ViSE: Virtualized Sensing Environment
http://vise.cs.umass.edu

University of Massachusetts, Amherst
Prashant Shenoy, Mike Zink, David Irwin, Jim Kurose, and Deepak Ganesan

1 ViSE Hardware System Overview

ViSE is a mult-sensor/multi-user sensor network testbed in Western Massachusetts. The testbed will consist of 3 nodes with physical connections over long-range 802.11b with directional antenna as well as a backplane connection over a commercial cellular network. Each ViSE node is custom-built and consists of a number of different hardware components, listed below. In addition to listing each component and a brief summary of its capabilities, we also list relevant external links that include more information. Where appropriate, we elide certain details of the node hardware including cabling, custom-built power boards (e.g., for the Gumstix backplane), external enclosure connectors, etc.

In addition to the specific hardware components, the position of each node is an important part of our testbed's profile. We currently have one node positioned at the top of a firetower on Mount Toby in Amherst, MA. This node is roughly 10 km from one node on the UMass CS Building and one node on the MA1 Tower on the UMass Campus. The Mount Toby node is able to communicate with both the CS building node and the MA1 Tower node. The CS building node and the MA1 Tower node do not have line of sight and cannot communicate directly over 802.11b. A server inside the CS building is able to connect to the CS ViSE node over 802.11b. The MA1 Tower ViSE node has a wired Ethernet connection. These two servers will be the primary entry points for GENI user network traffic to/from the ViSE nodes.

1.1 Computational Hardware

- **Motherboard**. MSI GM965 Mini-ITX form factor. The motherboard supports Core 2 Duo Mobile processors and the Celeron M using type Socket P (478 pin) with a 800/533MHz front side bus. The chipset is an Intel GME965 northbridge and ICH8M southbridge. The board includes 2 DDR 533/667 SDRAM sockets. We use the motherboards Wake-on-LAN feature to power the node up and down using an external microcontroller—the Gumstix—described below.

  More information may be found at http://www.logicsupply.com/products/ms_9803.

- **Processor**. 1.86 Ghz Intel Celeron M (Merom) CPU 540.

  More information may be found at http://www.logicsupply.com/products/ms_9803.

- **Memory**. 2GB of DDR2 667 RAM.

- **Gumstix Backplane**. Each main node has a Linux Gumstix connected via serial and Ethernet to the main node to serve as a backplane. We are using the Gumstix Verdex Pro. The backplane connects to commercial cellular. Our current plan is to initially place only one cellular-connected on Mount Toby, which is physically inaccessible during the Winter.

  Details of the Gumstix are at http://gumstix.com/store/catalog/product_info.php?products_id=209.

- **Adlink Data Acquisition Card (DAQ) PCI-9812**. We use an ultra-high speed analog input card to read the high-bandwidth data produced by the radar.

More information about the card may be found at http://www.qproducts.sk/pci-9812-9810-9812a.html.

1.2 Peripherals/Sensors

- **Wireless NIC**. The motherboard includes a Mini-PCI slot. We are using the Atheros AR5BXB63 Mini-PCI card. The card uses the Madwifi driver under Linux.

  More information may be found at http://www.logicsupply.com/products/wn6302a_f4.

- **GPRS modem**. The node on the CS building includes a GPRS modem connected to the Linux Gumstix to provide backplane connectivity independent of the unprivileged GENI traffic. We use a data plan from ATT.

- **Point-to-Point Directional Antenna**. Each wireless NIC connects to a directional antenna for long-distance 802.11b communication. We are using an HG242G-NF Hypergain 24dBi Grid Antenna 2.4Ghz.

  More information may be found at http://www.hyperlinktech.com/familylist.aspx?id=146.

- **Weather Radar**. We are using a RayMarine Model 7845647 and Part E52067). We have modified a standard Marine/boat radar for weather sensing. The radar sends data to the Adlink DAQ card and is controlled by a custom PIC board that connects to the main board over USB. Commands may be forwarded using libusb in Linux.

  More about the radar may be found at http://www.raymarine.com/

- **DavisPro Weather Station**. The DavisPro weather station has sensors for temperature, humidity, pressure, wind, solar radiation, rainfall, rain rate, and wind chill. We are using a Vantage2 06152C weather station. The DavisPro uses the wview driver under Linux and connects via USB.

  More information is available at http://www.davisnet.com/weather/products/weather_product.asp?pnum=06152C.

- **Pan-Tilt-Zoom Camera**. We are using an Axis 212 Network IP camera on each node inside of a heated camera enclosure. The IP camera connects to one of the two available Ethernet outlets on the main board.

  Details of the camera may be found at http://www.axis.com/products/cam_212/.

1.3 Enclosure/Power

- **Enclosure**. Each node is housed in a Stahlin Enclosure. Each enclosure is mounted on a Campbell Scientific CM10 Tripod.

  More information can be found at http://campbellsci.com/cm10.

- **Solar Panel**. Each node draws power from an attached solar panel. A charge controller (the MPPT250-25A12V) regulates power from the solar panel to the main board, radar, and sensors.

  Details of the charge controller are at http://store.altenergystore.com/Charge-Controllers/Solar-Charge-Controllers/

- **Battery**. Each node includes a 12V ATX power supply from http://www.logicsupply.com/products/m1_atx and a battery charger that charges an attached car battery. Currently, all nodes have external A/C power as well; the solar panel/car battery are primarily used as backup and as a catalyst for future energy-aware work (which is not apart of the SOW for the current subcontract).

More information is available at at http://store.altenergystore.com/Charge-Controllers/AC-Charge-Controllers/Samlex-12V15A-3STG-AC-BATTERY-CHARGER/p1052/.

2 Sharing, Programming, and Configuring ViSE resources

ViSE is focused on slivering actuatable sensors and including them in GENI slices. This is the primary "resource" offered by the testbed in addition to access to each ViSE compute node. Each ViSE node will run an instance of the Xen virtual machine monitor and user slivers will be bound to a Xen virtual machine created at the user's request. Xen (or Linux) includes mechanisms to partition a virtual machine's memory allotment, CPU share, egress bandwidth, and storage volume size. In addition to these "resources" being available for a sliver, each Xen virtual machine will have one or more sensors attached to it; ViSE sensors include the PTZ camera, radar, and weather station. Initially, our focus is on offering the radar to users and (secondarily, as listed in our SOW) the PTZ camera.

Importantly, each sensor will be able to be actuated by the user from within their slice. Initially, we assume that all ViSE nodes will run the same flavor of Linux. Virtual sensors will be exposed as devices through the standard Linux device driver framework as `/dev/` files. Each sensor will expose a distinct interface that depends on its actuating capabilities; these interfaces will be exposed as `ioctls` to the relevant device driver. Initially, these interfaces will be simple and are largely TBD since their development is part of our project's description. Below we provide some initial insights into these interfaces based on the current physical interfaces exposed by the radar and comment on the degree of sharing capable of the device.

The radar essentially exposes three simple functions: `power`, `standby`, and `transmit`. Power turns the radar on, standby warms the radar up, and transmit spins and radiates the radar to gather reflectivity and voltage data. Our current radars expose two other actuators: `range` and `gain`. The PTZ camera interface will follow a similar model, but with different `ioctl` calls (e.g., move, capture, etc.). The initial approach to "sharing" the radar will map work from WFQ/SFQ to actuators to regulate the time that each slice controls a physical sensor. Thus, each sliver will contain a "share" of the sensor, just as it contains a "share" of the CPU.

The primary limitation to sharing the sensor (as opposed to the CPU or other resources) will be that a steerable actuator may only be in one position at a time. If two co-located slivers request different actuations at the same time there will be a performance impact. If two slivers have similar or overlapping actuations then the physical actuator may be able to satisfy both actuations without a performance impact. With this in mind, the degree of sharing capable with the device is somewhat dependent on degree of overlap in the user's needs.

3 Physical Connections and Software Interfaces to Control Framework

3.1 Physical Connections

Node-to-Node. The internal connections are over 802.11b using directional Point-to-Point antenna. The GPRS modem that we use as a backplane leverages the commercial Internet (through ATT).

Internet-to-Node. The servers that act as gateways to the nodes are connected to the UMass Department of Computer Science network. We envision a node in the CS department serving as a gateway node to the network and hosting the Orca software for a management authority. Any connection to the GENI backbone would have to go through these servers.

Internet-to-Backplane via Cellular. Our Operations and Management plane will use the backplane provided by the Gumstix as a way to debug and power up/down the nodes if necessary. The Gumstix backplane has both Ethernet and Serial connections as well as Wake-on-Lan capability. Additionally, the Mount Toby node (the most

inaccessible) includes a cellular-enabled power strip that is able to power, reboot, or cut power to the node via a cell phone text message.

4 Software Interfaces to the Control Framework

We will use Orca to schedule, configure, and access testbed node, and will share many of the internal and external interfaces to the Orca control framework. Internally, we will modify Orca's standard set of handlers (e.g., join, modify, leave and setup, modify, teardown) for Xen virtual machines to include directives to configure virtual sensor slivers for each requested slice; otherwise, we envision sharing much code with the current Orca framework. We will also modify Orca's standard broker/clearinghouse policy to divide control of virtual sensors across time and space. Orca exposes the options to request and configure different resources as opaque property lists. Our augmented broker/clearinghouse policy and handlers will include properties specific to our virtual sensors. These capabilities will be exposed initially through Orca's standard web interface for users to request slices.

One primary difference between our testbed and Orca's software is that our nodes are disconnected (e.g., the management authority is only able to communicate with Mount Toby node through another testbed node). We will make slight modifications to Orca's control plane software to enable the "disconnected" mode of operation.

5 Measurement Capabilities in Components/Test Equipment

Each sensor measures attributes of its physical surroundings. We expect this to be the most used feature of the testbed. We currently format the raw reflectivity and voltage data gathered by the radar as NetCDF files. Details of the NetCDF standard may be found at http://www.unidata.ucar.edu/software/netcdf/ and http://en.wikipedia.org/wiki/NetCDF. A brief sample output from the radar is below. The DavisPro outputs *.wlk files, which are simple text files with the relevant weather data. Finally, the PTZ camera produces sequences of *.jpg images.

While we currently have no plans to expose the Wireless NIC data (e.g., signal strength, bit rate, ESSID) we envision this being potentially useful for researchers of long-distance wireless communication and are exploring its possibilities.

Sample NetCDF data:

```
netcdf CS_20081118200052 {
dimensions:
        Radial = UNLIMITED ; // (15 currently)
        Gate = 150 ;
variables:
        float Azimuth(Radial) ;
                Azimuth:units = "Degrees" ;
        int Time(Radial) ;
                Time:units = "Seconds" ;
        int TimeUSec(Radial) ;
                TimeUSec:units = "MicroSeconds" ;
        float RawVoltage(Radial, Gate) ;
                RawVoltage:units = "Volts" ;
        float Reflectivity(Radial, Gate) ;
                Reflectivity:units = "dBz" ;

// global attributes:
```

```
                :RadarName = "CS" ;
                :Latitude = "49.49083" ;
                :Longitude = "-72.53800" ;
                :Height = 0.f ;
                :HeightUnits = "meters" ;
                :AntennaGain = 26.f ;
                :AntennaGainUnits = "dB" ;
                :AntennaBeamwidthEl = 25.f ;
                :AntennaBeamwidthAz = 4.f ;
                :AntennaBeamwidthUnits = "deg" ;
                :TxFrequency = 9.41e+09f ;
                :TxFrequencyUnits = "hertz" ;
                :TxPower = 4000 ;
                :TxPowerUnits = "watts" ;
                :GateWidth = 100.f ;
                :GateWidthUnits = "meters" ;
                :StartRange = 1.f ;
                :StartRangeUnits = "meters" ;
data:

 Azimuth = 138.1354, 138.1354, 138.1354, 138.1354, 138.1354, 138.1354,
    138.1354, 138.1354, 138.1354, 138.1354, 138.1354, 138.1354, 138.1354,
    138.1354, 0 ;

 Time = 1227038452, 1227038452, 1227038452, 1227038452, 1227038452,
    1227038452, 1227038452, 1227038452, 1227038452, 1227038452, 1227038452,
    1227038452, 1227038452, 1227038452, 1227038452 ;

 TimeUSec = 145438, 742353, 742656, 768753, 769040, 778393, 780048, 801755,
    802003, 802200, 833348, 834358, 852119, 852302, 852485 ;
```

6 Unique Tools/Services from ViSE

We envision users initially loading code that interacts with the virtual sensor interfaces through standard mechanisms (e.g., ssh/scp), although we may investigate more sophisticated workflow tools as they become available. Our work will leverage Orca's web interface to provide users an interface for requesting resources and slices.

Our focus is primarily on the interface to the virtual sensors in each sliver. To this end, we expect to develop user-level libraries (e.g., libsensor) for the sensors to allow users to program against them at application-level, rather than using system-calls and ioctl calls directly.