ProFuN TG: Programming Sensornets with Task Graphs for Increased Reliability and Energy-Efficiency

Atis Elsts, Farshid Hassani Bijarbooneh, Martin Jacobsson, and Konstantinos Sagonas
Department of Information Technology, Uppsala University, Sweden

***Abstract*—Sensor network macroprogramming methodologies such as the Abstract Task Graph hold the promise of enabling high-level sensor network application development. However, progress in this area is hampered by the scarcity of tools, and also because of insufficient focus on developing tool support for programming applications aware of performance requirements.**

In this demo we present ProFuN TG (Task Graph), a tool for designing sensor network applications using task graphs. ProFuN TG provides automated task mapping, sensor node firmware macrocompilation, application simulation, deployment, and runtime maintenance capabilities. It allows users to incorporate performance requirements in the applications, expressed through constraints on task-to-task dataflows. The tool includes middleware that uses an efficient flooding-based protocol to set up tasks in the network, and also enables runtime assurance by keeping track of the constraint conditions.

Through task allocation in a way that optimizes an objective function in a model of the network, and adaptive task reallocation in case of link, node, or sensor failures the tool helps to make sensornet applications both more energy-efficient and reliable.

I. Introduction

Programming wireless sensor network applications is difficult, especially if certain reliability and data quality properties are desired together with energy efficiency. Tool support is required to help the application programmer to address these conflicting requirements. We build on the dataflow programming paradigm and adopt the Abstract Task Graph (ATaG) [1] WSN macroprogramming methodology. We implement ATaG in ProFuN TG [1] , a tool that addresses the needs of sensor network programming, deployment and maintenance. ProFuN TG not only allows users to describe the functionality of an application with a task graph, but also comes with support for mapping these task graphs on network nodes, for macrocompilation of their code, and for their deployment both on simulated and real networks.

We go beyond the original specification of the ATaG-based compilation framework [2] and enable *performance-aware* ATaG applications, by allowing the user to incorporate application-level performance requirements in the applications. These requirements are expressed in form of constraints on delay and packet delivery rate (PDR), and set on dataflows between tasks. In this way, we join together existing ideas about runtime assurance through performance monitoring with high-level programming support for WSN.

[1] http://parapluu.github.io/profun/

During run time, these requirements are used to enable efficient (i.e. reactive, rather than continuous) feedback from the network to the central system. We implement a middleware for the runtime support; it sets up tasks in the network, manages task-to-task communication, and determines whether the conditions of the constraints hold, enabling runtime assurance through maintenance alert notifications. If configured to do so, it also periodically collects application performance statistics in the central system. The alerts and statistics are used for adaptive task remapping with the dual purpose to satisfy the constraints and to optimize the system.

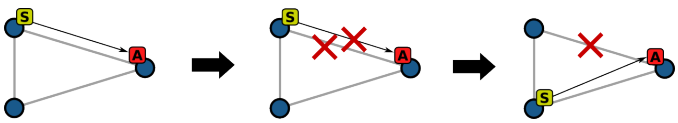

Fig. 1: Adaptive task reallocation on link failure

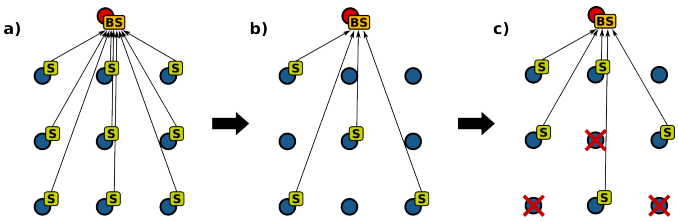

Fig. 2: Adaptive task redundancy

Thinking about applications as task graphs opens up a number of benefits:

- **Automated task mapping.** Using a single click, the user can map the task graph (Fig. 3) to a network in a way that optimizes energy-efficiency or another objective function (Fig. 4). The user also can constrain the mappings by describing requirements for node hardware components & properties, and setup a number of copies of a task per each node or per each region. The tool then automatically builds firmware images with the right components enabled.
- **Integrated performance requirements.** The user is allowed to specify the minimal acceptable path quality between a communicating pair of source & destination tasks, and to describe link-qualities either with a simple number or

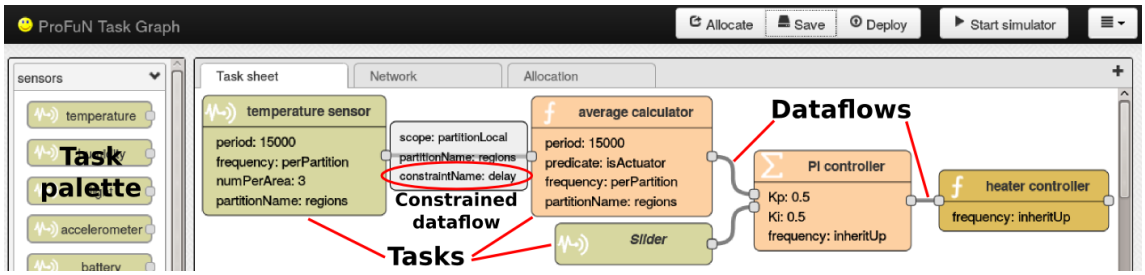

Fig. 3: **The task graph view of ProFuN TG**, showing a heating control application

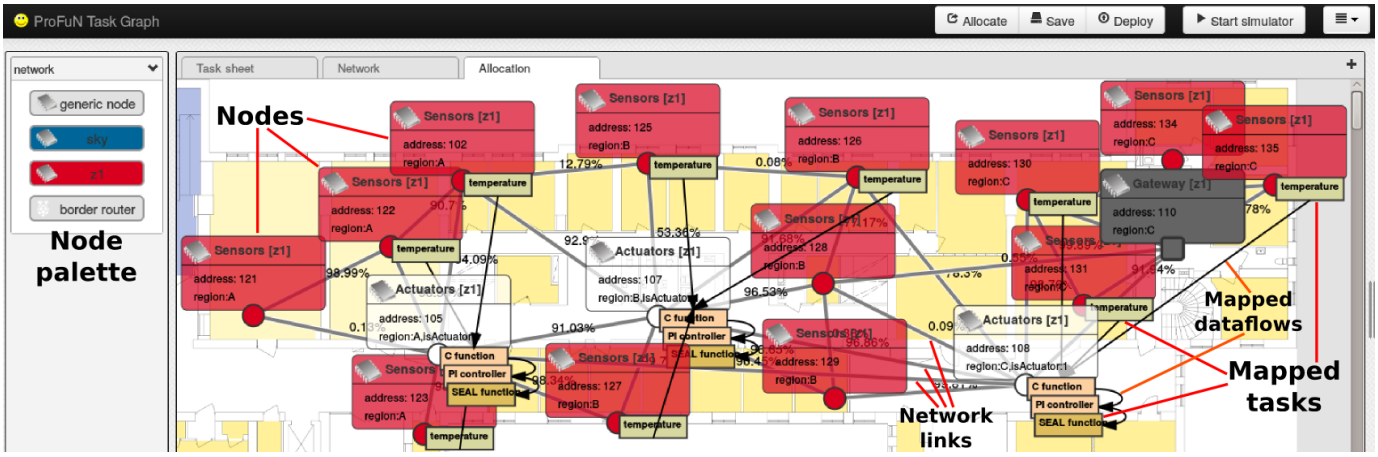

Fig. 4: **The network view of ProFuN TG**, showing the instantiated task graph of the application mapped on a test network

with complex probability distributions and let the to tool automatically infer the path quality.

- **Adaptations.** The system achieves higher reliability through automated task reallocation on link, node or sensor failures (Fig. 1).
- **Redundancy.** For increased reliability and coverage, the user with tool's support is allowed to create multiple task copies on redundant sensor nodes (Fig. 2a); for increased energy efficiency, the tool is allowed to reduce the number of copies as long as acceptable performance is achieved (Fig. 2b); on node failures, the task copies are automatically moved around (Fig. 2c).

This demo abstract is a companion to our publication in the SenseApp'15 workshop [3]. However, in this demo we plan to show a version of the tool extended with additional functionality, namely, "smart" sensor node selection for redundant task copies. Our algorithm is going to start by allocation a copy of the task on each feasible node, sampling the sensor a few times and collecting the data in the central system, and then, based on the sensor data and performance statistics, select a subset of the nodes to remain active. The selection is going to be done in a way to minimize correlations between the active sensors, but excluding sensor nodes with unacceptable routing path quality.

II. An example application

ProFuN TG task sheet view (Fig. 3) shows the task graph of an example application: an indoor heating control system. This application has two sensing tasks: *temperature* and *slider* (for desired T° adjustments), one actuation task: *heater*, and several data processing tasks: a function that *averages* multiple sensor values, and a *PI (proportional-integral) controller* that outputs the intensity of the required heating, using the difference between the input data and a reference value describing the desired T°.

Network view (Fig. 4) shows node placement in the network and the tasks of the application mapped on the nodes. We assume the application is deployed in a building with several regions (in this example, a region is defined as several co-located rooms), each of which has several sensor nodes.

The requirements of this application included:

- several types of nodes should be supported: a node equipped with sensors and a node connected to a heater actuator;
- each region should have at least one active heater task;
- each PI controller should receive input averaged from at least three temperature tasks located in the same region;
- the delay between the sensor tasks and the processing tasks should be $< 30\,\mathrm{s}$ with at least 99.5 % probability;
- the energy consumed by the network should be minimized, as long as the constraints above are satisfied.

Thanks to the help of the tool, it was easy to implement this application so that it supported all of these requirements.

III. Architecture

Under the hood, ProFuN TG uses a number of well-known software tools and libraries: Contiki for system-level func-

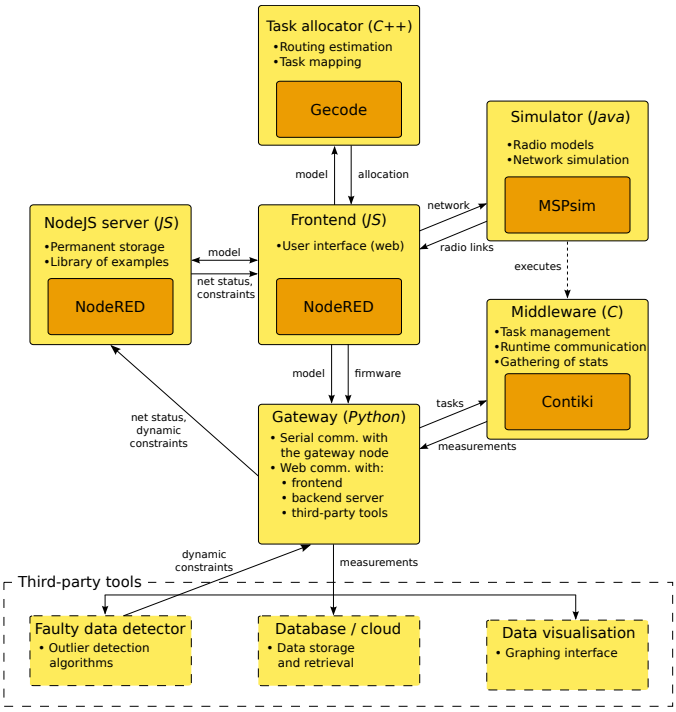

Fig. 5: Architectural overview of the tool

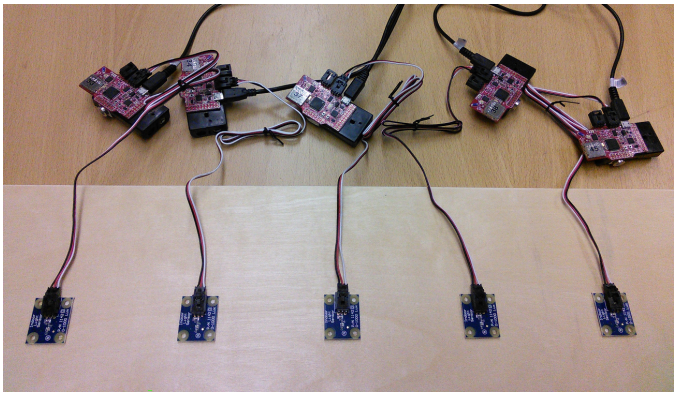

Fig. 6: **Demo setup:** sensor nodes with attached light sensors

tionality, Cooja for network simulation, Gecode for constraint solving (used in the task allocation algorithm). For the visual frontend, an adapted version of Node-RED is employed.

ProFuN TG joins these components in a distributed microservice architecture (Fig. 5). The components communicate by passing JSON messages through HTTP, with the exception of WSN middleware, which uses an efficient binary format.

The tool provides an HTTP interface for data export in JSON format. Through it, custom or third-party tools can access the data, provide feedback for ProFuN TG, and impose dynamic constraints on the task mapping algorithm.

IV. Demo Setup

We plan to demonstrate the functionality of the tool and its plugins by using a number of sensor nodes with attached light sensors (Fig. 6). Several aspects of the functionality will be demonstrated:

- **Automated task mapping and setup.** The web interface of the tool with task graph and network models will be shown. The task graph is going to describe an application for light intensity measurement and collection, and the network model is going to describe the demo network. Setting up this task graph in the demo network (i.e. on the physical nodes) will be repeated periodically and if requested by the attendees.
- **Adaptations.** Interested attendees are going to be invited to try to "break" the application by covering some of the light sensors and turning off some of the nodes, while the system is expected to demonstrate robustness by ignoring readings from the affected sensors and reallocating the sensing tasks to a different set of nodes. For this task, sensor fault detection functionality implemented as a plugin to the tool will be used. The plugin supports both threshold-based and non-parametric fault detection algorithms; for the latter, principal-component analysis based algorithm is used.
- **Redundancy.** The tool will choose a subset of the sensor nodes to initially use for light sensing tasks in a "smart" way. An algorithm implemented in a plugin of the tool is going to handle this by calculating correlations between sensor readings from different nodes and selecting the set of least correlated nodes, while also rejecting nodes with insufficient routing path quality. The attendees are going to be able to interactively influence the decisions of the algorithm by partially covering some of the light sensors or flashing light on them.

V. Concluding Remarks

ProFuN TG enables design of performance-aware task graph applications by allowing the user to write PDR and delay constraints on dataflows between tasks. The tool also enables deployment and maintenance of these applications by providing middleware that checks for faults at runtime and triggers reallocation in case a violation is detected. The deployment is done in a way that optimizes specific objective functions, in this way minimizing the energy required to keep the sensor network running. In this demo, we show both the design time (task graph and network modeling) and runtime (adaptive task allocation and reallocation) aspects of the tool.

Acknowledgments

The authors acknowledge support from SSF, the Swedish Foundation for Strategic Research.

References

[1] A. Bakshi, V. Prasanna, J. Reich, and D. Larner, "The Abstract Task Graph: a methodology for architecture-independent programming of networked sensor systems," in *USENIX EESR*, 2005, pp. 19–24.

[2] A. Pathak, L. Mottola, A. Bakshi, V. K. Prasanna, and G. P. Picco, "A compilation framework for macroprogramming networked sensors," in *IEEE DCOSS*. Springer, 2007, pp. 189–204.

[3] A. Elsts, F. H. Bijarbooneh, M. Jacobsson, and K. Sagonas, "ProFuN TG: A Tool for Programming and Managing Performance-Aware Sensor Network Applications," in *accepted for publication in SenseApp'15*, 2015.