A component-based approach to the design of networked control systems

Karl-Erik Årzén, Antonio Bicchi, Gianluca Dini, Stephen Hailes,
Karl H. Johansson, John Lygeros, and Anthony Tzes

***Abstract*— Component-based techniques revolve around composable, reusable software objects that shield the application level software from the details of the hardware and low-level software implementation and vice versa. Components provide many benefits that have led to their wide adoption in software and middleware developed for embedded systems: They are well-defined entities that can be replaced without affecting the rest of the systems, they can be developed and tested separately and integrated later, and they are reusable. Clearly such features are important for the design of large-scale complex systems more generally, beyond software architectures. We propose the use of a component approach to address embedded control problems. We outline a general component-based framework to embedded control and show how it can be instantiated in specific problems that arise in the control over/of sensor networks. Building on the middleware component framework developed under the European project RUNES, we develop a number of control-oriented components necessary for the implementation of control applications and design their integration. The paper provides the overview of the approach, discusses a real life application where the approach has been tested and outlines a number of specific control problems that arise in this application.**

I. Introduction

Networked embedded systems play an increasingly important role and affect many aspects of our lives. By enabling embedded systems to communicate, new applications are being developed in areas such as health-care, industrial automation, power distribution, rescue operations and smart buildings. Many of these applications will result in a more efficient, accurate and cost effective solution than previous ones. The European Integrated Project Reconfigurable Ubiquitous Networked Embedded Systems (RUNES) [8] brings together 21 industrial and academic teams in an attempt to enable the creation of large scale, widely distributed, heterogeneous networked embedded systems that inter-operate and adapt to their environments. The inherent complexity of such systems must be simplified if the full potential for networked embedded systems is to be realized. The RUNES project aims to develop technologies (system architecture, middleware, networking, control etc.) to assist in this direction, primarily from a software and communications standpoint.

Networked control systems impose additional requirements that arise from the need to manipulate the environment in which the networked systems are embedded. Timing and predictability constraints inherent in control applications are difficult to meet in general, due to the variations and uncertainties introduced by the communication system: delays, jitter, data rate limitations, packet losses etc. For example, if a control loop is closed over a wireless link, it should tolerate lost packets and be able to run in open loop over periods of time. Resource limitations of wireless networks also have important implications for the control design process, since limitations such as energy constraints for network nodes need to be integrated into the design specifications. The added complexity and need for re-usability in the design of control over wireless networks suggests a modular design framework.

In this paper, we propose a component-based approach to handle the software complexity of networked control systems. A general framework is presented and it is shown how it can be instantiated in specific problems that arise in control over wireless sensor networks as well as in control of network and communication resources. The proposed component framework hides network programming details from the control system designer. The components are well-defined entities that can be replaced without affecting the rest of the systems. It is shown how they can be developed and tested separately and integrated later. Building on the middleware component framework of RUNES, we develop a number of control-oriented components necessary for the implementation of control applications and design their integration. The paper provides the overview of the approach, discusses a real life application where the approach can be used and outlines a number of specific control problems that arise in this application. Companion papers [27], [12], [14], [15] provide the details of the implementation of specific components to address these control problems, as well as experimental validation results.

The paper starts by presenting a brief overview of middleware and component frameworks in general, and those targeted to networked embedded systems in particular (Section II). The RUNES tunnel disaster relief scenario that

Research supported by the European Commission under the integrated project RUNES, FP6-IST-004536.

K.-E. Årzén is with the Department of Automatic Control, Lund University, Sweden.

A. Bicchi is with the Department of Electrical Systems and Automation and Centro I.R. *E. Piaggio*, University of Pisa, Italy.

G. Dini is with the Department of Information Engineering, University of Pisa, Italy.

S. Hailes is with the Department of Computer Science, University College, London, U.K.

K. H. Johansson is with the School of Electrical Engineering, Royal Institute of Technology, SE-100 44 Stockholm, Sweden; `kallej@ee.kth.se`. Corresponding author.

J. Lygeros is with the Automatic Control Laboratory, ETH Zurich, Switzerland.

A. Tzes is with the Department of Electrical and Computer Engineering, University of Patras, Greece.

serves to focus our work is then described in Section III. An overview of control problems that arise in the scenario is also given in the same section: maintaining the connectivity of a wireless sensor network in an adverse environment, utilizing the resources of the network itself (e.g., wireless transmission power control) and those of mobile robots (e.g., to replace missing nodes). In Section IV we discuss the components that need to be implemented to address the specific control problems in the task on physical network reconfiguration; the details of the development of these components and their experimental testing are given in the companion papers. Some examples of general purpose, low-level control components are also presented in the section. Section V details a security component framework, which provides an interface to protect communications among nodes. The hardware and software integration for the demonstration of the physical network reconfiguration is given in Section VI. Its validation is shown in Section VII, which describes both a computer simulation of the scenario and some preliminary experimental results. Some concluding remarks are given in Section VIII.

II. Middleware and Components

A. Middleware

In a component-based software system, a component is a system element offering a predefined service and able to communicate with other components. A component is a unit of independent deployment and versioning. It is encapsulated and non-context specific. It follows that components can interact with other components without knowing much of their internal structure or their execution environment (for example, their operating system or network protocols). Clearly, devising such an abstract level of interaction is a non-trivial effort. In many cases, an effective solution can be found by the judicious application of a software abstraction layer, known as middleware. Middleware mediates the interactions of a component with its environment by providing a programming interface transparent to the operating systems and to the network protocols underneath. A comprehensive survey of middleware concepts (motivated primarily for networked embedded systems) can be found in [9]. Important examples of middleware currently in use are Java Remote Method Invocation (Java RMI) [5], Microsoft Component Object Model (COM) [6], and Common Object Request Broker Architecture (CORBA) [2]. These frameworks, however, are not specifically targeted to embedded systems or distributed control systems. The resource constrained implementation platforms common in embedded and distributed control systems imply additional, severe requirements on the middleware. To meet these requirements, extensions of general purpose middleware have been developed. One such example is real-time CORBA [30], which features prioritized scheduling policies for threads and export some control parameters in the communication protocols. Even real-time CORBA, however, has several shortcomings that make its use on demanding embedded system applications problematic [9].

Several application domains have emphasized the importance of developing software infrastructures specifically tailored to the needs of the domain. For example, the automotive industry has formed the development partnership AUTOSAR [1], to achieve modularity, scalability, transferability and re-usability of software functions in vehicles. AUTOSAR strives to provide an open system architecture for automotive systems based on standardized interfaces for the different system layers. A precise component definition and an appropriate composition framework are essential to answer a variety of questions on system architectures, e.g., on synchronization and network protocols [25].

Specific control and real-time requirements on the middleware have also been investigated in recent academic software prototypes. Etherware [16] is a middleware for networked control that was recently proposed. This middleware focuses on the ability to maintain communication channels during component restarts and upgrades and to recovery from failure situations. ControlWare [31] is a middleware that utilizes feedback control for guaranteeing performance in software systems. Though not specifically targeted to embedded systems, its usefulness has been demonstrated on web server and proxy quality of service management. A tutorial overview of software technologies for reusable and distributed control systems is given in [23].

Finally, from a theoretical point of view, semantic frameworks that support composition and abstraction operations are central to the formal modeling and analysis of such distributed systems. For embedded systems (where the logic functions encoded in the computational elements have to interact with a primarily analog environment) the most relevant frameworks are those developed in the area of hybrid systems. Several such frameworks have been proposed in recent years, to support the modeling, verification, system development and simulation efforts; for an overview see [29]. Some are general purpose, while others are targeted to specific application areas [18]. Most are also supported by simulation, verification or design computer tools. A link between these theoretical developments and the middleware frameworks discussed above is just emerging as an exciting and important research area.

B. Component frameworks for networked embedded systems

The main reason for using component-based approaches in software development is to enforce re-usability. A new software application is built from existing well-tested components. The components are composed (or assembled) into applications. It is often possible to aggregate components together, forming new components.

Component-based software engineering has been successfully used in several software development projects, primarily for desktop and eBusiness applications. Within real-time embedded systems, the use of component techniques is not well-developed. For desktop applications the COM technology is most widely used. COM components are often relatively large in size, each component encompassing a substantial amount of the application functionality. Another widely used class of component models are the models that have their basis in distributed object models. These include the CORBA Component Model (CCM) [3], Enterprise Java Beans (EJB) [4], and .NET [7]. The .NET model can be viewed as an distributed evolution from COM that is especially interesting due to Common Language Runtime (CLR). CLR is a virtual machine technology that can be compared to Java's Virtual Machine. It is Microsoft's implementation of the Common Language Infrastructure (CLI) standard, which defines an execution environment for program code. The CLR executes a bytecode format into which several languages can be compiled, e.g., C# and Visual C++. Through this it is possible to integrate software components developed in different programming languages. The drawback with the approach, compared to, e.g., Java-based approaches, is that it is operating system dependent, i.e., it is only supported for Windows-based systems. Components are viewed as extended objects that can be distributed. However, each individual object still resides on a single node in the network. In these types of component models object-oriented concepts, such as classes and inheritance, are integral parts.

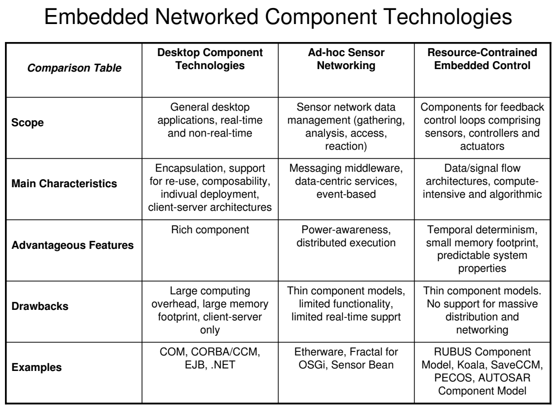
In component technologies for embedded systems, non-functional properties such as safety, timeliness, memory footprint, and dependability are of particular interest. Compared to the desktop component approaches described above the component models here are much more limited in functionality. Often the component models are intended for applications of an algorithmic nature. These applications are commonly modeled as data- or signal-driven block diagrams. Another name for this is a pipe and filter architecture. The individual components are typically smaller than in the previous component models, and the emphasis on component aggregation is larger. These component technologies are frequently inspired by the block diagram approach in Matlab/Simulink, the function block diagrams in the automation language standard IEC 61131-3, and by ordinary discrete logic gates. There are still no good examples of commercially successful component technologies for embedded systems. However, it is an area where considerable research currently is being performed.

For sensor network and mobile ad-hoc network applications, all the component technologies above are, in principle, applicable. Sensor networks are an example of a severely resource-constrained distributed implementation platform. If they are to host sensor fusion and control applications, it is quite clear that the component technologies developed for embedded systems are a natural option. Embedded control systems and sensor network applications, furthermore, have many similarities. In both cases, a component model centered around data flows is more natural than the focus on component function calls found in desktop component models. Following this path a possibility would be to develop a set of generic sensor, data fusion, control and actuator components or component types; examples along these lines are outlined in Section IV. The limited battery resources make power-awareness an important attribute of component models for sensor networks.

The different characteristics of desktop applications and sensor/actuator networks do, however, not necessarily imply that it is not possible to base a component model for sensor/actuator networks on more conventional component technologies; the development effort only becomes considerably larger. Rather than having built in support for data flows in the middleware, it has to be explicitly realized through component function calls. This is the approach that has been taken in the RUNES project.

In mobile ad-hoc network applications the resource constraints are normally less severe than in wireless sensor networks. More powerful CPUs with more memory and battery resources are often used. Hence, here the desktop-type of component technologies can be applied. The components of this type are often more application-oriented than the simpler and more generic sensor-controller-actuator components. In a mobile robot setting we may decompose the application into components for localization, path planning, collision avoidance, etc. These are the types of components of main interest the work presented here.

The table in Figure 1 summarizes embedded networked component technologies by listing their characteristics together with some advantages and disadvantages.


Embedded Networked Component Technologies

Comparison Table	**Desktop Component Technologies**	**Ad-hoc Sensor Networking**	**Resource-Contrained Embedded Control**
Scope	General desktop applications, real-time and non-real-time	Sensor network data management (gathering, analysis, access, reaction)	Components for feedback control loops comprising sensors, controllers and actuators
Main Characteristics	Encapsulation, support for re-use, composability, indivual deployment, client-server architectures	Messaging middleware, data-centric services, event-based	Data/signal flow architectures, compute-intensive and algorithmic
Advantageous Features	Rich component	Power-awareness, distributed execution	Temporal determinism, small memory footprint, predictable system properties
Drawbacks	Large computing overhead, large memory footprint, client-server only	Thin component models, limited functionality, limited real-time supprt	Thin component models. No support for massive distribution and networking
Examples	COM, CORBA/CCM, EJB, .NET	Etherware, Fractal for OSGi, Sensor Bean	RUBUS Component Model, Koala, SaveCCM, PECOS, AUTOSAR Component Model

Fig. 1. Comparison of embedded networked component technologies.

C. RUNES middleware components

Central to our efforts in developing a component-based framework for networked control is the RUNES middleware component model [10]. Even though sensor networks (and other ad hoc networks) are of central interest to RUNES, the RUNES middleware component model is closer in spirit to the desktop model discussed above, than to the embedded model. One reason for this is that the RUNES components are not only intended for the sensor nodes, but should also reside on the gateways and on the back-end computers. Another reason is that the RUNES components are also intended as a means for structuring parts of the RUNES middleware itself.

A component-based framework for networked control should enable quality of service definitions and negotiation between the designer of the control application and the middleware. The solution should combine the appropriate level of abstraction needed by control applications with a lightweight and scalable architecture. The middleware should provide the appropriate support for a wide variety of control applications, ranging from sensor networks to distributed control systems. To this end, it is of utmost importance to keep track of the level of introduced complexity. Memory consumption and communication latency are examples of fundamental parameters in the design. Our conclusion is that, even if some existing proposals attempt to cope with some of these issues, a middleware based on a comprehensive evaluation of the multifaceted requirements of networked control applications is still to come.

The RUNES middleware [10] is illustrated in Figure 2. The middleware acts as a glue between the sensor, actuator, gateway and routing devices, operating systems, network stacks, and applications. It defines standards for implementing software interfaces and functionalities that allow the development of well-defined and reusable software. The basic building block of the middleware developed in RUNES is a software component. From an abstract point of view, a component is an autonomous software module with well-defined functionalities that can interact with other components only through interfaces and receptacles. Interfaces are sets of functions, variables and associated data types that are accessible by other components. Receptacles are required interfaces by a component and make explicit the inter-component dependencies. The connection of two components occurs between a single interface and a single receptacle. Such association is called binding and is shown in more detail in Figure 6. Part of the RUNES middleware has been demonstrated to work well together with the operating system Contiki [22], which was developed for low-memory low-computation devices. The implementation of the component model for Contiki is known as the component runtime kernel (CRTK). This component framework provides for instance dynamic run-time bindings of components, i.e., during execution it allows components to be substituted with other components with the same interface.

III. Motivating scenario

This section describes the RUNES tunnel disaster relief scenario and gives an overview of some of the control problems that arise within the scenario.

A. Disaster relief scenario

One of the major aims of the RUNES project is to create a component-based middleware that is capable of reducing the complexity of application construction for networked embedded systems of all types. Versions of the component runtime kernel, which forms the basis of the middleware, are available for a range of different hardware platforms. However, the task is a complex one, since the plausible set of sensing modalities, environmental conditions, and interaction patterns is very rich. To illustrate one potential application in greater detail, the project selected a disaster relief scenario, in which a fire occurs within a tunnel, much

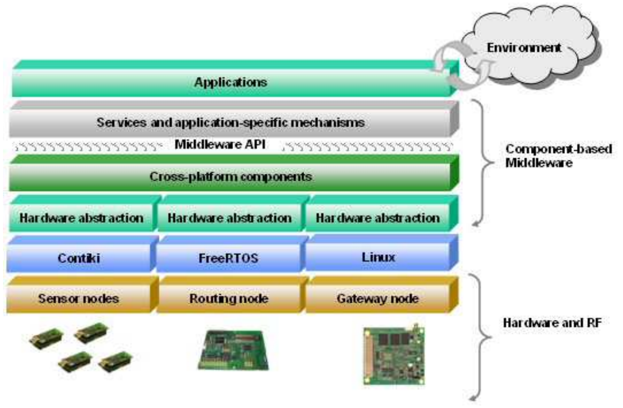


Fig. 2. Overview of the RUNES middleware platform. The component-based middleware resides between the application and the operating systems of the individual network nodes.

as happened in the Mont Blanc tunnel in 1999. In this, the rescue services require information about the developing scenario both before arrival and during rescue operations, and such information is provided by a network of sensors, placed within the tunnel, on robots, and on rescue personnel themselves. We explore the scenario in more detail below, but it should be noted this is intended to be representative of a class of applications in which system robustness is important and the provision of timely information is crucial. So, for example, much the same considerations apply in the prevention of, or response to, Chemical, Biological, Radiological, Nuclear or Explosive (CBRNE) attacks; likewise, search and rescue operations, and even industrial automation systems form application domains with similar requirements for predictability of response given challenging external conditions.

The fire-in-a-tunnel scenario deals with disaster relief activities in response to a fire in a road tunnel caused by an accident, as illustrated in Figure 3. For example, in the case of Mont Blanc, a very severe fire was caused as the result of the ignition of a lorry carrying margarine and flour. The resulting fire burned for two days, trapping around 40 vehicles in dense, poisonous, smoke, with a death toll of 37 people. Communications, lighting, and sprinkler systems failed within minutes of the fire starting with the result that Christian Comte, fire brigade chief at Chamonix, is reported to have said: *Sur le moment, on n'avait pas d'informations précises—on ne savait pas ce qui brûlait, ni à quel endroit, s'il y avait du monde à l'intérieur ou pas.* In other words, there was no precise information about what was happening: it was not clear what was burning, nor where it was, nor whether there were people inside the tunnel or not. As a consequence, firefighters entered the tunnel long past the time at which they could have made a difference, and themselves became trapped.

In the RUNES scenario, we project what might happen in a similar situation if the vision of the US Department of Homeland Security's SAFECOM programme becomes

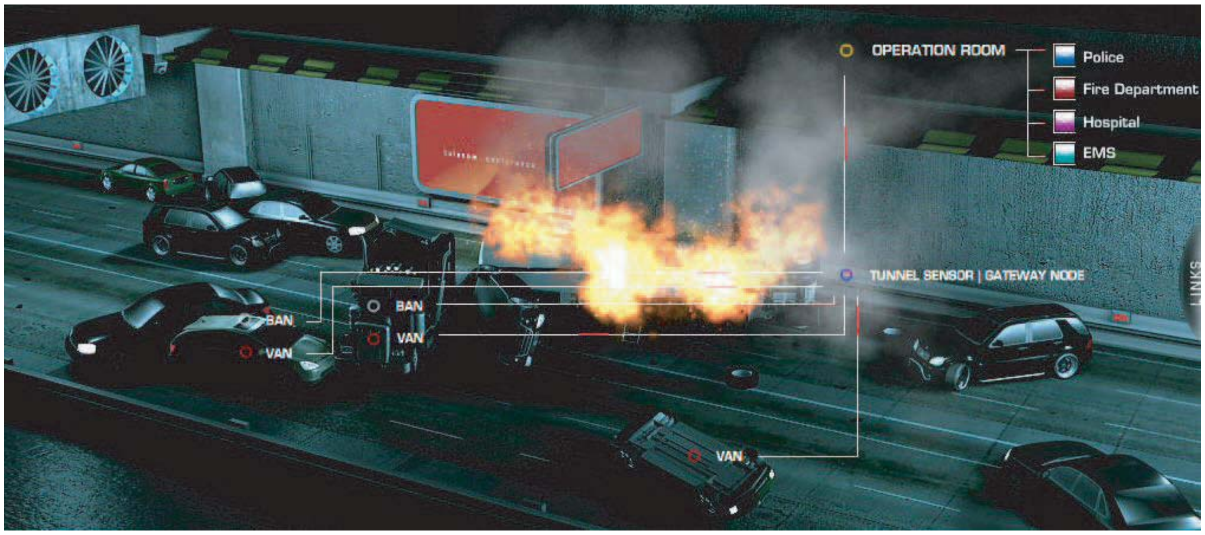


Fig. 3. Illustration of the RUNES tunnel disaster relief scenario.

a reality. The scenario is based around a storyline that sets out a sequence of events and the desired response of the system, part of which is as follows. Initially, traffic flows normally through the road tunnel; then an accident results in a fire. This is detected by a wired system, which is part of the tunnel infrastructure, and is reported back to the Tunnel Control Room. The emergency services are summoned by Tunnel Control Room personnel. As a result of the fire, the wired infrastructure is damaged and the link is lost between fire detection nodes (much as happened in Mont Blanc). However, using wireless communication as a backup, information from (for example) fire and smoke sensors continues to be delivered to the Tunnel Control Room seamlessly. The first response team arrives from the fire brigade and rapidly deploys search and rescue robots, following on foot behind. Each robot and firefighter carries a wireless communication gateway node, sensors for environmental temperature, chemical and smoke monitoring, and the robots carry light detectors that help them identify the seat of the blaze.

The role of the robots in this scenario is twofold: to help identify hazards and people that need attention, without exposing the firefighters to danger; and to augment the communications infrastructure to ensure that both tunnel sensor nodes and those on firefighters remain in contact with the command and control systems that the situation commander uses to make informed decisions about how best to respond. To accomplish this, the robots are moving autonomously in the tunnel taking into account information from tunnel sensors about the state of the environment, from a human controller about overall mission objectives, and from received signal strength measurements from the wireless systems of various nodes about the communication quality. The robots coordinate their activity with each other through communication over wireless links. Local backup controllers allow the robots to behave reasonably in the event that communication is lost.

B. Overview of control problems

The RUNES work in general and the disaster relief scenario in particular offer a number of interesting and challenging problems where control methods can make a key contribution. One can envision control algorithms being developed to control infrastructure resources; such as fans or fire extinguishing devices, control robot motion in order to localize hazards or localize injured humans and assist in removing them from the disaster area, and, last but not least, control network resources to ensure connectivity and timely delivery of crucial information. Here we will focus our attention to this last type of control problem, namely controlling network resources.

The control problem of interest is sketched in Figure 4. A set of nodes with wireless communication capabilities are deployed inside the tunnel for monitoring purposes. As soon as an emergency situation occurs, for example an accident involving many cars, the nodes need to transmit data regarding the tunnel conditions to a base station. In such a scenario, accurate and comprehensive information must be provided to the base station so that correct counter measures can be taken. It is of fundamental importance that the network would maintain connectivity, so that the flow of critical data to the base station is guaranteed. However, the network could be partitioned because of a malfunction of the nodes, caused by a fire, or because the presence of obstacles that deteriorate or even nullifies metrics of the Quality of Service.

In such a critical situation, the control application is responsible for restoring the network connectivity. This is done by sending a mobile autonomous robot inside the tunnel, see Figure 4. The robot is equipped with a radio transmitter–receiver so that it can maintain connectivity with

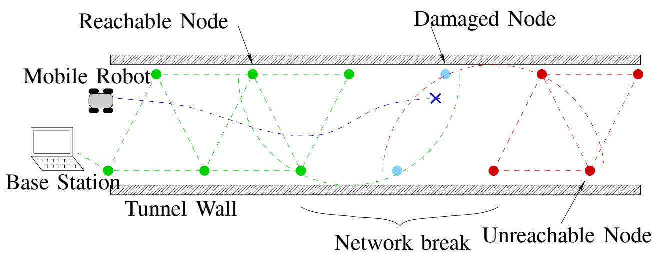


Fig. 4. Road tunnel scenario in which part of the deployed wireless network is disconnected due to two damaged network nodes. A mobile robot moves into the region of the damaged nodes to relay information from the unreachable nodes towards the base station.

the base station directly or through the deployed network. Once the base station determines the network break area, a target position for the mobile robot is computed. This is done by the network reconfiguration component. The robot then needs to navigate inside the tunnel until either it reaches the target position or it determines that the target position is out of reach because of obstacles.

Control applications impose additional requirements on the RUNES platform that arise from the need to manipulate the networked systems and/or the environment in which they are embedded. In the rest of the paper we present the organization of the control system components that need to be implemented in order to guarantee that network connectivity is reestablished. The core are the four components: network reconfiguration, localization, collision avoidance and power control. Details of the development of these components are given in the companion papers.

IV. Control components for maintaining network connectivity in adverse environments

This section describes the software architecture for the control components used for maintaining network connectivity, together with the functionality of each component. The control components outlined below follow the RUNES component model [10]. The four main control components deal with network reconfiguration, localization, collision avoidance and power control. Their integration is demonstrated through the network reconfiguration scenario described next. The section concludes with a discussion of the low-level component library containing sensor, data fusion, controller and actuator components; the higher level components of network reconfiguration, localization, collision avoidance and power control invoke the low level components in this library to accomplice their goals. Communication security issues are addressed by a specialized security component (which in turn comprises a number of subcomponents); this component is dealt with separately in Section V.

A. *Physical network reconfiguration scenario*

Mobile autonomous robots are sent inside the tunnel to restore connectivity, see Figure 4. The navigation of a robot inside the tunnel is made possible by two components. The first is the localization component, that provides the position and orientation of the robot inside the tunnel and information

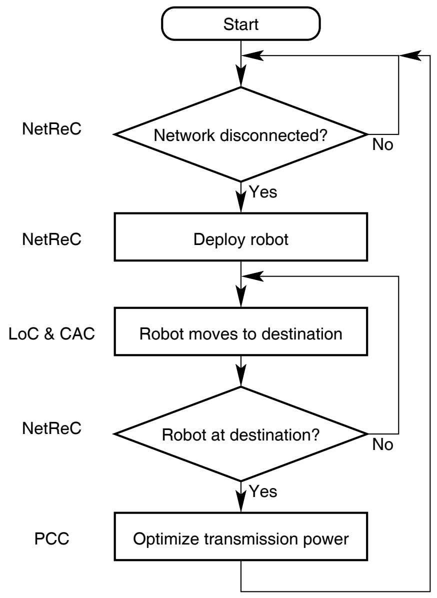


Fig. 5. Flow chart showing the actions taken in order to reestablish network connectivity. The acronyms to the left indicate the active control components.

about the presence of obstacles. The second is the collision avoidance component that ensures that the robot does not collide with obstacles or other robots. Once the mobile robot is in a suitable position it attempts to reconnect the network, by acting as a relaying node between the nodes in the disconnected parts of the network. At this stage, a third component, the power control component, is invoked, to reduce the energy consumption and lower the packet collision probability of the nodes at the boundary of the network. In case the network is not reconnected with the first robot, additional robots could be deployed in a similar fashion.

The flowchart in Figure 5 details the sequence of tasks in the reconfiguration scenario. The acronyms in the column to the left indicate the control component primarily responsible for the action. The scenario starts by the detection of that the network is disconnected. The network reconfiguration component (NetReC) then makes the decision that the first robot should be deployed. The robot moves autonomously to the destination using localization information about its position provided by the localization component (LoC). In parallel, it also uses the collision avoidance component (CAC) to avoid colliding with stationary objects or other moving agents. When the network reconfiguration component detects that the robot has reached a suitable goal position (possibly by adjusting the original destination point based on local information at the scene), it initializes the power control component (PCC). The radio transmission power is adjusted in the robot node and in its neighboring network nodes, in order to not only preserve battery power but also minimize interference among nodes. If the network is still disconnected after the power has been adjusted, the algorithm starts over and a new robot is deployed.