sleep(1);
        }
    }

    close(server_socket);
    close(tx_channel);

    return 0;
}
```

4. CSP+RDP (RELIABLE DATA PROTOCOL SOCKETS) OVER UDP

4.1. PROTOCOL



4.2. REQUIREMENTS AND USAGE

```
$ git clone https://github.com/borjatarraso/libcsp.git
$ cd libcsp
$ git checkout aalto1
$ cd libcsp
$ ./waf configure build --enable-rdp --enable-examples
$ cd build
$ ./csprdp server
$ ./csprdp client 127.0.0.1
```

4.3. IMPLEMENTATION

```
/*
csp_over_rdp.c - Example of CSP protocol running with RDP (Reliable Data
Protocol).
Copyright (C) 2014 Borja Tarraso Hueso <borja.tarraso@member.fsf.org>
This file is a part of Aalto1, Aalto University, Department of Radio Science
and Engineering
```

```
This source is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This source is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
*/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <pthread.h>
#include <stdint.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

#include <csp/csp.h>
#include <csp/csp_interface.h>

#define TYPE_SERVER 1
#define TYPE_CLIENT 2
#define PORT 10
#define BUF_SIZE 250

#define SERV_PORT 10000
#define MAXLINE 4096

#define CSP_PROTO 213

pthread_t rx_thread;
int data_socket;
```

```
struct sockaddr_in remoteaddr;

int csp_socket_tx(csp_packet_t *packet, uint32_t timeout);

csp_iface_t csp_if_socket = {
    .name = "socket",
    .nexthop = csp_socket_tx,
    .mtu = BUF_SIZE,
};

int csp_socket_tx(csp_packet_t *packet, uint32_t timeout) {
    if (sendto(data_socket, &packet->length, packet->length + sizeof(uint32_t)
+ sizeof(uint16_t), 0, (struct sockaddr *) &remoteaddr , sizeof(remoteaddr)) <
0)
      perror("sendto");

    csp_buffer_free(packet);
    return 1;
}

void * socket_rx(void * parameters) {
    csp_packet_t *buf;
    socklen_t length = sizeof(remoteaddr);
    int numbytes;

    buf = csp_buffer_get(BUF_SIZE);

    while ((numbytes = recvfrom(data_socket, &buf->length, BUF_SIZE, 0, (struct
sockaddr *) &remoteaddr , &length)) > 0) {
        csp_new_packet(buf, &csp_if_socket, NULL);
        buf = csp_buffer_get(BUF_SIZE);
    }

    return NULL;
}

int main(int argc, char **argv) {

    int me, other, type;
    char *message = "Testing CSP";
    csp_socket_t *sock;
    csp_conn_t *conn;
    csp_packet_t *packet;

    /* Run as either server or client */
        if (argc < 2) {
            printf("usage: server <server/client>\r\n");
            return -1;
```

```
        }

    /* Set type */
    if (strcmp(argv[1], "server") == 0) {
        me = 1;
        other = 2;

        data_socket = socket(AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in servaddr;
        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(SERV_PORT);

        bind(data_socket, (struct sockaddr *) &servaddr, sizeof(servaddr));

        type = TYPE_SERVER;
    } else if (strcmp(argv[1], "client") == 0) {
        me = 2;
        other = 1;

        data_socket = socket(AF_INET, SOCK_DGRAM, 0);

        memset(&remoteaddr, 0, sizeof(remoteaddr));
        remoteaddr.sin_family = AF_INET;
        remoteaddr.sin_port = htons(SERV_PORT);

        inet_pton(AF_INET, argv[2], &remoteaddr.sin_addr);

        type = TYPE_CLIENT;
    } else {
        printf("Invalid type. Must be either 'server' or 'client'\r\n");
        return -1;
    }

    /* Init CSP and CSP buffer system */
    if (csp_init(me) != CSP_ERR_NONE || csp_buffer_init(10, 300) !=
CSP_ERR_NONE) {
        printf("Failed to init CSP\r\n");
        return -1;
    }

    /* Start socket RX task */
    pthread_create(&rx_thread, NULL, socket_rx, NULL);

    /* Set default route and start router */
    csp_route_set(CSP_DEFAULT_ROUTE, &csp_if_socket, CSP_NODE_MAC);
```

```
    csp_route_start_task(0, 0);

    /* Create socket and listen for incoming connections */
    if (type == TYPE_SERVER) {
        sock = csp_socket(CSP_O_RDP);
        csp_bind(sock, PORT);
        csp_listen(sock, 5);
    }

    /* Super loop */
    while (1) {
        if (type == TYPE_SERVER) {
            /* Process incoming packet */
            conn = csp_accept(sock, 1000);
            if (conn) {
                packet = csp_read(conn, 0);
                if (packet) {
                    packet->data[packet->length] = 0;
                    printf("Received: %s\r\n", packet->data);
                }
                csp_buffer_free(packet);
                csp_close(conn);
            }
        } else {
            /* Send a new packet */
            packet = csp_buffer_get(strlen(message));
            if (packet) {
                strcpy((char *) packet->data, message);
                packet->length = strlen(message);

                conn = csp_connect(CSP_PRIO_NORM, other, PORT, 1000,
CSP_O_RDP);
                printf("Sending: %s (%d bytes)\r\n", message, packet->length);
                if (!conn || !csp_send(conn, packet, 1000))
                    return -1;
                csp_close(conn);
            }
            sleep(1);
        }
    }
    close(data_socket);

    return 0;
}
```

5. BUILD SYSTEM

Waf is used as a build automation tool. This tool is open source and uses the New BSD License. Documentation license is under CC-BY-NC-ND.

Wscript is required in order to build proper binaries, link libraries and define macros. A modified version of wscript has been made for Aalto-1 satellite project purposes:

```
#!/usr/bin/env python
# encoding: utf-8

# Cubesat Space Protocol - A small network-layer protocol designed for Cubesats
# Copyright (C) 2014 Borja Tarraso Hueso <borja.tarraso@member.fsf.org>
# Copyright (C) 2014 Aalto1, Aalto University, Department of Radio Science and
Engineering
# Copyright (C) 2011 GomSpace ApS (http://www.gomspace.com)
# Copyright (C) 2011 AAUSAT3 Project (http://aausat3.space.aau.dk)
#
# This library is free software; you can redistribute it and/or
# modify it under the terms of the GNU Lesser General Public
# License as published by the Free Software Foundation; either
# version 2.1 of the License, or (at your option) any later version.
#
# This library is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General Public
# License along with this library; if not, write to the Free Software
#  Foundation,  Inc.,  51  Franklin  Street,  Fifth  Floor,  Boston,  MA   02110-1301
USA

import os

APPNAME = 'libcsp'
VERSION = '1.0.1'

def options(ctx):
      # Load GCC options
      ctx.load('gcc')
      ctx.load('eclipse', tooldir='tools')

      ctx.add_option('--toolchain', default='', help='Set toolchain prefix')

      # Set libcsp options
      gr = ctx.add_option_group('libcsp options')
```

```
	gr.add_option('--cflags',   default='',   help='Add   additional   CFLAGS.
Separate with comma')
	gr.add_option('--includes',  default='',  help='Add  additional  include
paths. Separate with comma')

	gr.add_option('--disable-output', action='store_true', help='Disable CSP
output')
	gr.add_option('--enable-rdp',   action='store_true',   help='Enable   RDP
support')
	gr.add_option('--enable-qos',  action='store_true',  help='Enable  Quality
of Service support')
	gr.add_option('--enable-promisc',    action='store_true',    help='Enable
promiscuous mode support')
	gr.add_option('--enable-crc32',  action='store_true',  help='Enable  CRC32
support')
	gr.add_option('--enable-hmac',      action='store_true',      help='Enable
HMAC-SHA1 support')
	gr.add_option('--enable-xtea',   action='store_true',   help='Enable   XTEA
support')
	gr.add_option('--enable-bindings',    action='store_true',    help='Enable
Python bindings')
	gr.add_option('--enable-examples',    action='store_true',    help='Enable
examples')
	gr.add_option('--enable-static-buffer', action='store_true', help='Enable
static buffer system')

	gr.add_option('--with-os', default='posix', help='Set operating system.
Must be either \'posix\' or \'freertos\'')
	gr.add_option('--with-can',  default=None,  metavar='CHIP',  help='Enable
CAN  driver.  CHIP  must  be  either  socketcan,  at91sam7a1,  at91sam7a3  or
at90can128')
	gr.add_option('--with-freertos',                            metavar="PATH",
default='../../libgomspace/include', help='Set path to FreeRTOS header files')
	gr.add_option('--with-static-buffer-size',     type=int,     default=320,
help='Set size of static buffer elements')
	gr.add_option('--with-static-buffer-count',     type=int,     default=12,
help='Set number of static buffer elements')
	gr.add_option('--with-rdp-max-window',  type=int,  default=20,  help='Set
maximum window size for RDP')
	gr.add_option('--with-max-bind-port',   type=int,   default=31,   help='Set
maximum bindable port')
	gr.add_option('--with-max-connections',  type=int,  default=10,  help='Set
maximum number of concurrent connections')
	gr.add_option('--with-conn-queue-length', type=int, default=10, help='Set
maximum number of packets in queue for a connection')
	gr.add_option('--with-router-queue-length',     type=int,     default=10,
help='Set maximum number of packets to be queued at the input of the router')
	gr.add_option('--with-padding',  type=int,  default=8,  help='Set  padding
```

```
bytes before packet length field')

def configure(ctx):
	# Validate OS
	if not ctx.options.with_os in ('posix', 'freertos'):
		ctx.fatal('ARCH must be either \'posix\' or \'freertos\'')

	# Validate CAN drivers
	if not ctx.options.with_can in (None, 'socketcan', 'at91sam7a1',
'at91sam7a3', 'at90can128'):
		ctx.fatal('CAN must be either \'socketcan\', \'at91sam7a1\',
\'at91sam7a3\', \'at90can128\'')

	# Setup and validate toolchain
	ctx.env.CC = ctx.options.toolchain + 'gcc'
	ctx.env.AR = ctx.options.toolchain + 'ar'
	ctx.env.SIZE = ctx.options.toolchain + 'size'
	ctx.load('gcc')
	ctx.find_program('size', var='SIZE')

	# Set git revision define
	git_rev = os.popen("(git log --pretty=format:%H -n 1 | egrep -o
\"^.{8}\") 2> /dev/null || echo none").read().strip()

	# Setup DEFINES
	ctx.env.append_unique('DEFINES_CSP', ['GIT_REV="{0}"'.format(git_rev)])

	# Setup CFLAGS
	ctx.env.append_unique('CFLAGS_CSP', ['-Os','-Wall', '-g', '-std=gnu99'] +
ctx.options.cflags.split(','))

	# Setup extra includes
	ctx.env.append_unique('INCLUDES_CSP', ['include'] +
ctx.options.includes.split(','))

	# Add default files
	ctx.env.append_unique('FILES_CSP',
['src/*.c','src/interfaces/csp_if_lo.c','src/transport/csp_udp.c','src/arch/{0}
/**/*.c'.format(ctx.options.with_os)])

	# Add FreeRTOS
	if ctx.options.with_os == 'freertos':
		ctx.env.append_unique('INCLUDES_CSP', ctx.options.with_freertos)
		ctx.define('_CSP_FREERTOS_', 1)
	else:
		ctx.define('_CSP_POSIX_', 1)

	# Add CAN driver
```

```
	if ctx.options.with_can:
		ctx.env.append_unique('FILES_CSP', 'src/interfaces/csp_if_can.c')
		ctx.env.append_unique('FILES_CSP',
'src/interfaces/can/can_{0}.c'.format(ctx.options.with_can))

	# Store configuration options
	ctx.env.ENABLE_BINDINGS = ctx.options.enable_bindings
	ctx.env.ENABLE_EXAMPLES = ctx.options.enable_examples

	# Create config file
	ctx.define_cond('CSP_DEBUG', not ctx.options.disable_output)
	if not ctx.options.disable_output:
		ctx.env.append_unique('FILES_CSP', 'src/csp_debug.c')
	else:
		ctx.env.append_unique('EXCL_CSP', 'src/csp_debug.c')

	ctx.define_cond('CSP_USE_RDP', ctx.options.enable_rdp)
	if ctx.options.enable_rdp:
		ctx.env.append_unique('FILES_CSP', 'src/transport/csp_rdp.c')

	ctx.define_cond('CSP_ENABLE_CRC32', ctx.options.enable_crc32)
	if ctx.options.enable_crc32:
		ctx.env.append_unique('FILES_CSP', 'src/csp_crc32.c')
	else:
		ctx.env.append_unique('EXCL_CSP', 'src/csp_crc32.c')

	if ctx.options.enable_hmac:
		ctx.env.append_unique('FILES_CSP', 'src/crypto/csp_hmac.c')
		ctx.env.append_unique('FILES_CSP', 'src/crypto/csp_sha1.c')
		ctx.define_cond('CSP_ENABLE_HMAC', ctx.options.enable_hmac)

	if ctx.options.enable_xtea:
		ctx.env.append_unique('FILES_CSP', 'src/crypto/csp_xtea.c')
		ctx.env.append_unique('FILES_CSP', 'src/crypto/csp_sha1.c')
		ctx.define_cond('CSP_ENABLE_XTEA', ctx.options.enable_xtea)

	ctx.define_cond('CSP_USE_PROMISC', ctx.options.enable_promisc)
	ctx.define_cond('CSP_USE_QOS', ctx.options.enable_qos)
	ctx.define_cond('CSP_BUFFER_STATIC', ctx.options.enable_static_buffer)
	ctx.define('CSP_BUFFER_COUNT', ctx.options.with_static_buffer_count)
	ctx.define('CSP_BUFFER_SIZE', ctx.options.with_static_buffer_size)
	ctx.define('CSP_CONN_MAX', ctx.options.with_max_connections)
	ctx.define('CSP_CONN_QUEUE_LENGTH', ctx.options.with_conn_queue_length)
	ctx.define('CSP_FIFO_INPUT', ctx.options.with_router_queue_length)
	ctx.define('CSP_MAX_BIND_PORT', ctx.options.with_max_bind_port)
	ctx.define('CSP_RDP_MAX_WINDOW', ctx.options.with_rdp_max_window)
	ctx.define('CSP_PADDING_BYTES', ctx.options.with_padding)
```

```
	ctx.write_config_header('include/csp/csp_autoconfig.h', 	top=False, remove=False)

	# Check for endian.h
	ctx.check(header_name='endian.h', mandatory=False)

def build(ctx):
	# Build static library
	ctx.stlib(
		source=ctx.path.ant_glob(ctx.env.FILES_CSP, excl=ctx.env.EXCL_CSP),
		target = 'csp',
		includes= ctx.env.INCLUDES_CSP,
		export_includes = 'include',
		cflags = ctx.env.CFLAGS_CSP,
		defines = ctx.env.DEFINES_CSP,
		use = 'csp_size')

	# Print library size
	if ctx.options.verbose > 0:
		ctx(rule='${SIZE}  --format=berkeley  ${SRC}',  source='libcsp.a', name='csp_size', always=True)

	# Build shared library for Python bindings
	if ctx.env.ENABLE_BINDINGS:
		ctx.shlib(source=ctx.path.ant_glob(ctx.env.FILES_CSP),
			target = 'pycsp',
			includes= ctx.env.INCLUDES_CSP,
			export_includes = 'include',
			cflags = ctx.env.CFLAGS_CSP,
			defines = ctx.env.DEFINES_CSP,
			lib=['rt', 'pthread'])

	if ctx.env.ENABLE_EXAMPLES:
		ctx.program(source = ctx.path.ant_glob('examples/simple.c'),
			target = 'simple',
			includes = ctx.env.INCLUDES_CSP,
			cflags = ctx.env.CFLAGS_CSP,
			defines = ctx.env.DEFINES_CSP,
			lib=['rt', 'pthread'],
			use = 'csp')
		ctx.program(source = ctx.path.ant_glob('examples/csp_over_tcp.c'),
			target = 'csptcp',
			includes = ctx.env.INCLUDES_CSP,
			cflags = ctx.env.CFLAGS_CSP,
			defines = ctx.env.DEFINES_CSP,
			lib=['rt', 'pthread'],
			use = 'csp')
		ctx.program(source = ctx.path.ant_glob('examples/csp_over_udp.c'),
```

```
                target = 'cspudp',
                includes = ctx.env.INCLUDES_CSP,
                cflags = ctx.env.CFLAGS_CSP,
                defines = ctx.env.DEFINES_CSP,
                lib=['rt', 'pthread'],
                use = 'csp')
        ctx.program(source = ctx.path.ant_glob('examples/csp_over_ip.c'),
                target = 'cspip',
                includes = ctx.env.INCLUDES_CSP,
                cflags = ctx.env.CFLAGS_CSP,
                defines = ctx.env.DEFINES_CSP,
                lib=['rt', 'pthread'],
                use = 'csp')
        ctx.program(source = ctx.path.ant_glob('examples/csp_over_rdp.c'),
                target = 'csprdp',
                includes = ctx.env.INCLUDES_CSP,
                cflags = ctx.env.CFLAGS_CSP,
                defines = ctx.env.DEFINES_CSP,
                lib=['rt', 'pthread'],
                use = 'csp')

    # Set install path for header files
    ctx.install_files('${PREFIX}',      ctx.path.ant_glob('include/**/*.h'),
relative_trick=True)
    ctx.install_files('${PREFIX}/include/csp',
'include/csp/csp_autoconfig.h')
    ctx.install_files('${PREFIX}/lib', 'libcsp.a')

def dist(ctx):
    ctx.excl = 'build/* **/.* **/*.pyc **/*.o **/*~ *.tar.gz'
```

6. PERFORMANCE

At the moment it is not possible to test performance for CSP because CSP is not standardized protocol, so it is not possible to calculate benchmarks over it. However and after some research, it is possible in the future test the performance following those techniques:

- Implement support for CSP for already existing tool: iperf or nuttcp.
- Implement from scratch a new tool for benchmarking CSP protocol.
- Set up two MCUs talking CSP to each other and enabling standard profiling tools like FreeRTOS trace or use a GPIO pin to measure latency and CPU time used in different functions using an oscilloscope.

The performance to be investigated is:

CPU time
Memory usage

The performance will be based on:

No traffic / standby mode
Traffic throughput (in pkts/second)
Packet size (varying).

REFERENCES

The following references, technologies, information and documents were used to accomplish and achieve the goals of this delivery and its documentation:

Main software for libcsp:

- Libcsp: https://github.com/GomSpace/libcsp/
- Aalto1 libcsp fork: https://github.com/borjatarraso/libcsp/

Tools and libraries:

- Waf, automation build tool: https://code.google.com/p/waf/
- Dia, diagram creation program: https://wiki.gnome.org/Apps/Dia/
- GNU/Emacs, environment: http://www.gnu.org/software/emacs/
- GNU, operating system: http://www.gnu.org/
- Linux, kernel: https://www.kernel.org/
- Git, version control system: http://git-scm.com/
- Virtualbox, Virtualization system: https://www.virtualbox.org/
- GNU/Global, src browser: http://www.gnu.org/software/global/global.html
- LaTeX, typesetting system: http://latex-project.org/
- AucTeX, extensible package for TeX: https://www.gnu.org/software/auctex/
- Projectlibre, Project scheduling: http://www.projectlibre.org/

Standards:

- RFC-768, UDP: https://www.ietf.org/rfc/rfc768.txt
- RFC-793, TCP: http://www.ietf.org/rfc/rfc793.txt
- RFC-791, IP: http://www.ietf.org/rfc/rfc791.txt
- RFC-908, RDP version 1: http://tools.ietf.org/html/rfc908
- RFC-1151, RDP version 2: http://tools.ietf.org/html/rfc1151

Licenses:

- LGPL 2.1 license: http://opensource.org/licenses/LGPL-2.1
- New BSD license: http://opensource.org/licenses/BSD-3-Clause

Books:

- TCP/IP Illustrated, vol I. R. Stevens: ISBN-10: 0201633469
- Unix Network Programming vol I. R. Stevens: ISBN-10: 0131411551
- Unix Network Programming vol II. R. Stevens: ISBN-10: 0132974290

Guides:

- Beej guide, sockets programming: http://beej.us/guide/bgnet/

Other resources:

- Wikipedia, the universal encyclopedia: http://www.wikipedia.org/
- Stackoverflow, troubleshooting Q&A: http://stackoverflow.com/

Acknowledgements:

- Antonio Salueña: Systems designer at Ericsson
- Jose Uceda: Network Security Manager at PeopleCall
- Johan Christiansen: Electrical Engineer at GomSpace