Title
Kairos: A Macro-Programming System for Wireless Sensor Networks

Authors
Gummadi, Ramakrishna
Kothari, Nupur
Millstein, Todd
et al.

Publication Date
2005

DOI
10.1145/1095810.1118600

Peer reviewed

Kairos: A Macro-Programming System for Wireless Sensor Networks

Ramakrishna Gummadi* Nupur Kothari† Ramesh Govindan‡ Todd Millstein§

http://kairos.usc.edu

Wireless sensor networks research has, till date, made impressive advances in platforms and software services. Research in the area has moved on to consider an essential piece of sensor network technology—support for *programming* wireless sensor network applications and systems components at a suitably high level of abstraction. Two broad classes of programming models are currently being investigated by the community. One class focuses on providing higher-level abstractions for specifying a node's local behavior in a distributed computation. Examples of this approach include the recent work on node-local or region-based abstractions. By contrast, a second and less-explored class of research considers programming a sensor network in the large called *macroprogramming*.

This poster is devoted to understanding and evaluating the abstractions and mechanisms necessary to develop a macroprogramming environment that is not SQL-restricted, and which enables a developer to specify the *global behavior* of a distributed computation in sensor networks. For example, a simple global specification of a distributed computation that builds a shortest path tree rooted at node **N** in a sensor network could be stated in words as:

> for each node **n**, its *parent* is that neighbor whose distance to **N** is shortest.

We contend that such a global specification can be easily encapsulated within a *single*, *centralized* imperative program that presents a simple *sequential* execution model and a *centralized* memory model of node state of the distributed sensors. The system translates the centralized program capturing this specification into a decentralized and localized program specialized by a sensor node state. Copies of such a compiled program execute on each sensor node, along with some additional runtime support. In our example, the resulting distributed program might, for instance, cause a node to repeatedly invoke its runtime to query the node's current neighbors about their current distance to **N**, process the received distances, and pick that neighbor whose distance to **N** was smallest.

*University of Southern California. gummadi@usc.edu
†University of Southern California. nkothari@usc.edu
‡University of Southern California. ramesh@usc.edu
§University of California, Los Angeles. todd@cs.ucla.edu

The main motivation for such a programming methodology is that programmers and systems designers are often able to clearly describe (and reason about the correctness of) a sequential, centralized version of a distributed computation, but often find it difficult to implement (or understand the correctness of) a node-local program that realizes a desired global behavior using a low-level explicitly distributed message-oriented programming model.

The crux of our approach, called *Kairos*, is to provide a small set of *programming primitives* (only four) which extend a programmer's favorite programming language, and which enable a programmer to completely capture the necessary distributed control flow and data consistency semantics in the form of central, sequential, and synchronous reads and writes of sensor states. These four abstractions are a) the notion of a node type and an iterator on a node set as first class language objects, the ability to b) centrally read and write data and c) invoke local functions at arbitrary nodes, and d) a temporal abstraction of actions. Kairos involves a *compiler* that translates the centralized program using these primitives into a node-specific distributed version that codifies them into the necessary local interactions in the form of explicit inter-node control co-ordination and data coherence logic. At execution time, this synthesized logic is then enforced, optimized for network economy and energy efficiency, and implemented using explicit messages by the Kairos *runtime*—that constitutes the second half of Kairos—that exists at every node.

We have designed and implemented a first version of Kairos that extends Python and runs on the popular sensornet hardware platform of Stargates nodes and Mica2 motes. In this poster, we describe our implementation of the language extensions and the runtime system, and our evaluation of three distributed computations that exemplify system services and signal processing tasks encountered in current sensor networks: constructing a shortest path data routing tree, localizing a given set of nodes, and vehicle tracking. We demonstrate that Kairos' level of abstraction does not sacrifice *performance*, yet enables *compact* and *flexible* realizations of these fairly sophisticated algorithms. We also sketch facilities for automated failure recovery that we are currently investigating.

Kairos: A Macroprogramming Model for Wireless Sensor Networks

Ramakrishna Gummadi, Nupur Kothari, Ramesh Govindan, Todd Millstein

http://kairos.usc.edu

Motivation

• Macroprogramming: Allow all nodes to be programmed as a single unit

• Global program behavior captured as a single sequential task on a centralized memory model

• No need for explicit parallelization or synchronization code

• Challenge is designing the compiler and runtime components that generate and implement an equivalent concurrent distributed version …

• … and optimize resulting distributed execution at various levels (programmer-level, compiler-level, compiler-network level, and network-level) for the primary constraint of traffic economy in WSNs

Implementation

• Python based implementation of Kairos on Stargates, with motes as radio interfaces.

• Uses Python's extensibility and embedding APIs to execute a source-level Python script at every node.

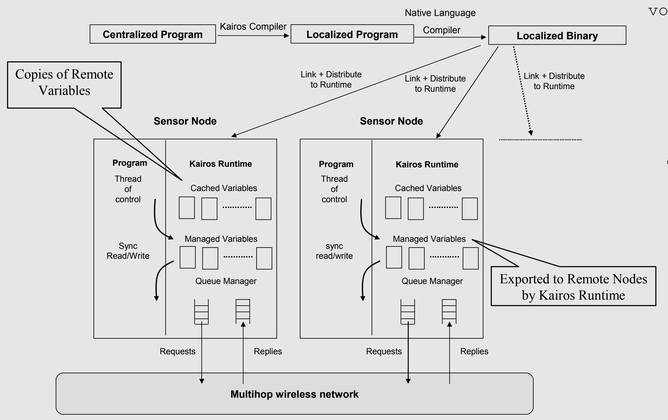
• 24 node test-bed of 16 Stargate motes + 8 Mica2Dots hanging off a multi-port PC serial card

•Emstar for E2E routing, topology management, and reliability

• Logical multi-hops over a single physical hop, using S-MAC as the MAC layer.

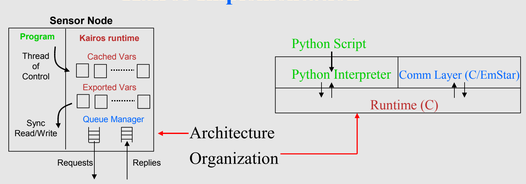
• Three representative programs: Data Routing Tree, Localization and Vehicle Tracking

Kairos Architecture



Testbed Setup

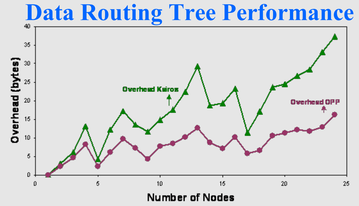
Kairos Implementation



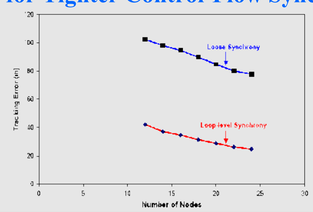
Procedural Code for Shortest-Path Data Routing Tree

```
void build_tree (node root=0) {
  node n, n';
  node_list avail_n=get_available_nodes();
  node_local node parent=NULL; //node_local variables denote…
  node_local uint dist_from_root=INF;//…state at each node
  dist_from_root@root=0; parent@root=root;
  for (;;) {//eventually converges; can also use…
            //…a stricter termination condition
    for (n=get_first(avail_n); n!=NULL;
         n=get_next(avail_n)) {
      node_list neigh_n=get_neighbors(n);
      for (n'=get_first(neigh_n); n'!=NULL;
           n'=get_next(neigh_n)) {
        if (dist_from_root@n' < dist_from_root@n+1) {
          parent@n=n';
          dist_from_root@n = dist_from_root@n'+1;
}}}}}
```

Data Routing Tree Performance



Need for Tighter Control-Flow Synchrony



Ongoing and Future Work

• Mote implementation: A Kairos compiler which can output nesC + Kairos runtime for motes

• Generic Failure Recovery: Automated recovery mechanisms in presence of various classes of failures

• Various levels of performance optimizations

• Exploiting Heterogeneity, Hierarchy, and User-level Energy/Resource Management

Mechanisms

• Four main programming primitives in Kairos:

- first-class datatypes: `node`, `node_list` (iterator on nodes) for topology independent programming

- `get_neighbors(node)` to obtain current one-hop neighbors of a node

- `var@node` to synchronously access node-local data

- a `time_queue` abstraction for temporal actions

• Kairos compiler translates the centralized sequential program into a node-localized version specialized at runtime by a node's state.

• Kairos runtime present at every node enforces inter-node control coordination and data coherence using explicit messages

- Kairos compiler and runtime optimize these messages for network economy

• Eventual Consistency (and other forms of relaxed consistency) semantics can be exploited

Results

• For the Data Routing Tree application, the displayed compact program with an unoptimized Kairos implementation correctly found a routing tree of better quality with only twice the byte overhead and 30% more convergence time than the default hand-coded One Phase Pull scheme in Directed Diffusion

• In the Vehicle Tracking application, `loose synchrony` (a form of eventual consistency) provided by Kairos alone is insufficient

- we use a stronger notion (Program Consistency) in the form of `loop-level synchrony`

- the `time_queue` abstraction permits a clean expression of this tighter consistency