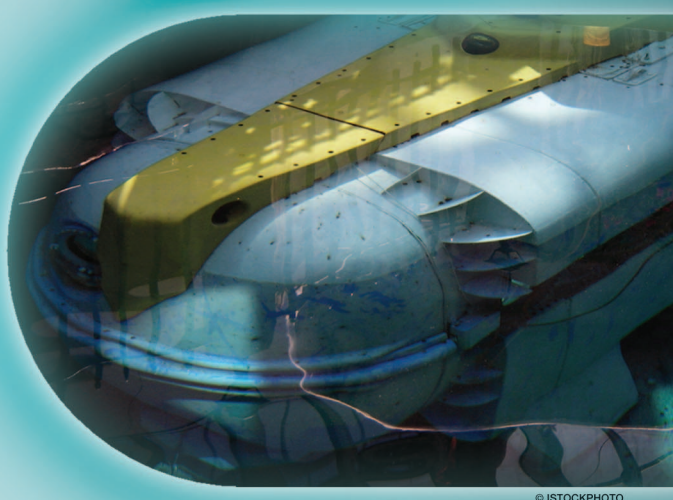
Heterogeneous Wireless Multirobot System

A Platform for Safety and Security in Distributed Communication Computing and Control

BY ANTONIO BICCHI, ANTONIO DANESI, GIANLUCA DINI, SILVIO LA PORTA, LUCIA PALLOTTINO, IDA M. SAVINO, AND RICCARDO SCHIAVI



Mobile Multirobot Systems

The convergence of communication, computing, and control is considered by many the future of information technology [1], [2]. This will provide the ability for a large number of sensors, actuators, and computational units interconnected, wirelessly or over wires, to interact with the physical environment. One of the main expected consequences of such convergence is the possibility to create large systems of many autonomous and interconnected units, which have capabilities of not only sensing [3] but also acting in and on the environment. Several research challenges raised by multiple autonomous mobile systems are stimulating a keen interest in the robotics research community, such as formation control and flocking (e.g., [4]–[6]), coordination (e.g., [7]–[9]), communication problems and protocols (e.g., [10]), and algorithm distribution (e.g., [1], [11]). These advances form the basis for addressing the application of multiagent robotic systems outside the labs in new scenarios, ranging from the exploration of unknown environments to surveillance, patrolling, and so forth.

In this article, we consider a scenario in which a group of vehicles move autonomously in a shared environment. Each vehicle is given a specific task to accomplish, on its own or in collaboration, such as monitoring the environment, reconstructing a map, searching for an object, or detecting light or heat sources. Agents can join or leave the group dynamically. Typical agents are inexpensive, unmanned vehicles equipped with embedded sensor systems with limited onboard processing units and short-range wireless communication capabilities. Contrary to what is often assumed in the current state of art, we accept the realistic requirement that the platform must accommodate for broadly heterogeneous vehicles in terms of different tasks to accomplish, different onboard sensors and computational capabilities, and also different dynamics or dimensions.

In particular, we focus on four crucial requirements on the design of a component-based platform for multiagent systems, which are safety, scalability, security, and reconfigurability. In our context, safety means that motions of the robots are executed so that any collision among them is avoided while they attend to their tasks. The need to manage a possibly large number of vehicles imposes scalability of the platform. An immediate consequence of this requirement is that solutions using a centralized traffic supervisor dispatching detailed instructions to all vehicles are unacceptable.

To fulfill their tasks, including collision avoidance, cooperative vehicles have to communicate, and ad hoc wireless network

Digital Object Identifier 10.1109/M-RA.2007.914925

1070-9932/08/$25.00©2008 IEEE

technologies are apparently good candidates to provide support for such architecture. Protecting wireless communication poses unique challenges. Unlike traditional wired networks, an adversary equipped with a simple radio receiver or transmitter can easily eavesdrop communication and inject or modify packets. Furthermore, to make them economically viable, embedded devices are often limited in their energy, computation, storage, and communication capabilities, and this leads to constraints on the types of security solutions that can be applied. To address these challenges, the platform should support security requirements in terms of secrecy, integrity, and authenticity of communications with respect to a potential active outsider.

A practical use of the platform also imposes the reconfigurability requirement. Because of their tasks, vehicles typically operate in critical environments that are subject to unpredictable changes of operational conditions. This requires the multiagent systems application to be able to reconfigure itself so as to meet the changing conditions. The proposed platform supports reconfigurability at different levels. At a physical level, vehicles may dynamically join and leave the group. Hence, candidate new members of the group can be accepted if safety is not compromised. At a logical level, a vehicle may need to reprogram tasks and the implementation of a given service because of the changed operational conditions. Also, reprogramming must not endanger the security of the vehicle software platform and thus of the whole platform.

The platform described in this article has been designed and implemented to fulfill the requirements described earlier. In particular, for the heterogeneity requirement, the platform must also be accessible to very simple vehicles with possibly low computational and data storage capabilities. To realize a platform that can deal with a larger class of mobile robots (from the very simple to the advanced-technology examples), services have been implemented taking into account possible technological limitations of the vehicles involved in the scenario.

Platform Architecture

In this section, the component-based agent architecture is described. A component is an encapsulated unit of functionality and deployment that provides services through its interface. This architecture abstracts away from the actual platform implementation and provides a general design framework for multiagent systems. Furthermore, the component-based approach supports and promotes encapsulation and modularity of design and implementation and thus makes it possible to integrate vehicle with hardware and software of completely different origin and make them safely and securely coexist and collaborate. In addition, if the basic runtime software allows it, components can be dynamically added and removed. This makes it possible to retask a robot and change the implementation of a service according to the changing operational conditions.

Agent Architecture

In Figure 1, the architecture of an agent in terms of its constituting components and their relationships is reported. Components are of different types. The network component provides network services for sending and receiving packets. The application components implement the task the agent has to fulfill. In the rest of this article, we abstract away application components with the component application in Figure 1 and do not specify them any further.

The control components and the security components allow a vehicle to access to, or even participate to the implementation of, the services described in the "Architecture Implementation" section. More precisely, the control components comprise the collision avoidance component (CAC), the self-localization component, and the neighbor-localization component and deal with collision avoidance and vehicle localization, respectively. The security components comprise the security controller component (SCC), the rekeying component, and the authenticated loading component (ALC) and deal with secure communication, rekeying, and secure software reconfiguration, respectively. Finally, the actuation components deal with the actual actuation of the motion commands issued by the CAC.

In the rest of the section, we provide a detailed description of components, interfaces, and services provided. We describe components interfaces, in a language-neutral interface description language (IDL). For each operation provided by an interface, the IDL allows us to specify the name and the type of both the arguments the operation takes as input (in parameters) and the values the operation returns as output (out parameters).

1) CAC: The CAC coordinates the motion of agents preventing collisions and guaranteeing that each agent eventually reaches the final configuration required by its task, providing to the agent a collision avoidance maneuver.

The component implements the following interface:

```
Interface ICollision{
  setParameters(in Parameters p);
  getParameters(out Parameters p);
  join(in Name n, in Configuration initial, in
Configuration final);
  leave(in Name n);
}
```

The operation `setParameters` initializes parameters necessary for the correct execution of each collision avoidance

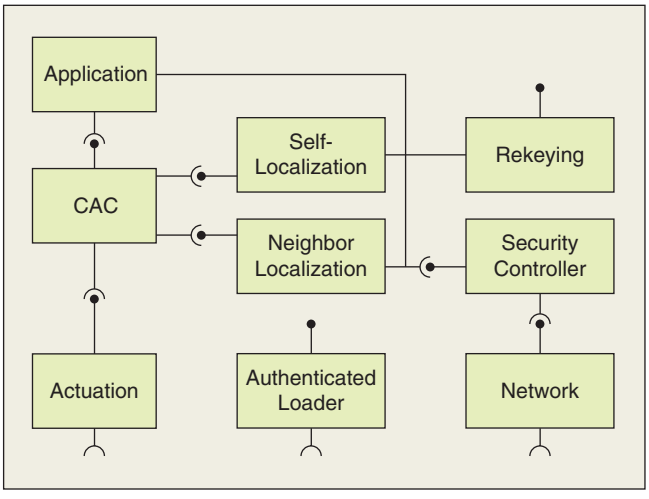


Figure 1. *The software architecture.*

algorithm the component implements. The actual implementation of Parameters depends on the specific collision avoidance algorithm (e.g., dimension and speed of the vehicle). The operation getParameters returns the current value of parameters. The operation join makes a vehicle named n to join the group. The operation takes the initial and the final configuration of the vehicle as input arguments. Finally, the operation leave makes vehicle named n to leave the group.

The CAC provides two services: the collision avoidance service that guarantees that no collision will occur among vehicles belonging to the group and the group membership service (GMS) that guarantees that every new joining does not endanger the safety property.

For every vehicle, the collision avoidance service requires both the current configuration of the vehicle and the configurations of its neighboring vehicles. The self-localization service provides the former information whereas the neighbor-localization service provides the latter. Such services are provided by the two following components, respectively.

2) Self-localization component: The self-localization component provides the agent with data about its own position. The component provides the following interface:

```
Interface ISelfLocalization{
  getSelfPosition(out Configuration c);
}
```

Operation getSelfPosition returns the current agent configuration.

3) Neighbor-localization components: The neighbor-localization component provides each vehicle with configurations of neighboring vehicles. The component provides the following interface:

```
Interface INeighbourLocalization{
  getParameters(out Parameters[ ] p);
}
```

The operation getParameters returns the parameters of all neighbors necessary for the correct execution of each collision avoidance algorithm.

4) Actuation component: The actuation component sets the desired linear velocity and angular velocity of the vehicle. The implementation of this component is strictly related to the dynamic of the vehicle. The component provides the following interface:

```
Interface IActuation{
  set(in int8 aVelocity, in int8 lVelocity);
}
```

Operation set sets as angular and linear velocity the values specified by the aVelocity and lVelocity arguments, respectively.

The SCC fulfills the communication security requirements in terms of confidentiality, integrity, and authenticity. The component implements the same interface as the network component:

```
Interface INetwork{
  send(in Message m, in Address a)
  receive(out Message m, out Address a)
}
```

By doing so, the SCC can be inserted and removed without affecting the other components. This allows us to reconfigure the software architecture by inserting the SCC only when needed.

Operationally, the SCC intercepts incoming or outgoing messages and applies to them the cryptographic transformations specified by the secure communication protocol. The actual specification and implementation of the protocol depends on several factors including the kind of embedded computing device and the hardware and software platform on which the SCC is deployed. The component can be implemented via software. However, if an hardware cryptographic device is present, the component can encapsulate and abstract the cryptographic services offered by that device.

5) The rekeying component: The rekeying component performs key distribution and revocation and updates the key repository on the vehicle. Usually, the keys are distributed in such a way that both confidentiality and authenticity are guaranteed. Operationally, the rekeying component receives a new key and performs the cryptographic transformation specified by the rekeying protocol to guarantee the key confidentiality. If required, it also verifies that the key comes from a trusted part.

In distributed rekeying protocols, the vehicles could generate the keys and securely transmit them to other nodes. In this case, the rekeying component provides the following interface:

```
Interface IRekeying{
  void renewKey(in Key k, in KeyValue v);
}
```

The operation renewKey renews key k with the new value v. The implementation of Key and KeyValue depends on the actual implementation of the rekeying protocol.

6) The ALC: On downloading a new software component through the network, a vehicle needs a proof that the component comes from a trusted source (component authenticity) and that the component has not been modified (component integrity). The ALC downloads a component from a remote trusted source, buffers the component during the downloading, verifies the component authenticity, and finally loads the component into memory from the buffer. The ALC can also guarantee the component confidentiality, if necessary. This component provides the following interface:

```
Interface IAuthLoad{
  load(in String cname, out ComponentType t);
}
```

The operation load downloads the component whose name is specified by the string cname and returns the component type.

Architecture Implementation

In this section, we present our implementation of the architecture services described in the "Agent Architecture" section.

Collision Avoidance Service

The collision avoidance service is one of the two services offered from the CAC. It coordinates the motion of vehicles within the group, preventing collisions and guaranteeing that each vehicle eventually accomplishes its individual task. The service implementation is based on a decentralized collision avoidance policy, called generalized roundabout policy (GRP), that has been recently proposed for vehicles evolving on the plane [14]. The GRP is now briefly reported for the reader's convenience. However, a complete, formal, and detailed description of it can be found in the cited literature with references to other existing decentralized conflict avoidance approaches.

Since we are dealing with heterogeneous vehicles, we consider vehicles with nontrivial kinematics; for example, they are not able to stop their motion and have nonholonomic constraints. Those assumption are not restrictive since vehicles that are able to stop or are holonomic can always perform trajectories obtained with the proposed policy. Indeed, we consider a number of vehicles moving on the plane at a constant speed, along paths with bounded curvature. The state of each vehicle is represented by the coordinates (x, y) and the heading angle θ. According to the policy, a first circle is assigned to each vehicle, called the safety disk, being the circle centered at the vehicle position (x, y) with heading given by θ. A collision is said to occur whenever two or more safety disks overlap.

As mentioned earlier, the proposed policy also applies to vehicles that cannot stop their motions. For dealing with such a case, the policy defines a reserved disk for each vehicle as the circle that contains the path traveled by the safety disk when its associated vehicle turns right at the minimum allowable curvature. The center of a reserved disk can easily be obtained from its vehicle state, and its heading is directly inherited from that of the corresponding vehicle. In spite of the vehicle constraint, the motion of the reserved disk can be stopped at any time by making the vehicle turn right at the minimum curvature rate.

Referring to Figure 2, suppose that each vehicle has to reach a desired final position and heading to accomplish its task. The motion strategy followed by the vehicle is based on four distinct modes of operation, each assigning a suitable value to the control input (i.e., curvature rate) of the vehicle. Each vehicle enters the straight mode if the motion along the line directed toward the desired configuration is permitted, that is, a motion in that direction does not cause an overlap with other reserved disks. Whenever its reserved disk becomes tangent to the one of another vehicle, a test is made based on the current motion heading θ. If a further movement in the direction specified by θ causes an overlapping, then the vehicle enters the hold mode. Otherwise, the vehicle is able to proceed and remains in the straight mode. When the hold mode is entered, the vehicle's curvature rate is set to the minimum allowable, and the motion of its reserved disk is stopped. As soon as the vehicle heading is permitted but not directed toward the target destination, the vehicle enters the roll mode and tries to go around the other reserved disk. This is achieved by selecting a suitable value for the curvature rate of the vehicle such that the two disks never overlap. An example of possible trajectory of a vehicle that moves according to the GRP is pictorially depicted in Figure 2.

The proposed policy satisfies the safety and scalability requirements [14]. In particular, the decentralized characteristic of the policy allows the CAC to be implemented on board of the vehicle. As a matter of fact, each vehicle is able to make a safe decision about its motion, based only on the locally available information. This information consists of the position and orientation of vehicles that are within a certain sensing or communication radius. For this reason, each vehicle communicates its state via the wireless network, though it is not required to explicitly declare its task or goal. The policy can be easily adapted in case of nonexact information on neighbors by enlarging the reserved disk radius and relaxing switching conditions between modes.

Group Membership Service

The second service offered by the CAC is the GMS. The motion strategy described in the "Collision Avoidance Service" section guarantees that no collision will occur among vehicles belonging to the group (safety property) and that all the vehicles eventually reach their final destinations (liveness property). These two properties are guaranteed provided that initial and final vehicles' configurations satisfy suitable conditions. In particular, safety is obtained if the vehicles' reserved disks do not initially overlap whereas liveness is guaranteed if the vehicles' destinations are not concentrated in the plane. Details on the target sparsity requested for liveness can be found in [14].

Taking into account the fact that vehicles can dynamically join or leave the group, the GMS purpose is to guarantee that such conditions are never violated. Thus, on joining, a new vehicle sends the configuration of its reserved disk to the GMS that verifies whether its entrance may compromise the overall system safety and liveness (join phase). Later, on reaching its final destination, the vehicle leaves the group and alerts the GMS that cancels its data (leave phase). This event may allow new vehicles

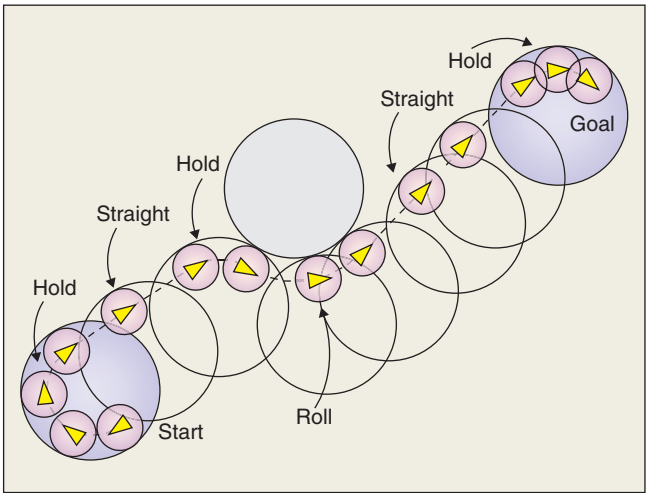


Figure 2. *Example of possible trajectory of a vehicle applying the proposed collision avoidance policy. Smaller circles are safety disks and larger circles are reserved disks.*

to enter the group. The GMS must be managed by a centralized server as the conditions guaranteeing safety and liveness are based on the information provided by all the vehicles.

Self-Localization Service

The self-localization component is responsible for providing the vehicle with the information about its own state. This component can be implemented in several different ways depending on the sensors mounted onboard the vehicle. In case of very simple vehicles, a possible implementation is a set of cameras that monitor the environment and detect position and direction of motion for all the vehicles. The data are collected in a centralized server that will send to any vehicle through the wireless network its own state information.

Otherwise, if some of the vehicles have some localization sensors, the component may have an onboard implementation based on the particular sensors. For example, in case of sonar, the localization component may be implemented as it has been proposed in [15].

Secure Communication Service

The secure communication service is provided by the SCC on board of vehicles that implements a secure communication protocol. Conceptually, a secure communication protocol is defined as a set of rules, each of which consists of a transformation, a cryptographic suite, and a set of selectors. A transformation specifies the set of cryptographic processing to be applied to messages before or after sending or receiving them to/from the network. A transformation can be either a cryptographic primitive or a combination of primitives. Cryptographic primitives can use cryptographic keys. A cryptographic suite specifies the actual cryptographic primitives, and the related keys, to be used in a transformation. Keys are specified by a key unique identifier. Finally, selectors make it possible to specify which messages a transformation has to be applied to. Selectors include at least the type of message (e.g., the port), the destination address, the source address, whether the message is incoming or outgoing, and so forth.

For example, let us assume that both confidentiality and integrity must be guaranteed for messages addressed to port p. One way to achieve these goals is to hash and encrypt the message according to the following transformation $t : \mathrm{E}(m \| \mathrm{H}(m))$, where E specifies symmetric encryption, H specifies hash, and $\|$ is the concatenation operation. For example, if we would like to use the hash function SHA-1 [16] and the symmetric cipher RC5 [17] keyed by the K key, then we specify the cryptographic suite $c : \langle \mathrm{e} = \mathrm{RC5}, \mathrm{keyid} = K; \mathrm{H} = \mathrm{SHA} - 1 \rangle$. Finally, selectors specifying that the relevant messages are addressed to port p are $s : \langle \mathrm{port} = p, \mathrm{direction} = \mathrm{outgoing} \rangle$. It follows that the secure communication protocol is specified by the rule $\langle s, t, c \rangle$.

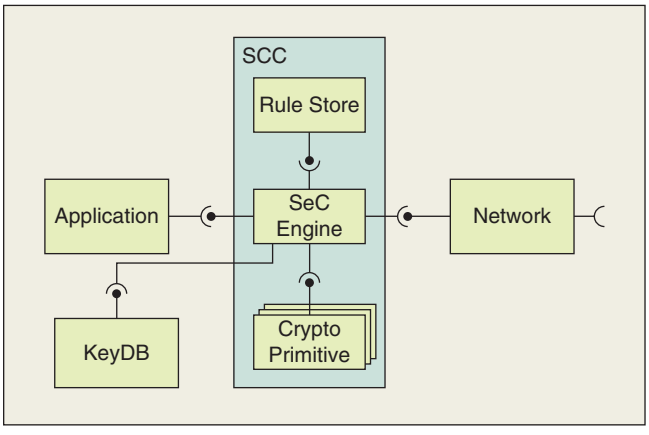


Figure 3. *The secure communication component.*

SCC is itself conceptually structured in components as specified in Figure 3. When the application sends or receives a message through network, SecEngine intercepts the message, retrieves the rule whose selector matches the message from the RuleStore, and applies the corresponding transformations to the message using the specified cryptographic suite implemented by CryptoPrimites. If the performed algorithms need cryptographic keys, SCC retrieves them from the KeyDB component.

Rekeying Service

The rekeying service is managed by the centralized rekeying server (RKS). So, the vehicles only have to verify the authenticity and the freshness of the received keys. That is, they have to verify that the key comes from the RKS and has not been used before, respectively.

Rekeying may occur either periodically, as requested by good cryptographic practices, or on events such as the leaving of a vehicle. So, GMS has to inform RKS that a vehicle leaves the group communication so that RKS distributes the new group key to all vehicles except the leaving one. The scalability of the rekeying service depends on the chosen rekeying protocol.

In our implementation, we chose S^2RP, the secure and scalable rekeying protocol for devices with low-computational capabilities [18]. S^2RP guarantees the key authenticity by using only one-way hash functions that are computationally affordable even by the simplest devices. In short, the key authentication mechanism levers on keychains, a technique based on the Lamport's one-time passwords. A keychain is a set of symmetric keys so that each key is the hash preimage of the previous one. Hence, given a key in the keychain, anybody can compute all the previous keys, but nobody can compute any of the next keys. Keys are revealed in the reversed order with respect to creation. Given an authenticated key in the keychain, the vehicles can authenticate the next keys by simply applying an hash function.

To reduce the communication overhead, RKS maintains a tree structure of keys according to S^2RP (Figure 4). Each internal node is associated with a keychain, whereas each leaf is associated with a vehicle. More in detail, a leaf is associated with the symmetric vehicle-key that the corresponding vehicle secretly shares with RKS. Let us refer to the last-revealed key associated with the node j as K_j and to the hash preimage of K_j as K_j^{nxt}. Each vehicle stores the key K_j if the subtree rooted at the node j contains the leaf associated with the vehicle-key. Hence, the key K_1 associated to the tree root is shared by all group members and it acts as the group key. Let us suppose vehicle D leaves the group. All its keys K_j with $j \in \{1, 2, 5\}$ are considered compromised and RKS has to securely broadcast the new keys K_j^{nxt} with $j \in \{1, 2, 5\}$. The rekeying messages are shown in the right-hand side of Figure 4, where $E(K, K^\star)$ is the encryption of key $K^\star$ by using the key K. So, the rekeying protocol is scalable because RKS has to broadcast

$\mathcal{O}(\log n)$ messages where n is the number of vehicles. The rekeying component constitutes the vehicle side of the rekeying service. The component is implemented by an security controller component SCC and an AuthKey component (Figure 5). SCC performs the secure communication and is responsible for guaranteeing the key confidentiality whereas the AuthKey component performs the check required by the keychain for verifying the key authenticity.

Authenticated Loading Service

The authenticated loading service is managed by the centralized authenticated loading server (ALS) that is responsible for guaranteeing the component authenticity.

A typical approach to authenticate a component on downloading consists in authenticating it as a whole. However, this approach requires that the component is entirely received before being verified, and this can be exploited by an adversary to mount a denial of service attack. More in detail, an attacker can make the device waste resources by causing it to buffer long strings of bytes that in the end fail the authenticity verification. An alternative approach is based on the observation that a component is typically transmitted in several packets [12]. If every packet is authenticated, a device stores only authenticated material and reduces the risk of denial of service at minimum. Nevertheless, this solution introduces overhead as now each packet needs to be authenticated.

A trade-off between security and performance can be achieved by authenticating bursts of packets. A burst contains a fixed predefined number N_B of packets. If N is the total number of packets conveying the components, N_B is comprised between 1 and N. Each burst is linked to the one that will be transmitted next by a hash function. ALS computes the hash of each burst and transmits the result with the previous burst. The hash value associated with the last transmitted burst is filled with the null value. It follows that, if the vehicle can authenticate the first burst, then it can sequentially authenticate all the subsequent bursts. On receiving a burst, the vehicle computes the hash and compares it with the hash value conveyed by the previously received burst. If the two values are equal, the received burst is authentic.

The authenticity of the first burst must be proven in a different way. In a scenario with many vehicles equipped with reduced computing capabilities, the digital signature might not be efficient. Therefore, we chose to prove the authenticity of the first burst by means of a message authentication code (MAC) computed with the pairwise key that the vehicle secretly shares with ALS or with the group key (see "Rekeying Service" section). Let us consider the example in Figure 6. Given $N = 6$ and $N_B = 3$, each burst contains two component-packets and the hash of the next burst. The first burst also contains an authenticator constituted by a MAC computed with the current group key. In this method, particular attention must be paid to whether an adversary can capture a device or not. Whenever a device is suspected of being compromised, the rekeying service has to revoke and then redistribute the group key to every device except the suspected one.

It follows that the proposed authentication scheme is efficient in that it requires $1 \leq N_B \leq N$ hash function computations and the authentication of the first burst. It is also flexible in that the value of N_B and the authentication method for the first burst (MAC or digital signature) are design parameters. It is also worthwhile to notice that the proposed scheme does not negatively affect scalability especially if one considers that the choice of the design parameters is not influenced by the number of vehicles to which a component has to be sent. Of course, a component could be potentially broadcast to all vehicles. In this case, the broadcast protocol is crucial for scalability. However, this is a general problem that is beyond our interests and is not addressed in this article. If necessary, we will resort to proposals in the literature [21].

The ALC constitutes the vehicle side of the authenticated loading service. In principle, the component includes an SCC and an AuthComp component (Figure 7). The SCC performs the secure communication protocol aimed at protecting packets

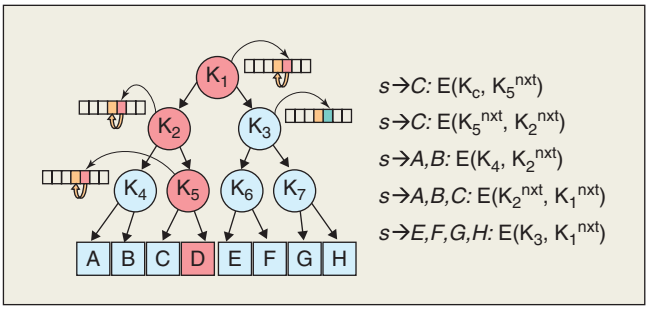


Figure 4. *Hierarchical structure of key chains in* S^2RP.

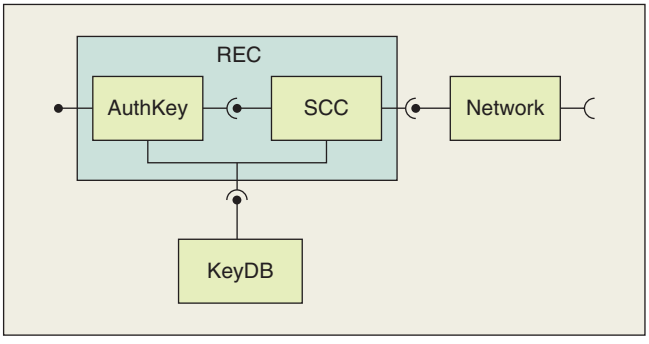


Figure 5. *The rekeying component.*

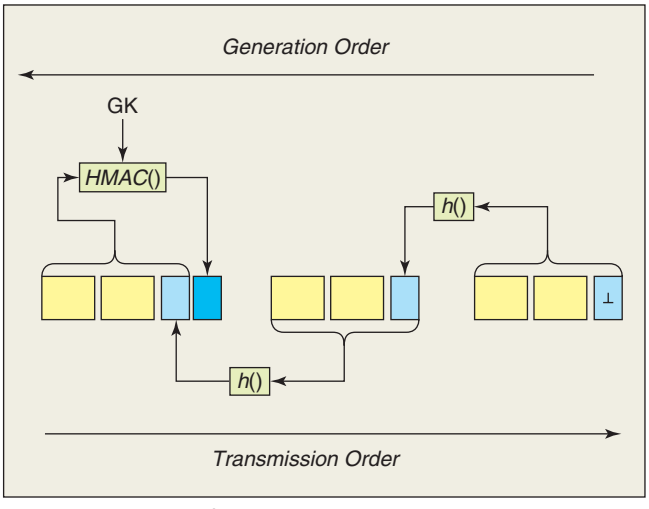


Figure 6. *A chain of bursts.*

carrying the component, whereas the AuthComp component is responsible for component buffering and authentication.

Platform Prototype

This section describes a platform prototype that has been realized according to the proposed architecture. The platform is composed of a fixed main infrastructure and a number of homogeneous mobile robotic vehicles. As already mentioned, our architecture implementation is tailored to a large number of low-cost vehicles equipped with limited sensor systems. Indeed, vehicle prototypes have been developed with such requirements. Details about vehicle hardware and software components are reported in the following subsections.

Vehicle Prototype

Robotic vehicles have been built, consisting of a chassis of 14 cm × 13 cm × 9 cm size that hosts motors, batteries, and electronics. The vehicles are also equipped with a Tmote-Sky sensor board, which enables communications with the 802.15.4 protocol, and some programmable system on chip (PSoC) mixed-signal array controllers that serve as servodriving, odometry, and CAC implementation (Figure 8). The Tmote-Sky board has been adopted for its high compatibility with the Zigbee protocol and low power consumption. An interface between microcontrollers and Tmote-Sky has been developed. To take advantage of every resources offered by all the units, the load of computing algorithms has been divided among the Tmote-Sky and PSoCs CPUs. With such an approach, performances have been improved with respect to the performance achievable only with the Tmote board. Indeed, a 40-Hz CAC computation and a 200-Hz servo driver control have been obtained. Extensive tests have been done on a test bed composed of two and three robotic vehicles (see Figure 9 for a two-vehicle example) as reported in the "Experimental Results" section.

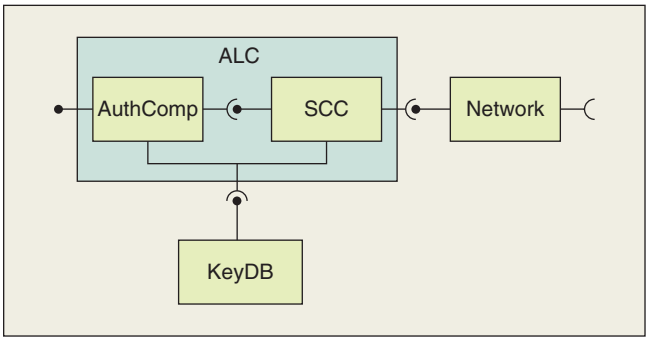


Figure 7. *The ALC.*

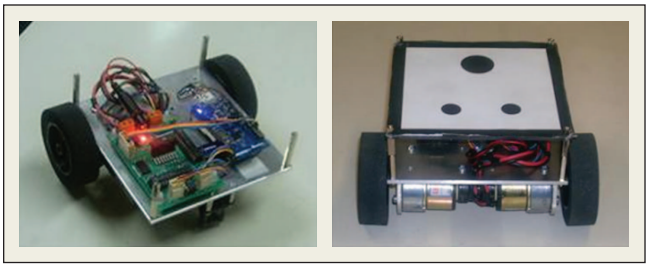


Figure 8. *Prototype of the vehicle.*

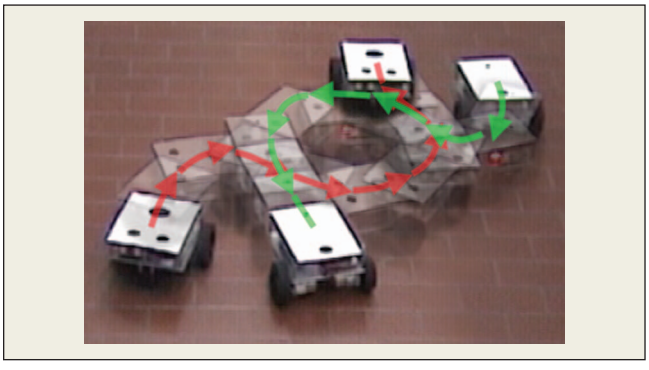


Figure 9. *GRP trajectories of two vehicles with assigned initial and final configurations.*

The chosen operating system is Contiki [19], which was developed for devices with low memory and computational capabilities. The implementation of the component model for Contiki is known as the component runtime kernel [13]. It has been chosen for the Contiki operating system since it is a lightweight and flexible operating system for tiny networked sensors and has a dynamic structure that allows to replace components during runtime.

As a consequence of the low-cost implementation of the vehicle, the peer- and self-localization have not been implemented on board. Indeed, the excessive cost and the insufficient precision of available sensor technologies have induced us to implement the location service (LS) localization service. The self-localization is achieved by means of aperiodic requests to a fixed infrastructure localization service that relies on computer vision to identify the vehicles' states. Furthermore, the peer-localization module is performed by listening to periodic messages of other vehicles communicating data about position and reserved disk radius.

From a security perspective, each vehicle implements an early prototype of the SSC (see "Secure Communication Service" section), the rekeying protocol (see "Rekeying Service" section), and the authenticated loading protocol (see "Authenticated Loading Service" section). The security controller uses Skipjack as a symmetric cipher to encrypt application and rekeying messages; the rekeying protocol (AuthKey component) uses SHA-1 to build and verify keychains; and finally, the authenticated loading protocol (AuthComp) uses a keyed-hash message authentication code based on SHA-1 to authenticate the first slice of a component. Furthermore, in our prototype, we do not consider it necessary to protect the confidentiality of a component during downloading. So ALC contains only the AuthComp component responsible for the authenticity of component slices.

Infrastructure

The infrastructure enables the LS by detecting the states of every vehicles and providing the common reference frame shared within the group. Second, the infrastructure enables the RKS by generating new keys and distributing them when necessary.

Off-the-shelf cameras have been exploited for monitoring the environment. Vision algorithms have been developed to

identify the state of every vehicle by means of markers placed over the chassis. By precisely calibrating the cameras, an accurate estimate of the position and orientation of each vehicle has been obtained. Despite the use of the low-cost cameras, the chosen algorithms are robust to illumination changes in an indoor test bed. Cameras and algorithms are hosted on a system composed of three PCs, connected in a LAN.

Communication Protocol

To test the platform, a simple ad hoc wireless communication protocol has been implemented for both periodic (required by the LS) and aperiodic communications (required by other services). The communication protocol realizes a time-division multiple access protocol briefly described in the following.

A central authority is responsible for the temporal synchronization and a time slice-based subdivisions. In large, multihop wireless networks, an accurate distributed time synchronization is a nontrivial problem [20]. Each time slice is composed of $2N + K$ slots, where N is the number of vehicles, and $K \geq 2$ is an integer value used to avoid starvation. Any time the group membership changes, the communication protocol assigns a slot index and the time slice duration to each vehicles. A time slice is composed of two phases: periodic communication phase and aperiodic communication phase. The periodic communication phase starts with a synchronization message, and it is composed of N slots. Every vehicle has its own slot to perform peer localization (broadcast of position and reserved disk radius), and it can submit a request to the central authority for self-localization and permission for further aperiodic communications. At the end of this phase, all vehicles have collected information regarding neighboring vehicles. In the aperiodic communication phase, the central authority replies to requests for self-localization (in no more than N slots), and it gives acknowledgments to vehicles that performed a request for an aperiodic slot.

Experimental Results

Components proposed in previous sections have been separately implemented and tested to verify their effectiveness before the integration of the overall platform. Details on technical data of the implemented components are reported.

In the proposed implementation of the SCC and the ALS, the time required for encrypting a packet of 48 B is 9.92 ms by using by SkipJack whereas the time for applying the hash function SHA-1 on a packet of 28 B is 14.3 ms. Furthermore, the time required for key authentication is 32.2 ms by using SkipJack as symmetric cipher and SHA-1 as hash function. To authenticate a component of 1,264 B, the computational overhead is 1.84 S.

The localization service is provided with resolution of 0.23 cm and 0.03 rd in an environment of 290 cm $\times$ 133 cm using two cameras. The truncation error during the transmission process is at most 0.04 mm for lengths and 10^{-5} rd for angles. The average errors measured during experiments is around 1 cm and 0.06 rd for lengths and angles, respectively. On a PC Pentium of 43 GHz with 1 GB RAM, the proposed implementation is able to process ten frames per second.

As reported in the "Platform Prototype" section, a basic component for the wireless network management has been implemented to allow wireless communication between agents. Each localization packet is composed of 12 B where the first is the identifier of the localized vehicle, while the others represent the status of the vehicle (x, y, and θ on two bytes) and the position of its center (x_c and y_c on two bytes). The short time needed to send a localization packet (approximately 0.006 s) allows the system to manage a large number of vehicles under the bandwidth constrains of the 802.15.4 wireless protocol.

Finally, several experiments have been performed to prove the effectiveness and the reliability of the overall platform. As the particular task for each robot involved does not influence the testing of the platform, a scenario in which two or three vehicles were assigned a final configuration without any specific task has been considered. The trajectory performed by two robots during an experiments is shown in Figure 9.

In all the conducted experiments, vehicles have forward velocity of 3cm/s, angular velocity during the hold mode of 0.385rd/s, reserved disk radius of 13 cm, and safety disk diameter of 15 cm. On each vehicle a battery pack of 9.6 V, 1,800 mA has been mounted to provide energy to the Tmote-Sky and the PSoCs. With such a power supply, this kind of experiments can be conducted for around 90min. It is important to notice that during the experiments, intensive use of the wireless communication is required by the architecture. Most of the energy supply is used for vehicles' motion, while the communication and the security protocols are less energy demanding.

Partial overlapping of reserved disks has occurred during experiments for at most 4.1cm because of nonexact integration of motion and delay on data communicated through the network. Indeed, as reported in the "Collision Avoidance Service" section, the GRP policy ensures the safety of the system only theoretically. In the real framework, the system safety can be recovered enlarging the reserved disk size according to estimated errors on the localization system and to a forecast of communication delays.

Conclusions and Future Work

A scalable platform for decentralized traffic management of a multiagent system has been proposed. Safety of the platform is achieved with a cooperative conflict avoidance policy. Security of communications among vehicles with respect to potential external adversaries is obtained through use of cryptographic keys and rekeying policies. A prototypical implementation of the architecture has been described, and some experimental results have been reported. Future work will be devoted to addressing further decentralization of the check-out and security procedures, intrusion detection, and noncollaborative collision avoidance protocols.

Acknowledgments

We thank M. Dell'Unto and G. De Paolis for having contributed in the implementation of the platform prototype. The work has been done with partial support from EC project RUNES (Contract IST-2004-004536) and EC Network of Excellence HYCON (Contract IST-2004-511368).

Keywords

Networked heterogeneous mobile robots, component-based platform, collision avoidance, safe communication.

References

[1] B. Sinopoli, C. Sharp, L. Schenato, S. Schaffert, and S. Sastry, "Distributed control applications," *Proc. IEEE*, vol. 91, no. 8, pp. 1235–1246, 2003.

[2] S. Graham and P. Kumar, Eds., "Proceedings of PWC 2003," in *Personal Wireless Communication* (Lecture Notes in Computer Science), vol. 2775. New York: Springer-Verlag, 2003.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Comput. Netw.*, vol. 38, no. 4, pp. 393–422, 2002.

[4] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Trans. Automat. Contr.*, vol. 48, no. 6, pp. 988–1001, 2003.

[5] H. Tanner, A. Jadbabaie, and G. Pappas, "Flocking in fixed and switching networks," *IEEE Trans. Automat. Contr.*, vol. 52, no. 5, pp. 863–868, 2007.

[6] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Trans. Automat. Contr.*, vol. 51, no. 3, pp. 401–420, 2006.

[7] S. Oh, L. Schenato, P. Chen, and S. Sastry, "Tracking and coordination of multiple agents using sensor networks: System design, algorithms and experiments," *Proc. IEEE*, vol. 95, no. 1, pp. 234–254, 2007.

[8] P. Ogren, E. Fiorelli, and N. Leonard, "Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment," *IEEE Trans. Automat. Contr.*, vol. 49, no. 8, pp. 1292–1302, 2004.

[9] J. Fax and R. Murray, "Information flow and cooperative control of vehicle formations," *IEEE Trans. Automat. Contr.*, vol. 49, no. 9, pp. 1465–1476, 2004.

[10] Y. Mostofi, T. Chung, R. Murray, and J. Burdick, "Communication and sensing trade-offs in decentralized mobile sensor networks: A cross-layer design approach," in *Proc. Int. Symp. Information Processing in Sensor Networks*, 2005, pp. 118–125.

[11] N. Lynch, *Distributed Algorithms*. San Mateo, CA: Morgan Kaufmann, 1996.

[12] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. 2nd Int. Conf. Embedded Networked Sensor Systems*, Baltimore, MD, 2004, pp. 81–94.

[13] P. Costa, G. Coulson, C. Mascolo, G. P. Picco, and S. Zachariadis, "The runes middleware: A reconfigurable component-based approach to networked embedded systems," in *Proc. 16th IEEE Int. Symp. Personal, Indoor and Mobile Radio Communications (PIMRC 2005)*, Berlin, Germany, 2005, vol. 2, pp. 806–810.

[14] L. Pallottino, V. G. Scordio, E. Frazzoli, and A. Bicchi, "Decentralized cooperative policy for conflict resolution in multi-vehicle systems," *IEEE Trans. Robot.*, vol. 23, no. 6, pp. 1170–1183.

[15] P. Alriksson, J. Nordh, K.-E. Arzen, A. Bicchi, A. Danesi, R. Schiavi, and L. Pallottino, "A component-based approach to localization and collision avoidance for mobile multi-agent systems," in *Proc. European Control Conference*, 2007, pp. 4285–4292.

[16] Secure Hash Standard, National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 180-1, 1993.

[17] R. Rivest, "The RC5 encryption algorithm," in *Fast Software Encryption—Second International Workshop*, vol. LNCS 1008. Heidelberg, Germany: Springer-Verlag, 1995, pp. 86–96.

[18] G. Dini and I. M. Savino, "S^2RP: A secure and scalable rekeying protocol for wireless sensor networks," in *Proc. 3rd IEEE Int. Conf. Mobile Ad-Hoc and Sensor systems*, 2006, pp. 457–466.

[19] A. Dunkels, B. Gröonvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in *Proc. IEEE Workshop on Embedded Networked Sensors*, 2004, pp. 455–462.

[20] R. Solis, V. Borkar, and P. Kumar, "A new distributed time synchronization protocol for multihop wireless networks," in *Proc. 44th IEEE Conf. Decision and Control*, 2005, pp. 2734–2739.

[21] J. Wu and F. Dai, "Efficient broadcasting in ad hoc wireless networks using directional antennas," in *IEEE Trans. Parallel and Distributed Systems*, 2006, vol. 17, no. 4, pp. 335–347.

Antonio Bicchi is a professor of automatic control and robotics at the University of Pisa. His main research interests include dynamics, kinematics, and control of complex mechanical systems, including robots, autonomous vehicles, and automotive systems; haptics and dexterous manipulation; and theory and control of nonlinear systems, in particular, hybrid (logic/dynamic, symbol/signal) systems. He has published more than 200 articles in international journals, books, and refereed conferences.

Antonio Danesi works in software specification for space launchers for European Space Agency. He obtained his Ph.D. in robotics and automation in 2007 at Centro "E. Piaggio," Università di Pisa, with a research on component-based control software on networked mobile agents for visual navigation and mapping. He participated in European project RUNES for the advanced control group.

Gianluca Dini received the Laurea degree in electronic engineering from the University of Pisa in 1990 and a Ph.D. in computer engineering from Scuola Superiore S. Anna, Pisa, in 1995. Since 2000, he has been an associate professor of computer engineering at the University of Pisa. His main research interests are in distributed computing, with particular reference to security and fault tolerance.

Silvio La Porta received the Laurea degree in computer engineering in 2006 from the Faculty of Engineering, University of Pisa. Currently, he is a Ph.D. student in computer engineering at the Department of Ingegneria della Informazione, Pisa. His research interests are in network security and network forensics.

Lucia Pallottino received the Laurea degree in mathematics from the University of Pisa in 1996/97 with a thesis in numerical analysis. She received a Ph.D. in robotics and industrial automation at the Department of Electrical Systems and Automation of the University of Pisa in 2002. She is currently a researcher in the Interdepartmental Research Center "E. Piaggio" in the University of Pisa. Her main research interests within robotics are in motion planning and control for nonholonomic vehicles, air traffic management systems, and quantized control.

Ida M. Savino received the Laurea degree in computer engineering in 2004 from the Faculty of Engineering, University of Pisa. Currently, he is a Ph.D. student in computer engineering at the Department of Ingegneria della Informazione, Pisa. His research interests are in security of networked embedded systems.

Riccardo Schiavi received the Laurea degree in computer engineering in 2004 from the Faculty of Engineering, University of Pisa. Currently, he is a Ph.D. student in robotics automation and bioengineering at the Centro Interdipartimentale di Ricerca "E. Piaggio" and at the Dipartimento di Sistemi Elettrici e Automazione. His research involves the development and control of embedded systems.

Address for Correspondence: Lucia Pallottino, Centro "Enrico Piaggio," via Diotisalvi, 2, 56100 Pisa, Italy. E-mail: l.pallottino@ing.unipi.it.