Demo Abstract: Sensor Network Programming with Flask

Geoffrey Mainland, Greg Morrisett,
Matt Welsh
School of Engineering and Applied Sciences
Harvard University
{mainland,greg,mdw}@eecs.harvard.edu

Ryan Newton
CSAIL
MIT
newton@mit.edu

Categories and Subject Descriptors

D.3.3 [**Software**]: Language Constructs and Features; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Languages

1 Introduction

A great deal of recent work has investigated new programming abstractions and models for sensor networks. However, the complexity of such systems demands a great deal of effort to develop appropriate compilers and runtime platforms to achieve good performance. We will demonstrate Flask [5], a new programming platform for sensor networks that decouples the design of a high-level programming environment from the the low-level details of generating per-node code and an efficient runtime system.

An overview of the Flask architecture is shown in Figure 1. Flask consists of four primary components: an intermediate language based on dataflow graphs (DFGs); a metaprogramming toolkit that allows programmatic construction of DFGs; a back-end compiler that generates code from DFGs that achieves performance comparable to that of hand-coded NesC; and a suite of commonly-used runtime services. By carefully decoupling the high-level programming abstraction from the low-level instantiation and generation of sensor node programs, multiple high-level programming environments can be readily layered on top of Flask. In addition, by providing a common set of runtime services, language and application developers can shift their focus from node-level primitives to network-wide behavior.

The primary goal in Flask is to decouple application and high-level language design from the details of node-level program compilation. This is accomplished through an *intermediate language* (IL) that can be compiled down to efficient code. The Flask IL is based on a dataflow graph (DFG) abstraction, consisting of a directed graph of *operators* connected by *wires*. Wires are strongly typed and may carry primitive values (e.g., ints or floats) or tuples of multiple primitive values. Operators can have multiple distinct inputs,

SenSys'07, November 6–9, 2007, Sydney, Australia.
ACM 1-59593-763-6/07/0011

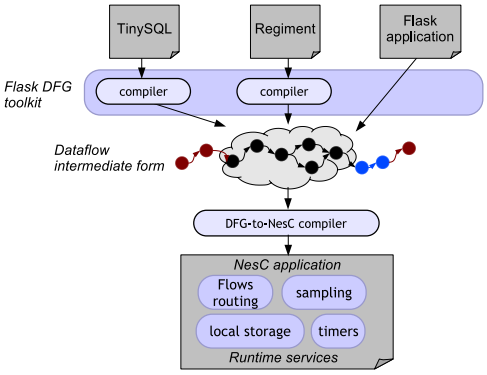

Figure 1. Overview of the Flask architecture.

but only one output. However, output wires may be freely split (i.e., fan-in and fan-out may be greater than one). Flask also provides *network wires*, wires that connect operators on different nodes.

Macroprogramming environments such as TinyDB [4], Tenet [2], Kairos [3], Regiment [7], and COSMOS [1] offer a range of programming models at different levels of abstraction and are often tailored for a fairly narrow range of target applications. Flask is targeted as a hybrid approach that simplifies the task of programming individual nodes using dataflow graphs, but provides facilities for composing these graphs across the network. As such, Flask is not a macroprogramming environment *per se*, but provides functionality that can be used to develop either node-level or network-level programs. Flask can also be used to build higher-level macroprogramming environments. We have used Flask as basis for implementations of systems similar in expressive power to TinyDB, Tenet and Regiment, showing that such systems can be quickly prototyped by targeting Flask's high-level DFG intermediate form. Flask has also been used to re-implement a previously fielded volcano monitoring application [8], greatly reducing the application's code complexity.

2 Demonstration

As part of the Flask we have developed an extensive framework for real-time visualization and manipulation of Flask applications. Motes running Flask-generated NesC applications can be individually manipulated both manually

from a GUI interface and automatically from a Python script, to facilitate automated testing. Our toolkit includes plugins that provide functionality such as viewing and tracing packets as they traverse the network, collecting performance statistics and collecting and injecting packets traces. It also allows users to write additional plugins with minimal effort.

For our demo we will show the capabilities of Flask and our visualization framework via two applications running live on Motelab [9]:

- **A FlaskDB Query:** We will run a live query that calculates the average temperature in Maxwell-Dworkin, the building where Motelab is located, and sends it to a base station. The novel aspect of this demonstration is that by leveraging Flask and its support for our visualization toolkit, we can perform live *validation* of the data received at the base station. Our GUI interface to the running query displays not only the average temperature calculated by the FlaskDB query using a spanning tree and sent to the base station, but also the true average temperature calculated from data collected directly from the motes via their serial port interfaces. The GUI also shows a visual real-time trace of multihop network messages as they flow hop-by-hop from originating nodes to the base station. We will manually disable nodes from the GUI to show how the application automatically recovers from node failure, a feature that is built into the Flask runtime.
- **A Regiment Plume Detection Algorithm:** We will demonstrate a version of the plume detection algorithm taken from [6], compiled and running on Motelab. In addition to the features already described, the plume detection visualization tool allows live injection of data into the network. We use this capability in a short Python script, loaded as a plugin, that simulates chemical plumes distributed over the sensor network and injects the appropriate simulated sensor data into individual nodes. The simulated plume and the "detection event" reported by the plume detection macroprogram are then both displayed in real-time.

For a more complete description of the Flask environment, please see [5].

3 References

[1] A. Awan, S. Jagannathan, and A. Grama. Macroprogramming heterogeneous sensor networks using cosmos. In *European Conference on Computer Systems (EuroSys)*, Lisbon, Portugal, March 21-23 2007.

[2] O. Gnawali et al. The TENET Architecture for Tiered Sensor Networks. In *Proc. ACM SenSys 2006*, Nov. 2006.

[3] R. Gummadi, O. Gnawali, and R. Govindan. Macroprogramming wireless sensor networks using Kairos. In *Proc. DCOSS'05*, 2005.

[4] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proc. the 5th OSDI*, December 2002.

[5] G. Mainland, M. Welsh, and G. Morrisett. Flask: A language for data-driven sensor network programs. Technical Report TR-13-06, Harvard University, May 2006.

[6] R. Newton, G. Morrisett, and M. Welsh. The regiment macroprogramming system. In *Proc. Sixth International Conference on Information Processing in Sensor Networks (IPSN'07)*, 2007.

[7] R. Newton and M. Welsh. Region streams: Functional macroprogramming for sensor networks. In *DMSN '04*, Toronto, Canada, August 2004.

[8] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proc. 7th USENIX OSDI*, Seattle, WA, Nov 2006.

[9] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A Wireless Sensor Network Testbed. In *Proc. the Fourth International Conference on Information Processing in Sensor Networks (IPSN'05)*, April 2005.