Int. J. Communications, Network and System Sciences, **2011, 4, 1-16**
doi:10.4236/ijcns.2011.41001 Published Online January 2011 (http://www.SciRP.org/journal/ijcns)

RISN: An Efficient Sensor Network Overlay with Support for Autonomous and Distributed Applications

Evens Jean[1], Ingmar Rauschert[1], Robert T. Collins[1], Ali R. Hurson[2], Sahra Sedigh[2], Yu Jiao[3]
[1]*The Pennsylvania State University*, *University Park*, *USA*
[2]*Missouri University of Science & Technology*, *Rolla*, *USA*
[3]*Oak Ridge National Laboratory*, *Oak Ridge*, *USA*
E-mail: *jean@cse.psu.edu*, *rauscher@cse.psu.edu*, *rcollins@cse.psu.edu*, *hurson@mst.edu*, *sedighs@mst.edu*, *jiaoy@ornl.gov*
Received November 22, 2010; *revised December* 14, 2010; *accepted December* 18, 2010

Abstract

Once deployed, sensor networks are capable of providing a comprehensive view of their environment. However, since the current sensor network paradigm promotes isolated networks that are statically tasked, the full power of the harnessed data has yet to be exploited. In recent years, users have become mobile entities that require constant access to data for efficient and autonomous processing. Under the current limitations of sensor networks, users would be restricted using only a subset of the vast amount of data being collected; depending on the networks they are able to access. Through reliance on isolated networks, proliferation of sensor nodes can easily occur in any area that has high appeals to users. Furthermore, support for dynamic tasking of nodes and efficient processing of data is contrary to the general view of sensor networks as subject to severe resource constraints. Addressing the aforementioned challenges requires the deployment of a system that allows users to take full advantage of data collected in the area of interest to their tasks. Such a system must enable interoperability of surrounding networks, support dynamic tasking, and swiftly react to stimuli. In light of these observations, we introduce a hardware-overlay system designed to allow users to efficiently collect and utilize data from various heterogeneous sensor networks. The hardware-overlay takes advantage of FPGA devices and the mobile agent paradigm in order to efficiently collect and process data from cooperating networks. The computational and power efficiency of the prototyped system are herein demonstrated. Furthermore, as a proof-of-concept, we present the implementation of a distributed and autonomous visual object tracker implemented atop the Reconfigurable and Interoperable Sensor Network (RISN) showcasing the network's ability to support ad-hoc agent networks dedicated to user's tasks.

Keywords: Pervasive Computing, Mobile Computing, RISN, Service Overlays, Overlay Architecture, Sensor Network, Embedded Systems, Mobile Agents, Target Tracking, Reconfigurable Hardware, Interoperability

1. Introduction

A sensor network is primarily composed of sensing nodes forming a network capable of reacting to environmental stimuli. The sensor nodes are generally low-power, low-memory devices with highly constrained computational capability. The data collected by the nodes is relayed to a special node in the network referred to as a base station or sink, for processing. Traditional sensor network architectures advocate the deployment of statically tasked nodes that form an isolated network. Hence, for every new application, a network would have to be deployed over possibly overlapping coverage areas. Node proliferation is generally undesirable, especially in areas such as metropolitan centers, wildlife refuges or when the overlapping networks may contain similar sensing apparatuses.

In general, the requirements and restrictions of applications are not necessarily aligned with those of the sensor network. Applications generally require the ability to efficiently access and process data from anywhere and at any time. The ability for networks to interoperate is thus

crucial, as applications are generally impervious to the origins of data, but are more concerned with the accuracy and reliability of the data. Further exacerbating the resource constraints typical of sensor networks are the severe constraints of end-user devices, which may be unable to support the computation of interest, thereby rendering the data available in the network unusable. The ability to remotely process collected data could allow users to offload computations, should they be unable to efficiently process the data locally. Such computations could be carried out by nodes with processing abilities that deviate from those of traditional sensor nodes [1,2].

While users may require swift reaction to environmental stimuli, this requirement is in conflict with the general approach in sensor networks of relaying sensed data over multiple hops to a remote sink for processing. Moving computation closer to where data collection occurs is an attractive possibility in reducing the reaction time of applications [2,3]. Furthermore, the needs of users will undoubtedly vary, and the stimulus that triggers collection of data by sensor nodes will change as well, thereby requiring nodes that can be tasked dynamically. Static tasking is not efficacious in a dynamic environment that supports the needs of dynamic users.

In essence, allowing users to efficiently harness data collected by sensor networks requires dynamic tasking of nodes subsisting in a dynamic environment, as well as efficient processing and interoperability to increase the amount of data available and prevent proliferation of nodes. To this end, we introduce a Reconfigurable and Interoperable Sensor Network (RISN) overlay architecture capable of harnessing and processing information from surrounding networks, referred to as sub-networks herein thereafter. RISN (read as risen) relies on Field Programmable Gate Arrays (FPGAs) to provide increased computational capability and hardware reconfiguration of nodes. Furthermore, relying on the use of mobile agents, RISN provides users the ability to migrate tasks to locations of interest, to collect and process data from surrounding nodes, and accomplish their desired goals. As herein discussed, RISN provides:

- Increased processing power through the use of hardware accelerators to help support applications with severe computational constraints,
- Interoperability with surrounding networks to enable data sharing and prevent node proliferation,
- Dynamic tasking and reconfiguration of nodes through the use of mobile agents capable of forming ad-hoc agent networks to intelligently collect and process data of interest, and
- Service provision to efficiently support common needs of applications.

Applications that constantly need to adapt their execution patterns based on current observations can greatly benefit from RISN's ability to support dynamic tasking and reconfiguration of nodes. In general, object tracking applications are interested in continuously maintaining the location of an object as it traverses a sensed environment. Such applications need to adapt to the object's motion as well as the heterogeneity of the sensing environment. By that we mean that the tracker should be able to adapt to the fact that an object may not be traceable at times from one or more nodes using a particular feature set. The location of the target could however still be determined through other feature sets or sensing abilities of the network such as heat or sound signatures. Drawing on that logic, we implemented a distributed tracking algorithm atop of RISN, that is capable of adapting to the environment, and maintain the location of the object of interest despite the potential heterogeneity of sub-networks through which the object may travel.

In presenting our work, we will first present the background and work related to our proposal in Section 2. Section 3 presents the requirements of the system while discussing the prototype implemented. Section 4 evaluates our proposal and introduces a cost model for assessing the potential benefits of RISN. Section 5 discusses the aforementioned target-tracking application implemented atop RISN; finally, Section 6 concludes this discussion, highlighting our contribution and future works.

2. Background & Related Works

Generally aiming at increasing the processing power of nodes, the notion of using Field Programmable Gate Array (FPGA) nodes has been adopted in several research proposals. VAPRES was introduced to that end and shown to reduce processing time [4], however adoption of the system for large deployments appears infeasible, since the network remains specialized based on its localized sensing ability. Commuri et. al., proposed the use of FPGA nodes that can be reconfigured to perform data aggregation based on incoming queries [5]. The aggregated data is then simply relayed to base stations for actual processing and thus no attempt is made to process data close to its point of collection.

Addressing the issue of network interoperability, TinyWeb adapted the notion of web services to the sensor network environment [6]. TinyWeb nodes use the Web Service Description Language to advertise their interfaces. The proposed system assumes that node responses are simple and specified in advance in order to reduce code complexity and data overheads. The Semantic Sensor Web (SSW) addresses interoperability through metadata and contextual information from networks. SSW however does not address communication heterogeneity

[6]. IrisNet aims at providing an interface allowing users to query data collected over various heterogeneous networks [7]. IrisNet does not address communication heterogeneity, and sensed data are processed remotely.

The use of mobile agents to render nodes reconfigurable has also been explored in the literature. Systems such as Agilla [8] and ActorNet [9], to cite a few, have introduced mobile agent platforms to allow nodes to be reconfigured. One shortcoming of such systems is that they do not attempt to increase the processing power of nodes, which is crucial if such nodes are to be able to process sensed data locally. The aforementioned systems do not attempt to take advantage of the fact that the applications a node can support will be limited by its sensing abilities. Furthermore the issues of collecting and processing data from multiple, possibly heterogeneous networks is ignored.

Target tracking deals with the issue of following a particular object as it moves within an environment. Within the scope of Sensor Networks, the environment is limited to any area where the nodes involved in the tracking are present. Numerous schemes, both centralized and distributed, have been put forth to allow sensor nodes to track a target efficiently while minimizing the power consumption of the network as a whole [10-12]. Such proposals have generally assumed that the network is homogeneous in terms of its sensing ability. Distributed approaches to target tracking have led to the use of mobile agents to perform the task [13], however, the system still relies on a homogeneous sensing network.

To summarize, FPGAs have been proposed to increase the processing power of nodes mainly for data aggregation or for specialized configurations, but not to allow dynamic tasking of nodes. Proposals to address network interoperability generally assume communication homogeneity. Lastly, the mobile agent paradigm has found its niche in sensor networks, particularly for target-tracking, although network interoperability issues have not been addressed in such proposals. Through the introduction of RISN, as presented in the following sections, our work seeks to foster a networking environment that is reconfigurable and facilitates interoperability among heterogeneous sensor networks while providing increased local processing power through which users can take advantage of the large sets of data being harnessed.

3. The RISN System

The ultimate aim of RISN is to promote sensor network interoperability while allowing efficient dynamic tasking of nodes and improving data availability to users. To achieve these goals, RISN is, in essence, a deployed network composed of FPGA nodes with higher computational capability than traditional sensor networks. RISN utilizes a hardware-overlay capable of communicating with surrounding traditional sub-networks. To abstract the underlying heterogeneity of sub-networks from user applications, RISN utilizes a uniform data representation model and a common communication medium for nodes in the RISN overlay, either of which may differ from those of underlying sub-networks. RISN extends the traditional definition of a sensor network by viewing the network as a system with four interacting entities, namely 1) the RISN hardware-overlay, 2) the sub-networks, 3) a base station, and 4) software agents acting on behalf of the users. The interaction among the four entities is aimed directly at improving data availability to the user and allowing efficient processing of harnessed data. **Figure 1** depicts the RISN overlay and its interaction with the base station, and the sub-networks. In the following subsections, we present the main components of RISN, while showcasing its implementation in a Proof-of-Principle system prototype geared towards image processing applications.

3.1. RISN Overlay

The RISN Overlay is at the core of the RISN system. It is comprised of FPGA nodes, heretofore referred to as RISN or overlay nodes interchangeably unless otherwise noted, each of which includes a general-purpose processor (GPP). The GPP supports a generic set of instructions to allow software reconfigurability, while the FPGA accomplishes the same feat at the hardware layer. As the nodes in the overlay are FPGA-based, we used the ML405 evaluation board available from Xilinx [14] to develop the prototype. The board contains a Power PC405 microprocessor that is used as our GPP. The overlay nodes encompass five major hardware/software components, which are:

- Agent System
- Service Architecture
- Low-Level Tasks (LLT)
- Interoperability Interfacing System (IIS)
- Local Sensors

3.1.1. Agent System

The mobile agent-programming paradigm is focused around the ability of a program to halt its execution, and then move to a new environment where execution can be resumed. Agents are prime candidates to allow deployment of user applications to remote systems, and are thus adopted by the RISN system. The use of agents allows RISN nodes to be dynamically tasked based on the needs of user applications. The agents do not migrate in the sub-networks; but instead travel through the overlay, the

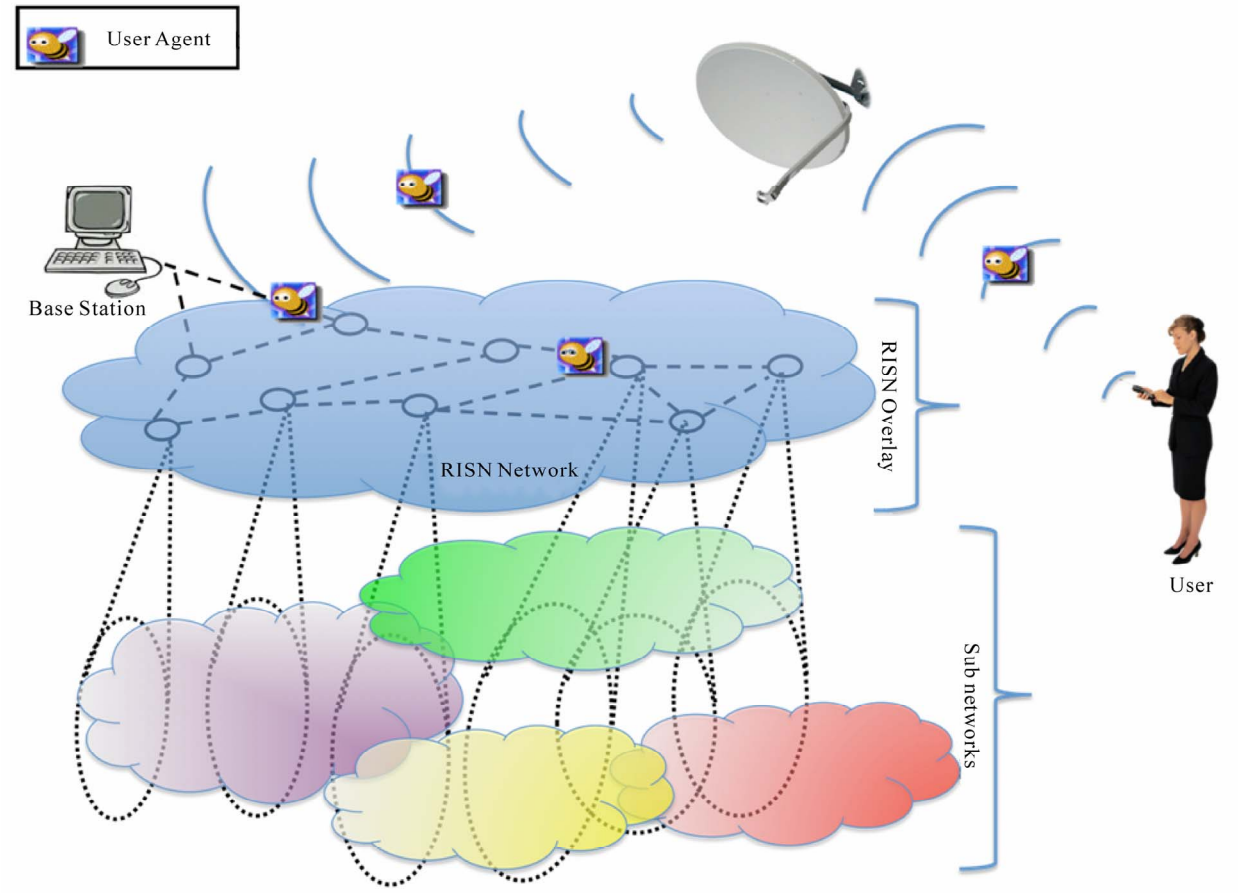


Figure 1. RISN network.

Base Station and the user devices.

The Agent System is intended to provide the interface between overlay nodes and user applications. Our choice of an agent platform to use was guided by our familiarity with agent platforms and the rankings of such platforms based on security, availability and other features, as conducted by Altmann *et al.* [15]. We thus opted for the use of Aglets [16], as our platform of choice. Upon migrating to a RISN node, the Aglets can use the resources available on the overlay node, along with the GPP, to perform the task at hand. Every overlay node contains a Static Service Agent (SSA) to allow migrating user agents to discover available resources on the current RISN node. Further details on the SSA are provided in the following subsection.

3.1.2. Service Architecture

The overlay nodes can be programmed to perform various functions, depending on their sensing abilities and the underlying sub-network. A subset of such functions can be provided as services to user applications, especially if there exists an efficient implementation of the function that may benefit the system. Such functions, as per our prototype, include the computation of histograms or the ability to track an object. RISN requires the presence of a service-based architecture in the overlay nodes to address that possibility. The service architecture must enable users to discover the services available in the system and on any particular RISN node. While the Base Station addresses the issue of detecting the services available in the system (see Section 3-B), service detection on a RISN node is met through provision of such information to local user applications using the SSA. The interaction between the SSA and user agents is accomplished through exchange of the following messages:

- GetAvailableServices: returns a list of names of services available (to the user) on the current overlay node. It is assumed that user agents know in advance how to interact with the services based on their names.
- GetServiceHandler: returns a *ServiceHandler* object associated with a particular service name. The object returned can be used to access the functionalities offered by the service.

As the SSA supports these two messages, the services in the system can be discovered and utilized through the

SSA; thereby allowing complex services to be composed from simpler ones. The ability of services to make use of each other's functionalities is one of the requirements of service-based architectures, herein handled through messages.

The *ServiceHandler* object, as implemented in the Proof-of-Principle prototype, is an abstract class through which users are provided access to the following functions:

- GetServiceName: retrieves the names of the services provided by the *ServiceHandler*.
- GetServiceMetaData: retrieves the metadata of the service being provided. Currently, the metadata is simply a string that must be parsed by user agents.
- Exec: instructs the handler to execute a particular command on an input object.

A RISN node's sensing ability and that of its underlying sub-networks limit the services that it can provide. Every overlay node provides communication and, when applicable, data collection services. Communication services allow users to send data to as well as receive data from underlying sub-networks. Data collection services provide access to data from local sensors. Within the prototype, the following functions of the *Service Handler* have a default implementation that can be overridden by IIS and local sensor handlers in order to provide communication and data Services.

- GetRawData: retrieves data from an underlying channel (local sensor or IIS).
- SendData: allows data to be sent to sub-networks, or if necessary, local sensors.

Other services that allow user applications to leverage the processing ability of LLTs are highly recommended, though not required, in implementations of the system. One should, however, note that the interaction of user applications with the LLTs of RISN nodes is expected to occur solely through services. The SSA is primarily intended to provide an interface for users to access LLTs, the IIS, and local sensors to support the RISN aim of yielding an interoperable sensor network with efficient processing of data.

3.1.3. Low-Level Tasks

Under the assumption that the sensing ability of a sensor node limits the applications that the node will be involved in, we introduce the notion of LLTs. For example, if the data streams of a node are all temperature readings, the node in question will primarily be monitoring changes in temperatures, and converting from one metric system to another. On the other hand, if the data streams originate from cameras, the node may be involved in object tracking or feature detection. Common tasks of applications can be abstracted and incorporated into the FPGA hardware as LLTs [1] to improve efficiency.

The LLT in our prototype supports image processing and is built with array processing in mind, providing users with the ability to perform various arithmetic operations on large or singular arrays. The LLT arithmetic operations are based on IEEE-754 single precision specification, with the ability to convert to and from 32-bit integers. The LLT also provides users with the ability to compute the histogram of an image based on a specified number of bins. The logic behind the provision of these particular operations as an LLT lies in the fact that these operations are the most basic commonalities for our goals. The Mean-Shift tracking algorithm [17] has been implemented using the basic LLT operations, showcasing the ability of the system to build complex services from simpler ones.

3.1.4. Interoperability Interfacing System

The Interoperability Interfacing System (IIS) is the component responsible for managing interactions between nodes in the overlay and those of the underlying sub-networks. In order to conduct its primary function of providing a communication medium between the overlay nodes and the sub-networks, IIS must deal with 1) interoperating communication protocols, 2) data format conversion, and optionally, 3) data aggregation. This is accomplished through coordination among the hardware and software modules incorporating the three tasks of the subsystem. IIS is designed with the ability to communicate through various protocols that may be in use by sub-networks in its vicinity. Communication with the underlying sub-networks may require IIS to encrypt/ decrypt data as per the requirements of such sub-networks. Once contact has been established with surrounding sub-networks, IIS provides user applications with the ability to communicate with such networks and leverage their observations and sensing abilities, which are unlikely to be resident in the overlay itself. This ultimately expands the amount of information available to users in accomplishing their tasks. While allowing data exchange between overlay nodes and underlying sub-networks is the primary task of IIS, it also reconciles heterogeneous data formats through automatic formatting of incoming and outgoing data streams. As a result, users can focus on coding the functionalities required by their applications instead of addressing data formats. Code snippets for format conversion can be eliminated from user applications; potentially reducing their size.

In the prototyped RISN, the IIS performs format conversion between the RGB-24 and YUV color formats, in the interest of presenting a unified representation of the image data. This conversion is performed using two

hardware accelerators, with the overlay nodes operating under the RGB-24 format. The format conversion is transparent to users and agent services. For simplicity, the prototype's IIS communicates solely over Ethernet, although in a real-world scenario, it could be designed to communicate over various communication media and protocols. Within the scope of our prototype, the system is limited to retrieval of image data from sub-networks by the overlay nodes. To reduce storage requirements, potentially conserve energy, and avoid processing of repetitive observations, IIS can also perform data aggregation, a highly desirable, yet optional, feature. Users have the option of accessing either aggregated or raw data, collected by the underlying sub-networks.

In short, IIS provides three main functionalities to the RISN system: 1) the ability to communicate with underlying sub-networks, 2) data format consistency, and optionally, 3) data aggregation to improve the overall performance of the system.

3.1.5. Local Sensors

The overlay nodes may also sense data from their environments. They are, however, not required to have any sensing ability and could exist solely for the purpose of maintaining network connectivity. The format of data sensed by sensors in the overlay must, however, be in compliance with the data format in use by the corresponding overlay. The prototyped RISN system utilizes overlay nodes with no local sensors for simplicity. A snapshot of the various components of an overlay node, and their primary interactions, is presented in **Figure 2**.

3.2. Base Station and Underlying Networks

The Base Station performs the same duties as its counterpart in a traditional sensor network, in that it essentially manages the network. We build on that notion to instill new functionalities in the Base Station to achieve our goals of interoperability and dynamic tasking. The Base Station in RISN executes an agent system with a stationary Base Station Agent (BSA). The BSA serves as an operating interface to users, providing the latter with pertinent information to help locate overlay nodes of interest and identify their capabilities. Interaction between the BSA and users occurs through the exchange of the following messages:

- LocateNode: Based on the geographic location or service names provided, the base station returns the identity of overlay nodes that could facilitate accomplishment of the user's task
- GetAllServices: Returns a list of all services available in the overlay nodes. Users can analyze the returned list to determine whether their applications can be supported. Note that users can still accomplish their tasks by relying more heavily on the GPP.
- GetNodeServices: Returns the services associated with a particular overlay node.

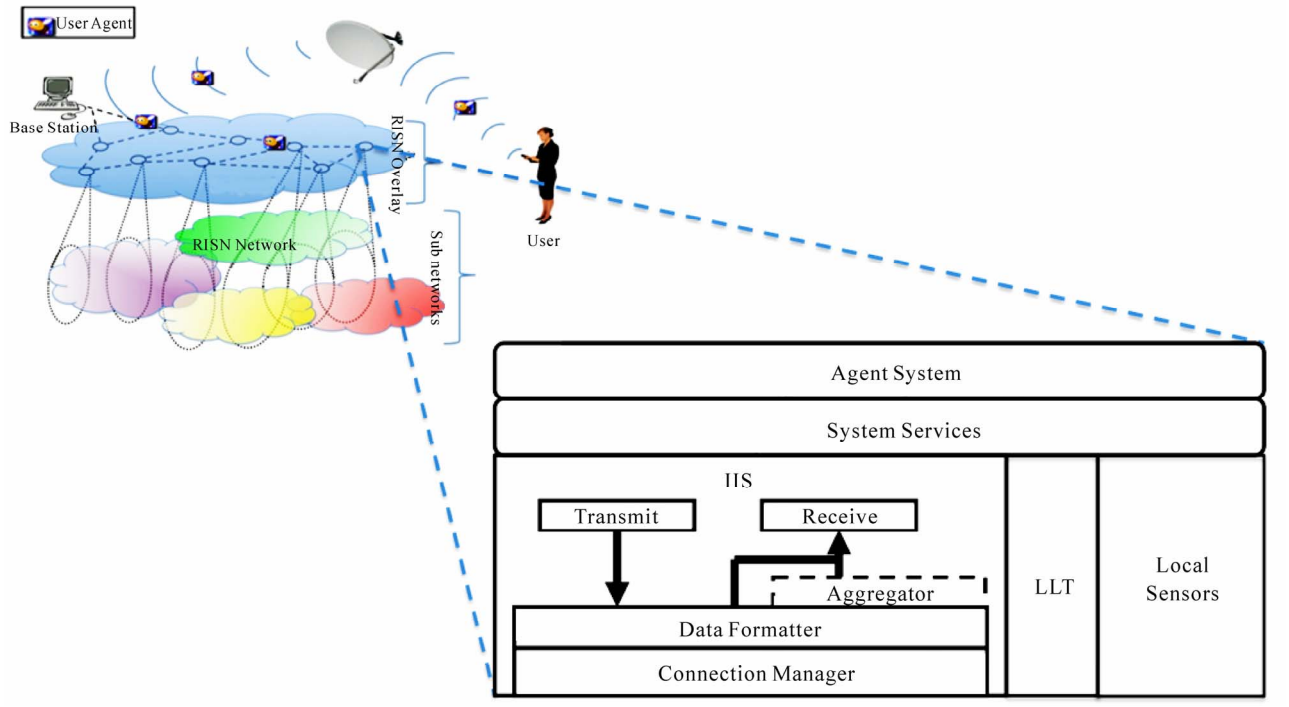


Figure 2. RISN network and components of a RISN overlay node.

In order to accomplish its task, the BSA needs to maintain the list of available services along with location and identity information for the overlay nodes. The Base Station in the prototyped system consists of a workstation with 1 GB of RAM and a 2.2 GHz Intel Xeon processor executing the Aglet server.

3.3. User Agents

The users in the system interact with other entities through the agent interface. To accomplish a task, a user contacts the Base Station and discovers the services available in the network, along with the location and identification of overlay nodes of interest. The user can then determine the overlay node to which an agent should be deployed. Users can deploy agents to the initial overlay node in one of two fashions; directly, if the overlay node is addressable from the user's location, or indirectly, by relaying the agent to the Base Station, where the agent can then migrate to the overlay node of interest. Note that the Base Station can address every overlay node in the system. The deployed agent can clone itself as necessary to form an ad-hoc agent network, as it carries out its goals. Authenticated agents must be allowed to traverse the overlay network and the RISN Base Station in search of data of interest. As RISN uses a homogenous data format, developers of user agents can focus on specifying the migration pattern of agents, along with access to and processing of information from any particular overlay node. While the computing device of the user may be resource-constrained and mobile, the user within the prototype developed resides on the same computer as the Base Station.

The user agent in the prototyped system is concerned with locating a target within the environment monitored by the overlay and underlying sub-networks. To implement the tracker, the user agent migrates to an initial overlay node, where the target of interest is expected to make its first appearance. The agent then migrates and clones itself accordingly in the overlay, for the purpose of maintaining and relaying the path taken by the target. Upon arriving at overlay nodes of interest, the user agent obtains the appropriate *ServiceHandlers* in order to access services harnessing the processing power and data of underlying components of the system. The *ServiceHandlers* in turn provide access to the services of the overlay node, by interacting with the "RisnNetwork" library through the Java Native Interface. The library provides access to the various drivers, implemented in C, that manage the IIS and LLT components. Note that in a full implementation of the system, the *ServiceHandler* would also interact with local sensors through the "RisnNetwork" library.

4. System Evaluation

4.1. RISN Overlay Node on the Xilinx ML405

The resources available on any FPGA board are limited. The same holds true for our underlying hardware platform, the Xilinx ML405. The ML405 board contains a Virtex-4 FPGA with 8,544 slices, and one PowerPC405 processor core, used as our GPP. The PowerPC405 processor is set to run at 300MHz, with 128 MB of RAM. The processor interacts with the peripherals on the system through the Processor Local Bus, running at 100 MHz. The prototyped overlay node encompasses hardware modules for IIS, LLT, Ethernet, RS-232 serial connection, and other system peripherals. The IIS and LLT were designed with memory limitations in mind. **Figure 3** depicts the resource utilization achieved by the Xilinx tools for the LLT subsystem, the IIS converters, and the general system, excluding the components reported separately. The programmed board is used to evaluate the computational and power efficiencies of our proposal, as presented in the following subsections.

4.2. RISN Overhead and Computational Efficiency

The proposed RISN system is general-purpose and useful for a broad array of applications, however, as noted earlier, our prototype specifically targets image processing. Our evaluation of the RISN overhead is intended to determine whether the agent-based computational model promoted by RISN degrades the performance seen by user applications, as compared to traditional models. Similarly, the goal of evaluating the system's computational efficiency is to determine the speedup in execution time afforded by the hardware accelerators as they are accessed through RISN.

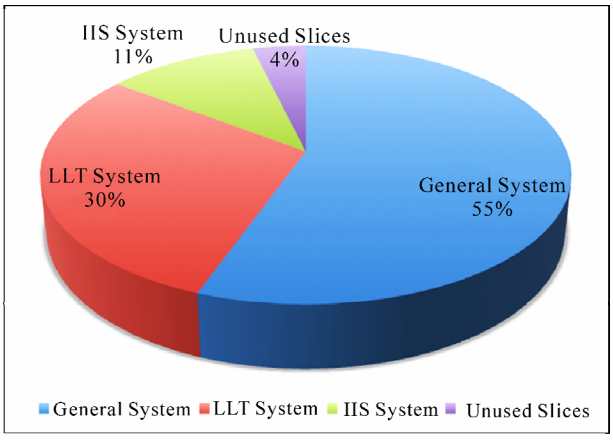


Figure 3. FPGA resource utilization.

As per our goal, three distinct execution times are measured: PPCSoftTime measures the time it takes a regular application to perform the computation of interest on the FPGA node. AgentSoftTime measures the execution time of a user agent that does not take advantage of the available hardware accelerators. Lastly, RISN_Time measures the execution time of a user agent harnessing the power of the hardware accelerators. Note that our experimental setup highlights the potential agent overhead as the difference between PPCSoftTime and AgentSoftTime. For each of the aforementioned execution times being measured, we experimented with different LLT floating-point operations, such as division, multiplication, addition, and square roots, with a varying number of array elements. We also measured the execution times of interest for the IIS operation of converting RGB-24 images to the YUV color space.

The results of our experiments are presented in **Figures 4** and **5**. **Figure 4** showcases the improved execution time that RISN can provide in maintaining format consistency across overlay nodes. AgentSoftTime converts 180000 pixels in 4.0746 seconds, while RISN_Time accomplishes the same feat in 0.5144, an 87.4% reduction in execution time. Figure 4 also highlights the negligible agent overhead incurred through the use of RISN's agent-based computational model, as the difference between AgentSoftTime and PPCSoftTime.

Figure 5 depicts the improvement in execution time RISN provides for applications involved in multiplying numbers or other operations in floating point format. For an array of 150000 elements, AgentSoftTime requires 1.576 seconds to compute their square; RISN reduces this execution time by 70.1% by performing the required computations in 0.471079 seconds. We should note that the RISN time presented in **Figure 4** and **Figure 5** does not include the time RISN takes to locate the appropriate *ServiceHandler* in the system, as this overhead, while not

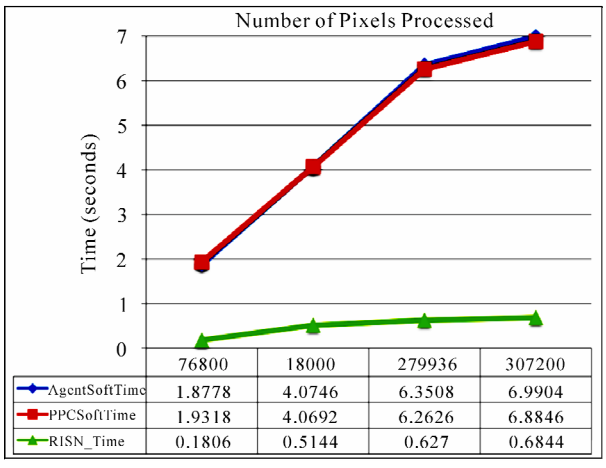


Figure 4. IIS computational efficiency.

a constant, is incurred only once per execution on an overlay node. Similar to **Figure 4**, the agent overhead displayed in **Figure 5** is negligible, especially in light of the improved performance afforded to user applications by the hardware accelerators. There is a considerable difference between the RISN_Time for computing square roots and software approaches, which we attribute to the fact that the square root is not a primitive operator in Java (The execution times of the square root operations are logarithmically scaled and presented in **Figure 5(d)**). Lastly, we must note that **Figure 5** also shows some variations in the execution time exhibited by RISN_Time. Such variations can be attributed to the following issues:

- The delay of the operating system in migrating data from user space to kernel space for usage by the hardware accelerators.
- Maintenance work by the Java Virtual Machine (JVM), as the agent system in use is Java-based.
- Inaccuracies of the JVM in reporting precise time durations.

4.3. RISN Power Consumption

The benefits and limitations of RISN are dependent upon the application and underlying networking infrastructure. We herein present a deployment model geared towards determining whether RISN is beneficial towards a particular application and networking system. The introduced model is used to study the power efficiency of our prototype.

4.3.1. RISN Strategic Deployment Model

RISN aims at improving data availability to users, while increasing responsiveness through hardware accelerators. In general, RISN must be cost-effective to warrant its application. Through RISN, applications can process data close to its point of collection; rather than relaying it to a processing center. In order to determine whether RISN's processing model should be employed for a particular application, we present a strategic deployment model that incorporates the two main factors in efficacy: communication load and power consumption. The proposed model analyzes the cost of utilizing RISN's local processing, versus relaying the data to a processing center.

$$C_{Comm} = NCB \tag{1}$$

The cost of communication is a function of the number of bytes, B, that need to be transferred. Since the data may need to traverse multiple nodes to reach its destination, the number of hops, N, must also be taken into account. C_{Comm}, as presented in Equation (1), represents the cost of sending data from a node to the Base Station, with

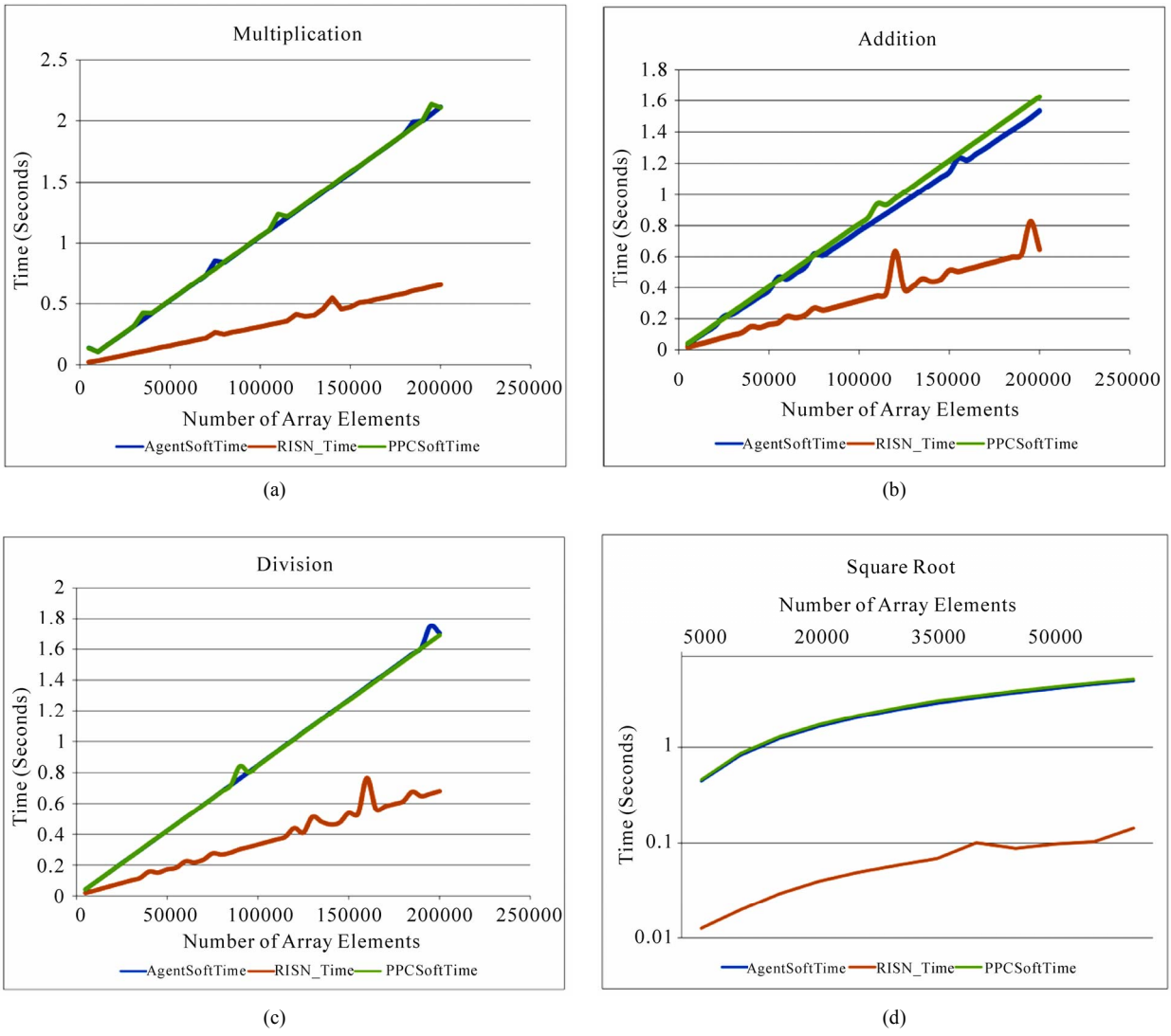


Figure 5. LLT computational efficiency.

C representing the cost per byte. At first glance, Equation (1) implies that simply reducing the amount of data that needs to be transferred is sufficient to justify the use of RISN for a particular application. However, the cost of communication is only one aspect of the system's overall cost. By implementing tasks in hardware, RISN introduces an execution cost in terms of power consumption. For every LLT implemented, there is an associated cost of dynamic and quiescent power usage. Dynamic power refers to the energy used by the LLT while it is in use, while quiescent power refers to the energy used by the hardware module when it is powered, yet inactive. The energy cost of implementing functionality in hardware also depends on the number of overlay nodes, M, that the application requires. This relationship is captured in Equation (2), where C_{Exec} is the power consumption; and *L* represents the load factor, *i.e.*, the likelihood that the LLT of an overlay node will be active. *D* and *Q*, respectively, represent the dynamic and quiescent power consumption of each overlay node.

$$C_{Exec} = M\left[(1-L)Q + LD\right] \tag{2}$$

Note that determining whether an application should make use of RISN's local processing is not equivalent to evaluating whether RISN itself should be used in the network. Instead, our deployment model aims to help designers determine, based on the amount of data to be transferred, and the expected overhead of power consumption, whether the data for a particular application should be relayed to a Base Station for processing, or be processed in the overlay.

In summary, the strategic deployment model introduces a means by which designers can determine whether the decrease in response time, and communication load

achieved by RISN justifies the overhead incurred in terms of power usage. Further insight on the deployment model can be acquired through the next subsection, which evaluates two tracking applications.

4.3.2. Communication Cost

Using the strategic deployment model, we compare the communication cost associated with performing target tracking, using the Mean-Shift tracking algorithm [17], based on RISN's processing model, with that of the traditional approach. The trackers implemented utilize the RGB-24 color format and process images of 640 × 480 pixels. The location of the target is maintained in both implementations as two 32-bit values representing the x and y coordinate. For clarity, we named the two trackers RISN_Tracker and Soft_Tracker, with RISN_Tracker being the implementation that uses the RISN processing model.

In our experiments, the cost associated with sending a byte of data over a network link is assumed constant for both RISN_Tracker and Soft_Tracker, as is the number of hops, N, that the data must traverse. C_{Comm} then becomes completely dependent upon the number of bytes that needs to be transferred. Both trackers, as implemented, are solely interested in the location of the target. RISN_Tracker needs to periodically relay eight bytes of data representing the target's new location. Soft_Tracker, however, must relay the image frames to be processed. As each pixel consists of 3 bytes, Soft_Tracker relays a total of 640 × 480 × 3 = 921,600 bytes, to the Base Station. The number of bytes that Soft_Tracker relays is directly dependent on the resolution of the cameras in the system. **Figure 6** depicts the theoretical effect of the image size on the communication cost associated with each tracker. The figure shows that RISN_Tracker is independent of image resolution, while Soft_Tracker is not. **Figure 6** also demonstrates that considerably less data needs to be relayed by RISN_Tracker to locate targets of interest. This is significant, as the energy cost associated with communication is generally high in any system; further detail on this statement is provided in the next section, where we evaluate the energy usage of the system.

4.3.3. Energy Cost of Execution

Determining the load factor of the system required extensive simulations. We used the Xilinx tools to generate the simulation model of the RISN node. We then relied on ModelSim to generate the Value Change Dump (VCD) used to estimate the power usage of the system through Xilinx's XPower power estimation tools. The tools estimated the toggle rate at 9.5% for the system as a whole. We use the toggle rate as our load factor for the system,

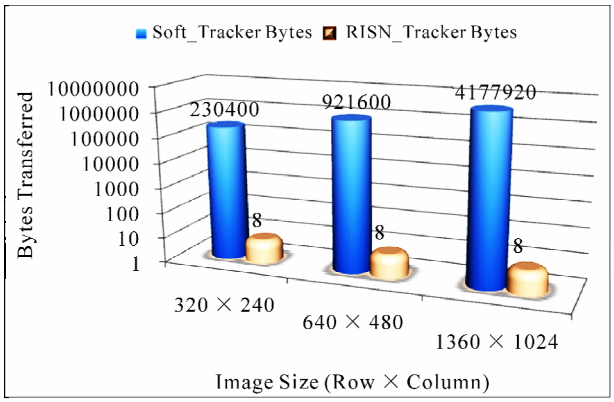


Figure 6. Communication cost.

as it measures the ratio of time that the system state changes relative to a clock input. **Table 1** shows the dynamic and quiescent power reported for various components of the RISN node (NR stands for Not Reported). Note that Ethernet uses more power than IIS and LLT combined; this is not surprising, as the energy cost of communication is expected to be high.

As the overlay nodes are FPGA-based, hardware modules, such as LLTs, are subject to quiescent power drainage. Intuitively, the benefits, with respect to the execution time, of performing a task in hardware must outweigh the potential drawbacks. In the case of the hardware modules in RISN, while communication load can be reduced, this must not occur at the expense of increasing the power usage of the system as a whole. Using Equation (2) and the estimated power usage from **Table 1**, the power consumption of an overlay node, with hardware accelerators, can be computed as (1 − 0.095) × (0.36191) + (0.095 × 1.7188) = 0.49081 W. On the other hand, the cost of execution with no hardware accelerators is (1 − 0.095) × (0.36191) + (0.095) × (1.7188 − 0.01717 − 0.00784) = 0.48844 W, as the application does not use the hardware modules.

The amount of energy used for an operation can then be determined, based on the time it takes for the operation to execute. **Figure 7** presents the energy (in J) used in the system, based on the number of array elements being processed. The figure clearly shows that even though RISN has higher power consumption, when the speedup afforded by the hardware accelerators is factored in; RISN actually uses less energy to perform the computation requested.

Lastly, **Figure 8** shows the theoretical effect of the percentage of dynamic power used by the RISN hardware modules (IIS and LLT). As the percentage of dynamic power used by the RISN hardware modules increases, the traditional software approach becomes more efficient in terms of required wattage when compared to RISN. Therefore, in order to maximize the benefits af-

Table 1. Estimated power usage.

	Quiescent (W)	Dynamic (W)
System as a whole	0.36191	1.7188
IIS	NR	0.00784
LLT	NR	0.01718
Ethernet	NR	0.03454

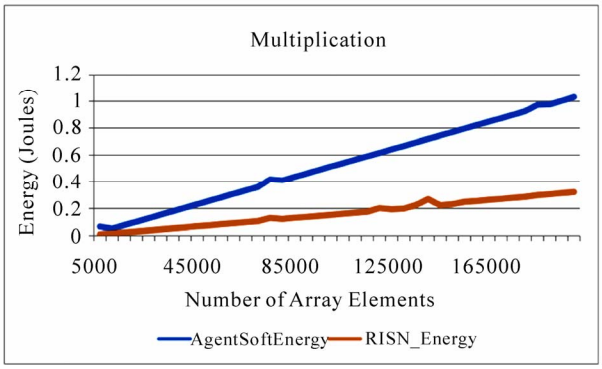


Figure 7. Energy used for multiplication.

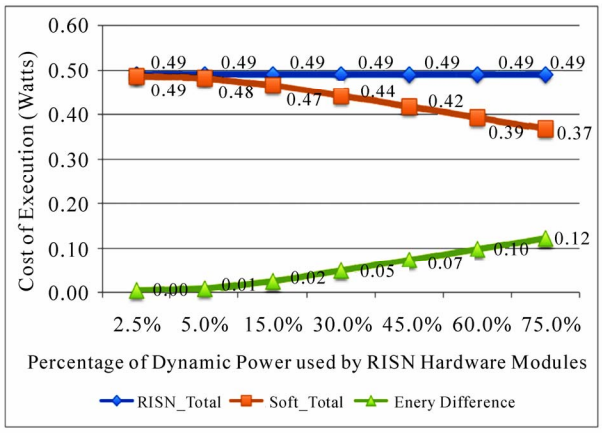


Figure 8. Theoretical effect of percentage of RISN hardware modules' dynamic power.

forded by RISN, namely, reduction of execution time and energy consumption, the percentage of a node's power used by the hardware accelerators should be minimized.

5. Autonomous and Distributed Target-Tracking

As a proof-of-concept of RISN's ability to ease development of distributed applications capable of executing in heterogeneous environments, we now present a distributed target-tracking implementation, independent of the RISN_Tracker discussed earlier, that continuously tracks and maintains the location of an object despite the network's heterogeneity and limited coverage of any one sensor.

5.1. Tracking System Architecture

Target tracking aims at continuously determining the location of an object of interest as it moves within an environment. With the assumption that the object of interest is the only mobile physical entity in the environment, target tracking must inevitably deal with the issue of the object moving out-of-range of one or more sensors or an entire isolated network. Furthermore, the potential heterogeneity of the sensors and their spatial deployment, their limited field of view and the potential for unexpected occlusions of the object of interest greatly complicates the task of determining the target's current location. However, since the primary function of all trackers is to return the location of an object in space, a distributed tracker can be independent of any one tracking algorithm implemented on a contributing sensor node. The distributed tracker can instead simply rely on the perceived location relayed by each sensor. As RISN leverages the data available from sub-networks, abstracts the heterogeneity of such networks while providing support for dynamic tasking and efficient processing, it is an attractive platform to develop such a distributed tracking application capable of continuously maintaining the location of the target despite the possibility of the object moving out-of-range of any one sensor as well as the heterogeneity of nodes' sensing abilities and required feature set used in tracking.

Our implementation relies on two main components, namely an Ad-Hoc Agent Network (AHAN), and a Tracking Service Handler (TSH) for each possible contributing overlay node. The two components interact to allow the location of the object of interest to be determined as it travels through the network despite the potential sensing heterogeneity of the nodes. The system works by deploying an agent to the overlay node with sensing coverage of where the object of interest is expected to appear initially. The deployed agent uses the appropriate representation of the object as specified in the Target field of the agent, to determine the current location of the object through communication with the TSH of the overlay node. Using the returned location, the agent clones itself and dispatches the clones to overlay nodes with sensing coverage of the target's perceived path as determined by TSH. The initial agent and its clones form the AHAN, which is described in the next section.

5.1.1. Ad-Hoc Agent Network

The Ad-Hoc Agent Network (AHAN) is made up of co-

ordinating agents dispatched to locations of interest. The network determines the current location of the target based on the individual tracking results received. The agents contain a Target field that represents the object being tracked. Target maintains past locations of the object and the possibly diverse set of features that may be used to locate the object. To illustrate the latter point, consider that an object's appearance and location, within the scope of computer vision, can be represented as parametric and non-parametric probability densities, as well as active appearance models [18]. Furthermore, various other features such as heat and sound signatures can also be used to represent the object. To deal with such a vast set of possible feature representations of an object, Target is represented as a class capable of maintaining various representations of the object of interest, each of which is accessible through their statically predefined names. The appropriate representation of the object can be retrieved in order to determine the object's current location on any particular overlay node by using the retrieveObjectModel (String modelName) method of Targets. Note that there is an underlying assumption that Target is pre-configured with any necessary object model parameterization that might be required by a TSH in order to perform tracking under possibly changing environments.

Using the Target and its path, clones of the initial agent are dispatched to appropriate neighboring overlay nodes forming the AHAN. The clones and the initial agent communicate through the following messages:

- cmdGetTargetLocation: instructs clones to determine the current location of the target of interest using the sensor data streams managed by the clone.
- DestinationAddress: specifies the address to which the clone should migrate to and which sensor data stream it will manage at the destination.
- cmdTerminateClone: terminates execution of the receiving clone and frees up used resources.

The initial agent migrates through the network as the object moves; clones are terminated when they are no longer capable of helping in determining the current location of the target. The initial agent creates a polygon consisting of the retrieved locations of the object for the current iteration. The center of the constructed polygon is used as the location of the target perceived by the system.

As we mentioned earlier, the agent network is built starting with an initial agent, executing on a node from which the target is supposed to be locatable. The initial agent consults with the TSH in order to determine which neighborhood overlay nodes can help in determining the location of the target. Details regarding the interaction between the agent network and the TSH are provided in the following section.

5.1.2. Tracking Service Handler

TSH abstracts the heterogeneity of various implementations of trackers or associated feature sets from the Agent Network, thereby allowing for the object to be located based on suitable features for the current overlay node. TSH is essentially a RISN tracking service available through a handler. TSH is initialized by specifying the size of the 2D virtual space being monitored as well as specification of the sensors available on the overlay node along with the address of neighboring overlay nodes. For each sensor on the local overlay node, the handler implements the appropriate and possibly optimal tracking algorithm for the sensor. The handler also determines and maintains the Field Of View (FOV) of each local sensor, defined as a mapping of the area covered by the sensor onto the virtual space. Lastly, the FOV covered by sensors of neighboring overlay nodes is retrieved through communication with remote SSAs. In essence the handler allows execution of the following commands:

- GetNodeFOV: retrieves the FOV of a particular sensor stream available on an overlay node.
- cmdFutureAddressesOfPoint: returns the address and IDs of known sensor streams whose FOVs intersect with the specified point.
- cmdFutureAddressesOfSegment: returns the address and IDs of known sensor streams whose FOVs intersect with the specified segment.
- cmdFutureAddressesOfLine: returns the address and IDs of known sensor streams whose FOVs intersect with the specified line.
- cmdFOVIntersectPoint: returns true if the FOV of the specified sensor intersects with the specified point.
- cmdFOVIntersectSegment: returns true if the FOV of the specified sensor intersects with the specified segment.
- cmdFOVIntersectLine: returns true if the FOV of the specified sensor intersects with the specified line.
- cmdTrack: retrieves the required feature representation of the target and attempts to locate the current location of the object in the neighborhood of the target's last known location. The new location of the object is returned, without any mapping to the virtual space.
- cmdMapFromVirtual: maps the specified location in world coordinates to the sensor's local coordinate indicated on the overlay node.
- cmdMapToVirtual: maps the specified sensor's local coordinate to the system's world coordinates.

Details of the interaction between the components of the distributed tracker are provided in the next section as we walk through an implementation of the tracker.

5.2. Tracking System Implementation

The tracking system herein implemented ultimately relies on the RISN framework herein introduced. As such, in introducing the tracking system implementation, this section also highlights RISN's ability to leverage data from isolated networks, intelligently process sensor observations and abstract the potential heterogeneity of underlying networks. The proposed distributed tracker is implemented through the use of 2 RISN nodes (Node-1 and Node-2) attached to the base station from our earlier experiments. Each RISN node manages 4 cameras whose FOV are mapped onto a two dimensional "virtual" space representing the X and Y world coordinates of the object. Node-1 manages Cam1, Cam2, Cam6 and Cam4; while Node-2 manages the remaining four cameras. The reason behind this setup is to emulate an overlay network consisting of the RISN nodes managing two potentially isolated and heterogeneous sub-networks with overlapping coverage areas.

The cameras act as sensors of the sub-networks controlled by RISN; while the 2 nodes embody nodes in the RISN overlay that the distributed tracker depends on in accomplishing its task. The cameras do not share the same resolutions; they were calibrated offline and their projection matrices, mapping individual image coordinates to world coordinates, were computed and loaded onto the appropriate TSH. The handler uses these matrices to determine the FOV of each corresponding camera and to perform bi-directional mapping from the camera's local coordinate to the world coordinates of the distributed tracker. In so doing, user agents only need to deal with world coordinates, as the TSH abstracts the heterogeneous image coordinate system of each camera.

Once the handler is setup, having acquired the FOV of neighboring sensors, the user agent is deployed on Node-1, which maintains Cam2. The user agent, upon arriving at Node-1, initializes the target based on the specified location of the target in the initial frame. The target in question is a red ball whose appearance is modeled using a color histogram. For the purpose of experimentation, we work with a target whose representation is independent of the viewing angle of any one camera, thereby abstracting issues that may arise due the fact that the histogram representing an object can be very different depending on the viewing angle used to compute the histogram. For each frame of the test video sequence, and the last known location of the object, the user agent intelligently determines which sensors on which overlay node can be used to track the target. If there is no clone managing a suitable sensor, one is dispatched; else a message is relayed to the clone asking for the updated location of the object.

Figure 9 presents a pictorial representation of the FOV of all eight cameras available in the system, with that of Cam2 highlighted in green. **Figure 9** also displays the locations of the object as it is tracked by the dispatched user agent based on the location of the object in the initial frame, represented as F1 in the figure. From the first frame, F1, to the 27th frame (F27), the object is visible in the FOV of Cam2. On the 28th frame however, the user agent must rely on the other cameras in the system to maintain the location of the object. It is worth noting that at frame 43 (F43), the target leaves the FOV of all cameras under the control of Node-1. In order to maintain the location of the object, as per our experimental setup, the user agent must thus rely on data from neighboring networks. The AHAN takes over by dispatching an agent to Node-2 based on the expected path of the object and its intersection with the coverage area of the sensors managed by Node-2. In our experiment, this resulted in the system tracking the object using Cam3, Cam5 and Cam8 from Node-2, thus leveraging, and intelligently processing data from isolated and heterogeneous networks to accomplish a common goal.

The initial location, (F1), of the object being tracked as seen by Cam2 is displayed in **Figure 10(a)**; while **Figure 10(b)** shows the location of the object seen by Cam2 after 27 frames (F27) have been processed. On the 28th frame (F28), the object is no longer visible by Cam2; however, it is still visible by Cam1 in the same sub-network. In the 43rd frame (F43), the object also leaves the FOV of Cam1, thereby becoming invisible to Node-1. The tracker is able to maintain the location of the object in world coordinates, despite the fact that it is no longer visible from the initial camera or sub-network.

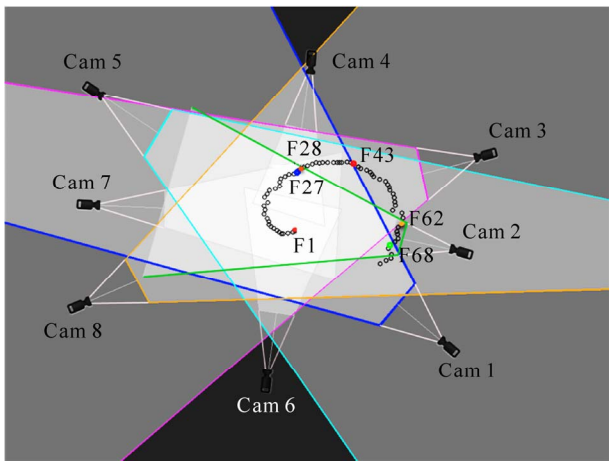
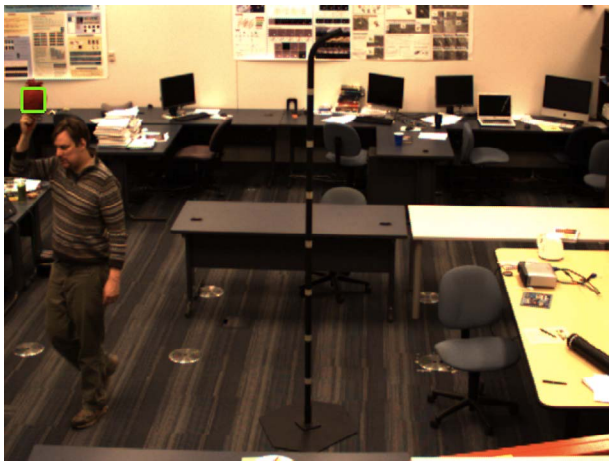


Figure 9. Overview of tracking system.

(a)



(b)

(c)

Figure 10. Distributed tracking system.

When the object reappears in frame 62, (F62), it is accurately located by the user agent using Cam2 as shown in **Figure 10(c)**, by relying on the information harnessed from the other sensors in the system. In frame 68 (F68), the object again becomes visible to Cam1.

As the handlers for each camera is responsible for implementing the suitable target-tracking algorithm, the system makes no assumption of the homogeneity of each tracker. As a result, each Tracking Service Handler could, in theory, implement a different tracking algorithm. However, in our implementation, we only used one tracking algorithm, namely the Mean-Shift algorithm. The only requirement is that the Target is able to supply to the handler the necessary parameters on which to operate by using the *retrieveObjectModel* method.

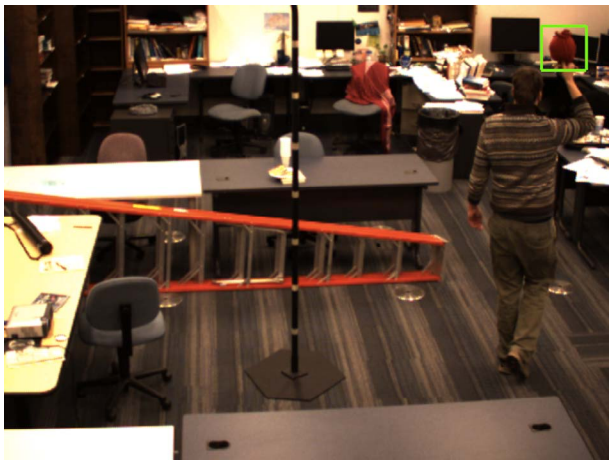
Although we assume that the representation of the object is independent from the sensors' point of views and that the sensing devices in the system consist solely of cameras; the size of the object however varies depending on its distance from any particular camera. Thusly, it is important that the TSH of each node is able to maintain and adjust the size of the object from each camera independently. This is accomplished by dynamically adjusting the bandwidth parameter of each Mean-Shift tracker using the method proposed in [17]. **Figure 11(a)**, **Figure 11(b)**, and **Figure 11(c)** showcase how the TSH adjusts to the changing size of the object in successive frames from Cam7.

6. Conclusions

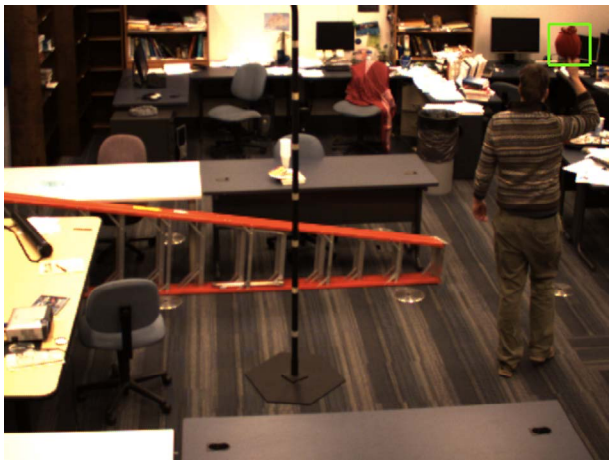
Our work has introduced a novel approach to sensor networks that aims to allow existing sub-networks to interoperate, while granting applications the ability to efficiently harness and process data. We have also proposed a strategic deployment model to help decide whether harnessing the processing power of LLTs can help improve the reaction time of applications while not imposing severe strain on the network's power consumption. Abstraction of the network and heterogeneity of data formats allows user applications to focus on their tasks.

The use of services enables developers to maximize the efficiency of the system by providing efficient implementations of common tasks, while balancing speed and power requirements. We have also shown how an autonomous distributed target tracking application that is independent of the sensing abilities of any one node can be implemented on the system. The tracking application discussed simply requires the presence of a Tracking Service Handler, which abstracts the sensing heterogeneity of nodes, along with the presence of all possible feature set representation of the object of interest, in order to maintain the continuous locations traversed by the object as it moves in and out of the view of any nodes.

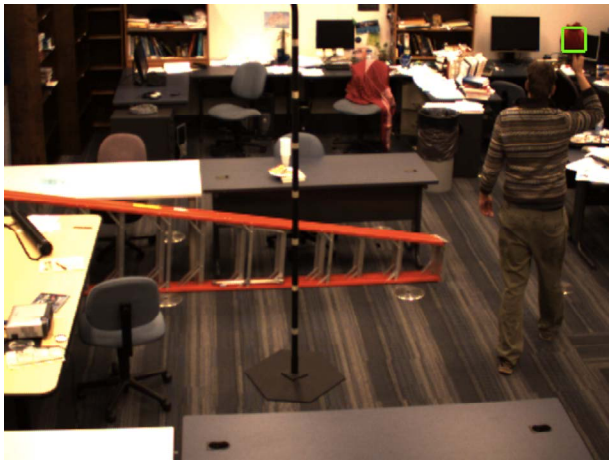
One stated goal of sensor networks is to occupy an



(a)



(b)



(c)

Figure 11. Dynamic adjustment of tracking bandwidth.

area with minimal disturbances to the environment and its occupants. With the deployment of a new network for every new task, this goal is fated to be breached by leading to proliferation of nodes. With RISN's ability to interoperate with other networks, the number of networks with similar sensing abilities that need to be deployed in an area can be greatly reduced, as the system facilitates interoperation and leverages available resource for processing by user agents. Furthermore, we have shown through our analysis that RISN can reduce execution time by over 70% and considerably reduce communication load over network links. The latter can be crucial to traditional sensor networks, as communication typically consumes considerably more energy than any other task in the system.

Future work will investigate if RISN can increase the lifetime of one or more power-starved networks. Addressing the security of overlay nodes will also be studied to prevent intruders from controlling sub-networks. The work described in this paper assumed that the communication protocols and data formats of underlying sub-networks are known in advance. Future work will address location of existing sub-networks by the IIS components, as well as reconciling heterogeneity of protocols and data without a priori knowledge of their exact nature. Methods for communicating encrypted data and preventing compromise of the encryption keys in the course of "discovering" new sub-networks will also be investigated. The prototype presented utilized wired Ethernet, as opposed to wireless communication. Future works will take into account the complications that RISN will face in wireless environments.

7. Acknowledgements

The National Science Foundation under the contract IIS-0324835 in part has supported this work.

We gratefully acknowledge the tools and hardware provided by Xilinx.

Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL) managed by UT-Battelle, LLC, for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

8. References

[1] E. Jean, R. T. Collins, A. R. Hurson, S. Sedigh and Y. Jiao, "Pushing Sensor Network Computation to the Edge," *Proceedings of 5th International Conference on Wireless Communications, Networking and Mobile Computing*, Beijing, 24-26 September 2009, pp. 1-4. doi:10.1109/WICOM.2009.5302659

[2] E. Jean, Y. Jiao, A. R. Hurson and V. Kumar, "Pushing Sensor Network Computation to the Edge while Enabling Inter-Network Operability and Securing Agents," *Proceedings of 3rd International Innovations and Real-Time*

Applications of Distributed Sensor Networks Symposium, Shreveport, 26-27 November 2007, pp. 66-75.

[3] Y. Jiao and A. R. Hurson, "Performance Analysis of Mobile Agents in Mobile Distributed Information Retrieval System—A Quantitative Case Study," *Journal of Interconnection Networks*, Vol. 5, No. 3, pp. 351-372. doi:10.1142/S0219265904001210

[4] R. Garcia, A. Gordon-Ross and A. D. George, "Exploiting Partially Reconfigurable FPGAs for Situation-Based Reconfiguration in Wireless Sensor Networks," *Proceedings of* 17*th IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, 5-7 April 2009, pp. 243-246. http://doi.ieeecomputersociety.org/10.1109/FCCM.2009.45

[5] S. Commuri, V. Tadigotla and M. Atiquzzaman, "Reconfigurable Hardware Based Dynamic Data Aggregation in Wireless Sensor Networks," *International Journal of Distributed Sensor Networks*, Vol. 4, No. 2, pp. 194-212. doi:10.1080/15501320802001234

[6] A. Sheth, C. Henson and S. S. Sahoo, "Semantic Sensor Web," *IEEE Internet Computing*, Vol. 12, No. 4, pp. 78-83. doi:10.1109/MIC.2008.87

[7] P. B. Gibbons, B. Karp, Y. Ke, S. Nath and S. Seshan, "IrisNet: An Architecture for a Worldwide Sensor Web," *IEEE Pervasive Computing*, Vol. 2, No. 4, pp. 22-33. http://doi.ieeecomputersociety.org/10.1109/MPRV.2003.1251166

[8] C. Fok, G. Roman and C. Lu, "Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications," *Proceedings of the* 25*th IEEE International Conference on Distributed Computing Systems*, Columbus, 10 June 2005, pp. 653-662.

[9] Y. Kwon, S. Sundresh, K. Mechitov and G. Agha, "ActorNet: An Actor Platform for Wireless Sensor Networks," *Proceedings of the* 5*th International Joint Conference on Autonomous Agents and Multiagent Systems*, Hakodate, 8-12 May 2006, pp. 1297-1300.

[10] S. Pattem, S. Poduri and B. Krishnamachari, "Energy-Quality Tradeoffs for Target Tracking in Wireless Sensor Networks," *Proceedings of* 2*nd International Workshop of Information Processing in Sensor Networks*, Palo Alto, 22-23 April 2003, pp. 32-46.

[11] H. Yang and B. Sikdar, "A Protocol for Tracking Mobile Targets Using Sensor Networks," *Proceedings of* 1*st IEEE International Workshop on Sensor Network Protocols and Applications*, Anchorage, 11 May 2003, pp. 71-81. doi:10.1109/SNPA.2003.1203358

[12] W. Zhang and G. Cao, "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks," *IEEE Transactions on Wireless Communications*, Vol. 3, No. 5, pp. 1689-1701. doi:10.1109/TWC.2004.833443

[13] L. Szumel, J. LeBrun and J. D. Owens, "Towards a Mobile Agent Framework for Sensor Networks," *Proceedings of* 2*nd IEEE Workshop on Embedded Networked Sensors*, Sydney, 30-31 May 2005, pp. 79-88. http://doi.ieeecomputersociety.org/10.1109/EMNETS.2005.1469102

[14] Anonymous Xilinx Documentation for ML405 Board, July 2008.

[15] J. Altmann, F. Gruber, L. Klug, W. Stockner and E. Weippl, "Using Mobile Agents in Real World: A Survey and Evaluation of Agent Platforms," *Proceedings of* 2*nd Workshop on Infrastructure for Agents*, *MAS and Scalable MAS at Autonomous Agents*, Montreal, 28 May-1 June 2001, pp. 10-16.

[16] D. B. Lange and M. Oshima, "Programming and Deploying Java Mobile Agents with Aglets," Addison-Wesley, Boston, 1998.

[17] D. Comaniciu, V. Ramesh and P. Meer, "Real-Time Tracking of Non-Rigid Objects Using Mean Shift," *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Hilton Head Island, 13-15 June 2000, pp. 142-149.

[18] A. Yilmaz, O. Javed and M. Shah, "Object Tracking: A Survey," *ACM Computing Surveys*, Vol. 38, No. 4, pp. 1-45.

