# SDfR -Service discovery for multi-robot systems

Stefan-Gabriel Chitic, Julien Ponge, Olivier Simonin

# SDfR - Service discovery for multi-robot systems

Stefan-Gabriel Chitic[1], Julien Ponge[1] and Olivier Simonin[1]
[1]*Université de Lyon, INSA-Lyon, CITI-INRIA,*
*F-69621, Villeurbanne, France*
{*stefan.chitic, julien.ponge, olivier.simonin*}*@insa-lyon.fr*

Keywords: Robotic fleet, Service Discovery, Service oriented Architecture.

Abstract: Multi-robots systems require dedicated tools and models for their design and the deployment. Our approach proposes service-oriented architecture that can simplify the development and deployment. In order to solve the problem of neighbors and service discovery in an ad-hoc network, the fleet robot needs a protocol that is able to constantly discover new robots in its coverage area. To this end we propose a robotic middleware, SDfR, that is able to provide service discovery. This protocol is an extension of the Simple Service Discovery Protocol (SSDP) used in Universal Plug and Play (UPnP) used in dynamic networks generated by the mobility of the robots. Even if SDfR is platform independent, we propose a ROS (ROS, 2014) integration in order to facilitate the use. We evaluate a series of overhead benchmarking across static and dynamic scenarios. We also present some use-cases where our proposal was successfully used.

## 1 INTRODUCTION

Nowadays more and more robotic systems tend to be composed of several robots moving and cooperating, generally called fleets of robots. They are able to perform one or multiple tasks together and to share information about complex missions.

We believe that the complexity of the internal communication inside the robot layer and external communication with other robots and the environment can be managed using appropriate family of middlewares.

One of the problems in developing multi-robot applications is the communication inside the fleet. The robots need to know the reachable peers in any type of communication infrastructure. In an ad-hoc network, this problem becomes challenging because the robots are highly mobile. A fleet can easily partition or merge with another fleet. Also single robots can become isolated or join an existing fleet.

Our vision is to propose a service oriented approach for robotic applications development because we think that they would benefit from the sand-boxing of processes as well as from being cross-platform and programming language agnostic. Existing middlewares like ROS, can be adapted to support a service orientated architecture where each node becomes a service.

There is a great need of having a suitable middleware that can abstract the neighbor and service discovery layer and offer high-level application interfaces (APIs) to robotic applications. The challenge is to provide an adapted system that can offer a suitable mechanism for network configuration. Second, this mechanism needs to maintain a list of connected peers and their services in a highly mobile scenario.

In this paper we propose a new service discovery middleware, called SDfR (Service Discovery for Robots). SDfR is able to make an auto-configuration for the connectivity to the fleet ad-hoc network, discover reachable neighbors and their services. We also provide a ROS integration node, simplifying the use of SDfR.

The paper is structured as follows: Section 2 presents the background of service orientated architecture and discovery in multi-robot systems, Section 3 discusses the service discovery approaches in middleware for robotics, Section 4 presents our proposal for service discovery in a fleet context, Section 5 presents the arhitecture and the implementation of SDfR, Section 6 presents the main results of overhead benchmarking. Section 7 presents the applicability of SDfR by describing a couple of use cases and Section 8 concludes the paper.

## 2 MOTIVATION

One problem of robotic fleets is how functionalities can be applied to make the collaboration between peers easier. Robots in fleet system that perform a mission together need to communicate with other peers. The communication can be done using a centralized node or directly using an ad-hoc network.

**Robotic applications as services** In our vision, robots need to advertise their functionalities as services in order to allow other members of the fleet to interact with them. In network based applications, service-oriented programming is now a largely accepted principle (Issarny et al., 2011).

Service-oriented architecture greatly simplifies the implementation of highly-adaptive, constantly-evolving applications (Frénot et al., 2010). It also reduces the process of developing and deploying new robotic applications. This architecture is very suitable to quickly cope with new developing models and requirements.

Services are the basic blocks of complex robotic behaviors and applications. This provides sandboxing for each software component which renders the robotic application more robust and tolerant to failure and still disposing of the flexibility in developing new components.

Services are platform independent and they can be described, discovered and composed dynamically. Having a service oriented architecture increases the ability to develop distributed software components in various programming languages and for heterogeneous target devices. In addition, higher levels of functionalities provided by service-oriented programming reduce the implementation of redundant software.

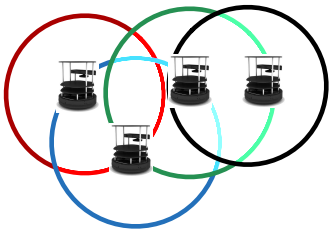

Figure 1: Ad-hoc fleet infrastructure with per-robot communication ranges.

In order to increase the mobility of the robots and to distribute the communication without having a central node, there is a need for the communication to be decentralized using ad-hoc networks. In this case, the robots do not have any image of their neighbors. Furthermore, the communication across peers is susceptible to route change and different peers can be used to relay a data package as shown in Figure 1. The ad-hoc network becomes the sum of peer to peer network with 2 robots at least.

**Fleet service discovery** In order to solve the problem of neighbors and service discovery in an ad-hoc network, a fleet robot needs a protocol that is able to constantly discover new robots in its coverage area, while maintaining a neighbor connectivity quality indicator. Since there is not any central node that can manage IP address allocation, the protocol should be able to negotiate an IP address inside the network and to have a conflict management tool in case of an IP collision. Our work focuses on ad-hoc multi-robot systems communication, especially on a family of middleware that accelerates the development and the deployment of new robotic applications.

## 3 RELATED WORK

In this section we present some of the major middlewares in robotics as well as some of existing service discovery protocols used in the field.

### 3.1 Middleware for robotics

All aspects of communication, application deployment and configuration can be facilitated using a proper middleware. The biggest difference between a classical middleware that runs in a cloud centralized infrastructure and a robotic one, is the mobility of the fleet and the decentralization of its components. An exhaustive survey of all the existing middlewares for robot contexts is clearly impossible because of the large number of existing middleware and frequent releases of new ones. A detailed survey can be found in (Chitic et al., 2014). Most of the middlewares are designed for mono-robot with little applicability in multi-robot systems.

One of the first middleware that emerged for robotics is Player/Stage (Kranz et al., 2006) project which is designed to provide an infrastructure, drivers and a collection of dynamically loaded device-shared libraries for robotic applications. It neither offers a service orientated architecture, nor a discovery for neighbors services system.

Another highly used middleware for robotics is ROS (*Robot operating system*). It is a recent flexible middleware for robot applications (Quigley et al., 2009; Cousins et al., 2010; ROS, 2014). It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot

behavior across a wide variety of robotic platforms. It provides hardware abstraction, device drivers, visualizers, message-passing, package management.

ROS (ROS, 2014) seems to be the emerging middleware for mono-robot with the most potential to become the most used framework for robotic fleets. However, it has no multi-robot coordination system and no automated testing environment, but it has already the advantage of having a large community that develops new packages for it.

To the best of our knowledge (Chitic et al., 2014), none of the existing middleware for robots provides a service discovery mechanism. ROS internal repository is represented as an internal node repository designed for internal service discovery, but it does not support neighbors discovery.

### 3.2 Service discovery and robotics

Classical protocols and middlewares for service discovery in distributed environments like data-grids, clouds or even smart environments have a *centralized* or *decentralized* registry that manages service description. *Decentralized* systems (e.g UPnP (Ahn et al., 2005), Jini (Pereira et al., 2011) or SLP (Romero et al., 2010)) can be *a purely distributed* solution where each node stores its own service repositories or a *hybrid* solution that includes super-nodes that aggregate information from other peers.

Another way to see a fleet of robots is like a service-oriented multi agent system. Such environments like Peer-to-Peer (P2P), Multi-Agent Systems (MAS) or Service-Oriented Environments (SOE) tend to approach the problem of service discovery in a *centralized*, *distributed* or *decentralized* way. *Centralized* mechanisms like super-peers (Gummadi et al., 2002), middle-agents (Klusch et al., 2006) or central registries (Rompothong and Senivongse, 2003) are limited in number of agents in the system and in terms of the number of requests. They also use a centralized node which can have serious impact if the central point becomes unreachable. *Distributed* approaches such as Distributed Hash Tables (DHT) (Maymounkov and Mazieres, 2002) offer more scalability and robustness by having multiple specific nodes that can manage the resources. *Decentralized* systems consider all the nodes to be equal. This approach provides more flexibility, but it has its downsides, since each node has only a partial view of the entire system. As mentioned in (del Val et al., 2014), an interesting way to discover service inside a decentralized and self-organized multi-agent system is to use homogeneity between agents.

In the robotic world, an approach for service discovery in centralized networks is to use classical Universal Plug and Play (UPnP) protocol. Since the concept of having the robotic tasks and processes as services is not mature yet, the main focus of research on service discovery in robotics is oriented towards the integration with the environment, like smart homes or smart cities. Reference (Borja et al., 2013) provides a case study of integration of service robots and smart-homes via UPnP. In these cases, the authors are not referring to a robot as part of a specific fleet, but as part of an environment, in which the robot is considered as an entity that can offer services.

The solutions presented above have their downsides when applied to ad-hoc multi-robot systems. Firstly, due to the mobility of the robots, the network connection is highly instable and robots can disconnect and reconnect very often. Existing protocols do not perform the same way in a highly dynamic environment and in a static one. As mentioned in (Issarny et al., 2011), the challenge is to set the trade-off between physical mobility and scalability. Secondly, existing protocols are not very adaptive. The discovery protocol should be ready to be used at any time and track its usage and failures. Existing protocols like UPnP, have a limited memory factor and when the connection is timed-out, the discovery process reinitializes itself at reconnection.

Next sections present our extension of SDP used in UPnP in order to overcome the limitation of existing service discovery approaches.

## 4 SERVICE DISCOVERY FOR ROBOTICS

Our main goal is to propose a mechanism that allows highly mobile robots to keep track of the reachable peers inside a fleet while using an ad-hoc infrastructure. This mechanism should also be able to provide a list of services available on each peer. Another objective is to propose a network configuration negotiation protocol. Due to the mobility of robots, classical peer to peer network configuration techniques are not suitable.

In this section we present the general description of our service discovery protocol for robotic applications, called SDfR (Service Discovery for Robots).

### 4.1 SDfR vs SSDP

We propose a protocol that does not flood the network and has an already seen memory feature build-in. SDfR protocol is a highly dynamic, adaptive and

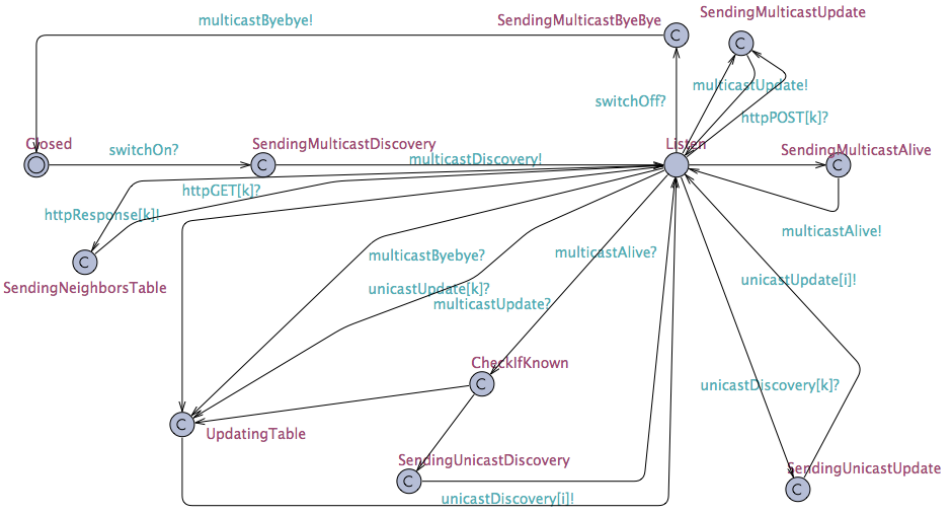


Figure 2: SDfR automaton.

scalable protocol adapted from Simple Service Discovery Protocol (SSDP) that is being used in UPnP. SDfR can be also used to provide service discovery with the smart-environment in which the robots are being deployed.

In order to limit the network use for the service discovery process, SDfR sends most of the internal messages in multicast, avoiding the overhead generated by unicast transmission in order to propagate the same message. In addition, to avoid failure in case of a disconnection due to the movement of the robots outside the coverage area, all the communications are done using UDP (User Datagram Protocol). Furthermore, to limit the network flooding when the protocol needs information from just one robot, a second transmission is enabled in unicast mode. SDfR does not need to reinitialize the entire discovery protocol when the connection is lost, because it disposes of a history map of all the already seen robots and their services.

In order to avoid services that are out of reach (e.g. service of robots that are present in the history map but are not present in the covered communication area), a connection indicator is computed for each robot represented by the success rate of pinging the connected peers.

SSDP and SDfR are similar because:

**Multicast transmissions** In order to avoid the overhead of retransmitting the same unicast message, most of the internal messages are multicast.

**HTTP-style messages** The messages that are being sent use an HTTP-style structure composed of headers and a body.

The main differences between SSDP and SDfR are:

**Limited multicast transmissions** To avoid failure in case of a disconnection due to the movement of the robots outside the coverage area, all the communications in SDfR are done using UDP (User Datagram Protocol) and only in request model.

**Unicast transmissions** To limit the network flooding when the protocol needs information from just one robot, a second transmission mechanism is enabled in unicast mode in SDfR.

**History map** SDfR does not need to reinitialize the entire discovery protocol when the connection is lost, because it disposes of a history map of all the already seen robots and their services. In order to avoid services that are out of reach (e.g service of robots that are present in the history map but are not present in the covered communication area), a connection indicator is computed for each robot.

## 4.2 Protocol Description

We design our protocol as a finite state timed automaton. A detailed description of the design process and the model validation is out of the scope of this paper. Figure 2 presents the automaton of our protocol.

SDfR protocol behavior is defined by the request method. Each method has at least one type of message that resides inside the request payload. Two methods representing the desired action of a request are used in SDfR : *M-SEARCH* and *NOTIFY*.

The *M-SEARCH* method is used for discovery requests to get the list of near by members and their services. The only message type associated with this method is *Discovery*.

The other method, *NOTIFY* is used to respond

to a *Discovery* request or to inform the others about changes in the current state of the robot. The message types associated with this method are: *Update*, *Alive* and *Byebye*.

- The *Update* message is sent as a response to a *Discovery* request or when the current services or capacities of the robot change.
- The *Alive* message is sent recurrently, as a beacon, in order to inform the others about the presence of the robot. The beacon sending rate can be set depending on the services need. The default value is at 10s.
- The *Byebye* message is sent when the robot stops gracefully, in order to inform the others about its disappearance.

When the protocol initializes, a discovery multicast message is sent, and then the protocol changes state into *listen* on a multicast as well as on an unicast socket. When the other robots receive a discovery message, they will respond with an update message.

When the protocol receives an update message, it passes into an atomic state, *Updating neighbors table*. When the protocol receives an alive message, it passes into *Check if known* state, that determines if the unicast IP of the sender is already known. If so, it changes into *Updating neighbors table*, otherwise it will send a unicast discovery message. The sender of the alive message responds by sending an unicast update message.

If the protocol traps a gracefully shutdown, a byebye message is sent and the other robots will update their neighbor table.

# 5 ARCHITECTURE AND BENEFITS OF SDFR

In this section we zoom in on the architecture and features embedded in SDfR by presenting its implementation as a service.

## 5.1 SDfR Architecture

Service Discovery for Robots is developed as a service itself. The service-oriented architecture approach for robotic software development is not very wildley used in the robotic community. The practice in this community is to develop built-in libraries in order to extend software features.

The main advantage of having a service-oriented architecture, as shown in Figure 3, is the compatibility

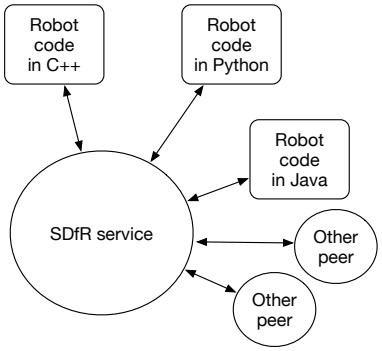


Figure 3: SoA in SDfR.

with other robotic services developed in different programming languages and running over different operating systems. This is a critical feature for a heterogeneous robotic fleet.

Furthermore, SDfR service can run separately of the other processes on the robot and all the messages are consumed by instances of the service on multiple robots. If it fails, it would not affect the other services running on the robot. This sand-boxing also ensures that the information sent by the protocol is not corrupted by any other third-parties.

SDfR service is composed of two layers as shown in Figure 4: an API layer that communicates with other services and a Discovery Protocol layer. Each layer has an independent life-cycle and communicates internally via a shared memory. The API layer responds to requests independently from the lower layer, using the information from the shared memory. The lower discovery protocol layer is in charge of communication with the other SDfR nodes on an elastic number of robots in order to discover the reachable peers and their services.

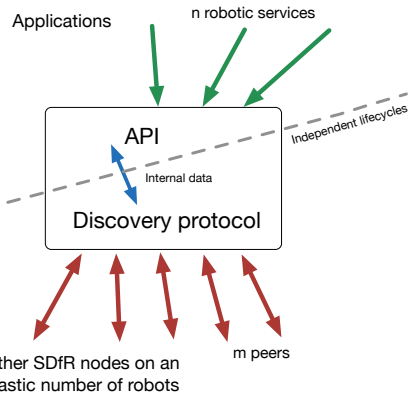


Figure 4: SDfR service architecture.

**RESTful communication API** A Representational State Transfer (Fielding, 2000) (RESTful) web-service was chosen for the API. It lets other services communicate with SDfR service because it is based on normal HTTP requests which enables intent to be inferred from the type of request being made and is completely stateless. All the responses are JSON messages.

**Implementation** The SDfR service is implemented in the 'Go' programming language (Pike, 2012) version 1.3.3. Go provides concurrent abstractions and safe memory management, something lacking in C/C++ and to a certain degree from Python. With 'Go', we can build all-in-one package that does not have any dependencies since the binary offers static linking for. Considering all the dependencies, the executable has still a small size in memory. Furthermore, we are able to build cross-platform executables which is an important aspect in deploying SDfR service across a heterogeneous platform of robots.

**Ad-hoc configuration management** Since the fleet is operating in an ad-hoc infrastructure, the peers need to be able to negotiate and auto-configure their network configuration. A robotic fleet ad-hoc network is different from a classic ad-hoc hot-spot because the robots are highly dynamic and the network can be partitioned or merged. The mobility of the peers needs to be taken into consideration in the negotiation protocol of the configuration. SDfR service, based on a simple configuration file, is able to automatically connect to an ad-hoc network. The secured WiFi SSID network is composed using the fleet id. This mechanism allows to have multiple fleets of robots in the same network space. Moreover, the robots can auto assign IP addresses. The standard network space is *10.<fleet id>.<x>.<y>*, where x and y are computed by each robot from its internal MAC address in order to avoid IP conflict (Thomson et al., 2007). Furthermore, if an IP conflict happens, the service has a mechanism to trigger an IP change on the robots. This mechanism is available all the time since an IP conflict can be triggered by a merge of 2 subnetworks.

## 5.2 ROS integration

In order to make SDfR service user friendly, we have created a ROS node that communicates with SDfR service and can be used by other nodes via topics and services. When the node starts, it launches a instance of SDfR if it is not running and then it provides support for other ROS nodes to publish or unpublish their services and their capacities. Furthermore, our ROS node provides the neighbors list of services and is capable of allowing other ROS applications to search for a specific service with a specific configuration.

A producer node can publish its services or capacities in an asynchronous way using ROS topics because we consider that the registration is not highly bound to time. The same concept applies for the unpublish commands and for getting the list of neighbors. On the other hand, the search command for a specific neighbor and their services needs to be done in synchronous way using ROS services because the behaviors of the consumer node is depending on it.

# 6 EXPERIMENTS AND OVERHEAD EVALUATION

In the evaluation we want to measure what is the overload generated by the use of SDfR in a robotic fleet context. Another objective is to see the impact of using text-plain protocol in the upper and the lower layer of the SDfR service. In this section we present the main results of resource overhead for SDfR.

Figure 5: Turtlebots in the experimentation hall of CITI-Lab.

SDfR was benchmarked in two types of contexts:

- a static scenario where the robots do not move to evaluate the overhead in an ideal Wi-Fi communication scheme
- a real dynamic scenario (see Figure 5) where robots are moving and transmission can drop.

In both scenarios all peers should discover their neighbors and in the the second one, the neighbors discovery depends on the distance between peers.

The benchmarks were performed on Turtlebot 2 robots equipped with an Intel Core 2 Duo, 2.1 GHz CPU, 4Gb of Ram PC running on Ubuntu 13.04

Each test run was given 5 minutes to collect the data. In our test runs we use simulated services that try to register/subscribe into SDfR. We simulate three type of actions:

1. **Publish.** We simulate new service providers that try to *publish* with a delay time of 10 seconds. In order to simulate publishers, we use an Apache

server on each robot that responds to the auto-discovery URL of each publisher.

2. **Unpublish.** Each of the already published service providers could be *unpublished* with a random delay between 5 seconds.
3. **Subscribe.** We generate separated threads for each *consumer*. Each thread will constantly request the table of neighbors from SDfR, in order to stress at maximum our protocol.

In the static scenario we considered different numbers of robots: 2, 4, 6. Each robot had a total number of service providers and service consumers equal to 100 simulated services. For each number of robots we used different ratios between providers and consumers: 30%, 50% and 70% publishers.

In the dynamic test-case we used for each variant of SDfR 6 robots with a number of 100 simulated services per robot. 70% of the services on each robot were publishers. The robots moved randomly in a 200 square meters room with poles and other obstacles. The room (see Figure 5) was exposed to Wi-Fi interferences from other networks that occupy all of the 2.4Ghz channels.

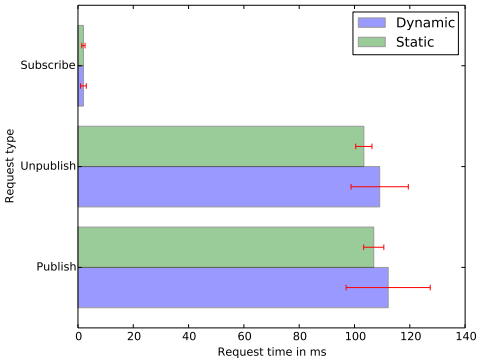


Figure 6: Average request time in SDfR.

A robotic application that provides a service for the fleet needs to register with SDfR. This must be done as fast as possible in order to avoid blocking the service when it starts. Another important overhead measure is the time of unregister request. This happens whenever a producer wants to remove itself from the SDfR registry. One of the most important metric for robotics application from a latency point of view is the time to request the list of neighbors and their services. In a real scenario, a producer registers once for its life-time cycle, but a consumer may request multiple times the list of reachable neighbors and services. The subscription response time can have a impact on the time to complete a fleet task.

Figure 6 reveals the resulting time consumed by a service to publish or unpublish its service or get the neighbors services in the static and dynamic scenarios. In the dynamic scenario (Figure 6), the latency is higher but the difference from the static scenario is less than 30%. This can be explained by the increase in computation load on robots generated by the mobility of the fleet. All the request times remain in a certain variation interval even if the number of robots increases. The publish request takes from 95ms to 130ms, the unpublish request varies between 98ms to 122ms and the subscribe request is less than 5ms.

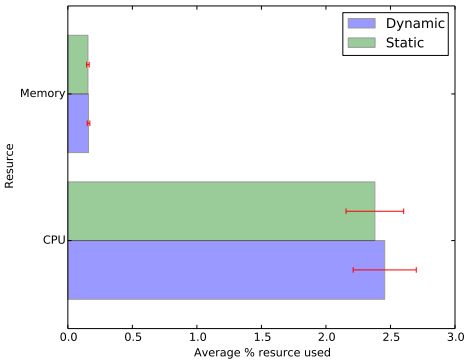


Figure 7: CPU and Memory usage.

In robotic applications, the computation power is critical. Fleet of robots are heterogeneous and can include different types of robots with different computational factor. Having a low CPU consumption discovery service, benefits the other processes involved in performing the fleet mission. Besides the CPU usage, another critical resource in fleets of robots is the memory. As an example, robotic fleets may include visual sensors like 3D cameras which are in high demand of memory. A service discovery protocol needs to have a low usage of the robot memory.

Figure 7 presents our results for % of CPU and Memory used. CPU consumption varies between 2.2% to 2.8% of CPU usage. We observe that the CPU usage is higher for the dynamic scenario because the CPU time is consumed by the mobility management and the number of CPU slots is less for other processes. We can also say that the CPU usage increases with the number of robots in the fleet with a rate of maximum 1% per robot. The memory usage is less than 0.3% (out of 4GB) and seems to remain constant even if the number of robots increases.

In a fleet context the communication between peers in ad-hoc network are very sensitive. The transmissions can be unreliable due to the mobility of the robots. This is why the network overhead needs to be as limited as possible in order to allow services to

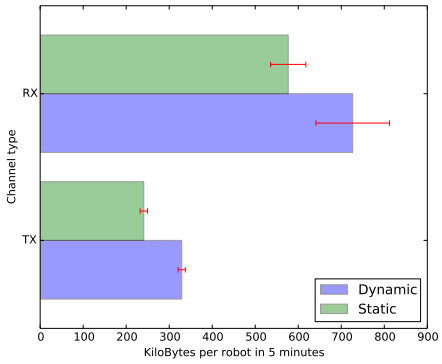


Figure 8: Transmitted and Received bytes per robot in 5 min.

exchange information. Our measurements include the average of transmitted and received bytes per robot in a test run of 5 minutes. To increase the quality of our measurements we use intermediary check-points for each metric at each 10 seconds.

Figure 8 presents the quantity of transmitted (TX) and received (RX) kilobytes per robot. The number of bytes varies from 220 kilobytes to 320 kilobytes for the transmitted bytes and from 540 kilobytes to 850 kilobytes for received bytes. This remains very limited considering the time of 5 minutes.

**Overhead conclusion** Our results show that SDfR is low latency brick, with a small overhead on CPU and memory that has a low network consumption. Moreover, the increase in number of robots in a fleet do not have a big impact on SDfR overhead. Being integrated with ROS makes SDfR a good asset in managing neighbors and service in multi-robot context.

# 7 USE-CASES

This section presents use-cases where SDfR and service oriented architecture were applied. These two use-cases are being studied in an ongoing project: 'CROME - Multi-view multi-robot scene understanding and fleet coordination' (Matignon et al., 2014). But the applicability of SoA and SDfR is larger than the examples presented below.

In general, we can use services in any robotic application that need to perform one or multiple tasks at once by having multiple nodes that manage different pars of hardware layers. In multi-robot context, SDfR can be used to simply discover the peers in our communication zone and avoid overloading the network with broadcasts or messages for unreachable robots. Furthermore SDfR provides a list of available services on each peer, thus a robot can decide if it can relay on a peer to perform together a task.

## 7.1 Peer-to-Peer monitoring in a fleet

This scenario consists in having a fleet of robots performing a mission in a decentralized network. The environment is big enough to make direct communication between all peers impossible.

In these condition, a monitor station (see Figure 9) cannot have a global image of what is happening in the fleet. In order to provide the monitor with information about each robot status, status messages and alerts need to be routed in a peer to peer context to the monitoring station.

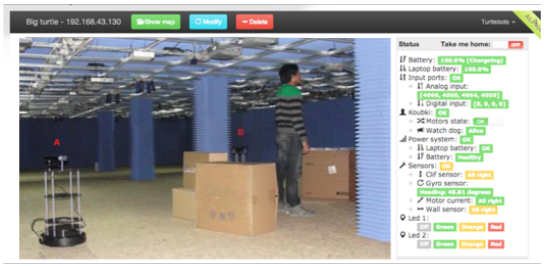

Figure 9: Peer 2 peer monitoring interface.

In this case, on each robot inside the fleet, services are represented by the nodes that manage the hardware elements of each robot and that expose their status. In addition, a peer to peer monitoring service is present. It collects the data from other nodes and analyzes it in order to trigger alarms if necessary.

SDfR is used for all the hardware nodes to register in order to be discovered by the monitoring service. The monitoring service first registers itself into SDfR and then it gets constantly the lists of available service on the robot. In addition, this service uses SDfR to discover the peers that are in the communication area.

The peer-to-peer monitoring service searches for the neighbors with an instance of this same service. If it finds one, it routes in a state-less connection all the data from its nodes to the neighbor. It also tries to build a communication route for alert to get in a state-full environment to the monitoring station.

## 7.2 Complex scene observation by robotic fleet

This scenario consists in having a fleet of robots observing a complex scene using multiple sensors in order to have a better observation. Combing the perception of each robot, the fleet has a better vision of the

scene. The robots need to move in order to maximize the observation of the scene. In a particular case, the fleet observe the pose of a human and his joints. Maximizing this means having the highest number of articulations visible by the fleet(Matignon et al., 2014).

In this case, on each robot we have a node that gets the sensor perception, a node that manages the movement of the robot and a upper AI layer service that decides if the robot needs to move in order to maximize the vision.

SDfR is used by the AI layer to discover the movement node and the perception node. Furthermore, SDfR is used to search for the other reachable peers in the fleet. The AI layer searches for similar instances in other peers in order to perform a group decision. Using SDfR allows an elastic number of robots to participate in the fleet in this type of mission.

# 8 PERSPECTIVES AND CONCLUSION

In this paper we presented the challenges to make a service discovery protocol for robot fleet systems. We discussed the limited applicability of existing service discovery protocols in the context of robot fleets and then, we proposed a new protocol called SDfR that is suitable for service discovery inside an ad-hoc networked fleet.

The evaluation results shows that SDfR API generates a small latency and the SDfR service has a small impact on the CPU and memory used. Furthermore, SDfR has a low bandwidth consumption in both static and dynamic scenarios. Base on our benchmarking, we believe that SDfR can provide neighbor and service discovery for multi-robot applications without stressing the system. ROS integration and ad-hoc network auto-connect features facilitate the usage of SDfR.

We intend to further use SDfR in our current approach of improving the families of middleware to facilitate the development of multi-robot systems. SDfR can be used as part of bigger systems like remote application deployment environment in ad-hoc robotic network where robots can be dynamically provisioned with new applications while remaining in operation environment. SDfR can also be used to monitor the running services on each robot. We are also interested in opportunities for SDfR not just for robotics but also in other fields like mobile wireless sensor networks.

# REFERENCES


Ahn, S. C., Kim, J. H., Lim, K., Ko, H., Kwon, Y.-M., and Kim, H.-G. (2005). Upnp approach for robot middleware. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1959–1963. IEEE.

Borja, R., de la Pinta, J., Álvarez, A., and Maestre, J. (2013). Integration of service robots in the smart home by means of UPnP: A surveillance robot case study. *Robotics and Autonomous Systems*, 61(2):153 – 160.

Chitic, S., Ponge, J., and Simonin, O. (2014). Are middlewares ready for multi-robots systems? In *Simulation, Modeling, and Programming for Autonomous Robots - 4th International Conference, SIMPAR 2014, Bergamo, Italy, October 20-23, 2014. Proceedings*, pages 279–290.

Cousins, S., Gerkey, B., Conley, K., and Garage, W. (2010). Sharing software with ros [ros topics]. *Robotics & Automation Magazine, IEEE*, 17(2):12–14.

del Val, E., Rebollo, M., and Botti, V. (2014). Enhancing decentralized service discovery in open service-oriented multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 28(1):1–30.

Fielding, R. (2000). Representational state transfer. *Architectural Styles and the Design of Netowork-based Software Architecture*, pages 76–85.

Frénot, S., Le Mouël, F., Ponge, J., and Salagnac, G. (2010). Various Extensions for the Ambient OSGi framework. In *Adamus Workshop in ICPS*, Berlin, Allemagne.

Gummadi, P. K., Saroiu, S., and Gribble, S. D. (2002). A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. *ACM SIGCOMM Computer Communication Review*, 32(1):82–82.

Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadist, P., Autili, M., Gerosa, M. A., and Hamida, A. B. (2011). Service-oriented middleware for the future internet: state of the art and research directions. *Journal of Internet Services and Applications*, 2(1):23–45.

Klusch, M., Fries, B., and Sycara, K. (2006). Automated semantic web service discovery with owls-mx. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922. ACM.

Kranz, M., Rusu, R. B., Maldonado, A., Beetz, M., and Schmidt, A. (2006). A player/stage system for context-aware intelligent environments.

Matignon, L., Chitic, S., and Simonin, O. (2014). Cooperating in a group of mobile robots to identify the human skeleton model. http://youtu.be/5kmxIgLEerQ.

Maymounkov, P. and Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer.

Pereira, A., Costa, N., and Serôdio, C. (2011). Peer-to-peer Jini for truly service-oriented WSNs. *International Journal of Distributed Sensor Networks*, 2011.

Pike, R. (2012). Go at google. In *Proceedings of the 3rd Annual Conference on Systems, Programming, and*

*Applications: Software for Humanity*, SPLASH '12, pages 5–6, New York, NY, USA. ACM.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3.

Romero, D., Rouvoy, R., Seinturier, L., and Carton, P. (2010). Service discovery in ubiquitous feedback control loops. In *Distributed Applications and Interoperable Systems*, pages 112–125. Springer.

Rompothong, P. and Senivongse, T. (2003). A query federation of uddi registries. In *Proceedings of the 1st international symposium on Information and communication technologies*, pages 561–566. Trinity College Dublin.

ROS (2014). Robot operating system. http://www.ros.org/.

Thomson, S., Narten, T., and Jinmei, T. (2007). Ipv6 stateless address autoconfiguration. IETF RFC 4862.