Architecture and Implementation of an Embedded Wormhole

Hugo Ortiz
ortiz@lasige.di.fc.ul.pt
Faculty of Sciences
University of Lisboa

António Casimiro
casim@di.fc.ul.pt
Faculty of Sciences
University of Lisboa

Paulo Veríssimo
pjv@di.fc.ul.pt
Faculty of Sciences
University of Lisboa

***Abstract*— Recent advances in wireless communication technology have opened the way for mobile services and applications. This paper describes ongoing work in the context of the HIDENETS project, which aims at the development of end-to-end resilience solutions for distributed applications and mobility-aware services in ubiquitous communication scenarios.**

In HIDENETS we assume the use of off-the-shelf components (COTS) and wireless communication links, which are inherently unreliable and uncertain, therefore making it difficult to secure dependability properties. The solutions under development are based on a hybrid system architecture, which considers the existence of a subsystem with better properties than the rest of the system: a wormhole. A wormhole provides specialized timeliness and trustworthiness services that may be used to construct more dependable and resilient applications. We are currently developing an embedded implementation of a wormhole, which we describe and discuss in this paper. In particular, we focus on the internal architecture of the wormhole, on the services it provides and on some of the challenges that we have faced during development.

I. Introduction

The development of embedded technologies to support cooperative, ubiquitous, pervasive and yet dependable applications, raises many technical and scientific challenges. For instance, when dealing with the uncertainty of the environment, it is hard to provide any guarantees, especially nonfunctional, such as performance, timeliness, reliability or security.

The model followed throughout the HIDENETS [1] project, and materialized in this work by means of an well-defined architectural solution, allows to reconcile the timeliness and/or security requirements of the applications with the uncertainty of the environment. It is a hybrid distributed systems model, referred to as the wormhole model [2], in which different parts of the system have its own properties. Applications execute in the payload part (possibly asynchronous and insecure), making use of a minimal set of services provided by the wormhole part, a small subsystem that is more timely, more reliable and more secure than the payload. Therefore, critical application steps can be executed, with the help of the wormhole, with higher confidence (coverage) than in a homogeneous system. In fact, specific instantiations of this model have been previously described (see, for instance, [3] and [4]), to address timeliness and security requirements of applications, in particular.

In this work we focus on the architecture and implementation of a wormhole in an embedded device. The embedded device is equipped with a Real-Time Operating System (RTOS), which allows the development of time-related services with improved timeliness guarantees. The device also provides communication interfaces over which the implemented services APIs can be provided. In other words, given its specific characteristics, including its small complexity, the small number of critical tasks it executes and the clean interfaces with the remaining parts of the system, it is possible to provide services with better properties with high confidence (coverage).

The paper is organized as follows. We start with a brief review of the wormhole model (Section II) and with a discussion of important construction principles to achieve the desired properties for the wormhole (Section III). The architecture and the discussion of implementation issues are then addressed in sections IV and V. In particular, we provide a description of the services that we are currently implementing on the device, explaining their interfaces and service parameters. Conclusions are presented in Section VI.

II. The Wormhole Model

As stated above, the architecture proposed for this project is based on a hybrid distributed systems model [2] in which different parts of the system have different sets of properties and can rely on different sets of assumptions, namely in terms of fault model and synchronism. This has a number of advantages when compared to the approach based on homogeneous models, as explained in [2]. The

general idea behind the wormhole system model is to exploit the heterogeneous nature of systems along the space domain, not only based on the fact that not all parts of the system have the same security or timeliness properties, but proactively making systems look like that.

As it will be seen ahead, this work is based on this model, considering the system to be divided into the following two (logical) subsystems:

- **Wormhole subsystem**: the part of the system that has better properties and that provides timeliness and trustworthiness related services, also called *oracle* services
- **Payload subsystem**: the rest of the system.

Oracles are typically more secure and more timely than they would be if implemented in the payload part of the system. This is necessary so that they can be used from the payload part of the system to improve the reliability and safety of the whole system. Service guarantees are visible at the oracles interfaces.

The wormhole subsystem can be local or distributed. This means that oracle services may have a local or distributed nature. If distributed, then there will be a communication subsystem to connect all local instances of wormholes, which will also have different properties from the payload communication network.

The identification of the role of all oracle services is important: what kind of guarantees exactly they provide, and what are the hardware, operating system and network requirements in order to maintain such guarantees with a given coverage. This will be addressed ahead in the paper.

III. Securing Wormhole Properties

In this section we address some fundamental points in the construction of a component of this kind to ensure that the necessary properties, namely in terms of security and synchrony, are achieved.

A wormhole can be designed in diverse ways, using a higher or lower degree of hardware and/or specialized software. Essentially, these implementations will differ in the coverage of the assumptions of security and synchrony of the wormhole. For example, it will be normally easier to guarantee that one local wormhole cannot be compromised, if it is physically protected from the rest of the system and only accessible remotely through a network or some other interface. In general, in the development of a wormhole the following set of basic principles must be followed:

- **Shielding**: the wormhole construction is such that itself is protected from faults affecting security.

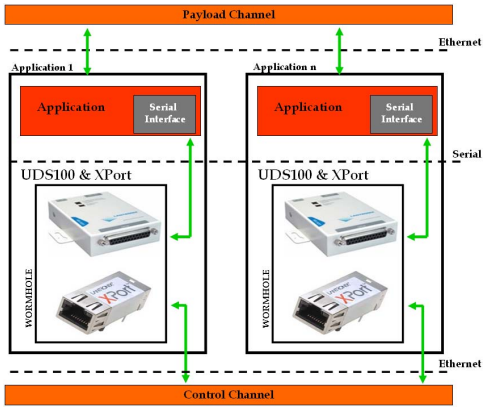

Fig. 1. Overall view of a system with a wormhole.

- **Validation**: the wormhole functionality is such that besides being simple, it allows the implementation of verifiable mechanisms.
- **Interposition**: the wormhole position is such that no direct access to critical resources can be made in default of the wormhole.

In order for the wormhole to behave in agreement with its specification it must have full control over the critical resources mentioned in the interposition. The shielding property substantiates the assumption that it is possible to prevent attacks or other kinds of interference that could result in intrusions in wormhole or loss of synchrony. The principle of validation implies that the conception and implementation of the wormhole must be simple so that it is verifiable, thus guaranteeing its properties in a justified way. For example: the number of lines of code must be necessarily limited and the internal structure of the protocols and procedures need to be easily perceptible.

The fact of being implemented in a embedded device with a low level of complexity and with only the strictly necessary interfaces, makes it easier to have a secure and timely component. In fact, a wormhole must have at least one interface, through which its services are provided to payload applications, and it may have a second interface, for connecting to other wormhole instances in the case of a distributed wormhole architecture. This is illustrated in Figure 1, for a concrete case in which two wormhole devices are represented, and their interfaces are instantiated.

In the figure we observe that:

- All the embedded devices are linked by an Ethernet network (through a switch), which constitutes the wormhole channel. This should be a private channel, for use only by wormhole instances and with well controlled traffic load in order to ensure the necessary improved properties. Note that other kinds of networks

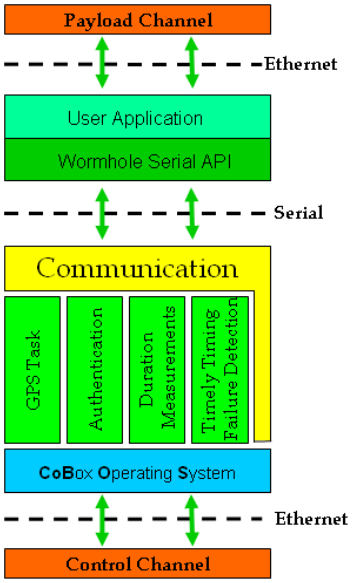

Fig. 2. Detailed architecture of an embedded wormhole.

could be considered, but specific measures for improving the characteristics of the communication would have to be taken.

- All the nodes where the applications run are linked by a payload network (again an Ethernet, in this example), but no special requirements for this network are necessary. In particular this may be an open network like the Internet.
- Each node has its own wormhole instance, an embedded device to which it is connected through a serial connection. This means that the payload application running on the node cannot access the resources contained in the wormhole except through this interface.

This setup is illustrates the implementation that was considered in this work.

IV. System Architecture

A more detailed view of the architecture of a system with an embedded wormhole is illustrated in Figure 2.

The wormhole subsystem is composed by an operating system, a communication module and a set of simple but critical services which will be used to support the development of dependable applications. The communication module manages incoming flows from both the user application and the remaining wormhole instances. In the specific implementation addressed here we consider a Timely Timing Failure Detection service, a Measurement service, an Authentication service and a positioning service. The first three are concerned with the provision of support for increased timeliness and security. The last one, may be an important service for the specific case of mobile applications, such as those considered in the HIDENETS project. In fact, the positioning information can be considered critical for some of these applications, and in that sense should be controlled and possibly secured by the wormhole. The services implemented in the wormhole are described in detail in the next section. Here we just provide a brief description of their goal:

Authentication: The application subsystem is more vulnerable than the wormhole subsystem. The goal of this service is to help the application to authenticate the information received from other nodes in the system, as well as to send authenticated information (in particular, positional one).

Duration Measurements: This service allows to measure the time interval between the occurrences of any two events of the system, as perceived by an application. The events can be local to a unique node, or may be distributed, in which case the wormhole will have to interact with peer wormhole instances.

Timely Timing Failure Detection: The detection of timing failures is very important for all applications that have timeliness requirements. The ability to detect these failures in a timely way, as done by this service, may be extremely important to prevent further contamination of the application, namely by timely reacting to the timing failure.

V. Prototype Implementation Details

For our implementation we have chosen a small embedded device, the Lantronix USD100, which is a compact size embedded device that has a serial and an Ethernet interface. The UDS100 device is based on the 80186 microprocessor with a 48MHz CPU and has 1024 Kbytes of Flash PROM. This computational power is already sufficient to implement the necessary services that we envision for our wormhole. The device already comes with a set of basic protocols already supported by default, which provide some communication related functionalities that may be useful in some situations, and that may open the development possibilities. The supported protocols include ARP, UDP, TCP, Telnet, ICMP, SNMP, DHCP, TFTP and HTTP.

A. Operating System

The operating system used in the adopted embedded devices is the CoBOS (CoBox Operating System) which is a cooperative [1] real-time operating system offering the needed functionalities for

[1]Being cooperative means that the scheduling algorithm is of the cooperative type, in which a running task does not lose the processor control until either some blocking function is called or willingly releases it.

achieving a real-time design. The task scheduling algorithm is a fully manual round-robin enabling the programmer to create completely deterministic and predictable applications. The provided primitives are `spawn()` that creates a new task, `kill()` that destroys a task and `nice()` that releases the processor in favor of the next waiting task. In addition, a safety mechanism is provided: a watchdog resets the device if some task does not release the processor before a fixed and known time interval. A fail-silent behavior is therefore ensured.

B. Wormhole Services

As refered above, the three services that are being implemented in the wormhole are:

Authentication: Before each new position is sent to the application, the local wormhole calls the primitive `signature ← signGPS_Position( gpsPosition )` which uses the md5 algorithm and a key (shared by all wormholes) to sign each position. *gpsPosition* is a data struct that holds all positional information (including the device id); a signed hash of the position is returned in *signature*. The primitive `result ←verifyPosition(gpsPosition, signature)` is called by the application each time it receives a signed position message from another application. This function receives as input a *gpsPosition* and its corresponding *signature* and returns a boolean that indicates if *gpsPosition* is a valid (authentic) position or not.

Duration Measurements: This service was implemented using two primitives and a table with *MaxMeasurements* entries. A duration measurement is started with a call to `tableEntry ← start_Measure (start)` which saves the initial timestamp corresponding to the start of the measurement (*start*) and returns the table entry where the timestamp was saved. This entry is thus used in order to stop the corresponding duration measurement by the call of the primitive `dm ←end_Measure (tableEntry)` which returns the measured value (*dm*).

Timely Timing Failure Detection (TTFD): This service was implemented using two primitives and also a table with *MaxTTFD_Entries*. To start a TTFD it is necessary to calls the function `tableEntry ← startTTFD( Start, Interval, FunctionID)` which receives as input, the *Start* timestamp (specifies the timestamp corresponding to the start of a TTFD), the *interval* (specifies the time interval after which the executing action is considered not to be timely anymore triggering the handler execution), and the *functionId* (function to be called as soon as a timing failure is detected). The return value, as in the Duration Measurements service, is the table entry where the parameters are saved and is used by the primitive `failed ← stopTTFD(TableEntry)` to stop a *TTFD* (*failed* is a boolean that indicates if there was a timing failure or not).

C. GPS Task

In many systems, the positional information is critical and so it should be in the wormhole domain. This service provides positional information, which can be real or emulated, depending on whether it is possible or not to connect a GPS receiver to the wormhole.

The assumption that this kind of embedded devices will soon be easily equipped with a GPS receiver, seems to be a reasonable one. In practice, the device that we used in this implementation does not have a GPS receiver and for that reason this receiver is emulated through a real-time task which periodically sends positions (within specific limits) as requested by the user application through the `getPosition()` function.

This task is initialized by defining an initial position through a function called `gps_setPosition()`. This task implements a simulated movement, so that each call to `gps_getPosition()` will return a possibly new position value. The rules that determine this movement are statically programmed in the task.

VI. Conclusion

In this paper we addressed the use of wormhole distributed systems model and framework in environments with uncertain timeliness using embedded devices with limited memory and low processing power. We described the architecture of the embedded wormhole and the main services that we have implemented in our prototype. We plan to use and demonstrate these results in the scope of the HIDENETS [1] European project.

Acknowledgment

This work was partially supported by LaSIGE and by the EU, through the HIDENETS project, IST-FP6-STREP-26797.

References

[1] HIDENETS (HIghly DEpendable ip-based NETworks and Services) web page. http://www.hidenets.aau.dk/

[2] P. Veríssimo. Travelling through wormholes: a new look ate distributed systems models. *SIGACT News*, 37(1):66-81, 2006

[3] P. Veríssimo and A. Casimiro. The Timely Computing Base model and architecture. IEEE Transactions on Computers, 51(8):916-930, August 2002.

[4] P. Veríssimo, N. F. Neves, and M. Correia. Intrusion-tolerant architectures: Concepts and design. In R. Lemos, C. Gacek, and A. Romanovsky, editors, Architecting Dependable Systems, volume 2677 of Lecture Notes in Computer Science, pages 3-36. Springer-Verlag, 2003.