IS-MOOS Documentation

Andrea Caiti, Vincenzo Calabró,
Gianluca Dini, Angelica Lo Duca, Andrea Munafó

August 30, 2012

Contents

Chapter 1

The IS-MOOS Framework

IS-MOOS (Inter-vehicle Secure MOOS) is an extension of MOOS (Mission Oriented Operating Suite) [Oxford Mobile Robotics Group, 2001]. While MOOS was originally designed as a pub/sub system to be used inside a vehicle, IS-MOOS is a pub/sub system supporting inter-vehicle communication.

In the remainder of the chapter we firstly describe the MOOS architecture and then we briefly illustrate the IS-MOOS architecture. For other details on how to write a MOOS application, please give a look at [Newman, 2009].

1.1 MOOS

The MOOS (Mission Oriented Operating Suite) framework is a centralized pub/sub system, allowing communication among modules inside a vehicle. A module is an application used by the vehicle for a specific purpose (e.g. collect information from the environment or calculate the position of the vehicle). A module can be one of those described in [Benjamin et al., 2010] or it can be implemented by the programmer.

MOOS is composed of a central server, called the MOOSDB, acting as the dispatcher and multiple clients, i.e. the modules of the vehicle. Figure 1.1 shows an example of the MOOS framework: there is the central server (MOOSDB) and three modules: the *Geo Position*, which monitors the position of the vehicle in the environment, the *Temperature Sensor*, which measures the temperature of the water, and the *Remote Shell*, which allows a remote user to access the vehicle in order to send commands or to retrieve information from the other modules.

The communication between the MOOSDB and each client is achieved through the Transfer Control Protocol (TCP), which requires an end-to-end connection between the client and the MOOSDB. When a client wants to join the system, it must perform an handshake with the MOOSDB. The handshake is made of two phases: a) *TCP handshake*, b) *MOOS handshake*. During the *TCP handshake* the client establishes a TCP connection with the MOOSDB, while during the

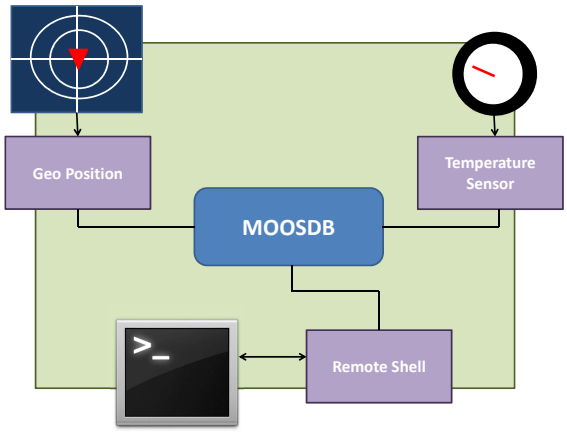

Figure 1.1: The MOOS framework inside a vehicle.

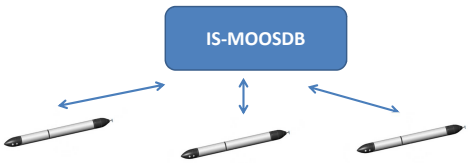

Figure 1.2: The IS-MOOS structure.

MOOS handshake, the client and the MOOSDB exchange some useful information (i.e. client identifier and clock for synchronization).

The MOOSDB has a single thread architecture so that it is able to serve one client per time, although each client is associated a TCP socket. If for some reasons the connection between the MOOSDB and a client gets lost (e.g. the client does not transmit for long periods), MOOS provides a mechanism allowing a client to automatically connect again to the MOOSDB. This mechanism guarantees that the end-to-end communication between each client and the MOOSDB is always up.

1.2 IS-MOOS

Figure 1.2 shows the IS-MOOS structure. IS-MOOS is designed to work in the underwater scenarios. However, it can also be used in other environments.

The communication between each client and the IS-MOOSDB is realized

through the User Datagram Protocol (UDP). UDP presents many advantages when used into an underwater environment. Firstly, it does not need an end-to-end communication so that all the problems related to the connection management are completely avoided. In practice, a client must not open and close a connection with the IS-MOOSDB. Secondly, UDP does not need a preliminary handshake between the client and the IS-MOOSDB so that the problem of frequent disconnections due to the acoustic channel are completely avoided. The main drawback of UDP is that it does not provide any mechanism for packet retransmission. However, this problem can be solved by performing the packet retransmission at the MAC and application layers.

In order to support UDP, the IS-MOOSDB architecture has been redesigned. The IS-MOOSDB has a multi thread architecture in order to serve many clients per time. In practice, each client c is associated a thread T_c. The client c must know in advance the address and the port which the associated thread T_c is listening to. If the client c wants to send a message, it is sufficient that it begins to transmit at the address and port which T_c is listening to, without perfoming any premilinary handshake. When the thread T_c has a message m for c, it sends c the message m together with the information for clock synchronization.

The described mechanism allows to establish a bond between the client c and the thread T_c. However, since there is not any preliminary handshake, it may happen that a client c' begins to send messages to the thread T_c. In order to avoid this problem, all the messages are authenticated through the mechanism described in the next section. Upon receiving a message the thread T_c verifies the authenticity of its source.

IS-MOOS also supports security. In particular it manages confidentiality, integrity and authenticity of messages. For more details about the cryptographic suite, please give a look at [Dini and Lo Duca, 2011].

Chapter 2

Setup a new Network

In order to configure a new network, you must have one MOOSDB and many MOOS clients. The MOOSDB and the MOOS clients should be physically located into different machines. However, you could also deploy them into a single machine. For the installation of the MOOSDB or a MOOS client, please refer to [Newman, 2009].

Each client must be associated in advance to an UDP PORT on the MOOSDB and must know in advance the IP address and the UDP PORT on the MOOSDB.

Both the MOOS clients and the MOOSDB must be associated to a configuration file, with extension .ini, generally named Mission.moos. In the remainder of the chapter we describe the configuration file for the MOOSDB and the MOOS client separately. Since both the client and the MOOSDB share the Security parameters, they are described into a separate section.

2.1 MOOSDB

The MOOSDB configuration file is described in [Newman, 2009]. The IS-MOOS framework adds the following paramters:

```
TRANSPORT_PROTOCOL = UDP|TCP
```

It specifies the transport protocol. It can be TCP or UDP.

```
NUM_PORTS = 10
MIN_PORT = 6000
```

If protocol is UDP you must specify the number of parallel communications allowed. This is specified by `NUM_PORTS`. if the number of clients is greater than this number, some clients must wait before be served if the protocol is UDP you must specify the minimum port value used assigned by the server to the client for communication If you do not know how to set up such value, leave default.

```
READ_TIMEOUT = 20
ABSENT_CLIENT_TIMEOUT = 420
```

The parameter `READ_TIMEOUT` specifies the timeout in seconds for read operations. A value less than 5 is not recommended, because MOOSDB is not able to accept connections The parameter `ABSENT_CLIENT_TIMEOUT` defines the timeout in seconds after which a client is considered disconnected.

The MOOSDB also supports some default topics, which do not need to be registered. All the connected clients are subscribed to the default topics. In order to specify some default topics use the following parameter:

```
DEFAULT_TOPICS = 1:topic1|2:topic2...
```

where each topic must be specified through the syntax n:topicn, where n is the number of default topic, while *topicn* is the name of the topic.

2.2 MOOS client

The MOOS client configuration file is described in [Newman, 2009]. The IS-MOOS framework adds the following parameters:

```
TRANSPORT_PROTOCOL = UDP|TCP
ServerPort = xxxx
```

It specifies the transport protocol. It can be TCP or UDP. If the protocol is UDP, you must specify the `ServerPort` parameter, which indicates the MOOSDB port at which the client must connect.

2.3 The Security Suite

The security suite must be configured in the configuration file. both in the client and the MOOSDB.

```
ENABLE_ENCRYPTION = TRUE|FALSE
ENCRYPTION_ALG = AES|BLOWFISH|CAST5|RC2
ENCRYTPION_KEY = XXXXX
```

These parameters specify if encryption of messages must be enabled or not. The supported encryption algorithms are: AES, BLOWFISH, CAST5 or RC2. The encryption key must be 32 bytes length. The encryption algorithm is symmetric so that all the clients share the same key.

```
ENABLE_INTEGRITY = TRUE|FALSE
INTEGRITY_ALG = SHA2|MD5
INTEGRITY_KEY = XXXXX
```

These parameters specify if integrity of messages must be enabled or not. In this case, a hash is appended to the original message. The supported integrity algorithms are: SHA2 and MD5. The integrity key must be 32 bytes length. The length of the hash is 16 (MD5) or 20 (SHA1) bytes. The hash of a message can be truncated, by setting the `DIGEST_TRUNCATION` parameter to `TRUE` and by specifying the length of the hash through the parameter `DIGEST_LENGTH`.

```
DIGEST_TRUNCATION = TRUE|FALSE
DIGEST_LENGTH = XXXX  (e.g. 4)
```

Bibliography

[Benjamin et al., 2010] Benjamin, M. R., Schmidt, H., Newman, P. M., and Leonard, J. J. (2010). Nested autonomy for unmanned marine vehicles with moos-ivp. *Journal of Field Robotics*, 27(6):834–875.

[Dini and Lo Duca, 2011] Dini, G. and Lo Duca, A. (2011). A cryptographic suite for underwater cooperative applications. In *IEEE International Symposium on Computers and Communications (ISCC'11)*, Kerkyra, Corfu (Greece).

[Newman, 2009] Newman, P. (2009). Introduction to programming with moos, http://www.robots.ox.ac.uk/ pnewman/moosdocumentation/programmingwithmoos/latex/programmingwithmoos.pdf.

[Oxford Mobile Robotics Group, 2001] Oxford Mobile Robotics Group (2001). The moos cross platform software for robotics research, http://www.robots.ox.ac.uk/ mobile/moos/wiki/pmwiki.php.