Lazy Pirate client in C++

```
//
//  Lazy Pirate client
//  Use zmq_poll to do a safe request-reply
//  To run, start piserver and then randomly kill/restart it
//
#include "zhelpers.hpp"

#include <sstream>

#define REQUEST_TIMEOUT     2500    //  msecs, (> 1000!)
#define REQUEST_RETRIES     3       //  Before we abandon

//  Helper function that returns a new configured socket
//  connected to the Hello World server
//
static zmq::socket_t * s_client_socket (zmq::context_t & context) {
    std::cout << "I: connecting to server…" << std::endl;
    zmq::socket_t * client = new zmq::socket_t (context, ZMQ_REQ);
    client->connect ("tcp://localhost:5555");

    //  Configure socket to not wait at close time
    int linger = 0;
    client->setsockopt (ZMQ_LINGER, &linger, sizeof (linger));
    return client;
}

int main () {
    zmq::context_t context (1);

    zmq::socket_t * client = s_client_socket (context);

    int sequence = 0;
    int retries_left = REQUEST_RETRIES;

    while (retries_left) {
        std::stringstream request;
        request << ++sequence;
        s_send (*client, request.str());
        sleep (1);

        bool expect_reply = true;
        while (expect_reply) {
            //  Poll socket for a reply, with timeout
            zmq::pollitem_t items[] = { { *client, 0, ZMQ_POLLIN, 0 } };
            zmq::poll (&items[0], 1, REQUEST_TIMEOUT * 1000);

            //  If we got a reply, process it
            if (items[0].revents & ZMQ_POLLIN) {
```

```
                //  We got a reply from the server, must match sequence
                std::string reply = s_recv (*client);
                if (atoi (reply.c_str ()) == sequence) {
                    std::cout << "I: server replied OK (" << reply << ")" <<
std::endl;
                    retries_left = REQUEST_RETRIES;
                    expect_reply = false;
                }
                else {
                    std::cout << "E: malformed reply from server: " << reply <<
std::endl;
                }
            }
            else
            if (--retries_left == 0) {
                std::cout << "E: server seems to be offline, abandoning" << std::endl;
                expect_reply = false;
                break;
            }
            else {
                std::cout << "W: no response from server, retrying…" << std::endl;
                //  Old socket will be confused; close it and open a new one
                delete client;
                client = s_client_socket (context);
                //  Send request again, on new socket
                s_send (*client, request.str());
            }
        }
    }
    delete client;
    return 0;
}
```