RoboCup Rescue Virtual Robots: Wireless Simulation Server Documentation

Max Pfingsthorn
m.pfingsthorn@jacobs-university.de

October 8, 2008

1 Introduction

The purpose of the Wireless Simulation Server (WSS) is to simulate wireless network links in a disaster setting. The current version of the WSS does not implement a very accurate signal degradation model. However, it does force participants to deal with the main issues of wireless links, such as limited range and the resulting need for either multi-hop routing or temporary autonomous behavior.

The server works by acting as a hub for communication between the robots. During the competition, all data must be communicated through the Wireless Simulation Server.

Each entity (robot or operator station) that wants to transmit data via the WSS has to open a control connection to it. The proper protocol for communication via that connection is explained below. Only this connection is used to send and receive simulation specific commands and data. All other communication between entities that is facilitated by the WSS follows standard TCP/IP protocols.

A sketch of the process of connecting to another robot is presented below:

1. Robot *A* opens control connection to WSS and registers with a port it is listening on.
2. Robot *B* does the same.
3. Robot *A* queries the WSS for the port to connect to on the address the WSS is running on in order to have the WSS forward data to Robot *B*.
4. Robot *A* connects with a standard TCP socket to the returned port.
5. The WSS accepts the connection only if Robot *B* is in range and opens another socket to the port Robot *B* indicated it is listening on.

6. Robot *B* accepts the connection from the WSS.
7. (Optional) In order to find out the name of the connecting entity, Robot *B* queries the WSS with the *remote port number* of the connection it just accepted.

Both robots can send data via this established connection. The WSS will read up to 2048 bytes at a time from each robot if available and forward the exact same data to the other side. As all connections are implemented as their own threads that use `select` to poll for readable data, bandwidth should not be a problem.

Important: When running the WSS, be sure to turn off your firewall. The WSS chooses random ports to listen on. Also be sure to allow a connection to the port your robot is listening on.

2 GUI

Figure 1 shows the GUI. By default, the WSS will start listening on port 50000. If another port is desired, it can be changed in the top text field. Below is a box to choose the signal degradation implementation, the *PropagationModel* (See section 5). The default *NoopPropagationModel* always allows all connections, returning a signal strength of 0 dBm for any distance. This is mainly intended for testing purposes. Other implementations might be configurable, in which case the “Configure” button would be enabled.

The “Connected Robots” field shows an up-to-date list of robots that are currently registered with the WSS.

“Ports for incoming connections” shows a list of connection to port mappings returned by the `DNS` query defined in Section 3.3. If a certain connection was requested and a port allocated before, the “Listening?” field shows if this connection is currently allowed by the *PropagationModel*. If this field shows *false*, the WSS has closed this port temporarily to prevent robots from connecting.

“Established connections” shows a list of currently active connections. Here, it also shows the port number used to connect to the second robot.

Both of the previous fields can be searched. By typing in the text fields above them, the lists will be filtered. To clear the search, press the “Clear” button.

The box below is a log window. Most of the actions performed in the background are documented here for debugging purposes.

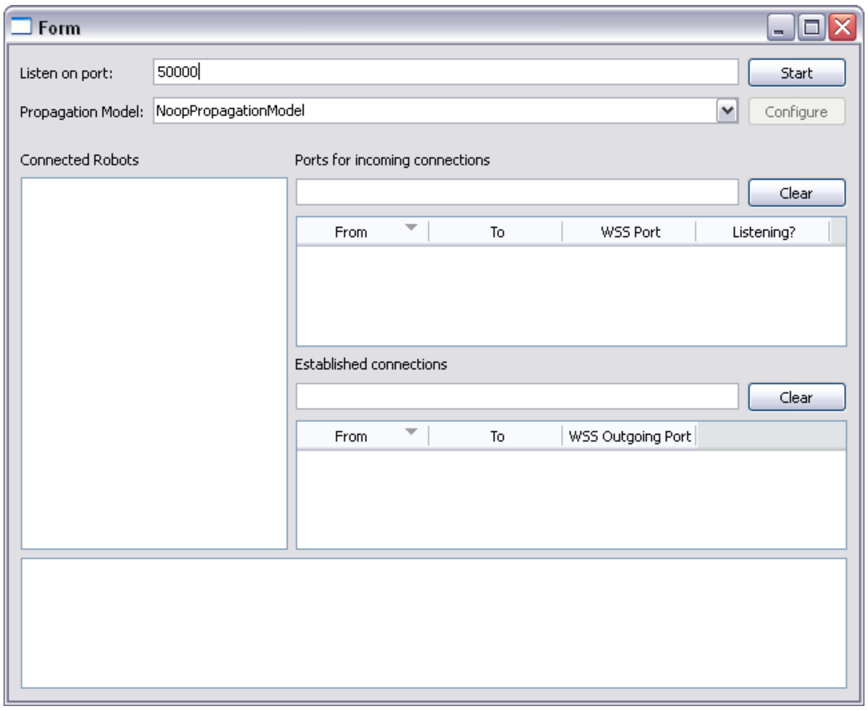


Figure 1: A screenshot of the version 0.1 GUI.

3 The Control Connection

The WSS employs a control connection with each entity to support its operation. This connection is used to

1. register an entity with a port it is listening on,
2. get the signal strength between two registered entities,
3. get a port number on the WSS machine in order to connect to a different entity, and
4. query which other entity connected given the originating port number,

Registered entities are from here on called *clients* of the WSS.

For each operation, clients are referenced by name used to identify their corresponding instances in the simulator. For example, if a robot was spawned in USARSim using the command

```
INIT {ClassName P2AT} {Name Robot1} ...
```

the corresponding WSS client name is also `Robot1`.

Just as with USARSim, all command strings and replies are terminated with a carriage return followed by a newline. In C escape sequences, this is equal to the string “\r\n”.

There will always be a reply from the server. Specific error situations are included below. If a general error occurs, such as you sent a message which was not recognized, the server will respond with an `ERROR` message:

`ERROR {Reason` *`SomeReason`*`}`

The reason is always a string without whitespace.

3.1 Client Registration

You **must** register your client before calling any other command. Also, you can only call this command once per connection request.

The command to initially register the identity of a connecting client is:

`INIT {Robot` *`RobotName`*`} {Port` *`PortNumber`*`}`

Optionally, the *PortNumber* may also be a comma separated list of port numbers without whitespace. I.e.:

`INIT {Robot` *`RobotName`*`} {Port` *`PortNumber1,PortNumber2`*`}`

All following operations will use the given name. Incoming connections will be forwarded to the specified port(s) on the client’s machine. If you try to register a name already in use, an error message will be sent. If successful, the server will reply:

`INITREPLY {Status OK}`

3.2 Getting the Signal Strength

Getting the signal strength between the current (i.e. set by `INIT`) robot and another is done via the command:

`GETSS {Robot` *`OtherRobotName`*`}`

The reply of the server will be:

`SS {Robot` *`OtherRobotName`*`} {Strength` *`SignalStrength`*`}`

where *`SignalStrength`* is a double formatted as text, *NaN* if the *`OtherRobotName`* is unknown, or *INF* if the name is the same as given in the `INIT` command.

3.3 Connecting to Another Station

In order to connect to another robot (or the operator), you need the port number on the WSS machine for this exact connection. Each port the WSS listens on is for exactly one pairing of robots or the operator. Actually, the WSS listens on two ports for each pairing, one for each direction. The WSS will match the IP address of the connecting client with the one used for the current control connection to the expected client for a minimum of security.

To get the port WSS is listening on for you (the robot specified in `INIT`) to connect to any other station in the simulated wireless network, use:

`DNS {Robot` *`RobotName`*`}`

Optionally, you may add a field to the `DNS` message to ask for a connection to a specific port on the named robot:

`DNS {Robot` *`RobotName`*`} {Port` *`OtherRobotPortNumber`*`}`

The reply from the server will be:

`DNSREPLY {Robot` *`RobotName`*`} {Port` *`PortNumber`*`}`

where *PortNumber* is the port on the WSS machine you should connect to. If the current signal strength is above a predefined threshold, the port will be open, listening for connections. If not, the port will be closed.

If the *RobotName* given in the query doesn't exist, is the same as given in `INIT`, or the desired robot does not listen on the requested port, the *PortNumber* will be `-1`. Additionally, the message will include a field called `Error` with a reason to distinguish these cases:

`DNSREPLY {Robot` *`RobotName`*`} {Port -1} {Error` *`Reason`*`}`

3.4 Asking which Robot is Trying to Connect

Since all incoming connections to the port specified in `INIT` originate from the WSS, it is important to be able to distinguish between different robots trying to connect. For that purpose, the WSS implements a reverse DNS lookup.

`REVERSEDNS {Port` *`PortNumber`*`}`

queries the WSS for the name of the robot trying to connect given the originating port number on the WSS machine. This port number can be read from the socket with the C system call `getpeername()` in Unix and Windows. There are equivalents in all major programming languages.

On success, the WSS will reply:

`REVERSEDNSREPLY {Port` *`PortNumber`*`} {Robot` *`RobotName`*`}`

If the given port number is not in use, or used for a connection to another robot than given in `INIT`, the reply will be:

`REVERSEDNSREPLY {Port` *`PortNumber`*`} {Error UnknownOrIllegalPort}`

4 Making the Connection

Actually connecting to another robot is very straightforward once the corresponding port has been queried from the WSS. This process uses plain sockets which are available on any programming platform.

Once a socket has been connected to the corresponding port on the WSS machine, you are free to use it as you please. Any form of data sent over the connection will be read from the incoming socket on the WSS and written straight to the outgoing one connected to the target robot.

If the signal strength should fall below some predefined threshold, both incoming and outgoing connections will simply be closed. The WSS will also stop listening for new connection attempts while the signal strength is too low. As soon as the signal strength is up again, the WSS will start listening on the same port as before, and reconnection will be possible again.

5 Propagation Models

5.1 Introduction

Propagation Models implement a way to compute the signal attenuation between two connecting stations. The main way to extend the WSS is by implementing new *Propagation Models* as plugins.

Implementations of the main *PropagationModel* interface communicate with the main WSS via the *RobotLinkModel* interface, which allows the *PropagationModel* to terminate and resume connections between two robots, kick out robots from the WSS by name, e.g. after their battery is empty, or get a list of currently registered robots.

In turn, the *PropagationModel* allows the main WSS to query for the current signal strength between two robots and whether a connection between two given robots is currently possible. Also, the interface of the *PropagationModel* is designed to allow the implementation to be multi-threaded and includes methods to stop and start the underlying implementation. Configurations have to be checked in the start method such that incomplete settings can be detected before the main WSS starts. Configurable implementations can also override methods to display a configuration dialog.

5.2 *NoopPropagationModel*

The most basic implemented model is the *NoopPropagationModel*. It always shows a signal strength of 0 dBm (i.e. no attenuation) for all connections. This is useful for debugging your network implementation without having to run USARSim at the same time.

5.3 *DistanceOnlyPropagationModel*

The *DistanceOnlyPropagationModel* implements the calculation of signal strength based on the distance between two robots in the simulator. It maintains a connection to the simulator to update the positions of robots continuously.

The signal strength is computed given a number of configurable parameters:

- S_c: The signal cutoff in dBm. This is usually -93 as it is a physical property of the wireless channel. In the configuration dialog, this is referred to as "Cutoff".
- P_{d_0}: The path loss in dBm at d_0 meters from the source.
- d_0: The distance from the source at which the path loss in P_{d_0} is reached.
- N: The attenuation factor of the environment. This parameter decides how fast the signal is attenuated over distance.

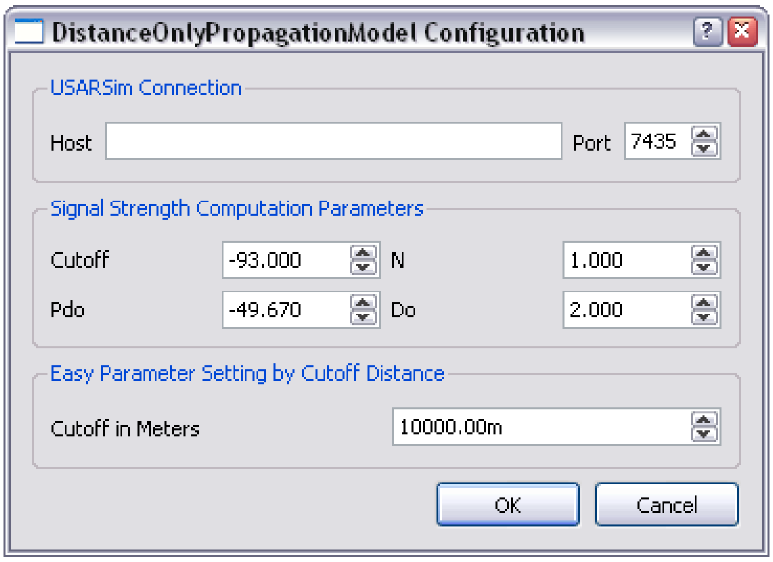
The complete equation to compute the path loss at a certain distance d_m from the source of the signal is:

$$S = P_{d_0} - 10 \cdot N \cdot \log_{10}\left(\frac{d_m}{d_0}\right)$$

When S falls below S_c, the *DistanceOnlyPropagationModel* will terminate any connections between the two robots in question and also stop the threads listening on the ports which correspond to this robot pair. This means that subsequent attempts to connect to a robot which is out of range will fail with a "Connection refused" error since that port is no longer open on the WSS. Once S is high enough again, e.g. after the robots moved closer together, the ports will be opened again for new connections.

Figure 2 shows the configuration dialog for this propagation model. The first section configures the connection parameters to USARSim. Here, it is especially important to know the right port. **Do not use port 3000**, which is the standard port to connect to for spawning robots and getting sensor data. Port 7435 is the standard port to use here. You must type in a numerical IP address, the WSS does not resolve DNS names.

The middle section allows the configuration of the computation parameters mentioned above. While it is possible to change the "Cutoff" parameter (S_c above), it is recommended to leave it at $-93dBm$.


DistanceOnlyPropagationModel Configuration

USARSim Connection

Host

Port 7435

Signal Strength Computation Parameters

Cutoff -93.000

N 1.000

Pdo -49.670

Do 2.000

Easy Parameter Setting by Cutoff Distance

Cutoff in Meters 10000.00m

Figure 2: The configuration dialog for the *DistanceOnlyPropagationModel*.

The last section facilitates changing the desired distance at which the *DistanceOnlyPropagationModel* terminates connections. Any changes in the above parameters is immediately reflected here. Also, changing the desired distance here will automatically update the parameter N above.

Once OK is clicked, the parameter values are saved in the *DistanceOnlyPropagationModel*. It also saves these settings persistently on your computer. In Windows, this is done via the Registry. This way, your previous settings are available again the next time you start the WSS.

5.4 Future Implementations

Future implementations include but are not limited to:

- *RayTracingPropagationModel*: A method which runs ray traces in the simulator to detect the number of obstacles between two entities in order to attenuate the signal more accurately.

Other implementations, such as a debugging one, which allows to selectively break connections between specific entities is also possible.

6 License

The application and this documentation is released under the GNU General Public License (GPL) version 2.