Towards a Virtualized Sensing Environment

David Irwin, Navin Sharma, Prashant Shenoy, and Michael Zink

University of Massachusetts, Amherst
{irwin,nksharma,shenoy}@cs.umass.edu,zink@ece.umass.edu

Abstract. While deploying a sensor network is necessary for proof-of-concept experimentation, it is a time-consuming and tedious task that dramatically slows innovation. Treating sensor networks as shared testbeds and integrating them into a federated testbed infrastructure, such as FIRE, GENI, AKARI, or CNGI, enables a broad user community to benefit from time-consuming deployment exercises. In this paper, we outline the challenges with integrating sensor networks into federated testbeds in the context of ViSE, a sensor network testbed we have integrated with GENI, and describe our initial deployment experiences. ViSE differs from typical embedded sensor networks in its focus on high-bandwidth steerable sensors.

Key words: Testbed, Sensor Network, Federation, Radar.

1 Introduction

Apart from the additional burden of developing, operating, and maintaining an experimental shared testbed, researchers do not typically co-opt their field-deployed sensor networks to serve as shared testbeds for at least three reasons. **First**, the benefits of multiplexing node resources are unclear, since computational, bandwidth, and energy constraints restrict embedded platforms to operating a few passive sensors that collect similar data regardless of the application. Thus, controlling the data gathering process at each sensor node has little value, and end-user applications may simply "share" the collected data at the network's sink.
Second, embedded platforms generally do not support the hardware mechanisms, e.g., MMU or TLB, or software abstractions, e.g., virtual memory or processes, necessary for either basic control plane functions or efficient multiplexing, making it undesirable or even impossible. Software platforms for embedded nodes often do not provide the separation between kernel-space and user-space that testbed's use as the linchpin for safe execution of untrusted user software.
Third, the specialized nature of a sensor network deployment often lends itself to only a single application rather than multiple diverse applications. For example, the choice of sensors and node placement for a network designed specifically for monitoring the vibrations of a volcano [3] are unlikely to be suitable for, say, tracking the movements of nearby wildlife [4].

However, there are a range of "large" sensor types that exist today that do not adhere to the extreme power and form factor constraints of embedded sensor

nodes and, instead, (i) expose programmable sensor actuators to applications, (ii) support the basic mechanisms/abstractions for safe software execution, and (iii) incorporate multiple types of sensor supporting a range of applications. Most notably, networks of steerable sensors are distinct from their embedded counterparts by allowing applications to dictate the type, quality, and quantity of data they collect by steering the sensor to different points in space.

Unlike passive sensors that collect the same data regardless of the application, the data gathering process for each steerable sensor is highly application-dependent. Further, the energy required to move a steerable sensor's mechanical motor necessitates the use of higher-power computing components that support both the hardware mechanisms and software abstractions critical for a testbed control plane. Finally, node platforms are generally powerful enough to operate a range of sensors useful for different applications. Examples of steerable sensors include both steerable weather radars and pan-tilt-zoom video cameras. Recent work has proposed using networks of small steerable weather radars to fill coverage gaps in the NEXRAD radar system [5], while the U.S. Border Patrol uses networks of pan-tilt-zoom cameras to monitor both the northern and southern border [6].

Thus, the basic characteristics of steerable sensor networks do not preclude exposing them as shared testbeds for use by external researchers. As with their embedded counterparts, steerable sensor networks are costly to deploy, operate, and maintain, which magnifies the potential benefits of sharing them with external users. For instance, he hardware cost alone for CASA's steerable radars is $250,000 and does not include infrastructure, operational, or labor costs [5], and the cost for the Border Patrol's 20-mile "virtual fence" using pan-tilt-zoom cameras is over $20 million [6]. In general, much like early mainframe computing systems, the cost and expertise necessary to build and maintain sensing systems significantly restricts the scope of users available to experiment with them.

We also believe that continuing advancements in low-power embedded sensor nodes will eventually mitigate, or even eliminate, the characteristics that discourage testbeds using small form factor field-deployed sensor nodes. For instance, in Section 4.1, we briefly discuss a prototype of a small form-factor sensor node we have built capable of (i) separating control plane functions from user functions using two distinct processors and (ii) supporting multiple types of "high-power" sensors off harvested energy. Steerable sensor networks provide an early opportunity to address challenges, such as control plane separation, multiplexing shared sensor actuators, and dealing with unpredictable energy availability, that are, or will be, relevant to all types of sensor network testbeds.

The focus of this paper is the design and implementation of the ViSE [1] testbed and its integration with ORCA [7, 8], a candidate control framework for GENI [9]. ViSE focuses on *virtualizing* steerable sensors to benefit from the strong resource and fault isolation of modern virtualization platforms. Virtual machines isolate ViSE's control plane from untrusted users, and untrusted users

[1] See `http://geni.cs.umass.edu/vise`. ViSE stands for **Vi**rtualized **S**ensing **E**nvironment.

(a) Prototype Node (b) Topology

Fig. 1. ViSE contains nodes (a) spread throughout the Amherst, MA area (b).

from each other. Each ViSE node includes a weather radar, a pan-tilt-zoom camera, and a weather stations that applications programmatically control to sense aspects of their surrounding environment. Thus, ViSE users request slices composed of virtual machines bound to not only isolated slivers of each node's CPU, memory, storage, and bandwidth, as in other GENI testbeds [10, 11], but also one or more attached sensors.

ViSE reflects our view that new innovation in sensing systems is necessary to take advantage of new innovation in the Internet and cloud computing, since sensors must transmit their data over the Internet to the computers that ultimately process the data to some end. Section 2 gives an overview of ViSE, while Section 3 details ViSE's current integration with Orca/GENI, using Orca's extensible slice controller implementation, as well as deployment issues we have observed in practice. Section 3 then briefly outlines three specific challenges for ViSE, and other sensor network testbeds, moving forward. Finally, Section 4 discusses concludes.

2 ViSE Testbed Overview

The ViSE testbed currently includes one Internet-accessible node that acts as a gateway to three sensor nodes located on the roof of the UMass-Amherst Computer Science Research Center at 140 Governors Drive in Amherst, MA (42 23' 42.33" N, 72 31' 50.96" W at elevation 62 meters), on the MA1 Radar Tower on the UMass-Amherst campus (42 23' 30.95" N, 72 31' 2.53" W at elevation 120 meters), and on the Mount Toby firetower in Sunderland, MA (42 29' 17.00" N, 72 32' 14.96" W at elevation 385 meters). The distance between Mount Toby and the UMass-Amherst Computer Science Research Center is 10.37 kilometers and the distance between Mount Toby and the MA1 Tower is 10.83 kilometers. There is no link between the MA1 Radar Tower and the Computer Science Research Center since there is no line of sight.

Figure 1(a) shows the ViSE node on the roof of the UMass-Amherst Computer Science building. The relative location of each node is depicted in Fig-

ure 1(b). We are planning to add one additional node on the Pelham firetower, approximately 10km east of the MA1 tower, and other nodes on campus buildings in the near future. Each node includes three distinct sensors, a Davis VantagePro2 Weather Station, a Sony SNC-RZ50N Pan-Tilt-Zoom Camera, and a Raymarine RD424 Radome Radar Scanner. Testbed nodes use 802.11b over directional antenna for communication. Importantly, programmatic control is available for each sensor, allowing users to build sensor control into their experimental applications. The ViSE testbed is part of the GENI prototype and integrates with GENI's ORCA control framework, which we describe in the next section.

ViSE's primary usage scenario is as a platform for experimenting with closed-loop control of adaptive sensor networks using non-traditional steerable sensors. Experiments actuate sensors to capture data at a specific time, location, spatial region, etc., stream that data over a wireless network to compute clusters for analysis, and use the new results to actuate and refocus sensors on important regions as conditions change. For example, recent work [12] explores how shared high-bandwidth sensor systems can intelligently prioritize and compress data when not enough bandwidth exists to transmit all of the sensor data. ViSE also targets the study of multi-sensor experiments, such as performing longitudinal climate studies or studying the fidelity of the long-distance wireless link under different atmospheric conditions including rain, snow, wind, or fog [13]. Finally, ViSE is also useful for experimenting with long-distance wireless communication using directional antennas—the testbed includes two links over 10 kilometers long. Setting up long-distance links, like those in ViSE, is cumbersome since they require line-of-sight from elevated vantage points.

3 GENI/ORCA Integration

As with other GENI testbeds [7, 10, 11], ViSE uses virtualization to separate its control plane from each user's data plane. The control plane has mechanisms to start and stop user VMs, create and destroy VM root disks, attach and detach sensors to VMs, and configure network connections. Users currently request a *slice* of the testbed by logging into ViSE's web portal and issuing a slice request. Each ViSE slice consists of a virtual machine on each node bound to an isolated *sliver* of each node's CPU, memory, storage capacity, network bandwidth, as well as one or more attached sensors [2]. Note that since, currently, ViSE has a chain topology every slice *must* include a virtual machine on each ViSE node—otherwise, users would have no means of accessing virtual machines at the end of the chain. ViSE users may log into the gateway of their slice using a secure shell session at `vise-testbed.cs.umass.edu` on a specified port. ViSE currently uses `ssh` as the method for users to bootstrap their own services and/or inject their own code into a slice, although we are working on integrating the Gush experiment workflow tool [14]. Once inside the gateway, ViSE nodes operate within a private IP address space: the node on the roof of the Computer Science

[2] See [9] for complete description of terminology.

Research Center has IP address 192.168.2.25, the node on the firetower at Mount Toby has IP address 192.168.2.26, and the node on the MA1 Tower on the UMass-Amherst campus has IP address 192.168.2.27.

3.1 Requesting Slices

To gain access to ViSE, each user sends a `ssh` public key to `vise@cs.umass.edu` and receives in return a unique username and password to access ViSE's web portal. Slice requests made through the portal under each login account are tagged by an Orca slice controller, discussed below, with the public key associated with that account. As with other testbeds, on slice creation, each user's `ssh` public key is copied to the root disk image of each of its virtual machine, allowing it to access the machine with its corresponding private key. ViSE currently uses Linux VServers as its virtualization technology because it simplifies attaching privileged sensing devices, such as cameras and radars, to virtual machines. We initially used Xen as our virtualization technology. However, we were forced to switch to VServers since the device driver for our radar's analog-to-digital convertor uses DMA operations that Xen does not currently support. Since ViSE focuses on exposing sensors to users, the OS-level virtualization provided by VServers is sufficient. Note that VServers and Xen exhibit similar resource and fault isolation capabilities [15].

We implement ViSE's web portal as a simple front-end to a customized Orca *slice controller*. The portal uses PHP to log user slice requests to a backend MySQL database, which the ViSE-Orca slice controller polls for new requests every tick of its internal clock. By default, Orca's internal clock ticks every 10 seconds, although, as with a conventional operating system, the tick rate is configurable. Upon detecting a new request, the slice controller reads the request from the database and issues it to Orca's instantiation of the GENI *Clearinghouse*. GENI testbeds that use Orca are able to delegate the right to allocate their resources to the Clearinghouse. As a result, GENI users—including ViSE's web portal and slice controller—request slices from the Clearinghouse, and not from the testbed itself. Note that Orca does not *require* testbeds to delegate their resources to the Clearinghouse. If necessary, testbed's may retain control of resource allocation. However, the Clearinghouse serves as a focal point for a GENI facility to implement GENI-wide resource allocation and authorization policies.

3.2 Clearinghouse Integration

If testbed's retain control of resource allocation then GENI cannot implement coordinated global policies, such as ensuring that slice requests spanning different testbeds and networks at multiple institutions are allocated atomically with the same start time. One alternative to a Clearinghouse approach that implements GENI-wide policy is to force the burden on individual end-users to request resources from each individual testbed. Using this approach, if testbed's do not coordinate with each other, end-users will have no guarantee of being

granted resources from each testbed at the same time. ORCA currently operates a GENI Clearinghouse for multiple institutions, including Ohio St. University, Duke University, University of Massachusetts at Amherst, Northwestern University, and the University of Houston, at the Renaissance Computing Institute in Chapel Hill, North Carolina. Once the Clearinghouse approves or declines a ViSE slice request, it sends the response, in the form a signed ticket, to the ViSE slice controller, which then logs a state change for the request to the web portal's database to notify the portal, and hence the user, of the request's current status.

The Clearinghouse allocation policy for ViSE always approves requests from the ViSE slice controller for the next available unallocated time slot. If the request is approved, the portal notifies the user of the start time of its slice. To activate the slice, the ViSE slice controller redeems the ticket it received from the Clearinghouse with ViSE's *aggregate manager*. The slice controller also attaches important user-specific configuration properties to this redeemed ticket; in ViSE, the most notable property is the public key the user initially registered, described above. ViSE's architecture in combination with GENI/ORCA's architecture, separates the slice controller, which accepts and issues slice requests, from the aggregate manager, which uses ViSE's control plane to allocate and configure virtual machines, to accommodate slice controllers operated locally by end-users. Thus, rather than ViSE operating a single monolithic web portal that end-users leverage to issue slice requests, end-users may operate their own slice controllers that programmatically issue slice requests for ViSE or other testbeds.

The underlying assumption of ORCA's architecture is that the only thread common to all GENI testbeds is that they share resources and do not use them forever. As a result, all delegations and requests in ORCA are represented as leases with well-defined start and end times. Since ORCA does not make any assumptions about the attributes of a resource, integrating ViSE's non-traditional sensing resources with its slice controller, Clearinghouse, and aggregate manager implementations was straightforward. ORCA includes extensible implementations of each server that exposes lease handler interfaces that execute, for each request, at the start and end of its lease at both the slice controller and the aggregate manager.

3.3 Slice Controller Integration

For ViSE's slice controller, we implement lease handlers that update the web portal's database when leases start and end to notify users of the status of their slice through the web portal. For ViSE's aggregate manager, we implement lease handlers that setup and teardown virtual machines on each ViSE node, attach sensors to them, and write the user's public key to each VM's root disk. The aggregate manager setup handler snapshots a copy-on-write template virtual machine disk image using Linux's logical volume manager to create the root disk for each virtual machine in a slice. The template disk image includes simple scripts and code segments that provide examples for how to control each sensor. Once active, the aggregate manager notifies the slice controller, which logs the notification to its database.

Initially, we are limiting the *degrees of freedom* available to users through the ViSE web portal, even if ORCA's default slice controller provides them. Examples of degrees of freedom include the ability to (i) request leases with variable lengths and variable start times, (ii) extend leases with a variable duration, (iii) lease virtual disks or sensors independently of virtual machines, (iv) cancel a lease before its end time, (v) add or remove nodes from a slice on or before a lease extension, or (vi) increase or decrease the size of a node's sliver, i.e., share of CPU, bandwidth, memory, etc., attached to each virtual machine. Instead, ViSE's portal currently forces users to issue slice requests for four hour durations starting in the next available slot. Users cannot extend leases beyond this four hour period or submit additional requests before their current lease has finished to prevent hoarding.

Some of the degrees of freedom above do not apply to ViSE, although they may apply to other testbeds. In particular, as previously mentioned, we allocate all ViSE nodes in each slice (v) rather than allocate partitions of the network. Initially, we are only allocating dedicated nodes, so (vi) does not currently apply, although we are investigating multiplexing actuators, which we discuss in Section 4.2. While (i), (ii), (iii), and (iv) may be useful, they burden the user with formulating complex requests, and the Clearinghouse with implementing sophisticated allocation policies that require mechanisms to prevent users from hoarding resources. We plan on integrating these functions for users as necessary. Our approach is to start simple as we attract users, and allow their experiences to motivate the degrees of freedom we ultimately add and support.

4 Challenges

While ORCA's minimalist design based on leases, which only assumes that users do not use resources forever, simplified our initial integration of ViSE with GENI, we have identified at least three challenges moving forward. We outline current challenges in ViSE related to control plane separation, fine-grained actuator multiplexing, and unpredictable energy supplies below. Although ViSE focuses predominantly on steerable sensors, we view these challenges as also being applicable to embedded sensors as well.

4.1 Control Plane Separation

As with other testbeds, sensor network testbeds must separate their control plane from each user data plane. ViSE accomplishes this separation on each node using virtual machines. However, since ViSE nodes connect wirelessly, it currently does not separate control plane communication. The control plane operations occur over the same wireless channel used by each testbed user. In related work, we built a small form factor sensor platform that uses two distinct nodes and radios for control plane and data plane communication [16]. The advantage of the dual node/radio approach is that the properties of the control plane's node/radio can

be well-matched to its needs. For example, in our work we matched a mote-class control plane node/radio with a more powerful embedded node, capable of running Linux, for the data plane.

The control plane node could always remain on because its energy demands were small compared with the demands of the data plane node, which required an appropriately sized solar panel. Since the control plane only executes a small number of operations, the more powerful computing platform of the embedded node was not necessary. Further, the control plane radio had low bandwidth, but long range, rather than the high bandwidth, but short range, radio of the embedded node. The low bandwidth radio is appropriate for a control plane that only sends short commands to nodes, while the long range is appropriate for providing greater connectivity in the case of node failures. Finally, separating the control plane onto a different processor decouples the control plane from the software on the main embedded node, allowing faults to be tolerated on the embedded node.

We are investigating using the same approach in ViSE. However, in the case of ViSE, we have prototyped a solution using our embedded node from the work above—the Linux Gumstix platform—as the control plane node, since for ViSE the Gumstix draws significantly less power than our x86-class nodes. For our radio, we have prototyped using a GPRS modem connected to AT&T's cellular network. Much like the mote-class control plane node in our prior work, the GPRS modem exhibits low bandwidth but a wide range. An advantage in ViSE of using a separate control plane radio is that it allows testbed applications to control the main wireless radio's operational settings, such as its operating frequency, bit rate, or MAC-layer protocol. Currently, ViSE does not allow applications control of these settings because ViSE's control plane uses them, and, if they are changed ViSE's control plane becomes disconnected. However, long-distance wireless researchers may find the control useful. ViSE also uses automated reboot switches, triggered by cell phone text messages, as a last resort to rebooting a node that cannot be contacted, since physical access to the nodes is time-consuming in good weather and nearly impossible in poor weather or during winter.

4.2 Fine-grained Actuator Multiplexing

ViSE currently allows testbed users dedicated control of each steerable sensor. However, we are integrating support for fine-grained multiplexing of sensor actuators, which we call MultiSense [17]. MultiSense enables multiple virtual machines to each control and steer their own *virtual sensor*. The virtual machine hypervisor, or privileged management domain, then multiplexes steering requests at a fine-grain to provide the illusion that each virtual machine controls a dedicated, albeit, slower sensor. MultiSense optimizes a proportional-share scheduler, which we call Actuator Fair Scheduing or AFQ, for steerable sensors by adding support for judicious batching and merging of requests, anticipatory scheduling, and state restoration. Our results from a MultiSense prototype show

that for ViSE's pan-tilt-zoom camera it enables a tracking application to photograph an object moving at nearly 3 mph every 23 ft along its trajectory at a distance of 300 ft, while supporting a security application that photographs a fixed point every 3 seconds. Of course, MultiSense diverges from strict fairness between virtual machines by reordering steering requests, in part, by batching together requests that are "near" each other. An in-depth description of MultiSense and a complete experimental evaluation is available in [17]. Once we integrate MultiSense with ViSE, we will be able to allocate sensor "slivers" in the same way that we allocate slivers of CPU, memory, or bandwidth.

4.3 Unpredictable Energy Supplies

Finally, while there is currently A/C power available for each ViSE node, we have also connected nodes to both solar panels, wind turbines, and batteries. We are studying how to operate nodes purely off harvested energy to enable us to deploy nodes where external A/C power is not available. One consequence of operating a testbed using unpredictable energy supplies is that both testbed users and the GENI Clearinghouse expect predictable behavior. For example, ViSE's aggregate manager delegates the right to allocate its resources to a GENI Clearinghouse for a fixed period of time. Recall that this delegation is important for implementing GENI-wide policies, such as atomically allocating slices that span multiple testbeds. However, determining the duration of this time period is dependent on ViSE's available energy supply, which is not entirely predictable.

One option is to choose the duration based on the current reserves in the battery to ensure that the Clearinghouse will not allocate nodes to users when there is no energy to run them. An alternative option, which we are exploring, is to use weather forecasts to increase the duration that ViSE can delegate to GENI. If weather forecasts are accurate, they should provide a means for increasing the duration of possible requests ViSE can satisfy. Additionally, the simple policies above allocate resources assuming that testbed users operate nodes at their full utilization. If nodes are not operated at their full utilization, then additional energy is available for subsequent experiments. With a GENI-wide Clearinghouse, these updates must propagate to the Clearinghouse to allow subsequent testbed users to take advantage of the additional resources. One avenue for future exploration is the performance impact of these updates on a federated Clearinghouse.

5 Conclusion

In this paper, we have described the motivation behind ViSE, a steerable sensor network testbed we are building in Western Massachusetts, described ViSE's integration with Orca, a candidate GENI control framework, and outlined three challenges ViSE presents moving forward. We are actively working on ViSE's integration with other substrates and testbeds, including other GENI testbed's, cloud computing environments, and networks, such as NLR and Internet2, to allow cross-testbed slices.

Acknowledgments. This material is supported by the NSF grant CNS-0834243.

References

1. Werner-Allen, G., Swieskowski, P., Welsh, M.: Motelab: A Wireless Sensor Network Testbed. In: Information Processing in Sensor Networks, pp. 483-488 (2005).
2. Ertin, E., Arora, A., Ramnath, R., Naik, V., Bapat, S., Kulathumani, V., Sridharan, M., Zhang, H., Cao, H., Nesterenko, M.: Kansei: A Testbed For Sensing At Scale. In: Information Processing in Sensor Networks, pp. 399-406 (2006).
3. Weren-Allen, G., Lorincz, K., Johnson, J., Lees, J., Welsh, M.: Fidelity And Yield In A Volcano Monitoring Sensor Network. In: Operating System Design and Implementation, pp. 381-396 (2006).
4. Juan, P., Oki, H., Wang, Y., Martonosi, M., Peh, L., Rubenstein, D.: Energy-efficient Computing For Wildlife Tracking: Design Tradeoffs And Early Experiences With Zebranet. In: Architectural Support for Programming Languages and Operating Systems, pp. 96-107 (2002).
5. NSF Center For Collaborative Adaptive Sensing Of The Atmosphere (CASA), http://casa.umass.edu, Accessed (2009).
6. Magnuson, S.: New Northern Border Camera System To Avoid Past Pitfalls. In: National Defense Magazine (2009).
7. Irwin, D., Chase, J., Grit, L., Yumerefendi, A., Becker, D., Yocum, K.: Sharing Networked Resources With Brokered Leases. In: USENIX, pp. 199-212 (2006).
8. Chase, J.: Orca Control Framework And Internals. In: Duke University Technical Report (2009).
9. The GENI Project Office. GENI System Overview. The GENI Project Office, Technical Report GENI-SE-SY-SO-02.0 (2008).
10. Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., Wawrzoniak, M.: Operating Systems Support For Planetary-scale Network Services. In: Network Systems Design and Implementation, pp. 253-266 (2004).
11. White, B., Lepreau, J., Stoller, L, Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An Integrated Experimental Environment For Distributed Systems And Networks. In: Operating System Design and Implementation, pp. 255-270 (2002).
12. Li, M., Yan, T., Ganesan, D., Lyons, E., Shenoy, P., Venkataramani, A., Zink, M.: Multi-user Data Sharing In Radar Sensor Networks. In: Embedded Networked Sensor Systems, pp. 247-260 (2007).
13. Kang, W., Stankovic, J., Son, S.: On Using Weather Information For Efficient Remote Data Collection In WSN. In: Workshop on Embedded Networked Sensors (2008).
14. Gush, http://gush.cs.williams.edu/, Accessed (2009).
15. Soltesz, S., Potzl, H., Fiuczynski, M., Bavier, A., Peterson, L.: Container-based Operating System Virtualization: A Scalable, High-performance Alternative To Hypervisors. In: EuroSys, pp. 275-288 (2007).
16. Sharma, N., Gummeson, J., Irwin, D., Shenoy, P.: SRCP: Simple Remote Control For Perpetual High-power Sensor Networks. In: European Conference on Wireless Sensor Networks, pp. 358-374 (2009).
17. Sharma, N., Irwin, D., Shenoy, P.: VSense: Virtualizing Stateful Sensors With Actuators. University of Massachusetts, Amherst, Technical Report UM-CS-2009-033 (2009).