Scriptable Sensor Network Applications for Rapid Development of Internet of Things

Sanat Sarangi and Subrat Kar

Bharti School of Telecommunication Technology and Management,

Indian Institute of Technology Delhi, Hauz Khas,

New Delhi - 110016, India

E-mail: sanat.sarangi@ieee.org and subrat@ee.iitd.ac.in

Received: December 15, 2013 Accepted: March 9, 2014 Published: April 30, 2014

DOI: 10.5296/npa.v6i1.4763 URL: http://dx.doi.org/10.5296/npa.v6i1.4763

Abstract

Internet of Things (IoT) refers to the paradigm of having sensor-enabled devices represented on a global inter-connected web. A sensor network is a customized network of sensor nodes deployed for a specific objective. Due to the variety of application areas, objectives and end-user requirements of sensor networks can differ widely between deployments. IoT, on the other hand, envisions a homogeneous presentation of information from sensor nodes for mining and analysis. It promises support for machine-to-machine (M2M) communication between sensor nodes. To tackle such conflicting requirements, we present RAPIDSNAP—a scriptable application framework for rapid deployment of sensor networks—which acts as a smart middleware to adapt sensor networks for IoT. RAPIDSNAP provides script-based extensions to facilitate development of customized sensor network applications while providing bridge-interfaces for integrating them with IoT repositories such as Wisekar. We illustrate the impact of RAPIDSNAP and Wisekar by using them to propose extensions to a sensor network based health application—Gaitsense. We conclude by comparing RAPIDSNAP with other frameworks.

Keywords: Internet of Things, sensor network, application framework, scriptable application, RAPIDSNAP, Rapidapp, JavaScript, Gaitsense, Wisekar

1. Introduction

Application development and deployment is more challenging for resource-constrained sensor networks than it is for legacy telecommunication networks. User applications for sensor networks have to be able to seamlessly adapt to the changing deployment conditions, sensing requirements, and instrumentation capabilities. At the same time, information from heterogeneous sensor networks has to be assimilated into a single standardized system to facilitate rich data analytics. The paradigm of creating an information-web of sensor-enabled devices, which naturally includes interconnecting multiple sensor networks, has been termed—Internet of Things (IoT). Characteristic attributes of IoT include having a virtual identifier for each "thing" (here, a sensor node) on the Internet and providing support for machine-to-machine (M2M) interaction between things. Due to the heterogeneous deployment characteristics and objectives of sensor networks, development of a unified framework that seamlessly achieves the goals of IoT is non-trivial.

We have developed a flexible IoT repository called Wisekar [1], hosted at http://wisekar.iitd.ac.in, for structured archival and exchange of sensor events. Wisekar provides an API through which events can be streamed over the Web into datasets called Active Sets where each Active Set can store information in an application-specific standard, such as SensorML [2] or EEML [3]. Wisekar allows information represented in multiple standards to co-exist. The configuration of a Wisekar Active Set can be tailored to the requirements of a given sensor network deployment, where each physical node corresponds to a unique virtual node defined in the Active Set.

In order to rapidly deploy customized sensor network applications, we have designed and implemented a framework called RAPIDSNAP [4]. RAPIDSNAP provides support for development of deployment-specific extensions called Rapidapps that act as communication-bridges between a sensor network deployment and Wisekar. We extend our discussion on protocols and algorithms associated with RAPIDSNAP, and illustrate how RAPIDSNAP and Wisekar are jointly used for IoT applications.

Gaitsense [5] is a sensor network based health application developed by us. It consists of a network of accelerometer-enabled sensor nodes called gait nodes mounted on mobile subjects for activity monitoring and fall detection. A gait node generates an event when the subject associated with it falls or changes its posture, for example, from standing to walking. The event travels through the relay (sensor) nodes to a central station where an action is initiated such as the generation of an email to call for emergency services. We note that notifying fall events can help prevent fatalities in the elderly. We extend the capabilities of Gaitsense by proposing novel extensions to it with the help of RAPIDSNAP and Wisekar. Specifically, we illustrate how Wisekar harvests information from a RAPIDSNAP deployment into its structured archival system. We also discuss how RAPIDSNAP is configured to use Wisekar for tasking sensor networks, to achieve M2M communication – one of the end-objectives of IoT.

The rest of the paper has six sections. Section 2 gives an overview of related work motivating the development of RAPIDSNAP. It discusses the advantages of an integrated RAPIDSNAP and Wisekar system over other frameworks, for IoT applications. Section 3 presents the RAPIDSNAP architecture and protocols in detail. Furthermore, ways to extend a

RAPIDSNAP network for heterogeneous deployments and role of Wisekar in achieving M2M communication between multiple RAPIDSNAP deployments are discussed. Customization of RAPIDSNAP and Wisekar for Gaitsense is presented in Section 4. Rapidapps are developed to realize the implementation of a clustering algorithm for Gaitsense and to selectively log Gaitsense events at Wisekar. Section 5 discusses other algorithms integrated with RAPIDSNAP. A comparison of RAPIDSNAP with other script-based frameworks is presented in Section 6. Finally, we conclude by discussing the impact of the proposed framework and highlighting its future scope in Section 7.

2. Background

Code generation frameworks [6, 7] for sensor networks and 'Service oriented sensor and actuator networks' (SOSANETs) [8] have been proposed. Argos [9] is an extensible middleware container into which hot-plug Java services are added or removed to meet deployment requirements. Agilla [10] middleware offers a flexible platform for sensor nodes which allows deployment specific mobile agents to be injected into the network. To simplify programming, sdlib [11] offers a library wrapper around NesC (in TinyOS [12]) with get()/give() interfaces. jWebDust [13] offers an n-tier sensor-network application architecture.

The combination of a scripting language and a strongly typed language has been used with simulation frameworks, such as discrete event simulators ns-2 (oTcl, C++) [14] and VisualSense (Python, Java) [15]. In SensorWare, nodes execute TinyTcl scripts over an RTOS [16]. Maté is a virtual machine on TinyOS that executes TinyScript and Mottle scripts [17]. Tapper [18], used with the Rappit deployment tool, and SCript [19] allow scripts to run on sensor nodes.

Information from sensor nodes terminates at a central station called a gateway. The gateway of a sensor network is a useful place to deploy algorithms for customized interaction with the network. Using a scripting (interpreted) language for developing such algorithms can be used to switch the algorithm(s) (and therefore, the network objective) in a live scenario without disturbing any other component of the deployment. Few frameworks provide such flexibility which allows effortless reconfiguration of a sensor network in action. Rappit is an embedded system application deployment tool that generates code for embedded systems (from sensor nodes to fast Ethernet) and has a user-side script interface in Python [20]. The need for customizable application frameworks for sensor networks has been highlighted in [8, 13].

Cross-platform script-based frameworks such as PhoneGap [21] have emerged. PhoneGap allows native applications to be written in HTML and CSS with phone hardware access support available through JavaScript. Like heterogeneous mobile platforms, the sensor network ecosystem is fragmented by numerous sensors, algorithms for different objectives, and multiple operating systems. RAPIDSNAP tries to standardize and implement multiple algorithms in a single integrated framework with support for deployment specific customization in JavaScript. PhoneGap facilitates seamless interaction between JavaScript and phone sensors whereas RAPIDSNAP achieves this between JavaScript and sensors in the network.

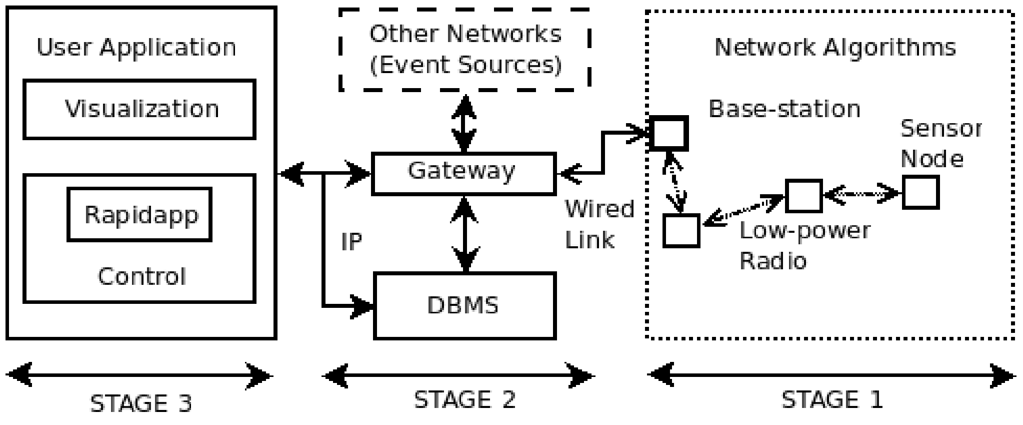


Figure 1: RAPIDSNAP Architecture

SPINE [22-24] is a deployment framework for signal-processing applications in Body Sensor Networks which provides a Java API for writing user applications. In contrast, the interpreted JavaScript-based Rapidapps used for developing RAPIDSNAP user applications allow their behaviour to be changed or updated at run-time. Furthermore, SPINE provides for one-hop communication between sensor nodes and the user application on a central Controller whereas RAPIDSNAP supports multihop communication through the integrated routing algorithms – AODVjr and PARTROUTE.

We note that SPINE2 [25] improves SPINE by adding support for node-platforms other than TinyOS and improving the software abstraction on the nodes for developers' convenience. A Java-based Building Management framework (BMF) has been proposed in [26] where OSGi plugins are used to extend the framework. Although the plugins in BMF are hot-swappable, development of a framework-extension with JavaScript in RAPIDSNAP is typically faster as recompilation of the extension during development or evaluation is not required. Also, JavaScript allows a large pool of web-developers to write sensor network applications. Integration of RAPIDSNAP with Wisekar vastly expands the application scope of RAPIDSNAP as a key component of pervasive IoT systems.

3. RAPIDSNAP Architecture

The RAPIDSNAP architecture has three stages as shown in Fig. 1. Stage 1 typically consists of a mesh of sensor nodes communicating with a designated node wired to the gateway called the base-station. The wired link is a *serial interface* in our implementation of RAPIDSNAP. We have implemented routing and localization algorithms on sensor nodes in Stage 1 with TinyOS 2.x [12] on TelosB nodes. The network protocol format defined by us in Section 3.1 is used by the algorithms. The protocol is independent of the underlying sensor-node hardware and operating system. Therefore, it can be used with other nodes and operating systems or a combination of them.

The gateway in Stage 2 acts as an interface between the user application on the legacy IP network (LAN, WAN) and other event sources, on an IP network for example. Every command from an application to the sensor network or sub-packet from the network, such as an event or a command response, to the application is logged in the DBMS and, in the process, tagged with a unique *packet ID* by the gateway. The algorithms realized with RAPIDSNAP do not require the network to be time-synchronized. In the absence of a reference network clock, the gateway acts as a global clock and timestamps each packet it receives.

The C++ based *User Application* in Stage 3 has two subsystems—the *User Subsystem* and the *Core Subsystem*. The core subsystem in the user application abstracts the user subsystem from the details of communication with the sensor network, legacy networks and the DBMS. For a given sensor network deployment, the user subsystem has the customized control and visualization components for the user. Customization is achieved with JavaScript based scripts called Rapidapps as shown in Fig. 1. Multiple Rapidapps, where each Rapidapp may implement a different algorithm, can be loaded into the user application and executed on the fly.

3.1 Communication Protocol for Sensor Nodes

In Stage 1, RAPIDSNAP standardizes the network layer of sensor nodes and the layers above it. We review the standards developed for packet formats in distributed systems to lay the groundwork for discussing inter-node communication with RAPIDSNAP. As sensor nodes are memory and energy constrained, energy spent in transmission and reception of packets plays a key role in determining the life of the network.

3.1.1 Background

There are three popular machine-to-machine (M2M) encoding formats—XML, JSON and Binary. An XML message contains both the data and the meta-data describing it. The self-descriptive nature of XML makes it platform independent. However, an XML parser is required for formatting and interpreting verbose XML packets, which is an overhead on memory and communication energy. Thus, the suitability of XML needs to be evaluated keeping in view the capability of sensor nodes and deployment requirements. Simple Object Access Protocol (SOAP) is an XML based standard for development of web services. An application consuming a web-service uses a SOAP client library to convert remote procedure calls into XML messages sent to the server.

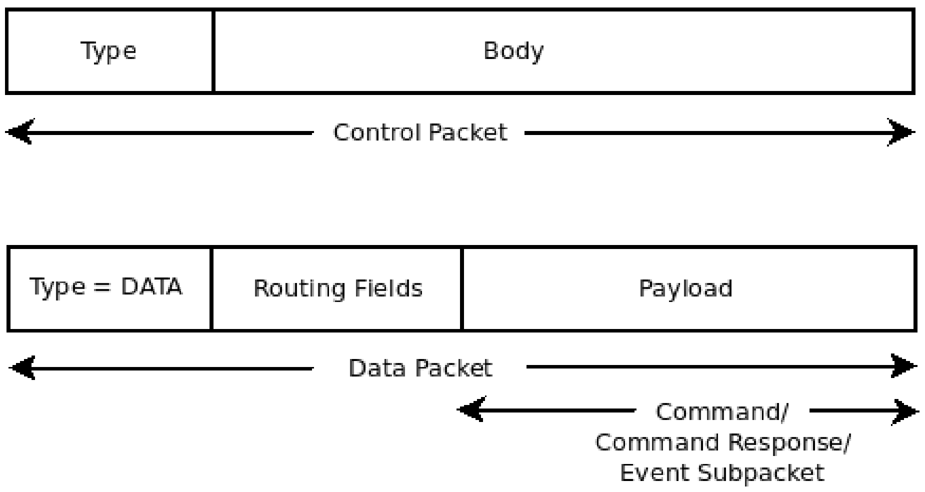


Figure 2: Sensor Network Packet Formats

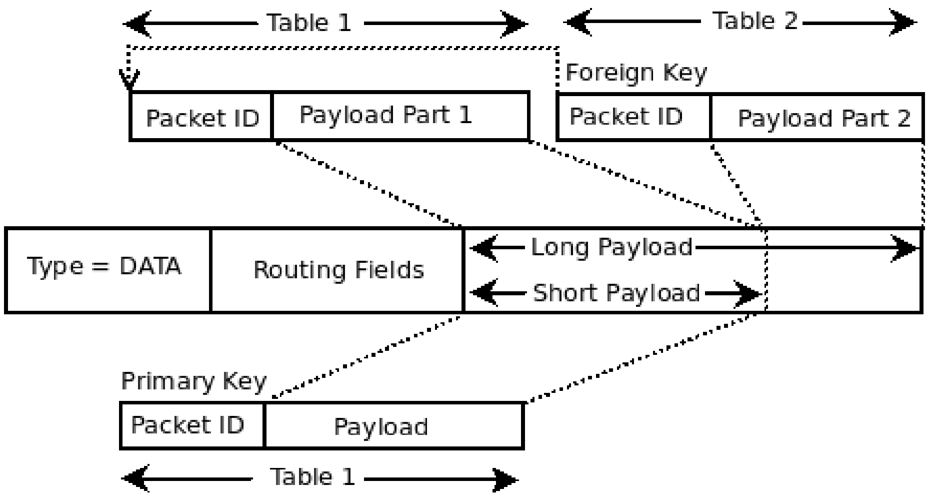


Figure 3: Mapping of DATA packet sub-packets to DBMS tables

XML has a clear advantage with web services since the client and the server are not energy-constrained, and XML allows structured messages to be exchanged over the Internet. JavaScript Object Notation (JSON) is popular textual data-interchange format derived from JavaScript that has a relatively compact notation and therefore lower communication overheads when compared to XML [27]. XML is considered to be a more suitable document-interchange format. JSON is popular with RESTful web-services as conformation to XML is not essential with REST unlike SOAP. JSON-RPC is a remote procedure call mechanism with JSON as the underlying message exchange format [28]. However, even JSON is less energy-efficient than a packed binary-encoded message where the communicating entities are aware of the message format. Java RMI and CORBA are RPC implementations where packed binary byte sequences are used. Both sender and the receiver are aware of the message format and its decoding method since the message is not self-descriptive.

RMI uses Java Object Serialization to flatten objects into a sequence of bytes and share them between JVMs [29]. CORBA is a standard that uses similar marshalling and unmarshalling operations at the sender and receiver [30]. RMI-IIOP allows RMI to interoperate with CORBA.

3.1.2 Communication Protocol and DBMS Schema Considerations

Due to resource-constraints in nodes, we choose the energy efficiency of binary encoding over flexibility of JSON or XML. The packed binary control and data formats used with RAPIDSNAP are shown in Fig. 2. Control packets are used by the routing, localization, clustering, and related distributed algorithms to optimally manage or support the network. The *Type* field determines the fields in the *Body* and the length of the packet. Data packets have a constant type—*DATA* which is followed by the *Routing Fields* segment that stores information needed to transport the data packet through the network to the base-station or vice-versa. Therefore, these fields are based on the routing protocol used. Since table-driven routing algorithms integrated with RAPIDSNAP—PARTROUTE [31] and AODVjr [32]—only require source, destination and the next-hop information in a packet, the

Routing Fields segment is short. For source-routing algorithms like DSR [33], the segment could be much longer. The *Payload* segment contains measurements from sensors.

Payloads in data packets are divided into *command*, *command response* and *event* sub-packets. Commands originate at the user application and terminate at a node in the sensor network while events and command responses originate at a sensor node and terminate at the user application. Command responses are generated only in response to received commands whereas events are generated asynchronously. RAPIDSNAP maps each sub-packet (event, command and command response) to one of a set of predefined formats. Storage optimization, schema design and evolution happens with respect to these formats. For example, a short and a long payload correspond to two different event formats which are mapped to database tables 1 and 2 as shown in Fig. 3. For each event, a unique packet ID is generated by the gateway. The event with the long payload is split across the two tables, where the packet ID for this event is stored in both the tables. Packet ID in table 2 acts as a foreign key to packet ID in table 1 which serves as the primary key.

3.2 Extending the Network with the RAPIDSNAP Gateway

Having the gateway in a separate stage allows RAPIDSNAP to address a variety of configuration scenarios. Fig. 4 shows a series of sensor networks connected to a single user application by cascading the gateways together. The IP link between the gateways could be an Ethernet or a Wi-Fi link. Furthermore, Network 1 and Network 2, associated with base-stations BS 1 and BS 2, need not be similar to each other in terms of their objectives or the architecture of their constituent sensor nodes. Thus, the gateway allows RAPIDSNAP to create large, hierarchical, distributed and heterogeneous sensor networks. From Fig. 1, we note that one gateway becomes an event source for the other.

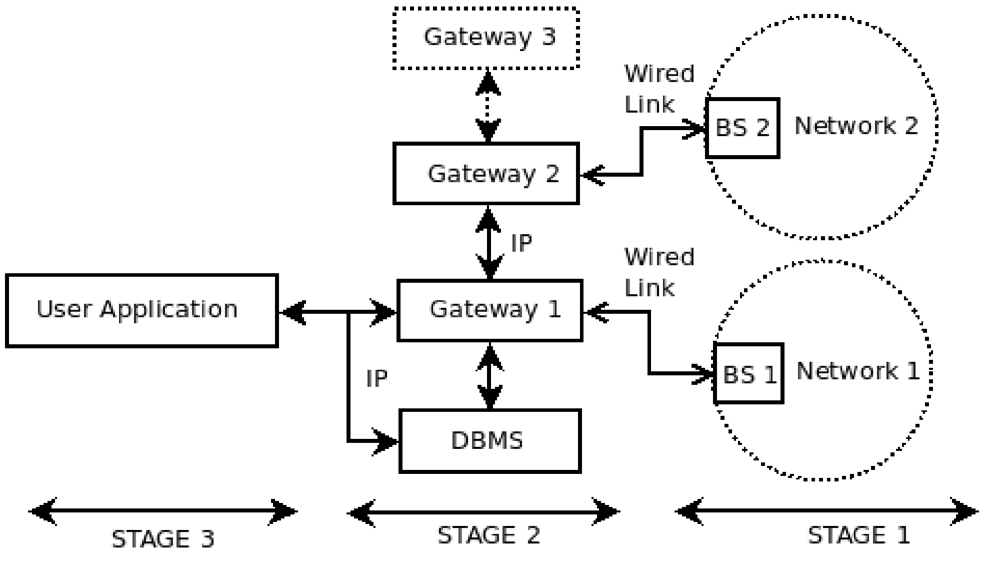


Figure 4: Extension of a RAPIDSNAP Network

3.3 Rapidapps for Customized Control and Protocol Translation

3.3.1 Customized Control

Javascript-based Rapidapp scripts allow RAPIDSNAP to address automated and customized control requirements. Multiple Rapidapps can be used to simultaneously address different target requirements. Each Rapidapp avails the core subsystem services through an instance of an *Adaptor* class made available by the core-subsystem [4]. Listing 1 shows an example Rapidapp where the function *handlePacket()* receives an event from the network and extracts its payload type. Available to handlePacket() is *adaptor*, an instance of Adaptor, which is used to obtain the numeric event type associated with string *NodeInfo*, stored in *nodeInfoEvent*. If the payload type matches *nodeInfoEvent*, an acknowledgement for the receipt of this event is printed. Adaptor provides methods to allow the context information of handlePacket() to be preserved across multiple handlePacket() invocations.

Each user application may have multiple Rapidapps, each executing a different algorithm. For example, there could be three Rapidapps 1, 2 and 3 as shown in Fig. 5 with the same entry method handlePacket() where the implementation of handlePacket() is different for each case. All Rapidapps receive (a) asynchronous events, commands and command-responses from the gateway and (b) periodic events from internal timers in the user application for the different algorithms they execute. Although commands for the sensor network are generated by the user application, it is the gateway which assigns a packet ID to the command and returns a copy of it to the Rapidapps for possible action.

3.3.2 Protocol Translation between Sensor Networks and IoT Repositories

Wisekar [1] offers a RESTful API interface for contribution of sensor events. We have extended the RAPIDSNAP user application to allow Rapidapps to invoke RESTful web-services. This enables multiple RAPIDSNAP installations at (perhaps) geographically distributed locations to communicate with Wisekar as shown in Fig. 6. Nodes in different RAPIDSNAP installations can use Wisekar to share information with each other simplifying the realization of IoT. In Section 4, we use Gaitsense as a case-study to discuss this integration mechanism.

Listing 1: Example of a Rapidapp

```
function handlePacket() {
    // adaptor object and packet fields as arguments
    // are passed implicitly to handlePacket
    var payloadType = arguments[1];
    var nodeInfoEvent = adaptor.getPayloadTypeFromString("NodeInfo");

    // process NodeInfo Events
    switch(payloadType) {
        case nodeInfoEvent:
        adaptor.print("NodeInfo event received");
    }
}
```

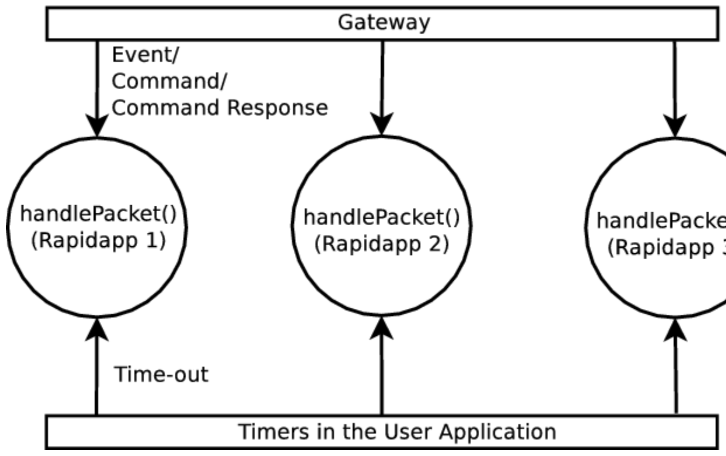


Figure 5: Multiple Rapidapps in the RAPIDSNAP user application where each Rapidapp executes a different algorithm

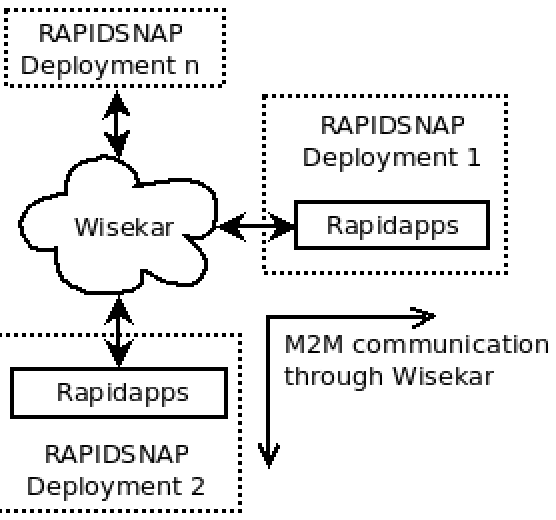


Figure 6: Protocol translation between RAPIDSNAP and Wisekar

4 Customizing RAPIDSNAP for a Health Application—Gaitsense

We use RAPIDSNAP and Wisekar to propose a set of extensions to the sensor network based Gait Assessment System—Gaitsense—introduced in Section 1. We first show how a Rapidapp is used to realize a clustering algorithm GAROUTE for energy-efficient routing of gait-node events. Then, we present another Rapidapp which filters and conditionally logs these events in Wisekar.

4.1 The GAROUTE Clustering Algorithm for Gaitsense

We have proposed a clustering algorithm GAROUTE [34] which uses mobility awareness of sensor nodes, added with accelerometers, to achieve energy-efficient clustering. We present the motivation for using GAROUTE with Gaitsense before presenting its integration with RAPIDSNAP. GAROUTE uses genetic algorithms to optimize a linear combination of three parameters of a sensor network with mobile nodes: Mean Communication Energy (*ME*), Clusterhead Fraction (*CHFrac*) and Total Clusterhead Speed (*CHSpeed*). The parameters are scaled by their respective factors S_{ME}, S_{CHF} and S_{CHS}. GAROUTE is simulated on a 50-node network shown in Fig. 7 with the sink (gateway) placed at (100, 100) [34].

GAROUTE prefers low-speed nodes as clusterheads. Fig. 8 shows how the parameters for a variant of GAROUTE, GAROUTE-V, vary with the mobility percentage. We note that the value of S_{CHS} is adjusted so that $ME\ X\ S_{ME}$ is a hard upper bound for $CHSpeed\ X\ S_{CHS}$ with 100% mobility. We note that if speed minimization out-weighs transfer energy minimization, it increases the total energy consumption of GAROUTE at the end of simulation. The residual energy of sensor nodes after 200 s of simulation for a network with 25% and 75% mobile nodes is shown in Fig. 9 and Fig. 10 respectively. The nodes follow a random-waypoint mobility model [34]. In Fig. 9, the large proportion of stationary nodes helps to rotate the role of clusterhead well. due to which the residual energy is evenly distributed across all nodes. In Fig. 10, on the other hand, energy of the smaller proportion of stationary nodes regularly preferred as clusterheads is drained faster. Therefore, a large variation in residual energy is observed. Assuming that a certain percentage of gait-node associated subjects is stationary at any given point of time and that the stationary subjects keep changing over time, clusterheads would be rotated regularly if gait nodes participated in clustering using GAROUTE.

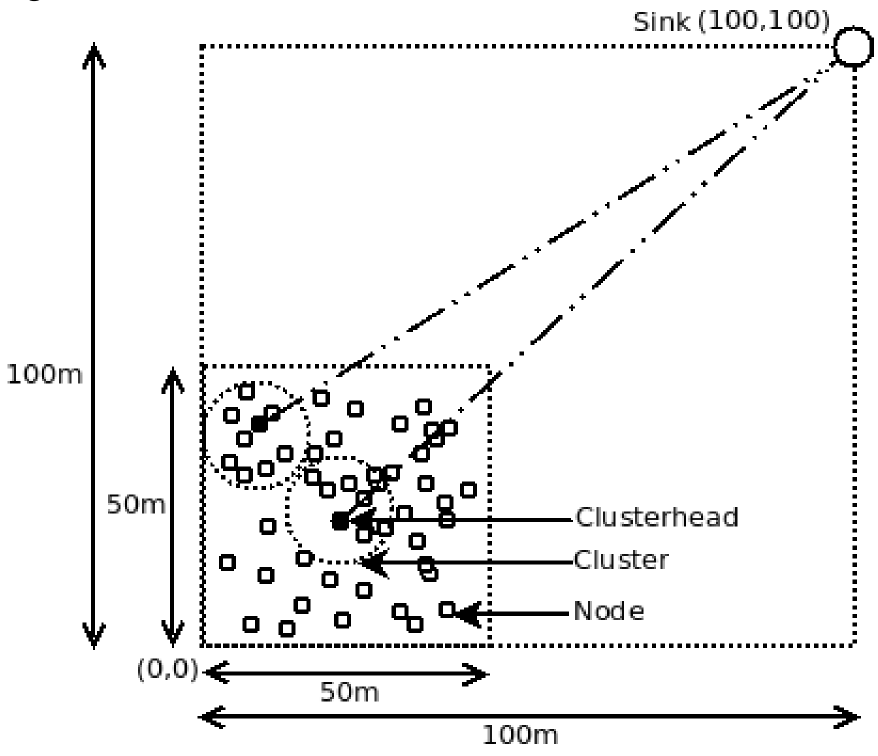


Figure 7: Deployment area configuration for evaluation of GAROUTE

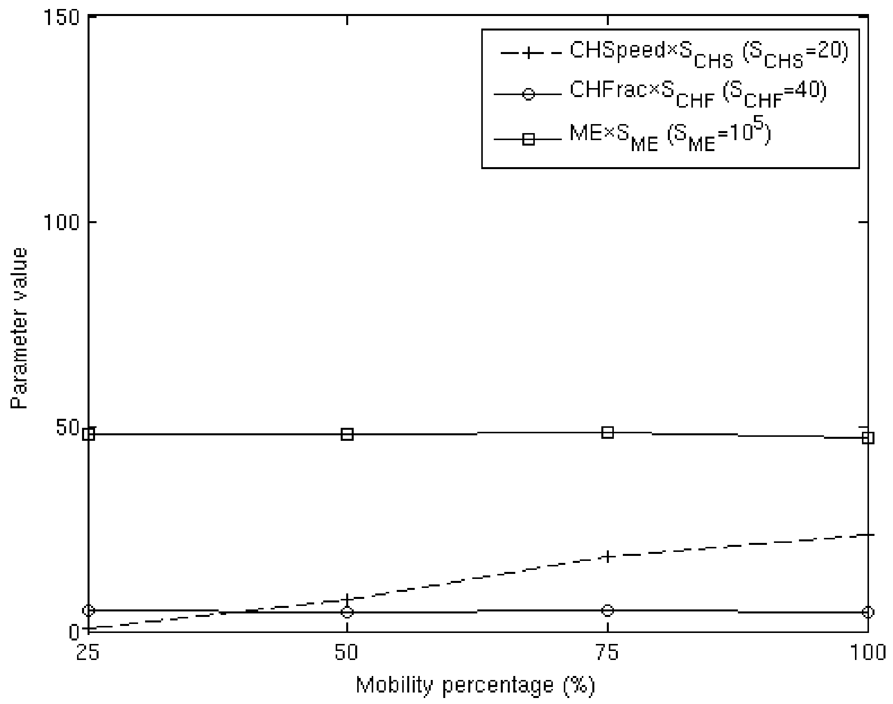


Figure 8: Variation of GAROUTE-V parameter values with mobility

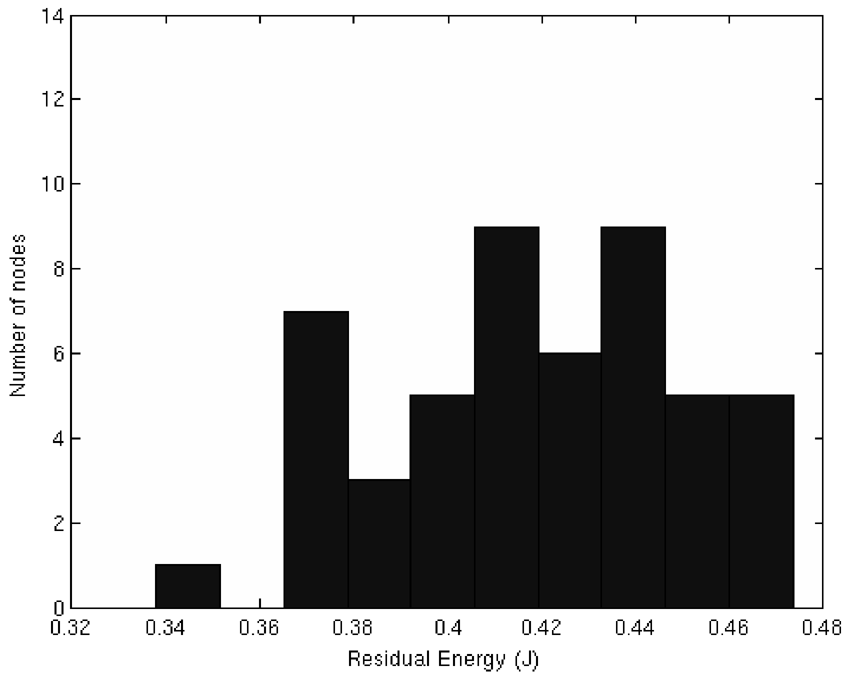


Figure 9: Residual Energy of nodes in GAROUTE-V with 25% mobility

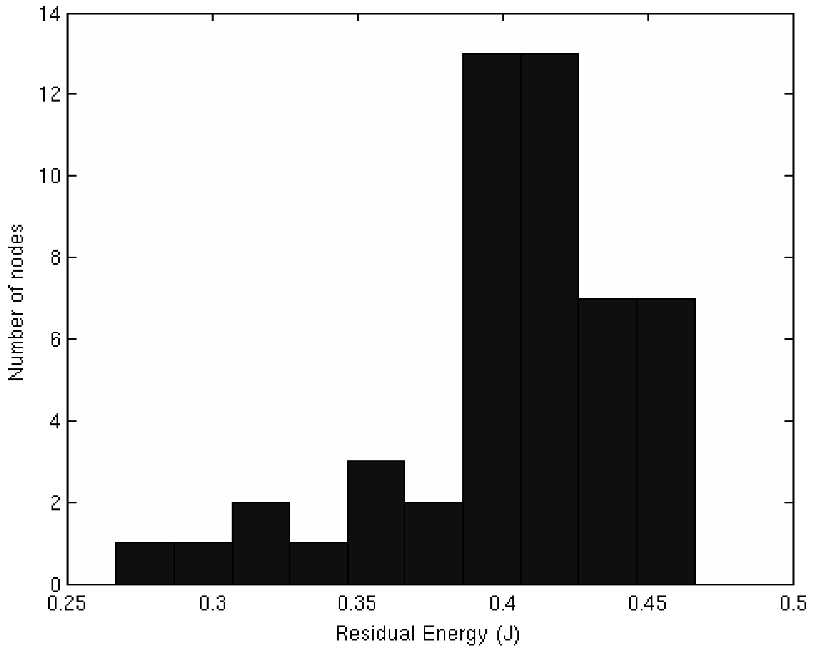


Figure 10: Residual Energy of nodes in GAROUTE-V with 75% mobility

4.2 Prototyping GAROUTE with RAPIDSNAP

To illustrate the orchestration of gait nodes for clustering with GAROUTE, we prototype a subset of GAROUTE with RAPIDSNAP. In the prototype, we consider a three-node network where the node with ID 2 becomes the clusterhead for two nodes 1 and 3. The sink of Fig. 7 is realized with stages 2 and 3 of RAPIDSNAP. The experimental setup for evaluation is realized with four TelosB nodes. One node connected to the gateway acts as the base-station, and three other nodes form a single cluster sensor network. All nodes operate at a common transmission power level at which they can hear each other. Although the nodes are stationary, we assign speed levels to them in order to illustrate their operation in a deployment scenario.

The clustering operations are captured by the sequence diagram in Fig. 11. Commands *StartGAROUTE*, *SetGAROUTEClusterhead* and event *NodeInfo* are defined at all stages of RAPIDSNAP. The parameters for a command or event in each message are indicated in brackets. The RAPIDSNAP user application and gateway run on a laptop termed as the *centralStation*. A Rapidapp at the user application is used to instrument the network. It initiates clustering by broadcasting StartGAROUTE. The nodes exchange NodeInfo event messages with each other to build their neighbourhood list and share their list with the Rapidapp in a NodeInfo event. The Rapidapp then chooses a clusterhead (node 3 in this case) and uses the SetGAROUTEClusterhead command to set it as the clusterhead for nodes 2 and 4. Fig. 12 shows the final cluster configuration and the entire log of communication between the Rapidapp and the sensor nodes in the visualization component of the user application.

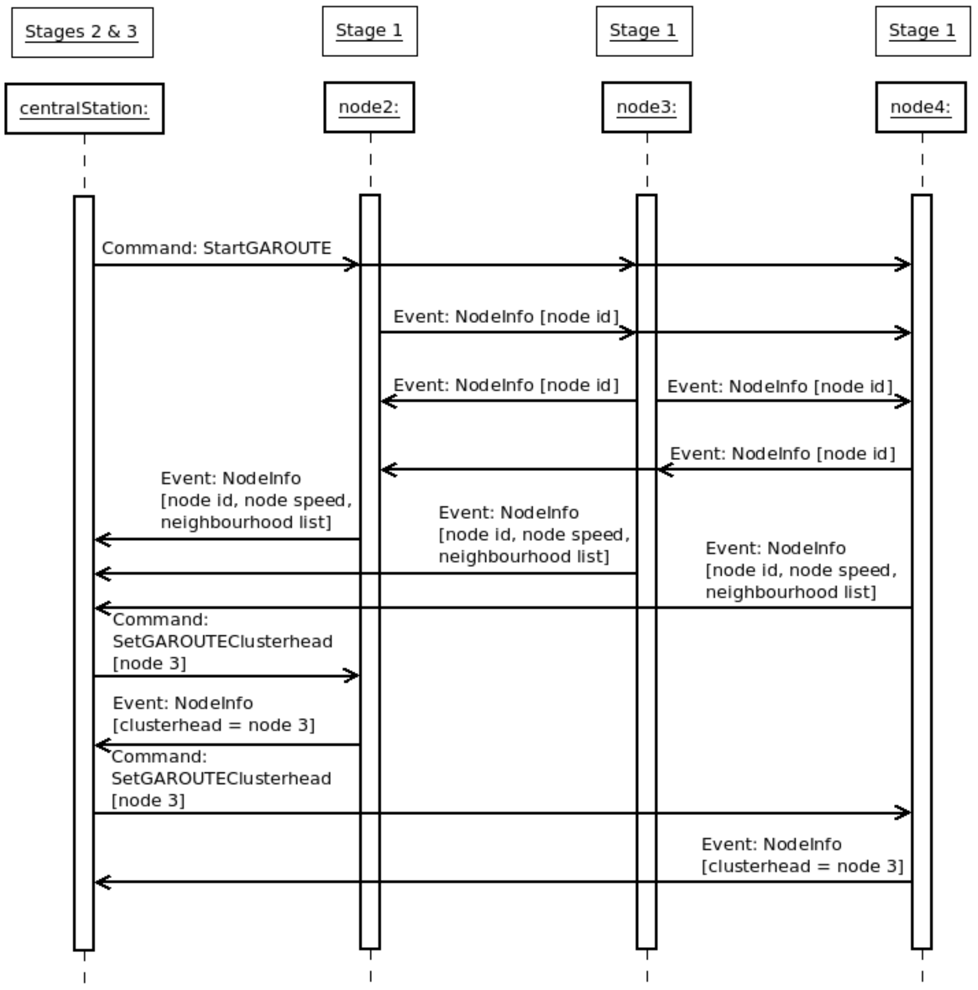
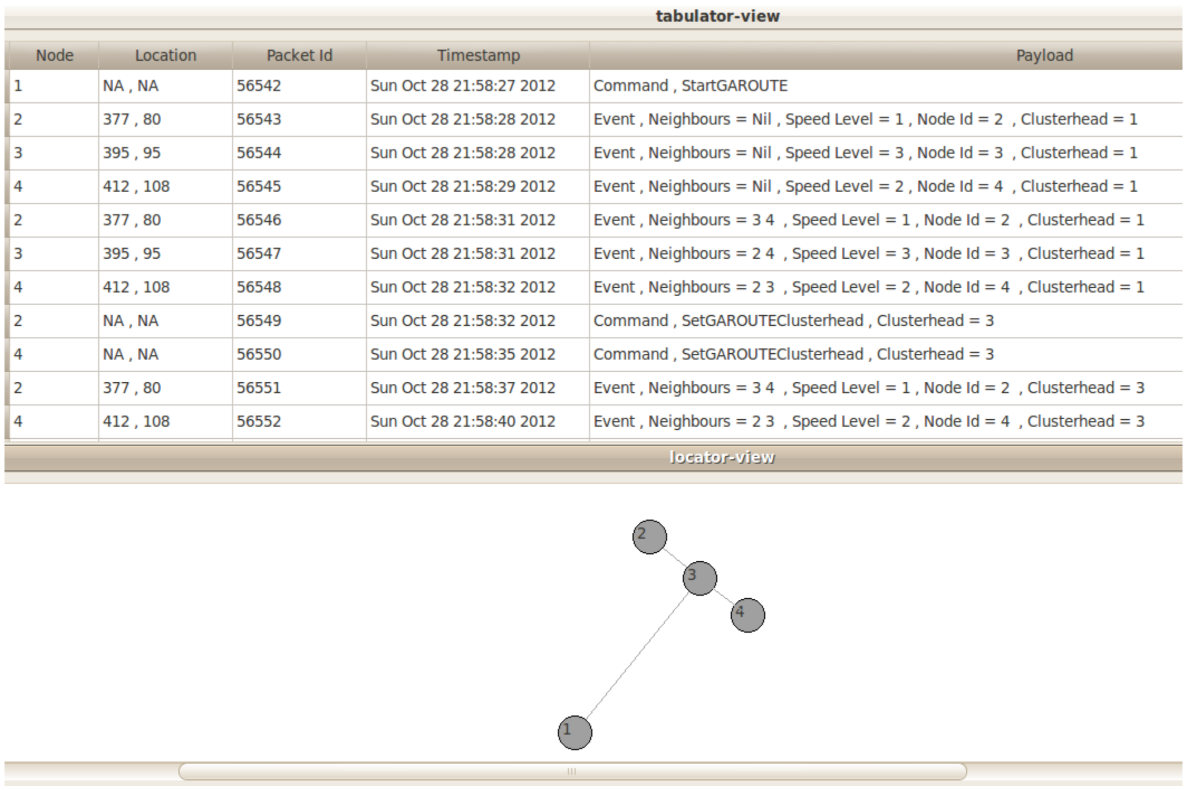

Stages 2 & 3
Stage 1
Stage 1
Stage 1
centralStation:
node2:
node3:
node4:
Command: StartGAROUTE
Event: NodeInfo [node id]
Event: NodeInfo [node id]
Event: NodeInfo [node id]
Event: NodeInfo [node id]
Event: NodeInfo [node id, node speed, neighbourhood list]
Event: NodeInfo [node id, node speed, neighbourhood list]
Event: NodeInfo [node id, node speed, neighbourhood list]
Command: SetGAROUTEClusterhead [node 3]
Event: NodeInfo [clusterhead = node 3]
Command: SetGAROUTEClusterhead [node 3]
Event: NodeInfo [clusterhead = node 3]

Figure 11: Communication between the GAROUTE Rapidapp and the sensor nodes

4.3 Event Notification with Wisekar

In addition to managing the sensor network behaviour, we show how Rapidapps act as middleware for bridging the communication between the network and Wisekar. We configure RAPIDSNAP to report fall events from gait nodes to Wisekar. For this, an *accelerometer* event packet is defined for communication between the three stages of RAPIDSNAP. Furthermore, we develop a Rapidapp to log all events in the Open Geo-spatial Consortium (OGC) defined standards-compliant format in Wisekar. As an example, whenever a subject falls, we need to log an event in Wisekar which captures the location of the event and also explicitly indicates that this is a fall event. For this, we define an XML fragment message sml_gaitsense.xml given in Listing 2 which is derived from two OpenGIS XML namespaces SensorML 1.0.1 and SWE 1.0.1. The SWE fields, enveloped in a *swe:DataRecord* tag, are defined in Listing 3. To configure Wisekar, we define an Active Set [1] with the XSD schema that corresponds to the XML Fragment in Listing 3. A node WISEKAR_NODE_ID is created in the active set for logging the gait events. This active set can now accept an event through the Rapidapp given in Listing 4. The incoming fall events from the Gaitsense network are packaged into GaitsenseSMLFrag.xml and logged in Wisekar through the *httpMethod* adaptor method.

Operations for all four HTTP verbs – GET, POST, PUT and DELETE – are supported by httpMethod. Just as the HTTP POST method is used to push an event into Wisekar, HTTP GET can be used by Rapidapps to pull information from Wisekar. Thus, Rapidapps can be used to achieve bidirectional communication between Wisekar and a node in the network. Wisekar, therefore, serves as a bridge between Rapidapps for development of smart M2M communication strategies in IoT applications.



tabulator-view

Node	Location	Packet Id	Timestamp	Payload
1	NA , NA	56542	Sun Oct 28 21:58:27 2012	Command , StartGAROUTE
2	377 , 80	56543	Sun Oct 28 21:58:28 2012	Event , Neighbours = Nil , Speed Level = 1 , Node Id = 2 , Clusterhead = 1
3	395 , 95	56544	Sun Oct 28 21:58:28 2012	Event , Neighbours = Nil , Speed Level = 3 , Node Id = 3 , Clusterhead = 1
4	412 , 108	56545	Sun Oct 28 21:58:29 2012	Event , Neighbours = Nil , Speed Level = 2 , Node Id = 4 , Clusterhead = 1
2	377 , 80	56546	Sun Oct 28 21:58:31 2012	Event , Neighbours = 3 4 , Speed Level = 1 , Node Id = 2 , Clusterhead = 1
3	395 , 95	56547	Sun Oct 28 21:58:31 2012	Event , Neighbours = 2 4 , Speed Level = 3 , Node Id = 3 , Clusterhead = 1
4	412 , 108	56548	Sun Oct 28 21:58:32 2012	Event , Neighbours = 2 3 , Speed Level = 2 , Node Id = 4 , Clusterhead = 1
2	NA , NA	56549	Sun Oct 28 21:58:32 2012	Command , SetGAROUTEClusterhead , Clusterhead = 3
4	NA , NA	56550	Sun Oct 28 21:58:35 2012	Command , SetGAROUTEClusterhead , Clusterhead = 3
2	377 , 80	56551	Sun Oct 28 21:58:37 2012	Event , Neighbours = 3 4 , Speed Level = 1 , Node Id = 2 , Clusterhead = 3
4	412 , 108	56552	Sun Oct 28 21:58:40 2012	Event , Neighbours = 2 3 , Speed Level = 2 , Node Id = 4 , Clusterhead = 3

locator-view

Figure 12: Final cluster configuration with node 3 as the clusterhead for nodes 2 and 4 after execution of the GAROUTE Rapidapp

Listing 2: sml_gaitsense.xml – a SensorML based Gaitsense message format

```
<sml:SensorML
  xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
  xmlns:swe="http://www.opengis.net/swe/1.0.1">
  <sml:member>
    <sml:components><sml:ComponentList><sml:component name="gaitsense">
      <sml:characteristics>
        <swe:DataRecord>
          [Highlighted section of GaitsenseSMLFrag.xml]
        </swe:DataRecord>
      </sml:characteristics>
    </sml:component></sml:ComponentList></sml:components>
  </sml:member>
</sml:SensorML>
```

Listing 3: GaitsenseSMLFrag.xml – XML Fragment structure for the Gaitsense Active Set in Wisekar

```
<swe:SensorMLFragment
  xmlns:swe="http://www.opengis.net/swe/1.0.1">
  <swe:DataRecord>
    <swe:field name="hasFallen">
      <swe:Boolean>
        <swe:value>true</swe:value>
      </swe:Boolean>
    </swe:field>
    <swe:field name="xloc">
      <swe:Quantity>
        <swe:value>100</swe:value>
      </swe:Quantity>
    </swe:field>
    <swe:field name="yloc">
      <swe:Quantity>
        <swe:value>200</swe:value>
      </swe:Quantity>
    </swe:field>
  </swe:DataRecord>
</swe:SensorMLFragment>
```

Listing 4: Gaitsense Rapidapp used to log 'fall' gait events in Wisekar

```
function handlePacket() {
  var subjectid = arguments[0];
  var payloadType = arguments[1];
  var condition = arguments[3]; //arguments 2 and 4 not used.
  var locX = arguments[5];
  var locY = arguments[6];
  var FALL = 1; //constant '1' denotes a fall condition
  var WISEKAR_NODE_ID = 33; //all Gaitsense events for 'subjectid' are
                            //registered with Wisekar node '33'
  var CUSTOM_TYPE = 9; //Wisekar type 'custom' (= 9) allows a
                       //user-defined event to be added

  accelEvent = adaptor.getPayloadTypeFromString("Accelerometer");
  switch(payloadType) {
    case accelEvent:
      adaptor.print("Accelerometer event received");

      if (condition == FALL) { //the subject has fallen
        var encodedString = adaptor.urlEncode([GaitsenseSMLFrag.xml with
         "swe" values for "xloc" and "yloc" set to locX and locY
         respectively]);

        statusString = adaptor.httpMethod("wisekar", "POST",
         "http://wisekar.iitd.ernet.in/api/home/
         resource.php/resource/event?key=[API_KEY]&
         nodeId="+WISEKAR_NODE_ID+"&typeId="+CUSTOM_TYPE+
         "&status="+subjectid+"&xmlFragment="+encodedString);
      }
    break;
  }
}
```

5 Algorithms integrated with RAPIDSNAP

Apart from GAROUTE, algorithms integrated with RAPIDSNAP include AODVjr [32] and PARTROUTE [31] for routing, and LORECOS [35] for localization. The software architecture of each sensor node is shown in Fig. 13. An *Algorithm Abstraction Layer* separates network level concerns, handled by the algorithms, from the application specific information termed as the *Node Application Payload*. LORECOS adds a set of localization extensions to AODVjr. PARTROUTE and GAROUTE are mobility aware routing and clustering protocols. Mobility awareness is derived from an external accelerometer integrated with the node. PARTROUTE uses mobility awareness of sensor nodes to achieve energy-efficient routing by reducing route-formation time. It creates traces over stationary nodes, and nodes on the traces are called *representatives* which send replies to a route-request source on behalf of the destination. To create routes, stationary nodes in PARTROUTE use S-ROUTE-REQUEST (S-RREQ) messages which also assist the trace-formation process [31]. M-ROUTE-REQUEST (M-RREQ) used by mobile nodes in PARTROUTE are comparable with the RREQ messages used by all nodes in AODVjr.

For AODVjr and PARTROUTE, Fig. 14 compares the delay for an event to be reported at the user application – the Decision Support System (DSS) for the network – with a gait node upto 4 hops away (i.e. with 3 intermediate relay nodes). With PARTROUTE, the gait node is configured as a mobile node, and the relay nodes are configured as stationary nodes. Thus, the relay nodes use S-RREQ messages to form routes while the gait node uses M-RREQ. Fig. 14 captures the 95% CI for Mean Event Transfer Delay between the gait node and the DSS. The (stationary) relay nodes form a stable trace to the base-station connected to the DSS. Therefore, irrespective of the hop-distance between the gait node and the DSS, M-RREQ from the gait node is fielded by a *representative* on the trace at a single hop. In AODVjr, on the other hand, only RREQ messages are used by all nodes [32]. Therefore, the RREP for an RREQ from the gait node is always generated by the base-station. Thus, with AODVjr, time required for the delivery of the first event from the gait node increases with hop-distance. We note that both AODVjr and PARTROUTE use expanding ring search based route-discovery to conserve communication energy. Expanding ring search results in the step-like delay pattern observed for AODVjr.

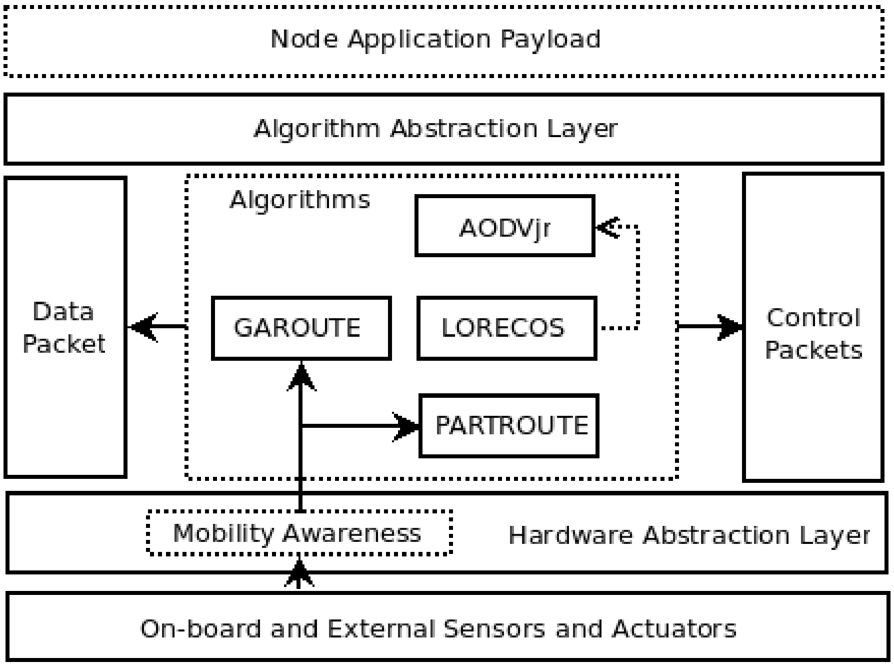


Figure 13: Software Architecture of a Sensor Node

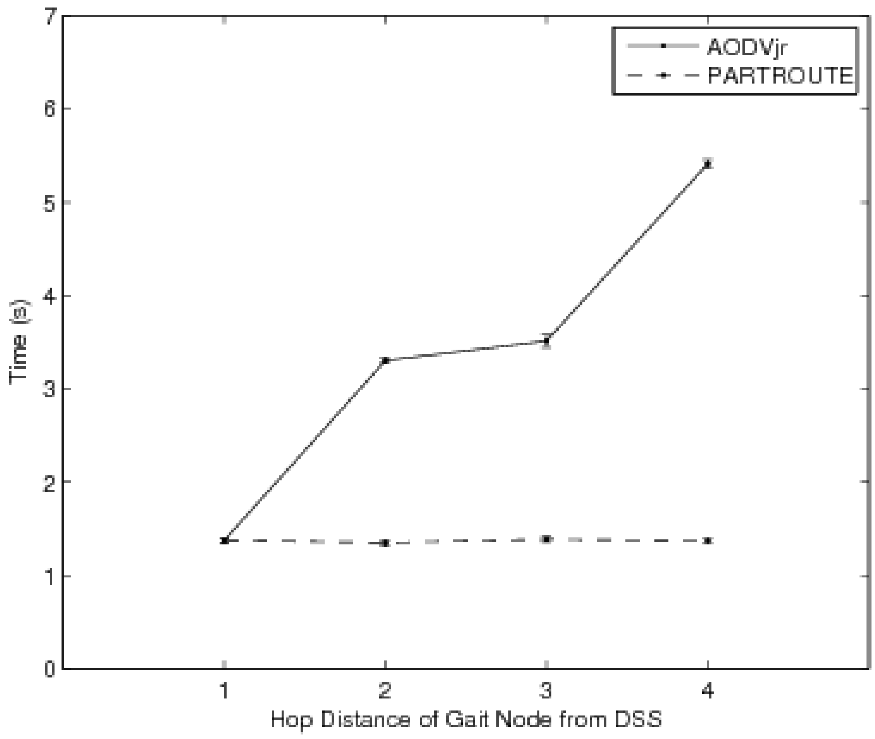


Figure 14: Delay estimates for gait event to reach the user application (DSS)

Table 1: Comparison of script-based frameworks with frameworks having similar scripting roles grouped using the same colour

Year	**Framework**	**Framework Purpose**	**Application Area**	**Scripting Ecosystem**	**Usage of Scripting**
1997	Ns-2 [14]	Simulation	Networks	oTcl, C++	Simulation configuration
2003	SensorWare [16]	Deployment	Sensor Networks	TinyTcl	Algorithm on sensor node
2004	VisualSense [15]	Simulation	Sensor Networks	Python, Java	Algorithm on sensor node
2004	Maté VM [17]	Deployment	Sensor Networks	TinyScript, Mottle	Algorithm on sensor node
2005	Rappit [20]	Deployment	Embedded Systems	Python	User (Console) Interface
2006	SCript [19]	Deployment	Sensor Networks	SCript	Algorithm on sensor node
2006	Tapper [18]	Deployment	Sensor Networks	Tapper commands	Algorithm on sensor node
2009	PhoneGap [21]	Deployment	Mobile Platforms	JavaScript, Platform OS	Phone app. development
2011	HomeWeb [36]	Deployment	Home Sensor Networks	JavaScript, Java (GWT)	Browser Interface
2012	RAPIDSNAP	Deployment	Sensor Networks	JavaScript, C++	User (GUI) Interface

6 Comparison of RAPIDSNAP with Other Script-based Systems for Sensor Networks

To summarize, we compare the characteristics of script-based systems motivating the development of RAPIDSNAP in Table 1. The frameworks are loosely classified into user-side scripting frameworks, sensor-node level scripting systems, simulation systems, and mobile application frameworks shown in blue, green, gray, and yellow respectively. Python has emerged as strong scripting platform for both simulation and deployment frameworks. JavaScript, the mainstay of client-side scripting on the web, has captured a major share of the mobile space with frameworks like PhoneGap. Development in Google Web Toolkit (GWT) happens completely in Java, and the client-specific Java code is compiled into JavaScript for execution on the browser. Homeweb is a GWT based sensor network application. RAPIDSNAP extends the scope of JavaScript to the development of rich native sensor network client applications.

7 Conclusion

A scriptable framework—RAPIDSNAP—for developing sensor network applications has been presented and its integration with an IoT repository Wisekar is discussed. The impact of the integrated system is presented in the context of a health application—Gaitsense. The RAPIDSNAP architecture is generic enough to encompass large and heterogeneous sensor networks deployments. We are working to integrate it with other systems like multimedia networks to further widen its application scope. We hope RAPIDSNAP and Wisekar evolve as a joint framework to address the potential challenges in Internet of Things.

References

[1] Sarangi S. and Kar S., "Wireless sensor knowledge archive," The First International Conference on Electronics, Computing and Communication Technologies (CONECCT 2013), Bangalore, January 17-19, 2013, pp. 1-6. http://dx.doi.org/10.1109/CONECCT.2013.6469315

[2] SensorML spec. Available at: http://www.opengeospatial.org/standards/sensorml/

[3] Extended Environments Markup Language (EEML). Available at: http://www.eeml.org/

[4] Sarangi S. and Kar S., "A scriptable rapid application deployment framework for sensor networks," International Conference on Advances in Computing, Communication and Informatics (ICACCI 2013), 2013, pp. 1035–1040. http://dx.doi.org/10.1109/ICACCI.2013.6637319

[5] Kar S., Sarangi S., and Bisht A., "Fall detection and activity monitoring with wireless sensor networks," Indian Journal of Medical Informatics, Vol.7, No.3, pp.128–139, 2013.

[6] Gummadi R., Gnawali O., and Govindan R., "Macro-programming wireless sensor networks using Kairos," Distributed Computing in Sensor Systems, pp. 126–140, 2005. http://dx.doi.org/10.1007/11502593_12

[7] Mostafizur M., Mozumdar R. , Lavagno L. , and Vanzago L. , "Rapid application development for wireless sensor networks," Factory Automation, pp. 103–120, 2010. http://dx.doi.org/10.5772/9514

[8] Rezgui A. and Eltoweissy M., "Service-oriented sensor–actuator networks: Promises, challenges, and the road ahead," Computer Communications, vol. 30, no. 13, pp. 2627–2648, 2007. http://dx.doi.org/10.1016/j.comcom.2007.05.036

[9] Munch-Ellingsen A., Eriksen D., and Andersen A., "Argos, an extensible personal application server," Middleware 2007, pp. 21–40, 2007. http://dx.doi.org/10.1007/978-3-540-76778-7_2

[10] Fok C., Roman G., and Lu C., "Rapid development and flexible deployment of adaptive wireless sensor network applications," Twenty-fifth IEEE International Conference on Distributed Computing Systems (ICDCS 2005), Columbus, Ohio, USA, 2005, pp. 653–662. http://dx.doi.org/10.1109/ICDCS.2005.63

[11] Chu D., Lin K., Linares A., Nguyen G., and Hellerstein J., "Sdlib: a sensor network data and communications library for rapid and robust application development," Fifth International Conference on Information Processing in Sensor Networks (IPSN '06), Nashville, Tennessee, USA, 2006, pp. 432–440. http://dx.doi.org/10.1.1.110.6074

[12] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System architecture directions for networked sensors,” Ninth International Conference on Architectural support for programming languages and operating systems, ser. ASPLOS IX, New York, NY, USA, 2000, pp. 93–104. http://dx.doi.org/10.1007/11502593_12

[13] Chatzigiannakis I., Mylonas G., and Nikoletseas S., “jWebDust: A java-based generic application environment for wireless sensor networks,” Distributed Computing in Sensor Systems, pp. 376–386, 2005. http://dx.doi.org /10.1007/11502593_29

[14] McCanne S. and Floyd S. , “ns network simulator.”
Available at: http://www.isi.edu/nsnam/ns/

[15] Baldwin P. , Kohli S. , Lee E. A., Liu X. , Zhao Y. , Ee C. C. T., Brooks C. H., Krishnan N. V., Neuendorffer S. , Zhong C. , and Zhou R., “VisualSense: Visual modeling for wireless and sensor network systems,” University of California, Berkeley, Technical Report: UCB/ERL M05/25, 2005.

[16] Boulis A. , Han C. , and Srivastava M., “Design and implementation of a framework for efficient and programmable sensor networks,” First International Conference on Mobile Systems, Applications and Services, San Francisco, CA, USA, 2003, pp. 187–200. http://dx.doi.org/10.1145/1066116.1066121

[17] Levis P., Gay D., and Culler D., Bridging the gap: Programming sensor networks with application specific virtual machines, Computer Science Division, University of California, Technical Report: UCB/CSD-04-1343, 2004.

[18] Xie Q., Liu J., and Chou P. H., “Tapper: a lightweight scripting engine for highly constrained wireless sensor nodes,” Fifth International Conference on Information Processing in Sensor Networks, ser. IPSN ’06. Nashville, Tennessee, USA: ACM, 2006, pp. 342–349. http://dx.doi.org/10.1109/IPSN.2006.243846

[19] Dunkels A., “A low-overhead script language for tiny networked embedded systems,” Swedish Institute of Computer Science, Technical Report: T2006:15, Sep. 2006.

[20] Chou P., Xie Q., and Hahn J., “Rappit: framework for synthesis of host-assisted scripting engines for adaptive embedded systems,” Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS’05), New York, 2005, pp. 315–320. http://dx.doi.org/1084834.1084912

[21] “Phonegap.” Available: http://phonegap.com/

[22] Gravina R., Guerrieri A., Fortino G., Bellifemine, F., Giannantonio, R. and Sgroi, M. “Development of Body Sensor Network Applications using SPINE” IEEE International Conference on Systems, Man and Cybernetics, Singapore, 2008, pp. 2810-2815. http://dx.doi.org/10.1109/ICSMC.2008.4811722

[23] Bellifemine, F., Fortino, G., Giannantonio, R., Gravina, R., Guerrieri, A. and Sgroi, M. “SPINE: A domain-specific framework for rapid prototyping of WBSN applications” Software - Practice and Experience, Vol. 41, No. 3, pp. 237-265, 2011. http://dx.doi.org/10.1002/spe.998

[24] Fortino G., Giannantonio R., Gravina R., Kuryloski P. and Jafari R., "Enabling Effective Programming and Flexible Management of Efficient Body Sensor Network Applications". IEEE Transactions on Human-Machine Systems, Vol. 43, No. 1, pp. 115-133, 2013. http://dx.doi.org/10.1109/TSMCC.2012.2215852

[25] Fortino, G., Guerrieri, A., Bellifemine, F. and Giannantonio, R., "Platform-independent development of collaborative Wireless Body Sensor Network applications: SPINE2," IEEE International Conference on Systems, Man and Cybernetics, San Antonio, USA, 2009, pp. 3144-3150. http://dx.doi.org/10.1109/ICSMC.2009.5346155

[26] Fortino, G., Guerrieri, A., Ohare, G.M.P. and Ruzzelli, A. "A flexible building management framework based on wireless sensor and actuator networks," Journal of Network and Computer Applications, Vol. 35, No. 6, pp. 1934-1952, 2012. http://dx.doi.org/10.1016/j.jnca.2012.07.016

[27] Nurseitov N., Paulson M., Reynolds R., and Izurieta C., "Comparison of JSON and XML data interchange formats: A case study," in ISCA Twenty-second International Conference on Computer Applications in Industry and Engineering (CAINE'09), San Francisco, CA, USA, 2009, pp. 157–62.

[28] "JSON-RPC." Available at: http://json-rpc.org/

[29] Waldo J., "Remote procedure calls and java remote method invocation," IEEE Concurrency, Vol. 6, No. 3, pp. 5–7, Jul-Sep 1998. http://dx.doi.org/10.1109/4434.708248

[30] Vinoski S., "CORBA: integrating diverse applications within distributed heterogeneous environments," IEEE Communication Magazine, Vol. 35, No. 2, pp. 46–55, Feb 1997. http://dx.doi.org/10.1109/35.565655

[31] Sarangi S. and Kar S., "Mobility aware routing with partial route preservation in wireless sensor networks," Ubiquitous Computing and Communication Journal, Vol. 6, No. 2, pp. 848–856, 2011.

[32] Chakeres I. D., and Klein-Berndt L., "AODVjr, AODV simplified," SIGMOBILE Mobile Computer Communication Review, Vol. 6, No. 3, pp. 100–101, 2002.

[33] Johnson D. B. and Maltz D. A., "Dynamic source routing in ad hoc wireless networks," Mobile Computing, Vol. 353, pp. 153–181, 1996. http://dx.doi.org /10.1007/978-0-585-29603-6_5

[34] Sarangi S. and Kar S., "Genetic algorithm based mobility aware clustering for energy efficient routing in wireless sensor networks," Seventeenth IEEE International Conference on Networks (ICON), 2011, Singapore, Dec. 2011, pp. 1–6. http://dx.doi.org/10.1109/ICON.2011.6168497

[35] Sarangi S. and Kar S., "Location estimation with reactive routing in resource constrained sensor networks," International Conference on Sensors and Related Networks (SENNET'09), Vol. II, Vellore, India, Dec. 2009, pp. 563–567.

[36] A. Kamilaris, V. Trifa, and A. Pitsillides, "Homeweb: An application framework for web-based smart homes," Eighteenth International Conference on Telecommunication (ICT 2011), Ayia Napa, Cyprus, 2011, pp. 134–139. http://dx.doi.org/10.1109/CTS.2011.5898905