Swarm Simulation Communication Report

Open Source Robotics Foundation
Mountain View, CA

July 15, 2015

1 Introduction

We aim to study the problem of collective control of large teams ("swarms") of air and ground robots. Our focus is the scalability of such teams, in particular with regard to: (i) communication systems and (ii) coordination algorithms. We aim to characterize the behavior of such systems, identifying how they change (and fail) with growth in size. The primary deliverable of this study will be a report on the subject of swarm control and communication that will provide guidance for future projects involving swarms of robots.

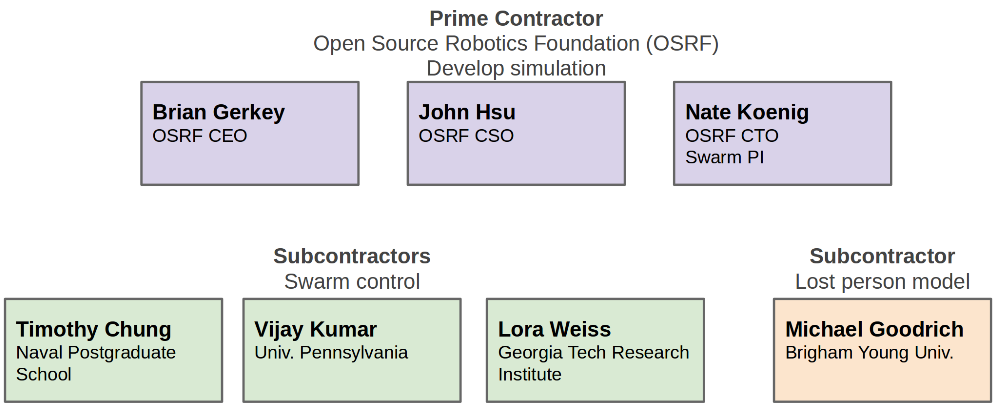


Figure 1: Team organization and responsibility.

The study will take place in simulation, using a combination of a robot simulator and a network simulator. The simulated task will be cooperative search for one or more lost persons in an outdoor environment. Our team (Figure 1), led by the Open Source Robotics Foundation (OSRF), includes research groups at Brigham Young University (BYU), the Georgia Tech Research Institute (GTRI), the Naval Postgraduate School (NPS), and the University of Pennsylvania (UPenn).

OSRF will develop, distribute, and support the simulation environment, as well as manage the project, collect experimental results, and report our findings to DARPA. BYU is responsible for providing the model to control the lost persons as they move around the simulated world and provide domain expertise on performance metrics for search and rescue. The other three university team members are each responsible for applying an existing coordination algorithm to the problem of lost-person search using the simulation provided by OSRF.

In this report, we describe and justify our plans for simulating robot swarms, with particular attention to which aspects of communication are simulated, and how they are modeled.

See the Appendix at the end of this document for the conclusions of the review process.

2 Robots & Environments

Modifications to this section are found in the Appendix under Report Corrections.

To simulate the cooperative search task, we model three types of robot vehicles:

- **ground** : A wheeled or tracked vehicle that can move through a variety of outdoor terrain.
- **fixed-wing** : An airplane-like vehicle that can fly, but not hover.
- **rotorcraft** : A helicopter-like vehicle that can fly, including hovering.

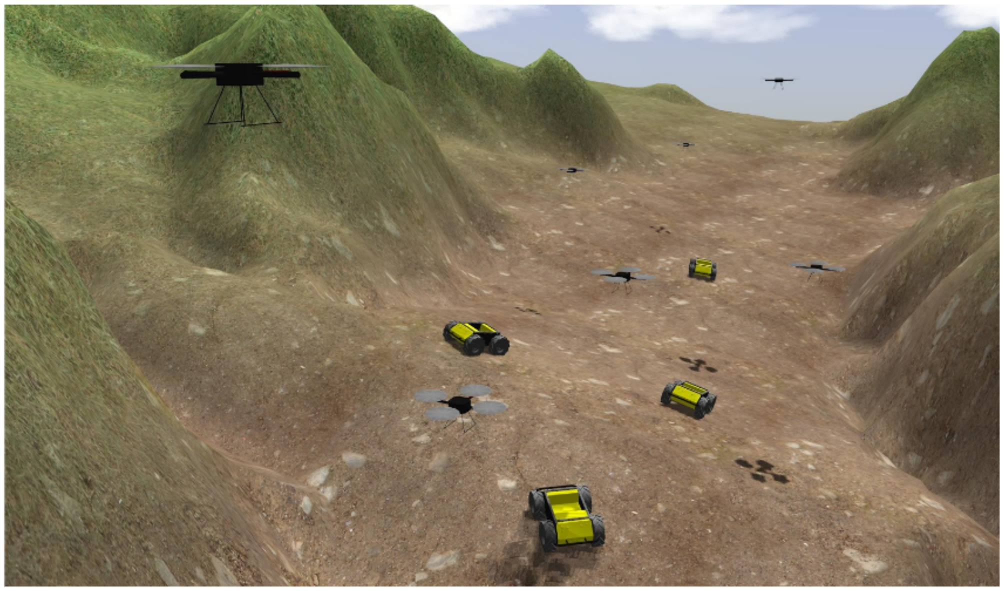


Figure 2: Visualization of an outdoor ground-robot swarm of the type that will be simulated.

These vehicles operate in a large and varied outdoor environment that includes multiple types of terrain, such as mountains, valleys, forests, and open areas (Figure 2).

For the present report, these purposefully vague definitions will satisfy our needs. We do not need to carefully model any specific vehicles, nor any particular geographic area. For example, we might model fixed-wing vehicles as able to travel faster and more efficiently than rotorcraft vehicles, but we will not model the details of a specific fixed-wing or rotorcraft vehicle. What we need to capture is that the vehicles will move with respect to each other in a variety of ways as they operate in the environment. As a result, there are many possible physical arrangements of the swarm, with vehicles interacting ground-to-ground, ground-to-air, and air-to-air, along with the effects of the local terrain on mobility, perception, and communication. For example, two ground robots might be positioned near each other in an open space or they might be in neighboring valleys, separated by a mountain.

To simulate the robots and their environment, we use an enhanced version of Gazebo, a high-fidelity open source 3D robot simulator.[1] Gazebo is commonly used in robotics research and development, including DARPA programs such as DARPA Robotics Challenge, HAPTIX, and Mentor2. In addition to its existing capabilities in robot modeling, simulation, and visualization, Gazebo's extensive configurability via plugins makes it especially suitable to the present application.

The Gazebo simulator has historically focused on accurate dynamic and sensor simulation. Some aspects of simulation fidelity are reduced in order to accommodate swarms of robots. We view this reduction in fidelity as acceptable given the focus in this project on swarm control and communication strategies. Extensions to Gazebo include the ability to adjust physics fidelity and utilize low-fidelity sensors as well as simplified models of ground and aerial vehicles. This work allows dynamic simulation to be turned off, camera sensors to operate as (noisy) Boolean object detectors, and environments to be constructed as a property field.

[1]`http://gazebosim.org`

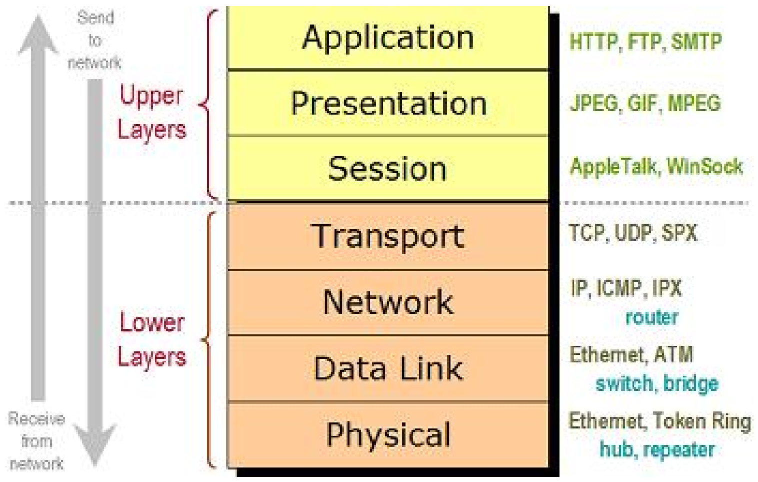


Figure 3: The OSI/ISO seven-layer network model. We are concerned with simulating the lower four layers, to present to the user a transport-layer interfaces for exchanging messages.

3 Networking

To model communication among the kinds of vehicles identified in Section 2, we need to consider the space of available wireless networking technologies. The main techniques to implement wireless communication are radio frequency (RF), optical, acoustic, and magnetic induction communication.

Optical communications, such as infrared, are not affected by electromagnetic interference. However, external light sources (common in outdoor environments) can create interference to optical systems. Acoustic systems are common in teams of underwater vehicles but have a relatively very slow and narrow bandwidth in comparison with RF. Magnetic induction communication allows for low-power and low-cost communication systems. However, magnetic induction limits the operating range to a few meters. RF technologies allow long-range communications if needed, low latency, and a good trade-off between power consumption and bandwidth. Given the range and bandwidth requirements of our study, we will focus the networking survey on RF-based wireless technologies.

In the following sections, we discuss the options available at each of the four lower layers of the network, following the OSI/ISO seven-layer network model (Figure 3).

3.1 Physical and data link layers

The physical layer of a wireless communication system deals with signal modulation and encoding for data transmission. The data link layer allows data transfer between nodes located on the same network segment. The Institute of Electrical and Electronics Engineers (IEEE) 802 standard covers a subset of the most common and open specifications for wireless communication, which are the focus of attention in our survey.

802.11 standard

The 802.11 standard uses 2.4, 3.6, 5, and 60GHz bands and DSSS, FHSS or OFDM modulation. In outdoor environments, its range of operations is 100-250 meters. The stream data rate varies from 1 Mbit/s (802.11b) to 6.75 Gbit/s (802.11ad). 802.11 uses two modes of operation: infrastructure or ad-hoc. Infrastructure mode requires an access point that serves as a bridge between the mobile nodes. Ad-hoc mode does not require any infrastructure as nodes transmit directly peer-to-peer. Wireless Fidelity (WiFi) is a popular technology based on the 802.11 standard.

802.15.1 standard

The 802.15.1 standard, commonly referred to as Bluetooth, was originally designed for short-range and cheap devices, such as computer peripherals. This range of applications is known as wireless personal area networks (WPANs). Multiple standards are available at 2400 and 2483.5 MHz and GFSK, DQPSK or 8DPSK modulation. Bluetooth typical range of operation is 1 meter (class 3 radios), 10 meters (class 2 radios), or 100 meters (class 1 radios). The stream data rate varies from 1 Mbit/s (v1.2) to 24 Mbit/s (v4.0 LE). The communication mode is based on a master/slave model, where the master node can communicate with up to seven slave nodes in a round-robin fashion creating a piconet. The devices can switch roles and a previous slave can become a master. An additional topology named scatternet is also possible, in which multiple piconets connect to form a scatternet, with certain nodes able to relay data between the members of both piconets. Bluetooth technology, managed by the Bluetooth Special Interest Group, is available using the 802.15.1 standard.

802.15.3 standard

The 802.15.3 standard is designed for high data rate and short to medium range applications (10 meters), such as audio and video delivery in home networking. It uses the 3.1-10.6 GHz band and pulse position or time modulation. The stream data rate is up to 480 Mbit/s. Star or peer-to-peer are the most common network topologies. 802.15.3 uses piconets, but unlike 802.15.1, it supports peer-to-peer communications among the members within the same piconet. Ultra Wideband (UWB) is an example of technology built atop IEEE 802.15.3.

802.15.4 standard

The 802.15.4 standard is designed for low-rate wireless personal area networks (LR-WPANs) using low-cost and low-speed devices. It operates using 800, 900 and 2400 MHz and DSSS, BPSK or OQPSK modulations. The range of operation varies from 10-100 meters depending on transmitter power, receiver sensitivity and the environment conditions. The maximum data rate is 250 kbit/s. The standard defines two network topologies: star (using a coordinator node) or peer-to-peer. The ZigBee protocol works over the 802.15.4 standard.

Discussion

Table 1 summarizes the most important parameters of the standards evaluated for our study. 802.11 and 802.15.3 provide higher data rates and 802.15.1 and 802.15.4 lower data rates. 802.11 standard technologies can reach up to 250m in outdoor environments. 802.15.1 and 802.15.4 allow up to 100m communication range, but they are technologies intended for 10m scenarios. Some of the standards impose a strong restriction in the number of devices within each network. For example, 802.15.1 and 802.15.3 only allow 8 nodes per piconet. In terms of power consumption, 802.15.1 and 802.15.4 offer the lower absolute numbers, as are designed for portable devices. However, 802.11 and 802.15.3 have better efficiency in energy consumption (energy per byte) but introduce more traffic overhead resulting in higher absolute energy consumption numbers. For our study, 802.11 and 802.15.4 are suitable standards to be equipped in each member of the swarm. Both of them allow thousands of nodes within the same network and offer enough range and data rate for the search and rescue application. 802.11 will allow higher data rates at the cost of increased energy consumption, whereas 802.15.4 will require a more dense networks with lower data rate communications. On the other hand, the autonomy will be better due to the lower power consumption of the radios using 802.15.4

Standard	Frequency	Modulation	Max range	Max data rate	Products
802.11	2.4 / 3.6 / 5 / 60 GHz	DSSS / FHSS / OFDM	100-250 m	6.75 Gbit/s	WiFi
802.15.1	2.4 / 2.5 GHz	GFSK / DQPSK / 8DPSK	1/10/100 m	24 Mbit/s	Bluetooth
802.15.3	2.4 / 2.5 GHz	BPSK / QPSK	10 m	480 Mbit/s	UWB
802.15.4	0.8 / 0.9 / 2.4 GHz	QPSK	10-100 m	250 Kbit/s	ZigBee

Table 1: Summary of physical and data link standards.

The IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) protocol deserves a special mention in this survey. This protocol allows the use of regular IPv6 addresses and interoperation with existing IP networks. In this study, this might be particularly interesting for enabling communications with ground stations or special purpose nodes that will relay data over the internet. 6LoWPAN can be used with 802.15.4 for allowing link-layer mesh routing under IP topology.

3.2 Network layer

The network layer allows data transfer between nodes that are not directly connected by the same network segment. Routers are typically in charge of forwarding data from one segment to another using routing algorithms. In peer-to-peer topologies, routing is a responsibility of the regular nodes.

In our study, a simulated swarm of robots cannot afford a long range and high power radio. Once the robots start moving within the environment, they will naturally create a mesh network where only a subset of robots will be within the communication range of each node. As we cannot assume that there is available infrastructure in a search environment, a mobile ad-hoc network (MANET) seems the most appropriate option for our study. In particular, we will create a simulated ah-hoc network where nodes (robots) have a high 3D mobility, low power consumption, high computational power and self-localization capabilities. A routing protocol is required to direct network traffic among between source and destination via (potentially multiple) intermediate vehicles.

Probably the most common metric for classifying ad-hoc routing algorithms is whether the routing is performed as a regular task in the background or, on the other hand, is done on-demand. In the first group we can find the proactive algorithms and in the second group the family of reactive algorithms.

3.2.1 Proactive routing algorithms

This group is characterized by keeping up-to-date routing tables in the background. Each node tries to discover the entire network topology for finding the best routes to destination nodes at any time. Thus, the main advantage of proactive algorithms are their latency because client applications can immediately send data. This is a double-edged sword because the cost of maintaining the routes generates network overhead.

Destination-Sequenced Distance Vector (DSDV) is a classic reference protocol under this group. It follows the Distance-Vector routing approach, where each node periodically sends to its neighbors a tuple containing routing information for all possible destinations. Each route entry contains destination identifier, number of hops and the first neighbor of the route.

Optimized Link State Routing Protocol (OLSR) is another reference protocol in the literature of proactive routing algorithms. It follows the Link-State routing approach using "hello" messages to find its one- and two-hop neighbors. This connectivity information is flooded among the members and computed at each destination to create a topology map with the shortest path to each destination.

3.2.2 Reactive routing algorithms

Reactive protocols generate much less control traffic because they wait until an application demands communication. Then, the routing algorithm triggers the route discovery and when the route is found the data is exchanged. The caveat of these algorithms are the higher latency compared with proactive protocols. However, in scenarios with high node mobility involved, it is often better to discover a route right when is needed to guarantee that the route is valid.

Dynamic Source Routing (DSR) protocol is a pure on-demand algorithm in which routes are only discovered when needed. Besides its on-demand nature, the source node also establishes the full route that the message is going to follow to the destination. Intermediate nodes just forward the messages following the routing path contained in the message.

Ad Hoc On-Demand Distance Vector (AODV) only maintains routes to destinations that are not in active communication. Unlike DSR, intermediate nodes are required to keep an updated route to the destination. AODV uses sequence numbers to uniquely identify route requests. While forwarding route requests, sequence numbers are verified to avoid retransmissions of the same requests and reduce control overhead.

3.2.3 Discussion

Modifications to this section are found in the Appendix under Report Corrections.

We have summarized the motivation for using a MANET protocol and described the two main paradigms among them. A hybrid approach is also popular combining features from both families. A proactive technique is used inside a delimited region and an on-demand technique is required for communicating outside this zone. Table 2 enumerates some of the most common routing algorithms classified by its family.

Family	Ad-hoc network protocols
Proactive	**DSDV**, **OLSR**, Babel, WRP, GSR, FSR, HSR, ZHLS, CGSR
Reactive	**AODV**, **DSR**, CBRP, TORA, ABR, SSR
Others	ZRP, HSLS, OOPR, GPSR, Wave Relay

Table 2: Common ad-hoc routing algorithms

The mobility models of the vehicles in our study, their computing power and their energy models are going to dictate which network protocol performs better. A proactive protocol has lower latency but creates more control overhead, as well as battery consumption. A reactive protocol has lower energy consumption and imposes minor overhead on the network at the cost of bigger latencies for the first message. If the degree of mobility is high, a proactive protocol can fail because a route was learned some time ago and it might be no longer valid when needed. A hybrid approach could have the advantages or disadvantages of both worlds too; there are always trade-offs.

For our study, the best solution would allow us to plug different routing protocols in the simulation to mitigate the risk of prematurely committing to a single protocol when we do not know yet understand the implications for the task at hand. The number of protocols is enormous and despite differences in all of them, our first option is to select a subset of reference and well-known protocols such as DSDV, OLSR, DSR and AODV. These protocols are not new and they might be improved by newer algorithms. However, their effectiveness has been demonstrated over many years. In addition, it is easier to find implementations these algorithms in existing network simulators. The Wave Relay radio from Persistent Systems is an example of an apparently well-suited technology for our study. However, although the documentation claims to use 802.11 and ah-hoc protocols, it is difficult to judge if we will find any kind of limitations. In addition, network models for such proprietary technologies will be hard to find.

3.3 Transport layer

The transport layer allows applications, rather than hosts, to communicate with each other. This service is achieved by using ports. Other optional services found in some transport layers are reliability and flow control. In practice, the two options available are the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP). UDP is a protocol that does not offer any guarantees about data delivery. UDP is stateless and has the possibility of sending data to all neighbors (broadcast) or a group of them (multicast). TCP offers reliable transport as well as flow and congestion control. In addition, TCP is a connection oriented protocol, suitable only for unicast data exchange between a single pair of nodes.

For this survey, we prefer UDP. A wireless network with unstable connectivity is not a good scenario for TCP. It is well-known that TCP does not perform well in this type of scenario, with high packet loss and long delays due to retransmission. UDP is a lighter protocol, better suited for highly mobile vehicles and also a better fit for systems limited CPU power compared with a regular computer. In addition, UDP offers broadcast and multicast capabilities, which can be interesting for swarm applications.

4 Network Simulation Requirements

To simulate communication technology of the type described in Section 3, we require a network simulator with several features:

- **Physical models** : We require models of RF propagation loss and delay for relevant frequencies in relevant environments. We will focus on 2.4GHz and 5GHz because of their relevance to 802.11 networks, but also include the 900MHZ neighborhood because of its use by 802.15.4 networks. Propagation models should be configurable and swappable. Optionally, the effect of specific antenna characteristics could be included.
- **Data link models** : We require models of relevant data link layers, primarily their media access control (MAC) protocols. We will focus on 802.11 and 802.15.4 networks.
- **Network models** : We require support for IP, along with implementations of relevant MANET routing algorithms. Because we expect the choice of MANET algorithm to have a significant experimental impact, we wish to have as many choices, with as much configuration ability, as possible.
- **Transport models** : We require support for UDP.
- **Co-simulation** : We need to be able to integrate the Gazebo robot simulator and the chosen network simulator so that they can run simultaneously, along with controllers that are commanding the robots. The network simulator should be capable of running faster or slower than real time, as needed. We prefer to link the network simulator as a library into Gazebo, then step the network simulator from within a Gazebo plugin.
- **Development platform** : We need the network simulator to support our primary development platform, Ubuntu Linux.
- **License** : For reasons of inspection, modification, extension, and cost, we prefer that the network simulator be available under an open source license.
- **Support** : We require that the network simulator be supported, either by an active open source community or by a company.

5 Network simulators

Given the requirements from Section 4, we investigated the use of the following network simulators:

- EMANE[2]
- netem[3]
- NetSim[4]
- ns-3[5]
- OMNeT++[6]
- SteelCentral NetModeler Suite[7] (formerly OPNET Modeler Suite)

EMANE

The Extendable Mobile Ad-hoc Network Emulator (EMANE) is an open source tool, released under the BSD license, and actively maintained and supported by the Naval Research Lab (NRL). The extensive documentation, examples, and training materials provided with EMANE are excellent.

EMANE is structured in a layered manner that mirrors how real networks are built, from physical to data link / MAC to network to transport, with some ability to swap models at each layer. It contains models that are sufficient for emulating UDP traffic atop IP over 802.11 physical/MAC networks, including RF path loss with various antenna configurations. Support for 802.15.4 networks seems to be absent.

EMANE can be built from source on a variety of platforms, including Ubuntu Linux. In addition to the source, the EMANE team distributes binary packages for Ubuntu Linux.

A key characteristic of EMANE is that it is an *emulator*, not a simulator. It works by creating new virtual network interfaces through the operating system (e.g., on Linux, you get a new `ethN` interface for each emulated network node). As a consequence, existing network software can be used unchanged with EMANE, because it will see what looks like a regular physical network interface. So, for example, EMANE does not contain implementations of MANET routing algorithms like AODV, because the standard versions of those algorithms that ship with the operating system can be used.

The downside to emulation, for our project, is that it must run in real-time. All the software, from the emulation of the physical layer up through the application code, uses the computer's system clock. As a result, we cannot speed up EMANE when the environment complexity would permit faster-than-real-time operation. More importantly, we cannot slow it down when environment complexity requires slower-than-real-time operation.

The emulation-only nature of EMANE makes co-simulation simple in the sense that there is no need to synchronize two systems because they both run at real-time, according to the host clock. But strictly real-time co-simulation is inflexible and will not scale well as we explore large team sizes. If we exceed the available computational resources when using EMANE, rather than just running slower, we run the risk of incorrect behavior in the emulated network. It is possible to emulate large networks in real-time with EMANE by using clusters of physical computers, but at the cost of significant added system complexity to manage the cluster.

netem

netem is an open source tool, released under the GNU GPLv2 license, and maintained and supported by the Linux community as part of the the core Linux subsystem. netem offers the ability, through the command-line interface *tc* (for

[2]`http://www.nrl.navy.mil/itd/ncs/products/emane`

[3]`http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`

[4]`http://tetcos.com/netsim_gen.html`

[5]`https://www.nsnam.org/`

[6]`https://omnetpp.org/`

[7]`http://www.riverbed.com/products/performance-management-control/network-performance-management/network-simulation.html`

“traffic control”), to modify the behavior of existing network interfaces on a Linux system.

The purpose of netem is to support testing of software that is expected to operate in less-than-ideal network circumstances. To that end, it offers the ability to apply delay, loss, duplication, corruption, re-ordering, and rate-limiting to packets delivered through a network interface. These behaviors can be configured in a variety of ways, including rules that define which kinds of packets (e.g., based on specific from/to addresses) should be manipulated. netem does not provide implementations of MANET routing algorithms, because the standard versions supplied with the operating system can be used atop the network interfaces that netem is manipulating.

Co-simulation with netem is easy in the sense that no explicit synchronization or custom integration is required, because the operating system’s regular network interfaces are used. We would need to create extra virtual network interfaces to allow simulation of multiple vehicles on the same physical machine. But netem does not include many of the features that we require, including models of wireless physical and MAC layers and the ability to run faster or slower than real-time.

NetSim

NetSim is a proprietary network simulator that is sold and supported by Tetcos. NetSim supports UDP on IP over 802.11 networks and contains implementations of common routing algorithms: AODV, DSR, OLSR, and ZRP. It also supports ZigBee on 802.15.4 networks. NetSim can run faster or slower than real time as dictated by the environment complexity and available resources.

According to Tetcos, we can achieve co-simulation by letting NetSim control the main loop, periodically calling Gazebo’s update method to advance the robot simulation. It is not clear whether we can do the reverse, which is our preference from a system architecture point of view.

NetSim is supported only on Microsoft Windows using Microsoft Visual Studio. The Tetcos team has offered to share with us a custom Linux build of the core NetSim libraries, but this course of action carries a great deal of uncertainty.

ns-3

ns-3 is an open source network simulator, released under the GNU GPLv2 license. It follows the widely used ns and ns-2 simulators that date back to the 1990s. It is actively maintained and supported by a global community of developers and users, led by the NS-3 Consortium, which was founded by INRIA and the University of Washington. The documentation, examples, and tutorials for ns-3 are excellent. ns-3 can be built from source on a variety of platforms, including Ubuntu Linux.

Like EMANE, ns-3 is designed in a layered manner, with support for swapping out choices at each layer. For example, substituting one MANET routing protocol for another can be done by changing at little as one line of code. ns-3 contains an extensive set of models spanning the network stack from physical up to transport.[8] It supports UDP on IP over 802.11 networks, including RF path loss with various antenna configurations. Common MANET routing algorithms are modeled: AODV, DSDV, DSR, and OLSR. It supports 802.15.4 networks, as well as 6LoWPAN, which can be used to interoperate with IP-based networks.

Co-simulation with ns-3 appears to be possible in either direction: Gazebo can call on ns-3 to update itself, or the reverse. The network simulation can run faster or slower than real time, as dictated by environment complexity and available resources.

[8] `https://www.nsnam.org/docs/release/3.23/models/ns-3-model-library.pdf`

OMNeT++

OMNeT++ is a proprietary simulator network simulator that is actively developed and supported by Simulcraft, Inc. The system, including source code, is freely available for academic and non-profit use; commercial use requires the purchase of a license. OMNeT++ can be built from source on a variety of platforms, including Ubuntu Linux. The documentation and examples for the core of OMNeT++ is thorough, but many of the key components (e.g., 802.11, UDP) are incompletely documented.

OMNeT++ supports UDP on IP over 802.11 networks and contains implementations of common routing algorithms: AODV, DYMO, GPSR, DSDV, DSR, and OLSR. It also simulates 802.15.4 networks, though support for 6LoWPAN for IP interoperation is unclear.

The OMNeT++ software appears to be very modular, as evidenced by the fact that different communities have built their own special-purpose network simulators atop the OMNeT++ libraries. Co-simulation with OMNeT++ appears to be straightforward.

SteelCentral NetModeler Suite

SteelCentral NetModeler Suite (NetModeler) is a proprietary network simulator sold and supported by RiverBed (it was formerly developed by OPNET, which was acquired by RiverBed). NetModeler can simulate UDP on IP over 802.11 networks, and provides common routing algorithms: DSR, AODV, TORA, and OLSR. It is not clear whether 802.15.4 network are supported.

Co-simulation with NetModeler appears to be possible in either direction: Gazebo can call on NetModeler to update itself, or the reverse. The network simulation can run faster or slower than real time, as dictated by environment complexity and available resources.

NetModeler is supported on Microsoft Windows and RedHat Linux. There is some risk associated with running NetModeler on Ubuntu Linux, but it should be possible.

5.1 Discussion

The key features of the network simulators under consideration are summarized in Table 3. We can immediately discard netem for lacking the ability to model wireless physical and MAC layers. We can similarly discard EMANE for lacking the ability to do non-real-time simulation (though it should be considered for future projects if cluster compute resources can be easily made available to users to support real-time emulation of large systems). The lack of Linux support for NetSim make it infeasible for our purposes.

We are left with ns-3, OMNeT++, and NetModeler. We can likely execute the Swarm Simulation project with any of the three systems. But, given its superiority in feature set and license status, as well as its widespread use in academia and industry, we conclude that ns-3 is the best choice. Should ns-3 not meet our needs for some reason, we will fall back on OMNeT++, then NetModeler.

It is worth noting that while all the network simulators either provide or are able to use common MANET routing algorithms (AODV, DSDV, etc.), no simulator offers access to proprietary MANET routing algorithms, such as that used by the Persistent Systems Wave Relay radio, which has been deployed in some DOD applications. The choice of MANET routing algorithm will have an impact on the experimental results from this project, and if it is determined to be vital that we model a militarily relevant system such as the Wave Relay radio, then we will need access to the proprietary information necessary to implement the required models.

Name	Physical	MAC	IP	MANET	UDP	Co-simulation	Linux	License	Support
EMANE	🟩 Many RF propagation models	🟩 802.11, 802.15.4	🟩 Y	🟨 Use OS implementations	🟩 Y	🟥 Only real-time emulation	🟩 Y	🟩 BSD	🟩 Active community
NetSim	🟨 Some RF propagation models	🟩 802.11, 802.15.4	🟩 Y	🟩 AODV, DSR, OLSR, ZRP	🟩 Y	🟩 Appears to be feasible	🟥 Only Windows	🟨 Proprietary	🟩 Commercial support
netem	🟥 None	🟥 None	🟩 Y	🟨 Use OS implementations	🟩 Y	🟥 Only real-time emulation	🟩 Y	🟩 GNU GPLv2	🟩 Active community
ns-3	🟩 Many RF propagation models	🟩 802.11, 802.15.4	🟩 Y	🟩 AODV, DSDV, DSR, OLSR	🟩 Y	🟩 Appears to be feasible	🟩 Y	🟩 GNU GPLv2	🟩 Active community
OMNeT++	🟨 Some RF propagation models	🟨 802.11, 802.15.4 (6LoWPAN?)	🟩 Y	🟩 AODV, DYMO, GPSR, DSDV, DSR, OLSR	🟩 Y	🟩 Appears to be feasible	🟩 Y	🟨 Free for non-commercial use	🟩 Commercial support
NetModeler	🟩 Many RF propagation models	🟨 802.11 (802.15.4, 6LoWPAN?)	🟩 Y	🟩 DSR, AODV, TORA, OLSR	🟩 Y	🟩 Appears to be feasible	🟨 Only RedHat	🟨 Proprietary	🟩 Commercial support

Table 3: Comparison of network simulators. Features are color-coded to indicate whether they completely satisfy (green), somewhat satisfy (yellow), or do not satisfy (red) our requirements for simulation of communiation among swarms of robots.

6 Simulation plan

Modifications to this section are found in the Appendix under Report Corrections.

The experimental phase of this project will be achieved by running multiple simulation episodes with different number of robots, communication protocols and cooperative strategies. Each simulation experiment requires the definition of the number of robots involved and its type (ground vehicles, fixed-wings, rotorcraft), the selection of communication parameters, and the ability to execute team's code.

To summarize the results from previous sections, we will use ns-3 to simulate communication, offering the following features:

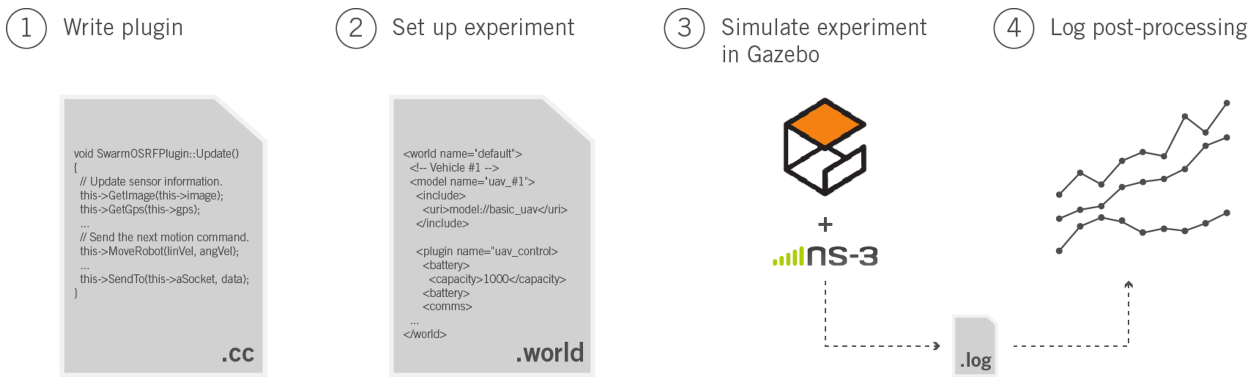
- antenna models;
- propagation and path loss models, including interactions with terrain;
- 802.11 and 802.15.4 physical and data link layers;
- MANET routing algorithms; and
- UDP transport.

Atop this stack of network simulation, developers of coordination algorithms will be presented with a communication API that supports sending and receiving unicast, multicast, and broadcast UDP messages. This interface will resemble the send / receive interface provided by many operating systems. Algorithm developers will also be able to configure the various layers of the stack, choosing antenna models, routing protocols, and so on to meet their needs. As the project progresses, we expect to reach consensus on the best choice for many of the options (e.g., antenna model), but based on their approach to coordination, teams will likely differ on their preference for some options (e.g., routing protocol).

Gazebo offers a flexible architecture for running different environments and configurations via SDF[9] world files. A world file is a test file that allows Gazebo to spawn models and to specify plugins that will modify model behavior in the simulated world. The teams will write these plugins using an API that OSRF will develop for this study. The plugins will allow the vehicle controllers to access their sensor information (images and GPS data), send motion commands, and send and receive messages.

Once the teams have the source code for their plugins ready, it will be possible to launch different experiments with different parameters without having to recompile code. For example, we can run one experiment with 50 ground vehicles, 30 fixed-wind vehicle, and 20 rotorcraft, then run again with a different population size and mix. Or we can run an experiment once using AODV for routing, then again using DSDV. All that will change between runs is settings in a configuration file. Each simulation episode will optionally log information that might be useful for experiment post-processing. Figure 4 shows the workflow that teams will use for conducting experiments during this study.

[9] `http://sdformat.org`



```
void SwarmOSRFPlugin::Update()
{
  // Update sensor information.
  this->GetImage(this->image);
  this->GetGps(this->gps);
  ...
  // Send the next motion command.
  this->MoveRobot(linVel, angVel);
  ...
  this->SendTo(this->aSocket, data);
}
```

```
<world name='default">
  <!-- Vehicle #1 -->
  <model name='uav_#1">
    <include>
      <uri>model://basic_uav</uri>
    </include>

    <plugin name="uav_control>
      <battery>
        <capacity>1000</capacity>
      <battery>
      <comms>
  ...
</world>
```

Figure 4: Swarm development process workflow

Internally, Gazebo will orchestrate the periodic execution of an `Update()` function on each model, the generation of sensor readings for each robot and the simulation of a time window. Additionally, Gazebo will trigger and pause the network event simulation during the same time window under simulation.

Next, you can find a draft of the C++ API that will be exposed to the teams to control the vehicles:

```
// Communication functions.
bool Bind(const std::string &_addr,
         const std::string &_port,
         void(*_cb)(const swarm::Socket &_socket),
         swarm::Socket &_newSocket)
bool SendTo(const swarm::Socket &_socket, const std::string &_data)
bood RecvFrom(const swarm::Socket &_socket, std::string &_data)

// Sensor functions.
bool GetImage(const swarm::Image &_img)
bool GetGPS(const swarm::Gps &_gps)

// Motion functions.
bool MoveRobot(const math::Vector3 &_linvel, const math::Vector3 &_angvel)

// Other functions.
std::string GetRobotType()
bool LogData(const std::string &_data)
```

Below you can find a snippet of a potential trivial version of a plugin that a team should write to control a vehicle:

```
/////////////////////////////////////////////////
bool SwarmOSRFPlugin::Load(physics::ModelPtr _model, sdf::ElementPtr _sdf)
{
  ...
  // Create a socket and register a callback for receiving data.
  this->Bind("192.168.0.2", 8000, &SwarmOSRFPlugin::OnDataReceived, this, this->aSocket);
  ...
}

/////////////////////////////////////////////////
void SwarmOSRFPlugin::Update(const common::UpdateInfo & /*_info*/)
{
  // Update sensor information.
  this->GetImage(this->image);
  this->GetGps(this->gps);

  // Do something useful with the sensor information.
  ...

  // Send the next motion command.
  this->MoveRobot(linVel, angVel);

  // Generate some data to send through my socket.
  std::string data;
  ...
  this->SendTo(this->aSocket, data);
}

//////////////////////////////////////////////////
void SwarmOSRFPlugin::OnDataReceived(const swarm::Socket &_socket)
{
  // Receive some data through my socket.
  std::string data;
  this->RecvFrom(this->aSocket, data);
  std::cout << "Data received: [" << data << "]" << std::endl;
  ...
}
```

Here you can find an example of how to load a world file and load the plugins needed to run a swarm simulation:

```
<world name="default">
  <!-- Spawn UAV #1 -->
  <model name="uav_#1">
    <include>
      <uri>model://basic_uav</uri>
    </include>

    <!-- Plugin for controlling the battery and communications in this robot -->
    <plugin name="vehicle_control" filename="libSwarmOSRFPlugin.so">
      <battery>
        <!-- Battery capacity in mAh. -->
        <capacity>1000</capacity>
      <battery>
      <comms>
        <physical>
          <data>
            <protocol>802.11</protocol>
            <network>
              <protocol>AODV</protocol>
              <ip>192.168.0.2</ip>
                <transport>
                  <port>8000</port>
                </transport>
            </network>
          </data>
        </physical>
      </comms>
    </plugin>
  <!-- Spawn the rest of vehicles-->
  ...

  <!-- Plugin for dealing with the communications among all robots -->
  <plugin name="swarm_comms" filename="libSwarmPlugin.so">
    <!-- Potential parameters -->
    ...
  </plugin>

</world>
```

Finally, a log playback component will be developed for Gazebo and the Swarm project. This piece of software will work in two phases. First, it will be able to intercept all the API calls to the swarm API, and will record them into disk. Every log entry will have a timestamp to capture the exact simulation time where the call was requested. In a second phase and after an experiment, a team will have the option of playing back an experiment from the log previously saved. In this case, the log entries will take place of the regular code executed in the `Update()` method of the team's plugin. Teams will be able to step forward, step back, pause or jump to a particular simulation instant for debugging or visualization purposes.

Appendix: Communication Report Review

On July 10th, 2015 the Swarm Simulation Communication Report was presented to a review board. The following changes to the swarm simulation communication strategy will be implemented based on the conclusions from this review.

RF Propagation

Fine details of RF propagation will be difficult to simulate. Instead of modeling RF propagation, or even the MAC layer above it, we will apply a gross statistical model that is bad enough. Real-world networks, especially large networks, have packet loss and network outages. A statistical model will be used to simulate an unreliable network.

The statistical model will consist of 10%-50% packet loss, periods of 100% packet loss, and will take into account physical obstacles. Terrain or buildings, which vehicles can not enter, that block line of sight between vehicles will result in no communication between those vehicles. Vegetation, such as trees, will increase packet loss.

Routing

The goal of this project is to explore the limits of swarm size, and determine when and how a swarm breaks down. MANET algorithms, which try to make a fully-connected network, are known to break down at about N=100. Using a MANET network would therefore limit the swarm size without exploring control limits or physical link limits.

The plausibility of deploying a simulated swarm in the real world, using existing radio technology, also must be maintained. In order to meet this constraint and avoid the MANET limit, a simple and plausible alternative will be used that consists of UDP packets with a nearest neighbor list. Communication between vehicles will use UDP, and each radio will have a list of one-hop neighbors. The set of one-hop neighbors determines who a given vehicle can communicate with. Any additional routing logic will be built by the swarm control teams. This strategy supports flexible swarm control and communication strategies, while being grounded in reality.

Report corrections

The following sections in the main body of the communication report have been superseded.

1. Section 2 summarizes the types of the robots and environment. This section now includes the loss of communication when line of sight does not exist due to terrain and buildings. Vegetation will also introduce packet loss.
2. Section 6 discusses the simulation plan, including the available communication methods. Based on the review, MANET has been replaced with UDP and a one-hop neighbor list. The propagation model will consist of 10%-50% packet loss, periods of 100% packet loss, and will take into account physical obstacles.