Noname manuscript No.
(will be inserted by the editor)

SOUL: Data Sharing for Robot Swarms

Vivek Shankar Varadharajan · David St Onge · Bram Adams · Giovanni Beltrame

Swarm robotics

Swarm robotics is an approach to the coordination of multiple robots as a system which consist of large numbers of mostly simple physical robots. It is supposed that a desired collective behavior emerges from the interactions between the robots and interactions of robots with the environment. Wikipedia

Received: date / Accepted: date

Abstract Interconnected devices and mobile robotic systems are increasingly present in many real-life scenarios. To be effective, these systems need to collect large amounts of data from their environment, and often these data need to be aggregated, shared, and distributed. Generally, multi-robot systems share state information and commands through a communication channel, but this is often too limited in many contexts (e.g. sharing large 3D maps). This paper introduces SOUL (Swarm-Oriented Upload of Large data), a mechanism that allows members of a fully distributed system to share large amounts of data with their peers. We leverage a BitTorrent-like strategy to share data in smaller chunks, or datagrams, with policies and bidding strategies that minimize reconstruction time. We performed extensive simulations to study the properties of the system and to demonstrate its scalability. We also report experiments on real robots with two realistic deployment scenarios: searching for objects in a scene, and replacing the whole identity of a defective robot.

Keywords Swarm robotics · Information sharing · Stigmergy · Multi-robot systems

Swarm intelligence

From Wikipedia, the free encyclopedia

Swarm intelligence (**SI**) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence. The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems.[1]

SI systems consist typically of a population of simple agents or boids interacting locally with one another and with their environment. The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents. Examples of swarm intelligence in natural systems include ant colonies, bird flocking, hawks hunting, animal herding, bacterial growth, fish schooling and microbial intelligence.

The application of swarm principles to robots is called *swarm robotics* while *swarm intelligence* refers to the more general set of algorithms. *Swarm prediction* has been used in the context of forecasting problems. Similar approaches to those proposed for swarm robotics are considered for genetically modified organisms in synthetic collective intelligence.[2]

1 Introduction

A robotic swarm is an inherently parallel, distributed system, where robots act independently and perform numerous physical tasks at the same time. Ideally, a robotic swarm should be greater than the sum of its parts, and able to solve problems that no robot of the same complexity could accomplish alone. A typical example is search and rescue missions, where a large area has to be covered: a single Unmanned Aerial Vehicle (UAV) needs extended autonomy and substantial processing capacity to analyze images of the terrain, but distributing the task over many UAVs requires fewer resources from each robot.

The advantages of a parallel, distributed system, can be even more conspicuous when considering a heterogeneous swarm. As an example, since payload is limited on flying platforms, a swarm can be partitioned in sub-groups with special attributes: one equipped with cameras, one with more processing and memory resources, and one without any payload, for longer flight time and covering longer distances. However, these specialized sub-swarms generate a new challenge: to manage large data transfers between them.

Concretely, many multi-robot applications often require heavy processing and sensors acquisition to fulfill their task. Mapping a region after a disaster from aerial photography is still mostly executed on ground stations, and can use large 4k images [Lliffe (2016)]. Scanning a damaged building can more easily be managed on board, but using a state-of-the-art RGB-D sensor and GPU-optimized algorithms [Surmann et al. (2017), Vempati et al. (2017)]. As for victims detection, various strategies use high resolution images, thermal sensors and powerful cellular antennas, together with computer-intensive learning algorithms to

Authors' accepted manuscript
Article published in *Autonomous Robots* (2019)
https://doi.org/10.1007/s10514-019-09855-2

The final publication is available at link.springer.com

adapt to the field conditions [Aguilar et al. (2017), Erdelj and Natalizio (2016)].

Furthermore, in critical missions such as emergency response, swarm intelligence has the great advantage of being able to cope with robot failures, but not without a loss of efficiency. However, if a fault detection mechanism is available, as simple as battery monitoring, each agent can warn the others about its possibly imminent failure and upload all of its local data and state variables to the swarm. With this last resort backup of the failed robots' identity and mission status, another agent can seamlessly replace it.

To address the data sharing requirements of these application scenarios we developed a strategy for the *Swarm-Oriented Upload of Large data* (**SOUL**). The main challenges of SOUL were to: 1- cope with dynamic network topologies, 2- optimize the data fragmentation and reconstruction, and 3- optimize the distribution of the datagrams (chunks of the injected data).

Peer-to-peer (P2P) file sharing mechanisms are well established in literature, with ample research to demonstrate their robustness and scalability [Reid (2015)]. SOUL leverages some of the strategies used in P2P networks (e.g. with the use of distributed hash tables), but integrates concepts of swarm intelligence to be optimized for heterogeneous swarms with multiple tasks.

In previous works [Pinciroli et al. (2016), St-Onge et al. (2018)], we developed strategies to reach consensus on environmental data or swarm-wide state variables. We showed that our mechanism (dubbed Virtual Stigmergy, and inspired by biological swarms) was robust to heavy packet loss, scalable, and greatly enhanced the robustness of a fleet when allocating a set of tasks. This mechanism, however, was suitable only for variables or small data structures. In this paper, we extend the Virtual Stigmergy to share, and reach consensus on, large data blocks.

In the following, we describe the research work that inspired us (Sec. 2), and then detail the SOUL model (Sec. 3). From the model we derive an implementation for the Buzz programming language [Pinciroli and Beltrame (2016b)] (Sec. 4) and discuss its performance in simulation (Sec. 5). Finally, we present two realistic experiments for the use of SOUL, executed on a swarm of wheeled robots (Sec. 6).

2 Related work

Swarm intelligence can provide both new challenges and new strategies for file sharing [Dhurandher et al. (2009)]. This section gives a brief overview of the related research that inspired SOUL, either in terms of application context (swarm tasks and backup mechanisms) or for their implementation strategy (P2P, consensus and bidding).

P2P

HTTP is unavoidably the most popular distributed query solution used for databases, but it still is a one-way client-server approach that does not exploit the number of units available in a swarm [Benet (2014)]. However, as in HTTP/1.0, SOUL manages each request in a separate connection. More recent P2P solutions targeting large files are more suitable for the distribution of large files over a network [Reid (2015)]. The fragmentation, distribution, and reconstruction of the data injected in SOUL is greatly inspired from the approach of the InterPlanetary File System (IPFS) [Benet (2014)]. In fact, distributed hash tables and torrent exchanges are the building blocks of our system, as for many P2P database solutions [Vu et al. (2010)]. However, as opposed to most web-based applications of P2P file sharing mechanisms [Ganesan et al. (2004)], in a robotic swarm we expect the up- and download times to be equivalent, and we assume that the swarm is cooperative (i.e. no agent is greedy in terms of bandwidth usage), thus removing the need for a fair compensation [Garbacki et al. (2006)].

Consensus

Consensus-based approaches have been proposed for a number of multi-UAV coordination problems such as resource and task allocation [Brunet et al. (2008)], formation control [Ren et al. (2005), Kuriki and Namerikawa (2015)], and determination of coordination variables [Wei et al. (2015)]. However, these approaches are specific and need to be implemented again in each new swarm architecture. The swarm-oriented programming language Buzz allows for an easier manipulation of the consensus strategy [Pinciroli and Beltrame (2016b)]. The work of [Davis et al. (2016)] proposed a shared table to reach consensus on the state of a swarm of UAVs. Their approach uses a single list to which each robot appends its own value, and the entries on the robots get updated in a query-and-response fashion. However, the approach in [Davis et al. (2016)] was not meant to cope with large files and does not support revising the appended values. Similarly, SOUL does not yet provide an update mechanism for the shared data, but earlier work made that possible for state variables, based on the use of Lamport Clocks [St-Onge et al. (2018)].

Auction-Based Task Allocation

Auction-based allocation is a popular solution to task allocation problems in robotic swarms, covering centralized, decentralized, and hybrid approaches [Zhang et al. (2009)]. Our solution is decentralized, but as described in [Zhang et al. (2009)], it

uses opportunistic centralization to locally manage each auction related to a new data blob injected in the network. Market-based bidding has found its use in software-agent research, with sophisticated algorithms for winner determination [Sandholm et al. (2000)]. Computer scientists working in the area generally refer to an auctioneer looking to *sell* the available tasks at the highest price, with various definition of cost, sometimes including a measure of the tasks' utility [Vig and Adams (2006)]. SOUL reverses the problem: it is no longer a mean to maximize the auctioneer revenue, but rather to find the lowest-priced subcontractor for each available job (e.g. to store a datagrams). The cost submitted by subcontractors consider a set of their available resources, as consideredin the work of [Lee (2018)] for a typical *buyer* strategy.

Hierarchical tasks for swarms

While most works focus on homogeneous groups of robots, SOUL targets heterogeneous swarms, with the goal of using the full capabilities of each robot. However, as bids often includes a resource vector in their computation [Lee (2018)], they are already well-fitted for heterogeneous scenarios. Somewhat along the path of cost computation in the bidding mechanism, one can use decision networks to attribute the tasks following each robot utility calculation [Sen and Adams (2013)]. We consider three hierarchical types of task: 1- the global swarm task, 2- the sub-swarm tasks, and 3- the individual tasks. For instance, in a search and rescue mission, the overall mission is the task assignment to the swarm, and one sub-swarm maybe be assigned an exploration task, within which some individuals are tasked to take pictures of particular hotspots. This view is oriented top-down, as opposed to the classical swarm robotics task implementation [Bayindir (2016)].

Backup and Update

[Brown and Sreenan (2013)] state that distributed systems are complex to update safely while in operation, partly because when a failure is detected, a unit must have access to a backup image to resume. The resource constraints of swarm robotics, and also those of Wireless Sensor Networks (WSN), render the availability of large backups difficult to implement.

In [Kamra et al. (2006)], the authors proposed a mechanism to maximize data recovery in WSNs deployed in hazardous environments with high probability of failure: they propose methods to combine data collected by neighboring nodes with the node's data using a set of logical operations, and to store them in the place of the nodes data. This allows the retrieval of data from failing nodes. However, this approach was not meant for large data blobs. Nevertheless, a similar system could be applied within SOUL to increase data persistence.

Large data sharing mechanisms are also mandatory for software updates during missions, in continuous integration. Indeed, to optimize the software integration workflow, "hidden commits" is a common strategy that generates large patches to be transferred over the network, which can be a serious issue for continuous integration mechanisms [Laukkanen et al. (2017)]. In fact, this work was partly inspired from the need of large data transfers within an update system proposed in one of our previous works [Varadharajan et al. (2018)]. Where we propose a systematic approach to perform Over-The-Air updates for robotic swarms during a mission. This work paves the way to extending the update system, by providing a solution to both hidden commits and backups.

3 SOUL model

The *Swarm-Oriented Upload of Large data* (**SOUL**) mechanism is a collection of discrete policies and bidding algorithms that adapt to the swarm's network topology. Each time a robot shares a file, referred to as a blob, this file is split into series of smaller chunks of data spread throughout the swarm. The problem consists of attributing each datagram to the right member of the swarm, or:

> Given a number of blobs $\{B_1,B_2,...,B_k\}$, a team of robots $\{R_1,R_2,...,R_n\}$, a function $C(B_i,R_j)$ that specifies the cost of storing datagrams of blob B_i on robot R_j, find the assignment that minimizes the amount of time consumed t_j to reconstruct a blob from a given robot j over the swarm.

3.1 General definition

While blobs can be continuously injected in the swarm network, each robot $r \in \{R_1, R_2, ..., R_n\}$ has limited storage s^r. Fig. 1 illustrates a possible distribution of three consecutive blobs with a representation of the robots' storage limit. The rectangular storage box above the robots indicates their storage occupancy, with colors and labels indicating where each blob is allocated. At a given time step t_1, R_1 injects a blob tuple $\langle 2,\text{blob}\rangle$ in the network. This blob gets allocated to R_1 ($a(2, b_2^1)$) and R_3 ($a(2, b_2^3)$) at the following time step t_2, and their storage gets occupied accordingly. A similar effect happens during the injection of blob 3 and 4, initiated by R_3 and R_1 respectively.

Definition 1 Given a set of robots $\mathscr{R} = \{R_1, R_2, ..., R_n\}$ with $s^r = s_l \ \forall r \in \mathscr{R}$ and a set

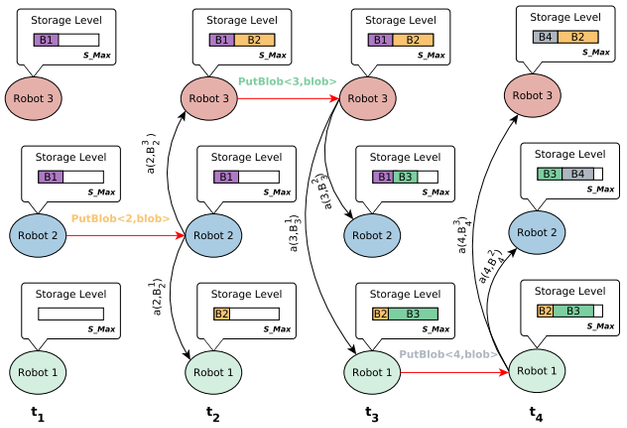


Fig. 1: Illustration of three consecutive blob sharing with a network of three robots.

of blobs $\mathscr{B} = \{B_1, B_2, ..., B_k\}$, a function $f : \mathscr{B} \rightarrow \mathscr{R}$ exists such that

$$\min \sum_{b \in \mathscr{B}} C_b^r(S, d_b), \quad \forall r \in \mathscr{R}, \tag{1}$$

where $C_r^b(S, d_b)$ is the cost of reconstructing blob b on robot r knowing the resource state vector $S = \{s_1, s_2, ..., s_n\}$ of all the robots in the network. The resource state of a robot is a generic vector that includes all characterstics relevant to the auction, such as battery level and storage space. $d_b = \{d_1, d_2, ..., d_l\}$ is the set of blob b datagrams, all of the same size, and $|d_b| = N_{c_b}$ is the number of datagrams of blob b.

This general description implies that the distribution of the datagrams should minimize the reconstruction cost (time taken to reconstruct) on all robots. Two factors influence this cost: the level of dispersion of the blob, and the proximity of the robots holding the datagrams (in terms of network topology). It is worth noting that this allocation mechanism considers that all members have identical priority access to each injected blob.

3.2 Heterogeneous Swarms

A heterogeneous group of robots is defined by the diversity of each of its members' abilities. While identical units are useful for redundancy and scale, the wide range of required sensing modalities to deploy an omnipotent robot makes it cumbersome and expensive. In a swarm, sub-groups or sub-swarms can have different specializations and still be managed as a whole in a distributed fashion, as widely demonstrated in nature's insect swarms [Kelley and Ouellette (2013)] and robot swarms with Swarm-Oriented programming [Pinciroli and Beltrame (2016b)]. For SOUL, one can think of a set of robots providing inputs, i.e. injecting the blobs, a set of robots with more processing power and memory to parse and process the data, and a generic set of robots maintaining the connectivity of the whole swarm. Consider these three sets of robots: capturers $\mathscr{C}$, processors $\mathscr{P}$, and networkers $\mathscr{N}$, such that $\mathscr{P} \cup \mathscr{C} \cup \mathscr{N} = \mathscr{R}$ and $|\mathscr{R}| = n$, with these sets in hand definition 1 can be reformulated as:

Definition 2 Given three sets of robots $\mathscr{P} = \{p_1, p_2, ..., p_u\}$, $\mathscr{C} = \{c_1, c_2, ..., c_v\}$ and $\mathscr{N} = \{n_1, n_2, ..., n_w\}$ with $s^r = s_l, \forall r \in \mathscr{N} \cup \mathscr{P}$ and a set of blobs $\mathscr{B} = \{B_1, B_2, ..., B_k\}$ injected from any $c_i \in \mathscr{C}$, there exists some $f : \mathscr{B} \rightarrow \mathscr{N} \cup \mathscr{P}$ such that

$$\min \sum_{b \in \mathscr{B}} C_b^p(S, d_b), \quad \forall p \in \mathscr{P} \tag{2}$$

where $C_b^p(S, d_b)$ is the cost of reconstructing a blob b on robot p, subject to the allocation function

$$\sum_{r \in \mathscr{N} \cup \mathscr{P}} a(b, A^r) \geq 1, \quad \forall b \in \mathscr{B}, \tag{3}$$

where A^r is the blob list of robot r, i.e. a list of all blobs allocated to robot r. The inequality of Eq. 3 indicates that a blob can be allocated to more than one robot in the swarm. Any such allocation mechanism generates a distribution of the blobs on the set of $r \in \mathscr{N} \cup \mathscr{P}$ robots that minimize the reconstruction on the processors. From the point-of-view of robot r, the vector of blob b datagrams it holds is q_b^r, subject to:

$$\sum_{r \in \mathscr{N} \cup \mathscr{P}} |q_b^r| = N_{c_b}, \quad \forall b \in \mathscr{B}, \tag{4}$$

with N_{c_b}, the number of datagrams for blob b, and

$$\sum_{b \in \mathscr{B}} |q_b^r| \leq s_l^r, \quad \forall r \in \mathscr{N} \cup \mathscr{P}. \tag{5}$$

The first constraint, Eq. 4, means that the sum of all the datagrams for a given blob b distributed on processors and networkers is exactly the number of datagrams of this blob. In order words, all the datagrams must be allocated, and a datagram can be allocated to only one robot. For the sake of simplicity, despite the fact that the datagrams could be replicated, we assume resource-constrained robots and we do not consider this level of redundancy in this work. The second constraint, Eq. 5, sets the maximum size of datagrams vector held by a robot as its storage capacity.
The objective of SOUL is to minimize the cost of reconstructing (Eq. 2) a blob b for all robots $p \in \mathscr{P}$, with S and d^b the storage vector of all robots and the datagram vector of blob b, respectively.

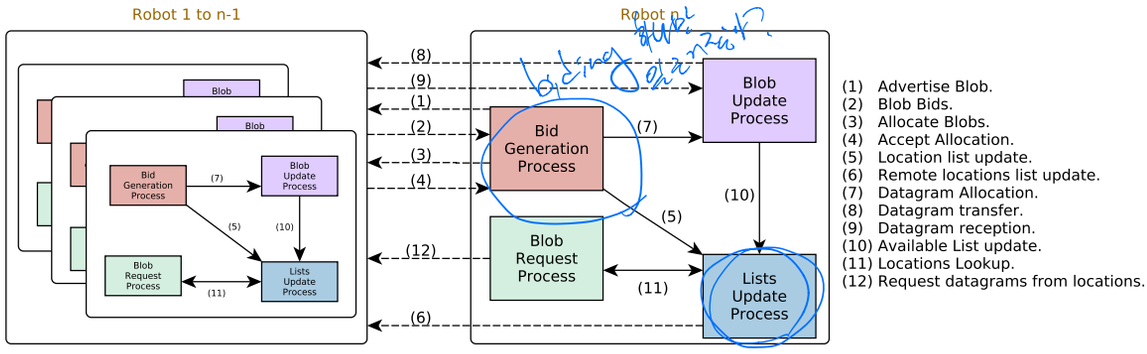


Fig. 2: The four processes and twelve message types involved in the SOUL mechanism.

3.3 Reconstruction cost

The function to be minimized (Eq. 2) is used in a decentralized auction-based algorithm to allocate each blob to the best (i.e. lowest cost) candidate. The proposed solution consists of an auctioneer, which determines how to distribute a blob's datagrams in the network to minimize the cost of reconstruction at robot $p \in \mathscr{P}$. Each robot can bid on the blob with its resource state vector s^r together with a vector of its proximity to the processors x^r. The auctioneer takes the bids from all the robots and computes the reconstruction cost. One can define this process as a market auction soliciting from a set of subcontractors to achieve the lowest price.

Definition 3 Given an auctioneer with a set of datagrams generated from a new blob b, $d_b = \{d_1, d_2, ..., d_k\}$, which require to be allocated for storage, and a group of subcontractors submitting service proposals, $E = \{e^1, e^2, ..., e^n\}$: A proposal is a triplet $e^j = \langle b^j, s^j, x^j \rangle$, where $b^j \in \mathscr{B}$ is the blob, s^j the resource state and x^j, the proximity of processors, together representing the subcontractor's *price*, p^j.

Definition 4 Given that the price $p^j \geq 0 \ \forall j \in \{1, 2, ..., n\}$. The winner determination problem is to label the proposals as winning ($a_j = 1$) or losing ($a_j = 0$) so as to minimize the auctioneer's expense under the constraint that each item can be allocated to at most one subcontractor:

$$\min \sum_{r \in \mathscr{N} \cup \mathscr{P}} e_b^r(s^r, x^r), \quad \text{with } b \in \mathscr{B} \tag{6}$$

subject to constraints of Eq. 4 and 5.

The cost expressed in Eq. 6 is a computation of reconstruction time that heavily depends on the network topology and the sub-swarms configuration. The SOUL mechanism to cope with this is detailed in the following section.

4 SOUL implementation

The SOUL mechanism consists of four processes: Bid Generation Process (BGP), Blob Update Process (BUP), Lists Update Process (LUP), and Blob Request Process (BRP), as illustrated in Fig. 2. Every member of the swarm runs the exact same algorithms and has an identical implementation of these four processes. When a robot wishes to share a blob within its swarm, the robot acts as the auctioneer for that blob and broadcasts a blob-advertising message. A robot r with available storage space responds with a bid value $q_b^r(s^r, x^r)$, with s^r and x^r their available storage space and their proximity to a processor, respectively. The auctioneer collects all the bids and sends the allocation request to the robots selected to store the blob's datagrams. A robot receiving such a message reserves the required space for the incoming blob's datagrams and, if the required size is not above its available space, sends an acceptance reply. Otherwise, the robot refuses the allocation request with a rejection reply. When receiving an acceptance reply, the auctioneer transfers the current datagram in the queue of the datagram vector d_b. If the auctioneer receives a rejection reply, it tries an allocation request to the next robot in a sorted list of bids. For every allocation accepted, before the blob's datagrams are transferred to another robot, the auctioneer adds the allocation to the locations list as a tuple ⟨id,number of datagrams⟩ and broadcasts this entry to all the robots in the swarm. If a robot needs to reconstruct a blob, it looks up the shared locations list and requests the corresponding datagrams from the robots holding them.

If the auctioneer robot does not receive enough bids to allocate all the datagrams within the bid allocation interval, it assumes either the communication is broken, or there is no available storage space. The first case with broken communication link can be ruled out using a connectivity maintenance controller [Ghedini et al. (2016)], so that we can assume

the existence of at least a communication path to every robot in the swarm. We tackle the second case by removing a blob with a certain set of policies from the robot network. The policies use local information such as the age of the blob and the priority of the blob to remove a blob. In this work, the policies we use are, the blob with lowest priority and highest age gets removed at first. The auctioneer robot creates a sorted list of blob indexes with the help of the above policies. In our approach we remove blobs once the robots hit their storage limits, but we could also write them to a file and re-inject them into the network when the available storage passes a given threshold.

4.1 SOUL Processes

4.1.1 Bid Generation Process

Algorithm 1 shows the pseudo code of the BGP. When a new blob is injected by a robot r_i, this latter executes the procedure on line 1. Robot r_i at first creates an MD5 hash with the content of the blob, which is used as the unique internal identifier for the blob as well as during the reconstruction to verify the integrity of the blob. Then, the blob is split into N_{c_q} identically-sized datagrams and inserted into the datagram vector d_b. Following this, the new blob is advertised to other robots and a new auction is created and added to the auction list. The procedure on line 6 is initiated when an advertise blob message is received. The robot j that receives the message checks its current available storage s^j and sends a bid triplet to the auctioneer. The procedure on line 11 collects all new bids in a bidlist when executed on the auctioneer robot.

Line 14 shows the datagram allocation procedure, that is the only periodic procedure executed in the BGP. When the auction list is not empty, the robot looks for any entry that has reached a timeout, and sorts the bidders list in ascending order of cost, such that it minimizes the reconstruction cost by solving the Eq. 2. The datagrams of the blob b are allocated to the robots in the sorted bidders list. The procedures on line 29 and 32 show, respectively, the behavior when a blob allocation is accepted, or rejected by a robot. When an allocation is accepted, the BGP requests the BUP to allocate s^j datagrams to robot j. Whenever an allocation is rejected, the datagrams are allocated to the next robot in the sorted list. If there are no more robots to be allocated in the list, one of the blobs is removed using SOUL policies as described in sec. 4.1.1 and the blob is allocated to the robots that were holding the now-removed blob. The procedure on line 43 is initiated when a robot requires a blob before the auction is complete. In other words, a robot receives a blob as soon as possible when required, irrespective of the state of the robots in the network.

Algorithm 1 The Bid Generation Process (BGP) pseudo code

```
1: procedure PUTBLOB(k, b)                          ▷ New blob b with key k
2:     (d_b, h_k) ← SplitHashBlob(b_k)
3:     AdvertiseBlob(b_k, h_k)
4:     ⟨Auctionlist⟩ ← NewAuction(b_k, h_k)
5: end procedure
6: procedure ADVERTISEDBLOB(b_k, h_k)
7:     if s^j > 0 then                              ▷ If storage space is not full
8:         SendBid(b^j, s^j, x^j)                   ▷ Bid for the blob
9:     end if
10: end procedure
11: procedure BIDRECEPTION(b^j, s^j, x^j)
12:     ⟨Bidlist_k⟩ ← (b^j, s^j, x^j)               ▷ Insert to bid list k
13: end procedure
14: procedure DATAGRAMALLOCATION(Auctionlist, Bidlist)
15:     for each A ∈ Auctionlist do
16:         if A.Time ≥ TimeOut then
17:             Bidlist_k ← SortBid(Bidlist_k)      ▷ Using Eq. 2
18:             i_k ← 0
19:             while Datagramstoallocate ≠ 0 do
20:                 r_i ← Bidlist_k[i_k].r_i
21:                 r_i ← aloc(r_i, s^j)            ▷ Allocate s^j size to r_i
22:                 i_k ← i_k + 1
23:             end while
24:         else
25:             A.Time ← A.Time + 1
26:         end if
27:     end for
28: end procedure
29: procedure ALLOCATIONACCEPT(b^j, s^j)
30:     allocatedatagram(s^j, j)
31: end procedure
32: procedure ALLOCATIONREJECT(b^j, s^j)
33:     if i_k < |Bidlist_k| then
34:         r_i ← Bidlist_k[i_k].r_i
35:         r_i ← a(r_i, B^k_{r_i})
36:         i_k ← i_k + 1
37:     else
38:         BR ← FindBlobToRemove                   ▷ Using Policies
39:         r_i ← BR.r_i
40:         r_i ← aloc(r_i, s^j)
41:     end if
42: end procedure
43: procedure AUCTINEERBLOBREQUEST(b^k, r_i)
44:     if AuctionComplete then
45:         for each j ∈ Locationslist do
46:             RequestDatagrams(j, s^j, r_i)       ▷ Request to r_i
47:         end for
48:     else
49:         TerminateAuction(k)
50:         SendDatagram(d_b, r_i)                  ▷ Send datagram vector
51:     end if
52: end procedure
```

4.1.2 Blob Update Process

BUP's procedures aims at managing the blob allocation, and datagram transmission and reception as illustrated in Algorithm 2. The procedure on line 1 transfers the next available s^j datagrams from the datagram vector to robot j (that has accepted the allocation) by updating the locations list and the available list (more on these lists can be found in sec. 4.1.4). A robot receiving a datagram $d_b[a]$ inserts it into the datagram vector d_b and updates its available list as shown in line 8, if the available list contains all the datagrams of the blob, then the state of the blob is set to be "ready". On reception of a datagram request, the robot checks for the availability of the requested number of datagrams and sends the datagrams to the requesting robot r_i.

Algorithm 2 BUPs pseudo code

```
 1: procedure ALLOCATEDATAGRAM(s^j, j)
 2:     j ← a(j, d_b[i] : d_b[i + s^j]) ▷ Transfer s^j datagrams to j
 3:     i ← i + s^j                     ▷ Store datagram index
 4:     RemoveDatagrams(d_b[i] : d_b[i + s^j])
 5:     UpdateAvailableList(d_b)
 6:     UpdateLocationsList(j, s^j)
 7: end procedure
 8: procedure RECEIVEDATAGRAM(d_b[a])
 9:     ⟨d_b⟩ ← d_b[a]          ▷ Insert d_b[a] datagram into d_b
10:     UpdateAvailableList(d_b)
11:     if AvailableListComplete then
12:         BlobState ← Ready
13:     end if
14: end procedure
15: procedure REQUESTEDDATAGRAM(s^j, r_i)
16:     if |d_b| == s^j then
17:         SendDatagram(d_b, r_i)
18:     end if
19: end procedure
```

4.1.3 Blob Request Process

When a robot wants to reconstruct a blob, the robot executes the getblob function as in Algorithm 3. At first, the robot checks whether all the datagrams of the blob are locally available by querying the state of the blob. If the state is "ready", then the blob is reconstructed from the datagram vector and then verified using the blob's hash. When not all the datagrams of the blob are available, this means that the auction is not complete or the locations list is not up-to-date. In both cases the robot requests the auctioneer for the blob, and the auctioneer, depending on its current state, either sends or requests the blob to be sent by the robots in the locations list. When the complete blob is available, the robot reconstructs it and makes it available.

Algorithm 3 BRPs pseudo code

```
 1: procedure GETBLOB(k)
 2:     if BlobState == ready then
 3:         b_k ← ReconstructBlob(d_b)
 4:         b_k ← VerifyHash(b_k, h_k)
 5:         return b_k
 6:     else
 7:         if LocationsListComplete then
 8:             for each j ∈ Locationslist do
 9:                 RequestDatagrams(j, s^j, r_i)
10:             end for
11:         else
12:             AuctineerBlobRequest(b_k, r_i)
13:         end if
14:         return 0
15:     end if
16: end procedure
```

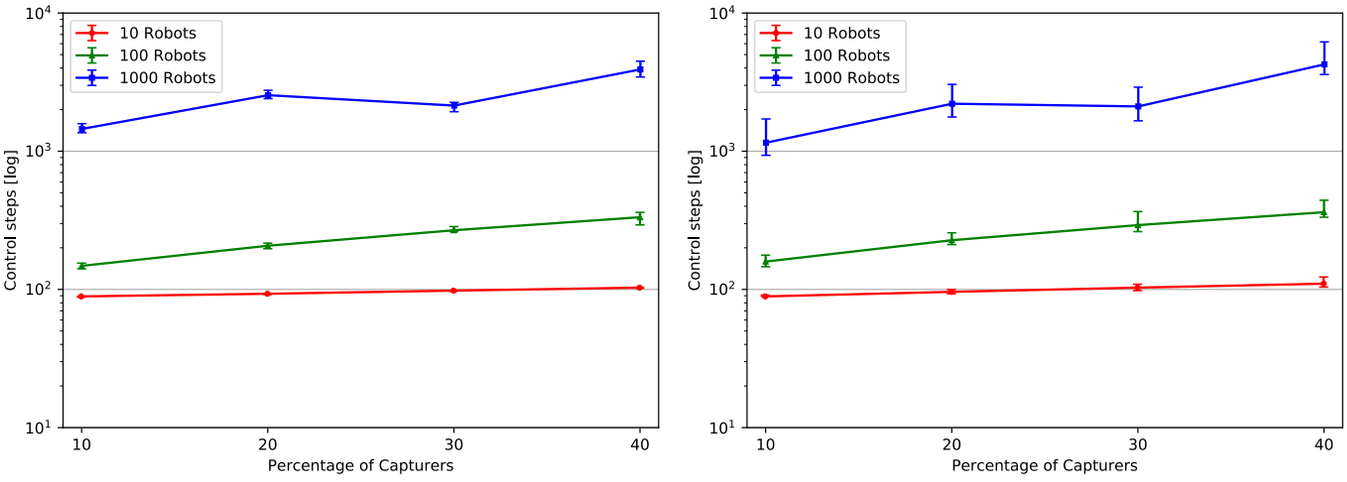
4.1.4 List Update Process

Each robot in the network maintains two lists (the available list and the locations list) and a state for every known blob. The available list contains the index of all available datagrams of the blob and the locations list contains an entry for each robot holding the datagrams of the blob. Every entry of the locations list contains the robot ID and the size of datagrams placed at that robot. The LUP maintains and updates the lists and the state whenever a new change occurs. The local lists act as a quick lookup for the availability of datagrams and their locations.

4.2 Buzz and SOUL

Buzz is a domain specific programming language for robotic swarms executed on a custom virtual machine. The Buzz Virtual Machine(BVM) is the core of Buzz that executes a byte code compiled from a Buzz script. Buzz provides a range of programming primitives tailored for robotic swarms: for more details on that we refer the reader to [Pinciroli and Beltrame (2016a)]. The SOUL implementation is integrated into the BVM as a programming primitive, making it available for extension and use by the research community.

In [Pinciroli et al. (2016)], the authors introduced an alternative use of the concept virtual stigmergy, a shared (key,value) pair among the robots in a swarm. Taking inspiration fromn [Pinciroli et al. (2016)], we designed SOUL as a shared `(key,blob)` within Buzz. Interestingly, the interface of SOUL is identical to that of the virtual stigmergy, but its internals substantially vary (with auction and allocation process). SOUL offers typical get/set routines to manage the blob tuple with a unique key and a set of function to inquiry the characteristics of blob (size, status, potential sink, etc.). When



(a) Cluster. (b) Scale free.

Fig. 3: Scalability analysis with 10, 100 and 1000 robots.

the auction is completed, a function (getblobseqdone) allows any robot to be aware of it.

5 Simulation Experiments

Simulation experiments aim at studying the performance of the SOUL model presented in the previous sections over diverse conditions. Simulations were performed with a physics-based multi-robot simulator, ARGoS3 [Pinciroli et al. (2012)].

Performance metrics

The performance measure used during the simulations was the time take for allocating the datagrams of a blob to the robots in the network, i.e. the convergence time of SOUL. The reconstruction time of a blob by any given robot could be computed as the communication delay between the robots holding the datagrams and the robot requiring it. It is desirable to minimize the convergence time to avoid delays in reconstructing a blob when required. During simulations, we used the number of control steps as a measure of convergence time, with each control step being 100 ms.

Communication model

We use a wireless device that creates a Mobile ad hoc network (MANET) [Barange and Sapkal (2016)] between the robots. A communication model that simulates the peer-to-peer communication was built within the simulator for the experimental evaluation. As in any MANET, the model provides both broadcast to neighbors within range, and uni-cast to any given robot in the network. Uni-cast messages within this model are sent to their destination by computing the shortest path. The messages within the SOUL model use broadcast in most cases, and unicast in a conservative manner, mostly when handling large datagrams, to minimize contention bandwidth use. We assume any given message between the robots in the network can be dropped with a certain probability, also known as the packet drop probability (or rate). Since the communication model is based on TCP/IP, we assume to have guaranteed delivery of messages and the effect of packet drop within the model was simulated as the delay incurred due to transmission errors. The communication model can be expressed as a weighted adjacency graph, for a network of n robots with an adjacency matrix $A = [\beta_{ij}] \in \mathbb{R}^{n \times n}$ with β_{ij} the packet drop probability [Davis et al. (2016)]. The entries in the adjacency matrix evolve over time depending on the topology of the network and the packet drop rates.

Experimental setting

We use two different topological configurations: cluster and scale-free. The former places the robots in a compact cluster with a uniform distribution $\mathcal{U}(-L/2, L/2)$, with L being the boundary of the arena. The latter, supposedly one of the most common topologies to naturally emerge from a swarm, distributes the robots in small clusters connected by hubs using Barabási-Albert preferential attachment algorithm [Barabási and Albert (1999)]. For details on these topologies, we refer the reader to [Pinciroli et al. (2016)] from where these topologies were adapted. During simulation, we use a packet drop probability $P \in \{0, 0.25, 0.5, 0.75\}$. Each robot has a storage limit (s_l^r) of 1000 datagrams, and the injected blobs (sized 1.02Kb) were 1022 datagrams each.

5.1 Scalability analysis

We studied the dependence of convergence time on two different topologies and the number of robots $N \in \{10, 100, 1000\}$ with set of simulation experiments. In particular, we assessed SOUL's scalability using static robots (fixed topology) and setting the percentage of capturer robots $\mathscr{C} \in \{10, 20, 30, 40\}$ varying the number of robots N over different trials. The percentage of processors $\mathscr{P}$ was set to 10 percent, and all the other robots without a role were tagged as networkers $\mathscr{N}$. For instance, with $\langle N = 100, \mathscr{C} = 40\rangle$, 40 robots were capturers injecting blob simultaneously, 10 robots were processors, and 50 were networkers. The role assignment was performed in such a way as to obtain random roles for robots over different trials. Each experimental setting was repeated 35 times with random placement and roles.

Results

Fig. 3 reports the results of our scalability experiment. In both cluster (fig. 3a) and scale-free (fig. 3b) topologies, the convergence time increases with the percentage of capturers, as more robots were simultaneously injecting blobs. Both the topologies incurred a similar change in performance for variations in the percentage of capturers and number of robots. In particular, with $N = 10$ both topologies consumed around 100 control steps to converge, whereas with $N = 100$ a steady increase in convergence time is observed with the increase in percentage of capturers. With $N = 1000$, the convergence time incurred large variations in scale-free topology, which could be explained by the change in sparse communication paths for the scale-free topology over different trials. Whereas in cluster topology, the variations in convergence time with $N = 1000$ is less in comparison with scale-free, which could be explained by the dense communication path allowing persistent data transfer over different trials. From the experimental evaluation results, we conclude that SOUL scales for upto 1000 robot with both the topological distributions.

Fig. 4 plots the datagram distribution on the robots during one of the experimental trial with $\langle N = 100, \mathscr{C} = 10\%\rangle$. During this trial, the blobs were continuously injected one after the other until the network dropped the first injected blob. The 100-robot network lost the first blob when injecting the 98th blob: the swarm was able to buffer 97 blobs, saturating the available storage space. The distribution of datagrams can be grouped into three classes that correspond to the three sloped distributions in Fig. 4 from left to right: the first group are the processor robots where the first 10 blobs were placed for fast reconstruction; the second group corresponds to the robots that are neighbors

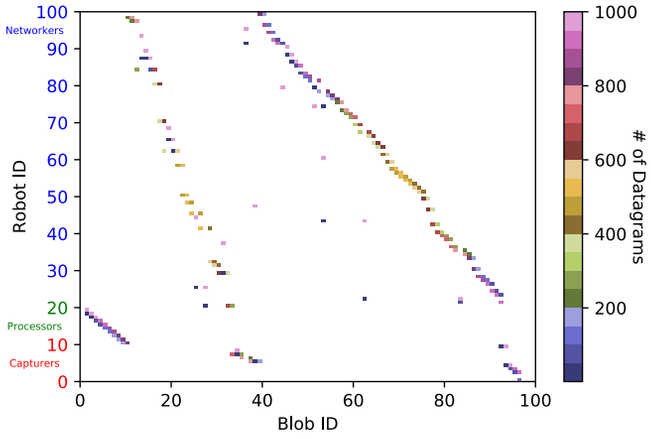


Fig. 4: Datagram distribution in a 100 robot network, as a result of continuous injection of blobs

of processor robots, which obtain the following blobs; the final group contains all the other robots having the lowest priority in the network. From this distribution classes, one could observe the effect of Eq. 2 minimizing the cost of reconstruction. It is worth noting that the allocation always starts from the highest robot ID because during the allocation using Eq. 2, robots break ties using robot IDs.

5.2 Dynamic Analysis

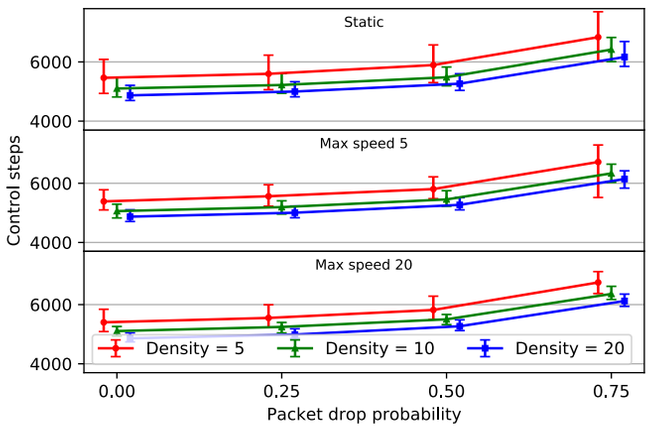
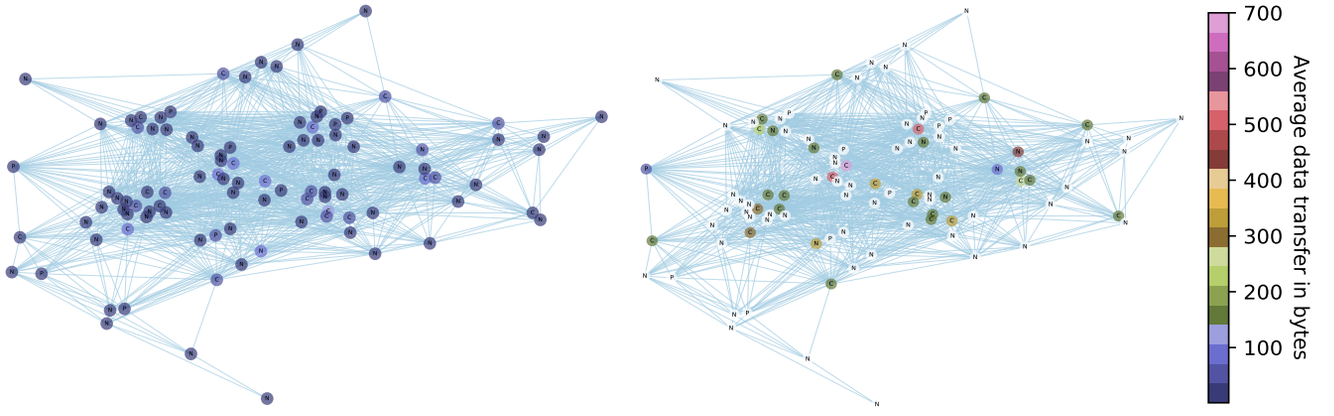
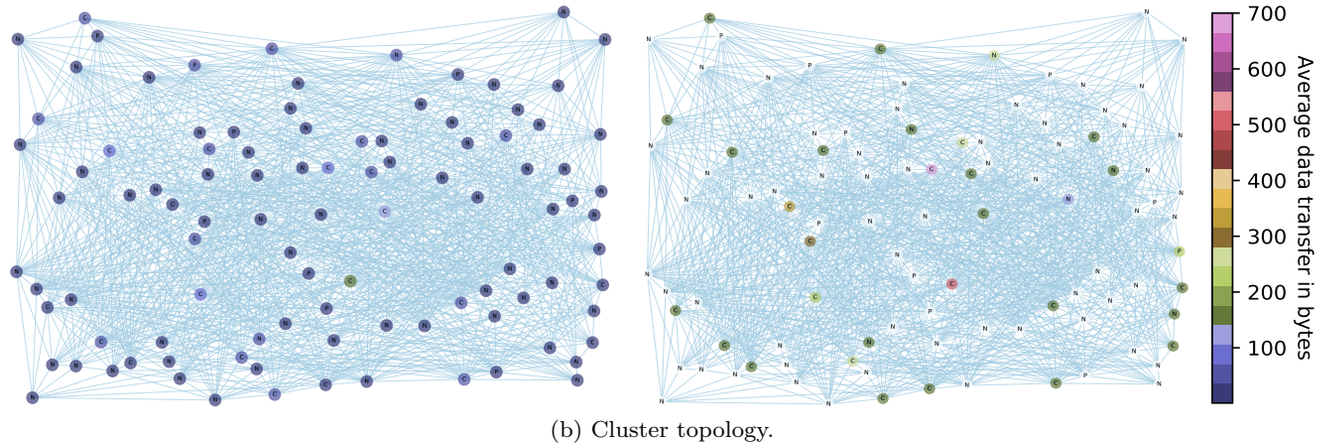


Fig. 5: Packet drop rate's influence on the time (control steps) required to allocate datagrams for various density of moving robots.

We performed a set of experiments to study the influence of change in topology on convergence time. During these experiments the robots had a simple controller executing the diffusion algorithm presented in [Howard et al. (2002)] to produce random motion



(a) Scale-free topology.



(b) Cluster topology.

Fig. 6: Average bandwidth consumption: left with SOUL, right as a result of one-to-one transmission.

spanning the arena. Blobs were injected continuously one after the other until the swarm dropped the first injected blob. The time taken for the loss of first injected blob is reported in fig. 5. The simulation arena during the motion experiments had four obstacles to stimulate changes in topology. To study the influence of the velocity of change in topology during these experiments, we considered different densities $D \in \{5$(lose distribution),10(medium distribution),20(tight distribution)$\}$ and maximum speeds for the robots $M \in \{5, 20\}$. The density of the robots was changed by moving the boundaries of the simulation arena as in [Pinciroli et al. (2016)]. In this set of experiments, N was set to 100 robots.

Fig. 5 reports the time steps required to remove the first blob with static robots (top) , moving at $M = 5$ (middle), and moving at $M = 20$ (bottom). The static robot case is analogous to the cluster topology reported in Sec. 5.1. In all cases, the swarm took similart time to lose a blob and to inject 97 blobs. With packet drop probability of up to 0.5, the network's performance degraded slowly, taking less than 6000 control steps to lose a blob. For P = 0.75 the time to lose a blob quickly increased to over 6000 control steps. The static topology reported large variability in convergence time with respect to the dynamic topology, probably due to the fixed communication link for a given trial. From this experimental evaluation, we conclude that the change in topology has minimal impact on the SOUL model.

5.3 Bandwidth consumption

Fig. 6a and 6b report the average bandwidth consumed by scale-free and cluster topologies respectively, in comparison with unicasting the blobs from a single robot. In this experimental trial, the blobs were injected from one of the robots in the network with $\langle N = 100, \mathscr{C} = 10, \mathscr{P} = 10$, and the maximum usable bandwidth was set to 700 bytes per second. This set of experiments measure the average bandwidth consumed until the first blob is lost, i.e. to inject 97 blobs. With a scale-free topology, the average bandwidth consumed by all the robots with SOUL is less than 100 bytes per second and with unicast the average bandwidth was over 100 bytes per second, with some robots using the

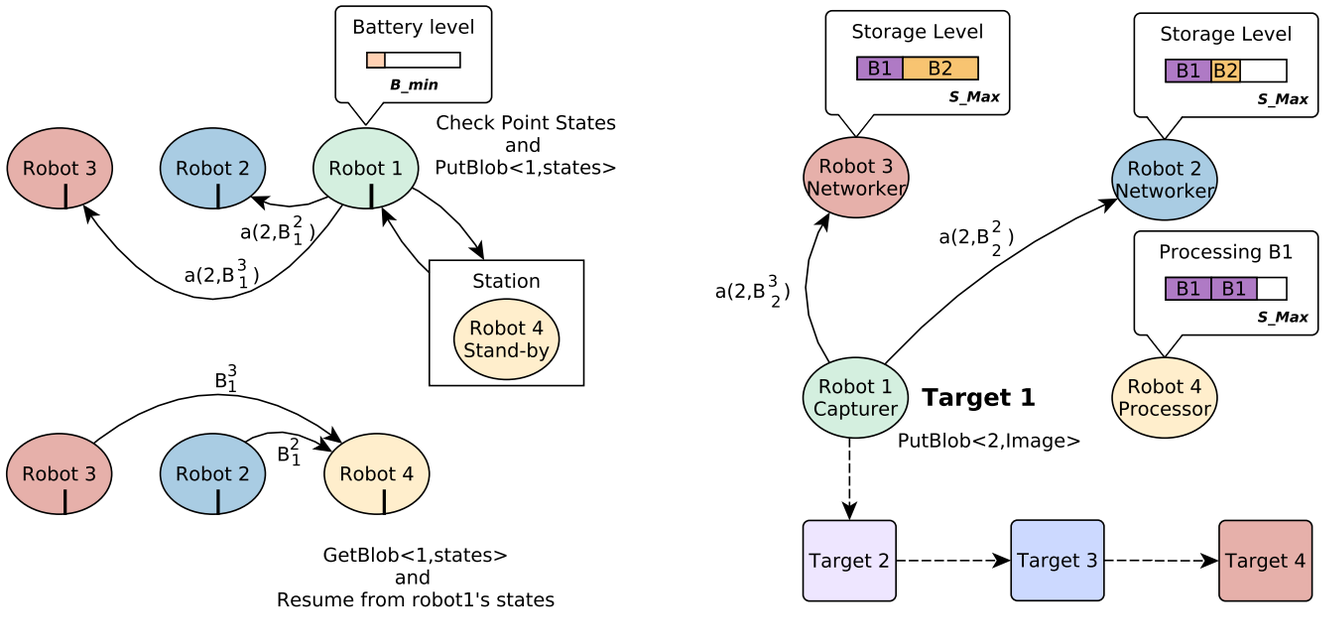


Fig. 7: Schematic representation of the two experiments conducted on the Khepera robots: SOUL backup to the left and YOLO Search, to the right.

maximum available bandwidth on average, and the majority of the robots consuming zero bandwidth. A similar effect happens with the cluster topology. SOUL lowered the average bandwidth consumed and evenly distributed the bandwidth requirements over the network.

6 Experiments

SOUL was studied under two realistic experimental scenarios with a swarm of 10 Khepera IV [Soares et al. (2016)] robots. Our experimental platform consists of an Optitrack system (an IR camera based tracking system), a centralized wifi-based software communication hub and a swarm of 10 Khepera robots. SOUL implementation requires situated communication [Støy (2001)], a common concept of swarms providing each inter-robot message with the relative distance and bearing of the sender. Situated communication is emulated by transmitting all the robots messages to each other swarm member through the hub, named Blabbermouth. Blabbermouth also computes the relative position from the tracking system and append it to each message. A laptop was running Blabbermouth dealing with situated communication and peer-to-peer network path using TCP/IP over wifi. This configuration by default generates a fully connected network topology, but the transmission range can easily be limited in software using each robot's position.

In order to test and demonstrate the range of application SOUL can be fit for, we designed two experiments: one to backup a failing robot (SOUL backup) and the second to manage large data transfer in an heterogeneous mission (YOLO Search). Fig. 7 illustrates both of the experimental scenarios: SOUL backup on the left and YOLO Search on the right.

6.1 SOUL backup

For this first experimental scenario, the robots were performing a cordon around an object of interest. Patrol and cordon exercices are frequent in emergency response and swarms are well fitted for the task [McDonald et al. (2017)]. Maintaining the surveillance perimeter around a sensible structure is challenging in terms of robot autonomy. As depicted in fig. 7, when one of the robot predicts an incoming failure (e.g. low battery), it serializes its current state and injects it into the swarm with SOUL. The robot identity is then available to be picked by any other member of the swarm. More then just the state of its current task, this backup includes the unique robot id, all of its global variables and the key data structures of its BVM. In [Pinciroli and Beltrame (2016b)], the authors state that describing the dynamics of a swarm through a finite state machine allows the developers to focus on the swarm behavior design rather than the individual robot behavior. Copying the key data structures of this finite state machine is what allows this back-up and resume mechanism with a new robot. In the experiment, a standby robot waiting aside the formation is

asked to replace the failing one and obtains its backup from SOUL. It can then resume the mission and be perceived by the swarm as the fallen member seamlessly. The chronology of the process is illustrated in fig. 9 with regards to the datagrams stored in the swarm.

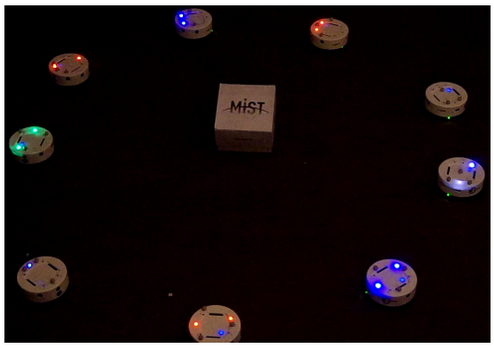


Fig. 8: Nine wheeled robots patrolling around a box. The LED patterns show their unique ID.

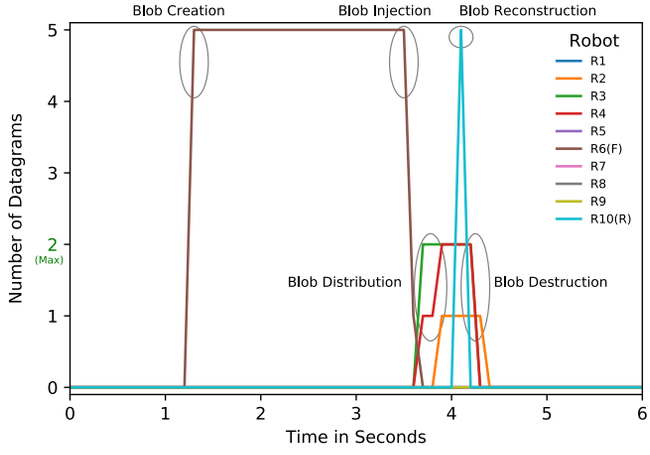


Fig. 9: Storage usage throughout one trial for the soul experiment.

As shown in fig. 8, the cordon is created with nine Khepera IV robots (numbered R1 to R9) in order to surround a box. A stationary robot (R10) waits aside the playground, as a replacement unit. Each robot blinks with a LED pattern indicating their unique ID and a frequency increasing with the mission time. When R6, the failing robot, observes a critical level of its battery, it triggers its identity backup (ID, states and BVM structure) and injects it as a serialized data blob into SOUL. As the auctioneer, robot R6 completes the datagrams allocation and distribution (known through operation *getblobseqdone()*) and then leaves the mission. When a robot leaves the swarm, all the other robots hold their position to allow the replacement robot to easily reach the swarm. While this is not mandatory it allowed us to get visual confirmation that the backup blob was successfully distributed. Robot R10 gets the blob datagrams and rebuild it using SOUL. It then sets its current state with blob's content, seamlessly taking R6's place in the swarm. Once robot R10 resumes the mission of R6, it joins the formation and start to patrol. This experiment was repeated ten times to confirm a consistent behavior.

In the supplementary video material, an excerpt from one of these SOUL backup experiments is presented. The LED patterns and their frequency show that the replacement robot takes the same ID (LED pattern) as its predecessor and starts at the same mission state (LED frequency). The LED are controlled by the behavioral Buzz script.

Results

For all the experiment trials, the robots were able to successfully backup their identity and resume the mission consistently. Fig. 9 reports the number of datagrams on each robots on one such trial. When R6 was close to fail it created a backup blob of five datagrams at 1.1s. The datagrams of this blob was then allocated to robot R2, R3 and R4, which were the closest to the replacement robot R10. R10 obtained all the datagrams from these three robots at 4s, and, after reconstruction, removed the blob, allowing all the other robots to remove their datagrams of this blob. Considering all trials, it took in average 2.3s to create, inject and distribute the blob with SOUL. Its full removal took in average 3.2s. The backup blob was always distributed over three robots, since the storage limit (s_l^r) was set as 2 datagrams for all robots in this experiment.

6.2 YOLO Search

The second experimental scenario represents a search mission executed by a swarm of wheeled robots. As shown in fig. 7, we arranged four targets, spread around the robots playground. The goal was to take picture and detect a key object in it at each location. However, no robot had both the camera and the processing power required. The swarm was heterogeneous: following their on-board hardware, the robots were assigned different roles. A single robot had a working camera, R4, and was attributed the capturer role. A single robot had a powerful processing GPU required to analyze the data, R10, and was attributed the processor role. All the other robots were the networkers, moving around randomly to create dynamic network topology. Indeed, the communication range of the robots was limited to be 1.1m, in order to test the behavior with dynamic topology. The processor was also moving randomly while waiting

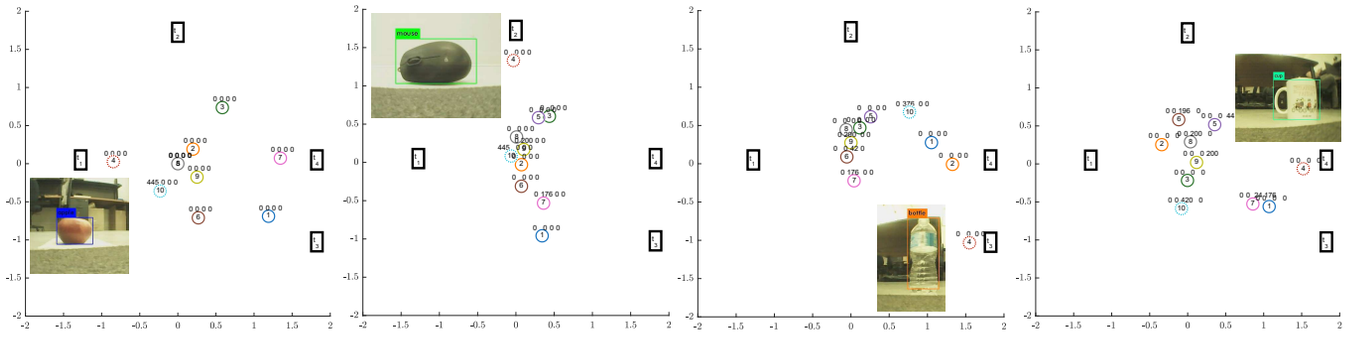


Fig. 10: Evolution of the blob distribution over the swarm on the second trial of the YOLO Search experiment.

for images to process using a GPU-optimized algorithm software: YOLO [Redmon et al. (2016)],a popular deep learning object detection system with state-of-the-art accuracy. The capturer was moving to pre-defined target positions searching for objects of interest and took a picture at each location.

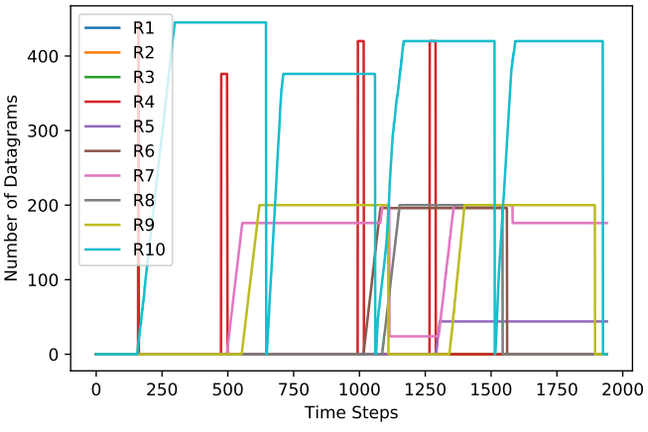


Fig. 11: Storage usage throughout the second trial for the YOLO Search experiment.

Ten Khepera IV robots were used: one capturer, one processor and eight networkers. The arena was 2m×2m with four target locations, as illustrated in fig. 10. Each location had a different object, selected from the training set of YOLO3. The processor had a Nvidia TX1 board to process the images. All robots were initially placed on the boundaries of the arena. The capturer moved from target to target, each time injecting a blob with SOUL. The processor and the networkers moved randomly in the arena. All robots were equipped with a collision avoidance mechanism, reacting to near obstacles with an incremental steering velocity. This experiment was also repeated ten times, each time generating different trajectories and completion time due to the random motions and the reactive collision avoidance.

The storage limit on all networkers was set to 200 datagrams while the images produced were around 500 datagrams. The capturer, R4, injected blobs into the swarm, as soon as it captured an image of the target. The processor, R10, waited for a blob to be available in the stigmergy whenever it was not processing an image already. As soon as the processor was done with an image, it removed it from the stigmergy. All the networkers can be used to store the blobs while the processor was busy with a previous one. When the processor was done processing a blob, the datagrams of this blob was removed from all networkers too.

In the supplementary video material, an excerpt from one of these YOLO Search experiments is presented.

Results

For each trial, the images were successfully captured, injected, reconstructed and processed from all four target locations. Fig. 11 shows the datagrams distribution over the whole swarm for one experiment. At the first target, R4 injects the first image blob. Since R10 is initially free it gets the image immediately, bypassing all the auction and the blob distribution discussed in sec. 4.1.1. For all the other three targets, the creation of a blob happens with R10 busy, it thus requires the datagrams to be allocated to networkers.

Fig. 10 shows the top view of the experimental arena with the four targets (rectangles) and the ten robots (circles). Each robot has four numbers above it, corresponding to the datagrams they hold of blob 1 to 4 (left to right). The capturer and processor are illustrated with dashed lines. Each snapshot corresponds to the moment a blob is injected in the swarm. Both fig. 11 and 10 represent the same trial. We can observe that the second datagram is attributed to robot R7 and R9, while the third is stored on robots R6, R7 and R8 and the last on R5, R7 and R9. In average, over the ten trials, 40 blobs were created, from which 21 were sent from the capturer to the processor directly. Over the remaining 19 that had to be distributed on the net-

workers (processor busy), the average time to allocate the datagrams is 2.7s (min:2.3, max:7.4s) and the average time to distribute them following their allocation is 12.9s (min: 8s, max: 18.3s). These variations are influenced by the network topology only. The supplementary material (Yolo.gif) contains an animation illustrating the robot trajectories and blob placements during this experimental trial.

7 Conclusions

This paper introduced a new mechanism to share large data in a swarms of robots called SOUL: Swarm-Oriented Upload of Large files. SOUL can be applied to a wide range of applications that needs to share large data in a fully distributed system. We described potential applications together with the background work on consensus and peer-to-peer mechanism that led to this implementation. SOUL uses an auction-based algorithm to distribute large data files on multiple robots around potential receivers. The data are split into datagrams leveraging a torrent-like mechanism. Numerous simulations demonstrated scalability and robustness to failures with different swarm size, packet drop rate, network topologies and heterogeneous configurations. We demonstrated the potential of SOUL under two realistic experimental scenarios with a swarm of ten Khepera IV robots: one to backup a failing robot (SOUL backup) and the second to manage large data transfer in an heterogeneous area covering mission (YOLO Search). Both shown SOUL has consistent performance, demonstrating its potential in reality.

SOUL now needs a mechanism to provide versioning, redundancy, a reallocation strategy, and security protocols. Once the datagrams of a blob is allocated its metadata can be extended to include a Git-like track of the changes. Robots movement will change the network topology and the auction attributing the datagrams should be run in cycle to maintain the cost of reconstruction minimal. As for security, future work will implement stream cipher based encryption of datasets to enhance the protection of the data.

SOUL is born from the challenges we faced with the deployment of heterogeneous swarms for emergency response scenarios. This solution is among the essential building blocks of a robust and flexible software solution for field deployment of robotic swarms in critical scenarios.

Acknowledgements We would like to thank NSERC for supporting this work under the NSERC Strategic Partnership Grant (479149). Simulation experiments were performed on supercomputer Mammouth-Ms from Université de Sherbrooke, managed by Calcul Québec and Compute Canada. We thank Calcul Québec and Compute Canada for making the resource available.

References

[Aguilar et al. (2017)] Aguilar WG, Luna MA, Moya JF, Abad V, Ruiz H, Parra H, Angulo C (2017) Pedestrian detection for uavs using cascade classifiers and saliency maps. In: Rojas I, Joya G, Catala A (eds) Advances in Computational Intelligence, Springer International Publishing, Cham, pp 563–574

[Barabási and Albert (1999)] Barabási AL, Albert R (1999) Emergence of scaling in random networks. Science 286(5439):509–512

[Barange and Sapkal (2016)] Barange MY, Sapkal AK (2016) Review paper on implementation of multipath reactive routing protocol in manet. In: Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on, IEEE, pp 227–231

[Bayindir (2016)] Bayindir L (2016) A review of swarm robotics tasks. Neurocomputing 172:292–321, DOI 10.1016/j.neucom.2015.05.116

[Benet (2014)] Benet J (2014) Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:14073561

[Brown and Sreenan (2013)] Brown S, Sreenan C (2013) Software Updating in Wireless Sensor Networks: A Survey and Lacunae, vol 2. DOI 10.3390/jsan2040717, URL http://www.mdpi.com/2224-2708/2/4/717/

[Brunet et al. (2008)] Brunet L, Choi HL, How JP (2008) Consensus-based auction approaches for decentralized task assignment. In: AIAA Guidance, Navigation, and Control Conference, August, pp 1–24, DOI 10.2514/6.2008-6839

[Davis et al. (2016)] Davis DT, Chung TH, Clement MR, Day MA (2016) Consensus-Based Data Sharing for Large-Scale Aerial Swarm Coordination in Lossy Communications Environments. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 3801–3808, DOI 10.1109/IROS.2016.7759559

[Dhurandher et al. (2009)] Dhurandher SK, Singhal S, Aggarwal S, Pruthi P, Misra S, Woungang I (2009) A Swarm Intelligence-based P2P file sharing protocol using Bee Algorithm. 2009 IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2009 pp 690–696, DOI 10.1109/AICCSA.2009.5069402

[Erdelj and Natalizio (2016)] Erdelj M, Natalizio E (2016) UAV-assisted disaster management: Applications and open issues. 2016 International Conference on Computing, Networking and Communications (ICNC) pp 1–5, DOI 10.1109/ICCNC.2016.7440563, URL http://ieeexplore.ieee.org/document/7440563/

[Ganesan et al. (2004)] Ganesan P, Gummadi K, Garcia-Molina H (2004) Canon in g major: designing dhts with hierarchical structure. In: Distributed computing systems, 2004. proceedings. 24th international conference on, IEEE, pp 263–272

[Garbacki et al. (2006)] Garbacki P, Iosup A, Epema D, van Steen M (2006) 2Fast : Collaborative Downloads in P2P Networks. Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06) pp 23–30, DOI 10.1109/P2P.2006.1, URL http://ieeexplore.ieee.org/document/1698587/

[Ghedini et al. (2016)] Ghedini C, Ribeiro CHC, Sabattini L (2016) Improving the fault tolerance of multi-robot networks through a combined control law strategy. In: 2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM), pp 209–215, DOI 10.1109/RNDM.2016.7608289

[Howard et al. (2002)] Howard A, Matarić M, Sukhatme G (2002) Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In: Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS), Springer, New York, pp 299–308

[Kamra et al. (2006)] Kamra A, Misra V, Feldman J, Rubenstein D (2006) Growth codes: Maximizing sensor network data persistence. In: ACM SIGCOMM Computer Communication Review, ACM, vol 36, pp 255–266

[Kelley and Ouellette (2013)] Kelley DH, Ouellette NT (2013) Emergent dynamics of laboratory insect swarms. Scientific reports 3:1073

[Kuriki and Namerikawa (2015)] Kuriki Y, Namerikawa T (2015) Experimental Validation of Cooperative Formation Control with Collision Avoidance for a Multi-UAV System. In: Proceedings of the 6th International Conference on Automation, Robotics and Applications, pp 531–536

[Laukkanen et al. (2017)] Laukkanen E, Itkonen J, Lassenius C (2017) Problems, causes and solutions when adopting continuous delivery: A systematic literature review. Information and Software Technology 82:55–79, DOI 10.1016/j.infsof.2016.10.001

[Lee (2018)] Lee DH (2018) Resource-based task allocation for multi-robot systems. Robotics and Autonomous Systems 103:151–161

[Lliffe (2016)] Lliffe M (2016) Drones in Humanitarian Action. Tech. rep., The Swiss Foundation for Mine Action, Geneva/Brussels, URL http://drones.fsd.ch/

[McDonald et al. (2017)] McDonald SJ, Colton MB, Alder CK, Goodrich MA (2017) Haptic Shape-Based Management of Robot Teams in Cordon and Patrol. Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction - HRI '17 pp 380–388, DOI 10.1145/2909824.3020243, URL http://dl.acm.org/citation.cfm?doid=2909824.3020243

[Pinciroli and Beltrame (2016a)] Pinciroli C, Beltrame G (2016a) Buzz: An extensible programming language for heterogeneous swarm robotics. In: International Conference on Intelligent Robots and Systems, IEEE, pp 3794–3800

[Pinciroli and Beltrame (2016b)] Pinciroli C, Beltrame G (2016b) Swarm-Oriented Programming of Distributed Robot Networks. Computer 49(12):32–41, DOI 10.1109/MC.2016.376

[Pinciroli et al. (2012)] Pinciroli C, Trianni V, O'Grady R, Pini G, Brutschy A, Brambilla M, Mathews N, Ferrante E, Di Caro G, Ducatelle F, Birattari M, Gambardella LM, Dorigo M (2012) ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence 6(4):271–295, DOI 10.1007/s11721-012-0072-5

[Pinciroli et al. (2016)] Pinciroli C, Lee-Brown A, Beltrame G (2016) A Tuple Space for Data Sharing in Robot Swarms. Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS) pp 287–294, DOI 10.4108/eai.3-12-2015.2262503

[Redmon et al. (2016)] Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 779–788

[Reid (2015)] Reid N (2015) Literature Review : Purely Decentralized P2P File Sharing Systems and Usability. Tech. rep., Rhodes University, Grahamstown

[Ren et al. (2005)] Ren W, Beard R, Atkins E (2005) A survey of consensus problems in multi-agent coordination. In: Proceedings of the American Control Conference, pp 1859–1864, DOI 10.1109/ACC.2005.1470239

[Sandholm et al. (2000)] Sandholm T, Sandholm T, Suri S, Suri S (2000) Improved Algorithms for Optimal Winner Determination in Combinatorial Auctions and Generalizations. Proceedings of the National Conference on Artificial Intelligence (AAAI) pp 90–97

[Sen and Adams (2013)] Sen SD, Adams JA (2013) A decision network based framework for multiagent coalition formation. In: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, pp 55–62, URL http://dl.acm.org/citation.cfm?id=2484920.2484933

[Soares et al. (2016)] Soares JM, Navarro I, Martinoli A (2016) The khepera iv mobile robot: Performance evaluation, sensory data and software toolbox. In: Robot 2015: Second Iberian Robotics Conference, Springer, pp 767–781

[St-Onge et al. (2018)] St-Onge D, Varadharajan VS, Li G, Svogor I, Beltrame G (2018) Ros and buzz: consensus-based behaviors for heterogeneous teams. In: [submitted to] International Conference on Intelligent Robots and Systems, IEEE, URL https://arxiv.org/abs/1710.08843

[Støy (2001)] Støy K (2001) Using situated communication in distributed autonomous mobile robots. Proceedings of the 7th Scandinavian Conference on Artificial Intelligence pp 44–52

[Surmann et al. (2017)] Surmann H, Berninger N, Worst R (2017) 3D mapping for multi hybrid robot cooperation. IEEE International Conference on Intelligent Robots and Systems 2017-Septe:626–633, DOI 10.1109/IROS.2017.8202217

[Varadharajan et al. (2018)] Varadharajan V, St-Onge D, Guss C, Beltrame G (2018) Over-The-Air Updates for Robotic Swarms. IEEE Software

[Vempati et al. (2017)] Vempati AS, Gilitschenski I, Nieto J, Beardsley P, Siegwart R (2017) Onboard real-time dense reconstruction of large-scale environments for UAV. IEEE International Conference on Intelligent Robots and Systems 2017-Septe:3479–3486, DOI 10.1109/IROS.2017.8206189

[Vig and Adams (2006)] Vig L, Adams JA (2006) Market-Based Multi-robotMulti-robot coalition formation. In: Distributed Autonomous Robotic Systems, vol 7, Springer, Tokyo, pp 227–236

[Vu et al. (2010)] Vu QH, Lupu M, Ooi BC (2010) Architecture of peer-to-peer systems. In: Peer-to-peer Computing, Springer, pp 11–37

[Wei et al. (2015)] Wei X, Fengyang D, Qingjie Z, Bing Z, Hongchang S (2015) A New Fast Consensus Algorithm Applied in Rendezvous of. In: 2015 27th Chinese Control and Decision Conference (CCDC), pp 55–60, DOI 10.1109/CCDC.2015.7161666, URL http://ieeexplore.ieee.org/document/7161666/

[Zhang et al. (2009)] Zhang K, Collins EG, Shi D, Liu X, Chuy O (2009) A Stochastic Clustering Auction (SCA) for centralized and distributed task allocation in multi-agent teams. Distributed Autonomous Robotic Systems 8 7(2):345–354, DOI 10.1007/978-3-642-00644-9-31