

Bug Report

Name : Saltel Bastien

Email address : saltel@ensta.fr

Software version : OPEN – R SDK 1.1.5 (r1)

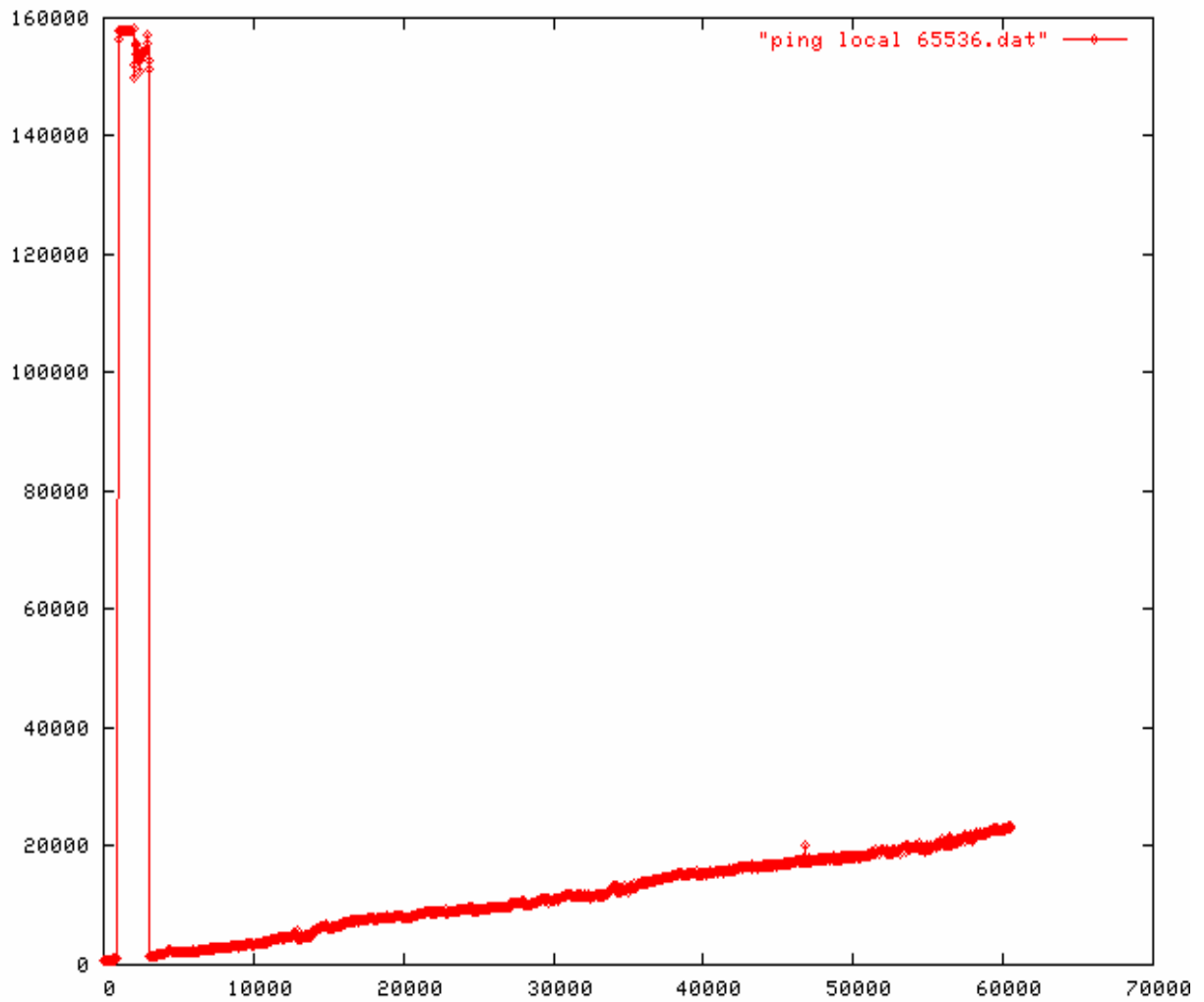
Aibo version : ERS – 7

Type of bug : Data fragmentation during packets transmission between two Open-R objects over a TCP / IP connection

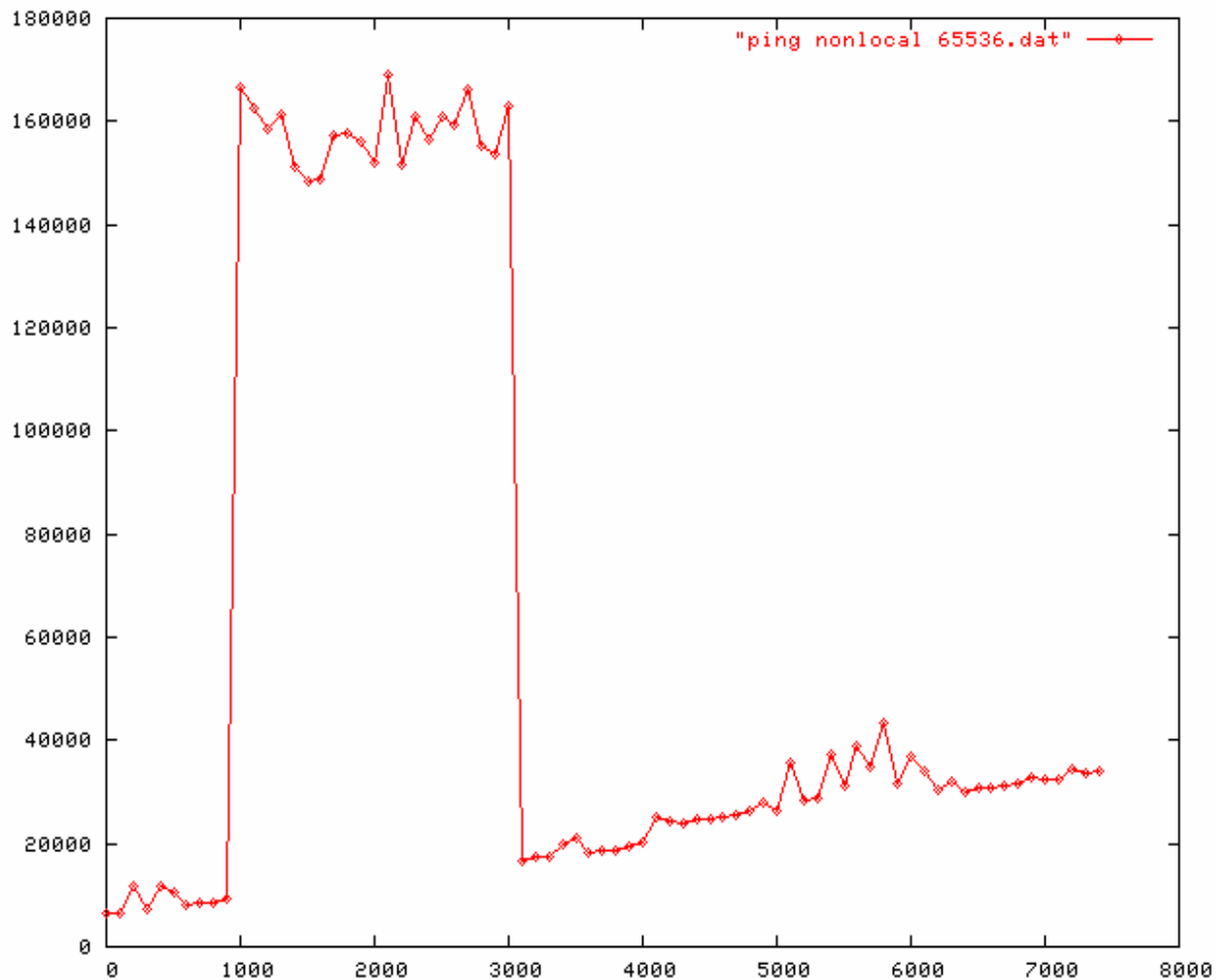
Summary : When an Open-R object sends a packet whose size stands between 1025 and 3072 bits through the IPStack, the receiver Open-R object abnormally receives two packets : the size of the first packet is 1024 bits and the size of the second one is the rest of the whole sent packet size. Moreover this bug seems to be totally independent of the value of the “Packetsize” variable when I instantiate a `entEnvCreateEndpointMsg`, a `antEnvCreateSharedBufferMsg` or a `antEnvCreateSharedBufferMsg` object.

Description : Having two Open-R objects which communicate by sending data to each other over a localhost TCP / IP connection, I notice that the duration of transmission isn't proportional to the packet size, especially when the size stands between 1025 and 3072 bits. Indeed, the receiver object calls the `ReceiveCont` function twice (`ReceiveCont` is normally invoked when a receive request has been completed) and the delay between these two function calls is very long, although the delay between the end of sending and the first call of `ReceiveCont` is correct and short. More exactly, during the first call of `ReceiveCont`, the object receives 1024 bits and the rest at the end of the second call of `ReceiveCont`. This bug occurs whatever the size of `Packetsize` is and even if the two Open-R objects are loaded into two different Aibo robots.

Measured results:



This graph represents the link between the duration of data transmission and the size of the packets with 65536 bit Packetsize. The sender and receiver Open-R objects are located into the same Aibo robot.



In this second graph, I measure the duration of transmission when the two Open-R objects are running into two different Aibo robots.

According to these two graphics, we can observe that the duration of transmission isn't effectively proportional to the packet size of the sent data.

Expected result :

The curve obtained by this experiment may normally be linear, or at least not discontinuous.

Reproduce code :

SENDER side :

```
SENDER::SENDER()
{
    packetSize_ = 65536;
    port_ = 54000;
    host_ = strdup("127.0.0.1");
    ipstackRef_ = antStackRef("IPStack");

    WhoAmI(&ID_);

    // Allocate OPENR send buffer
    antEnvCreateSharedBufferMsg sendBufferMsg(packetSize_);
    sendBufferMsg.Call(ipstackRef_, sizeof (sendBufferMsg));
    if (sendBufferMsg.error != ANT_SUCCESS)
    {
        OSYSDEBUG(("s : antError %d\n", "Can't allocate send buffer",
            sendBufferMsg.error));
        return ;
    }
    antSendBuffer_ = sendBufferMsg.buffer;
    antSendBuffer_.Map();
    antSendData_ = (byte *) (antSendBuffer_.GetAddress());

    // Create IPStack endpoint
    antEnvCreateEndpointMsg tcpCreateMsg(EndpointType_TCP, packetSize_ * 2);
    tcpCreateMsg.Call(ipstackRef_, sizeof (tcpCreateMsg));
    if (tcpCreateMsg.error != ANT_SUCCESS)
    {
        OSYSDEBUG(("s : antError %d\n", "Can't create endpoint",
            tcpCreateMsg.error));
        return ;
    }
    endpoint_ = tcpCreateMsg.moduleRef;
}

OStatus
SENDER::DoInit(const OSystemEvent& event)
{
    NEW_ALL_SUBJECT_AND_OBSERVER;
    REGISTER_ALL_ENTRY;
    SET_ALL_READY_AND_NOTIFY_ENTRY;

    return oSUCCESS;
}

OStatus
SENDER::DoStart(const OSystemEvent& event)
{
    unsigned int size = 50;
    char *str;
    unsigned long int j;
    unsigned long int deb;

    ENABLE_ALL_SUBJECT;
    ASSERT_READY_TO_ALL_OBSERVER;

    // Wait for the Receiver object to be ready
    for (deb = mtime(), j = deb; j < deb + 4000; j = mtime())
```

```

        ;

Connect();
for (int i = size; i < 60000; i += 50)
{
    str = (char *)malloc(i + 1);
    str[i] = '\0';
    memset(str, 'x', i);
    Send("%s\n", str);
    free(str);

    // Wait for the Receiver object to treat the sent packet
    for (deb = mtime(), j = deb; j < deb + 2000; j = mtime())
        ;
}
return oSUCCESS;
}

int
SENDER::Connect()
{
    // Connect to the Receiver Object
    TCPEndpointConnectMsg tcpConnectMsg(endpoint_, IP_ADDR_ANY,
                                           IP_PORT_ANY, host_, port_);
    tcpConnectMsg.Call(ipstackRef_, sizeof (TCPEndpointConnectMsg));
    if (tcpConnectMsg.error != TCP_SUCCESS)
    {
        OSYSDEBUG(("s : antError %d\n", "Can't connect to server",
                    tcpConnectMsg.error));
        Close();
    }
    return 0;
}

int
SENDER::Send(const char *command, ...)
{
    int len;
    char *str;

    str = (char *)malloc(128000);
    va_list arg;
    va_start(arg, command);
    len = vsprintf(str, command, arg);
    str[len] = '\0';
    va_end(arg);
    effectiveSend(str, strlen(str));
    free(str);
    return 0;
}

int
SENDER::effectiveSend(const char *buffer, int len)
{
    // Transfer to IPStack shared buffer
    memcpy(antSendData_, buffer, len);

    errorNotify("SENDER [%d] sent packet : len = %d\n", utime(), len);

    // Send message to IPStack
    TCPEndpointSendMsg tcpSendMsg(endpoint_, antSendData_, len);
    tcpSendMsg.continuation = (void *)this;
    tcpSendMsg.Send(ipstackRef_, ID_, Extra_Entry[entrySendCont],
                    sizeof (TCPEndpointSendMsg));
    return 0;
}

```

```

void
SENDER::SendCont(void *msg)
{
    TCPEndpointSendMsg      *tcpSendMsg = (TCPEndpointSendMsg *)msg;
    if (tcpSendMsg->error != TCP_SUCCESS)
    {
        OSYSDEBUG(("s : antError %d\n",
                    "Failed with SendCont",
                    tcpSendMsg->error));
        Close();
    }
}

OStatus
SENDER::DoStop(const OSystemEvent& event)
{
    DISABLE_ALL_SUBJECT;
    DEASSERT_READY_TO_ALL_OBSERVER;
    return oSUCCESS;
}

OStatus
SENDER::DoDestroy(const OSystemEvent& event)
{
    DELETE_ALL_SUBJECT_AND_OBSERVER;
    return oSUCCESS;
}

int
SENDER::Close()
{
    // Free the Send buffer
    antSendBuffer_.UnMap();
    antEnvDestroySharedBufferMsg      sendBufferMsg(antSendBuffer_);
    sendBufferMsg.Call(ipstackRef_, sizeof (antEnvDestroySharedBufferMsg));

    // Send the "Close" request message
    TCPEndpointCloseMsg      tcpCloseMsg(endpoint_);
    tcpCloseMsg.continuation = (void *)this;
    tcpCloseMsg.Send(ipstackRef_, ID_, Extra_Entry[entryCloseCont],
                     sizeof (TCPEndpointCloseMsg));
    return 0;
}

void
SENDER::TCPCloseCont(void *msg)
{
}

```

RECEIVER side:

```

RECEIVER::RECEIVER()
{
    packetSize_ = 65536;
    port_ = 54000;
    ipstackRef_ = antStackRef("IPStack");
    WhoAmI(&ID_);

    // Allocate OPENR receive buffer
    antEnvCreateSharedBufferMsg      recvBufferMsg(packetSize_);
    recvBufferMsg.Call(ipstackRef_, sizeof (recvBufferMsg));
    if (recvBufferMsg.error != ANT_SUCCESS)
    {

```

```

        OSYSDEBUG(("s : antError %d\n", "Can't allocate receive buffer",
                    recvBufferMsg.error));
        return ;
    }
    antRecvBuffer_ = recvBufferMsg.buffer;
    antRecvBuffer_.Map();
    antRecvData_ = (byte *)(antRecvBuffer_.GetAddress());

    // Create IPStack endpoint
    antEnvCreateEndpointMsg tcpCreateMsg(EndpointType_TCP, packetSize_ * 2);
    tcpCreateMsg.Call(ipstackRef_, sizeof (tcpCreateMsg));
    if (tcpCreateMsg.error != ANT_SUCCESS)
    {
        OSYSDEBUG(("s : antError %d\n", "Can't create endpoint",
                    tcpCreateMsg.error));
        return ;
    }
    endpoint_ = tcpCreateMsg.moduleRef;
}

OStatus
RECEIVER::DoInit(const OSystemEvent& event)
{
    NEW_ALL_SUBJECT_AND_OBSERVER;
    REGISTER_ALL_ENTRY;
    SET_ALL_READY_AND_NOTIFY_ENTRY;

    return oSUCCESS;
}

OStatus
RECEIVER::DoStart(const OSystemEvent& event)
{
    Listen();

    ENABLE_ALL_SUBJECT;
    ASSERT_READY_TO_ALL_OBSERVER;

    return oSUCCESS;
}

OStatus
RECEIVER::Listen()
{
    // Listen
    TCPEndpointListenMsg listenMsg(endpoint_, IP_ADDR_ANY, port_);

    listenMsg.continuation = (void*)this;
    listenMsg.Send(ipstackRef_, ID_, Extra_Entry[entryListenCont],
    sizeof(listenMsg));

    return oSUCCESS;
}

void
RECEIVER::ListenCont(void* msg)
{
    Receive();
}

int
RECEIVER::Receive()
{
    // Send the "Receive" request message
    TCPEndpointReceiveMsg tcpRecvMsg(endpoint_, antRecvData_, 1, packetSize_);
    tcpRecvMsg.continuation = (void *)this;
    tcpRecvMsg.Send(ipstackRef_, ID_,
                    Extra_Entry[entryReceiveCont], sizeof

```

```

(TCPEndpointReceiveMsg));
    return 0;
}

void
RECEIVER::ReceiveCont(void *msg)
{
    TCPEndpointReceiveMsg *tcpRecvMsg = (TCPEndpointReceiveMsg *)msg;
    if (tcpRecvMsg->error != TCP_SUCCESS)
    {
        if (tcpRecvMsg->error != TCP_CONNECTION_BUSY)
        {
            OSYSDEBUG(("s : antError %d\n", "Failed with TCPReceiveCont",
                tcpRecvMsg->error));
            Close();
        }
        return ;
    }
    errorNotify("RECEIVER [%d] receive packet : len = %d\n",
        utime(), tcpRecvMsg->sizeMin );

    // Loop to receive more data.
    Receive();
}

OStatus
RECEIVER::DoStop(const OSystemEvent& event)
{
    DISABLE_ALL_SUBJECT;
    DEASSERT_READY_TO_ALL_OBSERVER;
    return oSUCCESS;
}

OStatus
RECEIVER::DoDestroy(const OSystemEvent& event)
{
    DELETE_ALL_SUBJECT_AND_OBSERVER;
    return oSUCCESS;
}

int
RECEIVER::Close()
{
    // Free the Receive buffer
    antRecvBuffer_.UnMap();
    antEnvDestroySharedBufferMsg      recvBufferMsg(antRecvBuffer_);
    recvBufferMsg.Call(ipstackRef_, sizeof (antEnvDestroySharedBufferMsg));

    // Send the "Close" request message
    TCPEndpointCloseMsg      tcpCloseMsg(endpoint_);
    tcpCloseMsg.continuation = (void *)this;
    tcpCloseMsg.Send(ipstackRef_, ID_, Extra_Entry[entryCloseCont],
        sizeof (TCPEndpointCloseMsg));
    return 0;
}

void
RECEIVER::TCPCloseCont(void *msg)
{
}

Common code :

unsigned long int utime()
{
    static struct SystemTime time;
    GetSystemTime(&time);
}

```

```

    return time.seconds * 1000000 + time.useconds;
}

unsigned long int mtime()
{
    static struct SystemTime time;
    GetSystemTime(&time);
    return time.seconds * 1000 + time.useconds / 1000;
}

void
errorNotify(const char *format, ...)
{
    char      *str;
    va_list arg;

    if ((str = (char *)malloc(1024)) == NULL)
        return ;
    va_start(arg, format);
    (void) vsnprintf(str, 1024, format, arg);
    va_end(arg);
    OSYSPRINT("%s", str));
    va_end(arg);
    free(str);
}

```

Standard Output:

```

SENDER [18500053] sent packet : len = 52
RECEIVER [18500379] receive packet : len = 52

SENDER [20500077] sent packet : len = 102
RECEIVER [20501248] receive packet : len = 102

SENDER [22500077] sent packet : len = 152
RECEIVER [22501241] receive packet : len = 152

SENDER [24500083] sent packet : len = 202
RECEIVER [24501250] receive packet : len = 202

[...]

SENDER [54500099] sent packet : len = 952
RECEIVER [54501253] receive packet : len = 952

SENDER [56500103] sent packet : len = 1002
RECEIVER [56501248] receive packet : len = 1002

SENDER [58500103] sent packet : len = 1052
RECEIVER [58501252] receive packet : len = 1024
RECEIVER [58665648] receive packet : len = 28

SENDER [60500108] sent packet : len = 1102
RECEIVER [60501244] receive packet : len = 1024
RECEIVER [60642900] receive packet : len = 78

```

SENDER [62500109] sent packet : len = 1152
RECEIVER [62501243] receive packet : len = 1024
RECEIVER [62617557] receive packet : len = 128

SENDER [64500108] sent packet : len = 1202
RECEIVER [64501245] receive packet : len = 1024
RECEIVER [64698636] receive packet : len = 178

SENDER [66500111] sent packet : len = 1252
RECEIVER [66501243] receive packet : len = 1024
RECEIVER [66673656] receive packet : len = 228

SENDER [68500114] sent packet : len = 1302
RECEIVER [68501245] receive packet : len = 1024
RECEIVER [68649666] receive packet : len = 278

[...]

SENDER [20493332] sent packet : len = 2952
RECEIVER [20494149] receive packet : len = 1024
RECEIVER [20699510] receive packet : len = 1928

SENDER [22493169] sent packet : len = 3002
RECEIVER [22494164] receive packet : len = 1024
RECEIVER [22674697] receive packet : len = 1978

SENDER [24493183] sent packet : len = 3052
RECEIVER [24494176] receive packet : len = 1024
RECEIVER [24649797] receive packet : len = 2028

SENDER [26493171] sent packet : len = 3102
RECEIVER [26496956] receive packet : len = 3102

SENDER [28494249] sent packet : len = 3152
RECEIVER [28498164] receive packet : len = 3152

SENDER [30497175] sent packet : len = 3202
RECEIVER [30498653] receive packet : len = 3202

SENDER [32497194] sent packet : len = 3252
RECEIVER [32498641] receive packet : len = 3252

[...]