DCCL/GPB integration to ROS-AquaNet

Step-by-step guide

Version 0.2 – 06/08/2021

Outline

REVISION HISTORY

Version	Description of changes
V0.1	Initial version
V0.2	- added a creation date for a version - added linux distribution description - added Troubleshooting subsection to Section 1 - fixed the code in Appendix A - added installation command for protobuf-compiler in Section 2 - added a note about the protoc warning in Section 2

1. Downloading and installing DCCL/GPB

This section describes the installation instructions specifically for `Ubuntu 18.04 and 20.04` Linux distributions. Additional instructions for Debian systems can be found in the official DCCL repository over here:
https://github.com/GobySoft/dccl

First, download and install the latest DCCLv3 library from the official repository using the following commands:

- add packages to apt sources:

```
echo "deb http://packages.gobysoft.org/ubuntu/release/ `lsb_release -c -s`/"
| sudo tee /etc/apt/sources.list.d/gobysoft_release.list

sudo apt-key adv --recv-key --keyserver keyserver.ubuntu.com 19478082E2F8D3FE

sudo apt update
```

- install a full suite of DCCL packages

```
sudo apt install libdccl3-dev dccl3-compiler dccl3-apps
```

- you can check the installation by running:

```
$ dccl --version

3.0.15
```

1.1 Troubleshooting

In case the following signature errors occur when trying to add/update the source lists:

```
The following signatures were invalid: EXPKEYSIG F42ED6FBAB17C654 Open Roboti
cs <info@osrfoundation.org>


W: An error occurred during the signature verification. The repository is not
updated and the previous index files will be used. GPG error: http://packages
.ros.org/ros/ubuntu focal InRelease: The following signatures were invalid: E
XPKEYSIG F42ED6FBAB17C654 Open Robotics <info@osrfoundation.org>

W: Failed to fetch http://packages.ros.org/ros/ubuntu/dists/focal/InRelease
The following signatures were invalid: EXPKEYSIG F42ED6FBAB17C654 Open Roboti
cs <info@osrfoundation.org>

W: Some index files failed to download. They have been ignored, or old ones u
sed instead.
```

To fix the problem, please try to add the necessary ROS dependencies by following the steps from Section 1 of the ROS guide:
https://www.dropbox.com/s/2vmp8twu8isuqqb/ROS_guide_v0.4.pdf

2. Creating AquaNet DCCL message

DCCLv3 library presents an extension to Google Protocol Buffers (GPB) serialization/deserialization tool. It is aimed on preserving the amount of bits required to transmit a particular piece of information, described in the DCCL message fields. An information field can be described as a combination of a data structure (i.e., an integer, double, enumeration, etc.) and the range of values that data structure can take, such as minimum value, maximum value and its precision.

The range of a value and its precision affects the total number of bits required to send it. The higher the range or the precision are, the higher the number of bits required. The eventual size of a message, given the range and the precision, is calculated by DCCL on a compilation/translation stage, using `protoc` compiler.

The message structure is described by Google's `proto2` language. The aquanet message structure can have the following format, described in it:

```
import "dccl/option_extensions.proto";

message AquanetMessage {
  option (dccl.msg) = { codec_version: 3
                        id: 1
                        max_bytes: 32 };
  required uint32 ros_msg_id = 1 [(dccl.field) = { min: 0 max: 10 precision: 1 }];
  required double x = 2 [(dccl.field) = { min: -10 max: 10 precision: 1 }];
  required double y = 3 [(dccl.field) = { min: -10 max: 10 precision: 1 }];
  required double z = 4 [(dccl.field) = { min: -10 max: 10 precision: 1 }];
  enum VehicleClass { AUV = 1; USV = 2; SHIP = 3; }
  optional VehicleClass veh_class = 5;
  optional bool battery_ok = 6;
}
```

The source code for `aquanet.message.proto` file can be found in **Appendix A**.

After a message is described in the `proto2` format, it should be then translated to the corresponding libraries for a particular programming language used by a developer.

For the translation, the `protoc` compiler is used, which is installed by executing:

```
sudo apt install protobuf-compiler
```

In the case of ROS project, C++ language and CMake build systems are used. Therefore, a command to translate the original `proto2` message into C++ library would be the following:

```
protoc --cpp_out=. aquanet.message.proto -I . -I /usr/include
```

where:
- `aquanet.message.proto` – name of the DCCL/GPB message file
- `cpp-out` – flag which tells the proto-compiler to generate C++ headers and files

Note:
The `protoc` compiler may throw the following warning after finishing:

```
[libprotobuf WARNING google/protobuf/compiler/parser.cc:648] No syntax specif
ied for the proto file: test. Please use 'syntax = "proto2";' or 'syntax = "p
roto3";' to specify a syntax version. (Defaulted to proto2 syntax.)
```

This warning tells us that the Google proto language version has not been explicitly specified in the `.proto` file, thus, the version was set to `proto2` by default.
This warning can be avoided by prepending the following line to the `.proto` message:

```
syntax = "proto2";
```

When the command is successfully executed, the 2 new ***.h*** and ***.cc*** files will appear:

```
$ ls
aquanet.message.pb.cc  aquanet.message.pb.h  aquanet.message.proto
```

Those files can now be used in `ros-aquanet-adapter` to serialize, send, receive and deserialize the DCCL AquaNet messages.

3. Integrating DCCL-message source files into a ROS package

In order to integrate and use the DCCL-enabled data structures in ROS, the corresponding header files generated by `protoc` command, should be included into you ROS project source files.

For example, in the `ros-aquanet-adapter` package, the aquanet DCCL messages are added by including the following headers in the beginning of `start.cpp` program:

```
// dccl
#include "dccl.h"
#include "dccl_messages/aquanet.message.pb.h"
```

The source code of `start.cpp` with the included DCCL headers can be found in **Appendix B**. The source code also contains the examples on how to serialize/send and deserialize/receive DCCL messages over `aquanet-socket` interface. See the corresponding ***twistMessageReceived()*** and ***receiveAqua()*** methods.

3.1 Compiling ROS package with DCCL included

To compile a ROS package with the included DCCL libraries and message structures, some external libraries should be added into `CMakeLists.txt` file. They should be pointing to `dccl`, `protobuf` and `aquanet.message.pb` external source files.

Using the example with `ros-aquanet-adapter` package, the following lines should be added into `CMakeLists.txt` file, marked in red:

```
cmake_minimum_required(VERSION 3.0.2)
project(aquanet_adapter)

find_package(catkin REQUIRED COMPONENTS roscpp)

include_directories(include ${catkin_INCLUDE_DIRS})
catkin_package(CATKIN_DEPENDS)

add_library(aquanet.message.pb dccl_messages/aquanet.message.pb.cc)

add_executable(start start.cpp)
target_link_libraries(start ${catkin_LIBRARIES} protobuf dccl aquanet.message.pb)

install(TARGETS start RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

The complete `CMakeLists.txt` file for the `ros-aquanet-adapter` package can be found in **Appendix C**.

After the corresponding libraries are added, the project can be recompiled using the standard ros-catkin framework:

```
cd $ROS_FOLDER

./src/catkin/bin/catkin_make_isolated --pkg aquanet_adapter --install -DCMAKE
_BUILD_TYPE=Release
```

After a successful compilation, the last step would be to take the `libaquanet.message.so` library compiled by ros-catkin system, and link it to the shared `/usr/lib` library path:

```
sudo ln -s $ROS_FOLDER/devel_isolated/aquanet_adapter/lib/libaquanet.message.
pb.so /usr/lib/libaquanet.message.pb.so
```

The DCCL library can now be used inside ROS environment.

Appendix A. aquanet.message.proto file

```
import "dccl/option_extensions.proto";

message AquanetMessage {
  option (dccl.msg) = { codec_version: 3
                        id: 1
                        max_bytes: 32 };
  required uint32 ros_msg_id = 1 [(dccl.field) = { min: 0 max: 10 precision: 1 }];
  required double x = 2 [(dccl.field) = { min: -10 max: 10 precision: 1 }];
  required double y = 3 [(dccl.field) = { min: -10 max: 10 precision: 1 }];
  required double z = 4 [(dccl.field) = { min: -10 max: 10 precision: 1 }];
  enum VehicleClass { AUV = 1; USV = 2; SHIP = 3; }
  optional VehicleClass veh_class = 5;
  optional bool battery_ok = 6;
}
```

Appendix B. start.cpp file from ros-aquanet-adapter package

```
// This program initializes the aquanet stack of protocols and
// creates outbound/inbound topics for passing the data over aquanet
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <std msgs/String.h>
#include <iomanip> // for std::setprecision and std::fixed

// dccl
#include "dccl.h"
#include "dccl_messages/aquanet.message.pb.h"

// Socket communication
#include<stdlib.h> //exit(0);
#include<arpa/inet.h>
#include<sys/socket.h>
#include <iostream>
#include <sstream>
#include <limits>
// thread-related stuff
#include <thread>
// Aqua-sockets
#include <sys/un.h>
#include "aquanet_include/aquanet_log.h"
#include "aquanet_include/aquanet_socket.h"
char log_file[BUFSIZE];
char* log_file_name = log_file;

// Define inbound and outbound topics for communication over aquanet-enabled nodes
std::string inbound_topic = "aquanet_inbound";
std::string outbound_topic = "aquanet_outbound";

ros::Publisher aquanet_inbound_publisher_twist;

// Aquanet socket
int m_socket = -1;
struct sockaddr_aquanet m_to_addr;
dccl::Codec m_codec;

// A callback function. Executed each time a new velocity message arrives
void twistMessageReceived(const geometry_msgs::Twist::ConstPtr& vel)
{
    ROS_INFO_STREAM(std::setprecision(2) << std::fixed << "position=(" << vel->linear.x << "," <<
vel->linear.y << ")" << " angle=" << vel->angular.z);
```

```
    // Construct aqua-message
    std::string sent_msg;
    m_codec.load<AquanetMessage>();
    {
        AquanetMessage r_out;
        r_out.set_ros_msg_id(1);                // 1 - ros twist-message;
        r_out.set_x(vel->angular.z);            // set angular velocity to x
        r_out.set_y(vel->linear.x);            // set linear velocity to y
        r_out.set_z(0);
        r_out.set_veh_class(AquanetMessage::AUV);
        r_out.set_battery_ok(true);

        m_codec.encode(&sent_msg, r_out);
    }

    // Send aqua-message over aqua-socket
    if (aqua_sendto(m_socket, sent_msg.data(), sent_msg.size(), 0, (struct sockaddr *) &
m_to_addr, sizeof (m_to_addr)) < 0) {
        printf("failed to send to the socket");
        perror("m socket closed");
        exit(1);
    }
}

// Receive thread
void receiveAqua(int recv_socket)
{
    struct sockaddr_aquanet remote_addr;
    int addr_size = sizeof (remote_addr);
    // aqua message received values;
    char recv_buf[32]; // 32 bytes is the maximum aquanet message size, provided by DCCL-protobuf
    std::string recv msg;

    int out_value = 0;
    while (true)
    {
        // out_value = aqua_recvfrom(m_socket, &received_values, sizeof(received_values), 0,
(struct sockaddr *) & remote_addr, &addr_size);
        // std::cout << sizeof(received_bytes) << "\n";
        out_value = aqua_recvfrom(m_socket, &recv_buf, sizeof(recv_buf), 0, (struct sockaddr *) &
remote_addr, &addr_size);
        if (out_value < 0) {
            printf(log_file_name, "failed to read from aqua_socket");
            break;
        }
        else if (out_value == 0)
        {
            // nothing was received during the socket timeout
            printf("socket timeout\n");
            continue;
        }

        // reconstruct aqua-message
        std::string ret(recv_buf, sizeof(recv_buf));
        recv_msg = ret;

        m_codec.load<AquanetMessage>();
        if(m_codec.id(recv_msg) == m_codec.id<AquanetMessage>())
        {
            AquanetMessage r_in;
            m_codec.decode(recv_msg, &r_in);
            std::cout << r_in.ShortDebugString() << std::endl;

        // Publish the messsages to the inbound topic
        // TODO: identify ros-message type and re-generate the correpsonding ros-message
structure to be published
        geometry_msgs::Twist twist;
        twist.angular.z = 1.0*r_in.x();
        twist.linear.x = 1.0*r_in.y();
        aquanet_inbound_publisher_twist.publish(twist);
```

```
        }
    }
}

int main(int argc, char **argv)
{
    if (argc != 3)
    {
        std::cout << "Error! No local and/or destination addresses specified!\n";
        return -1;
    }

    // Start aquanet stack
    system("cd /home/ubuntu/ros_catkin_ws/src/aquanet_adapter/aquanet_scripts &&
./run_aquanet.sh");

    // Initialize the ROS system and become a node.
    ros::init(argc, argv, "aquanet_node");
    ros::NodeHandle nh;

    // Create socket
    if ((m_socket = aqua_socket(AF_AQUANET, SOCK_AQUANET, 0)) < 0) {
        printf("socket creation failed\n");
        perror("m_socket closed");
        exit(1);
    }

    m_to_addr.sin_family = AF_AQUANET;
    // Set local and dest aquanet addresses from CLI
    m_to_addr.sin_addr.s_addr = std::stoi(argv[1]);
    m_to_addr.sin_addr.d_addr = std::stoi(argv[2]);

    // Create a subscriber object.
    ros::Subscriber sub_twist = nh.subscribe("aquanet_outbound_twist/", 1,
&twistMessageReceived);
    aquanet_inbound_publisher_twist = nh.advertise<geometry_msgs::Twist>("aquanet_inbound_twist",
1);

    // Start the receive thread
    std::thread t1(receiveAqua, m_socket);

    // Let ROS take over.
    ros::spin();
}
```

Appendix C. CMakeLists.txt file from ros-aquanet-adapter package.

```
cmake_minimum_required(VERSION 3.0.2)
project(aquanet_adapter)

find_package(catkin REQUIRED COMPONENTS roscpp)

include_directories(include ${catkin_INCLUDE_DIRS})
catkin_package(CATKIN_DEPENDS)

add_library(aquanet.message.pb dccl_messages/aquanet.message.pb.cc)

add_executable(start start.cpp)
target_link_libraries(start ${catkin_LIBRARIES} protobuf dccl aquanet.message.pb)

install(TARGETS start RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```