VOLTTRON Thermostat

Teja Kuruganti
Oak Ridge National Laboratory

Presented to:
VOLTTRON™ 2017 - 4th Annual Technical Meeting on a Software Framework for Transactive Energy

May 16th 2017
Objectives

• Develop whole-building, retrofit-supervisory load control for improving energy efficiency and reducing peak demand by coordinating various building loads
  – Heating Ventilation and Air Conditioning (HVAC)
  – Commercial Refrigeration

• Develop grid responsive load control technology that can be deployed at large-scale to provide novel grid services (namely, ancillary services and renewable penetration)

• Deploy platform-driven technology for seamless self-aggregation of building-level loads for providing grid services

• Partnership with a building equipment manufacturer and an electric utility to demonstrate algorithms and techniques developed on an open-source control platform in real building sites.
An Multi-level Strategy

• Our goal is an integrated set of control strategies that realize the three main aims:
  – Peak demand reduction, on demand defrost
  – Energy efficiency
  – Provide services to the electrical grid

• A priority-based scheme for achieving peak demand reduction within a building

• A transactive approach to demand management
  – Priorities and nominal load are communicated to a wide-area “marketplace” where they serve as the “price” of supplying the service: price a function of priority and nominal load
  – When a demand shape is requested, the “market” clears at “price” that meets the request
  – Loads that are below the clearing price provide the service and receive the economic benefit
  – Price function constructed to favor shedding of active loads with lowest priority and highest nominal power (i.e., cheapest)
Control Formulation Priority-based Control – 2-step Implementation

- Earliest Deadline First (EDF) scheduling utilized
  - Dynamic scheduling algorithm based on priority queue
  - HVAC Constraints imposed into optimal selection
    - Both for activation and deactivation
    - Requests to deactivate may be ignored to avoid, e.g., short cycling a compressor
- Simultaneously track equipment status - past activity
- Activate HVACs according to priority and desired limit $N$ on number of simultaneously active units
- Algorithm is tested for arbitrary scaling

Temperature

HVAC #1

HVAC #2

Temperature

Dynamic Priority

Optimal Scheduling

$$p = \begin{cases} 
\text{ceil} \left( \frac{T - S}{0.1} \right) & \text{if } 0 < T - S < 1 \\
0 & T \leq S \\
10 & T - S \geq 1 
\end{cases}$$
V-Stat Development

- Peer-to-peer communication to share state of each RTU and estimation of future states
- Small form factor and plug compatible with Thermostat interface but provide “app loading” functionality
- Different agents are distinct entities and interact through messages on the bus
- Remote Protocol Calls can allow direct interaction between agents
- Agents can publish/subscribe to external VOLTTRON platforms via TCP/IP
Raspberry Pi Thermostat with VOLTTRON
Thermostat Relay Access

- Relay Board
  - Six relays OMRON G5V-1
  - Sensiron T/RH sensor

- Raspberry Pi
  - GPIO connect
  - VOLTTRON integration – Python
    - SW access to T/RH sensor
    - Relay IO files
**Temperature**

- Python temperature GPIO interface

```python
def __init__(self, device_number=0):
    # Opens the i2c device (assuming that the kernel modules have been
    # loaded)
    self.i2c = open('/dev/i2c-{}' * device_number, 'r+', 0)
    fcntl.ioctl(self.i2c, self.I2C_SLAVE, 0x40)
    self.i2c.write(chr(self._SOFTRESET))
    time.sleep(0.050)

def read_temperature(self):
    # Reads the temperature from the sensor. Not that this call blocks
    # for ~4ms to allow the sensor to return the data
    self.i2c.write(chr(self._TRIGGER_TEMPERATURE_NO_HOLD))
    time.sleep(self._TEMPERATURE_WAIT_TIME)
    data = self.i2c.read(2)
    if self._calculate_checksum(data, 2) == ord(data[2]):
        return self._get_temperature_from_buffer(data)

@staticmethod
def _get_temperature_from_buffer(data):
    # This function reads the first two bytes of data and
    # returns the temperature in C by using the following function
    # T = \frac{(S_1 - S_0) * 168.72}{ST2/2^16}
    # where ST is the value from the sensor
    #
    unadjusted = (ord(data[0]) << 8) + ord(data[1])
    unadjusted &= ~SH11_STATUS_BITS_MASK  # zero the status bits
    unadjusted *= 178.72
    unadjusted /= 1 << 16  # divide by 2^16
    unadjusted -= 46.85
    return unadjusted
```

**Relays**

- Python wrapper for WiringPi C Library

```python
@library
def relayIO = ctypes.CDLL('./relayIO.so')

@library
def relaySetup():
    relayIO.relaySetup()

@library
def relaySet(R):
    relayIO.relaySet(c=types.o_int(R))

@library
def relayClear(R):
    relayIO.relayClear(c=types.o_int(R))

@library
def relayRead(R):
    mode = relayIO.relayRead(c=types.o_int(R))
    return mode

@library
def relaySet([Int R])
    if (R==1):
        digitalWrite (23, HIGH)
    if (R==2):
        digitalWrite (27, HIGH)
    if (R==3):
        digitalWrite (24, HIGH)
    if (R==4):
        digitalWrite (28, HIGH)
    if (R==5):
        digitalWrite (29, HIGH)
    if (R==6):
        digitalWrite (31, HIGH)
    return:
```

- I2C protocol to talk to GPIO
- Request temperature from sensor and wait for response
- Wrap WiringPi lib in python using ctypes
- Resolve temp from bytes returned from sensor
- Example of setting relays using WiringPi lib
System Architecture

Agent: VOLTTRON Central

Agent: Thermostat
  - Hardware Interface

Agent: Scheduler
  - Control Algorithm

Agent: Local Control
  - Control Algorithm

External Platforms

GUI
Priority Based Control – Load Flattening

- The priority based control algorithm seeks to flatten electrical loads by quantifying the “need” to operate of particular electrical load, and then allow them to compete for permission based on distance from setpoint.
- After priority calculations are made, three reservoirs of loads are created:
  - Loads that **must** be activated (those at or in excess of maximal priority)
  - Loads that **must** be deactivated (those at zero priority)
  - Loads that may “compete” for activation permission (everything in between)
- **Ex:** HVAC system subject to priority constraints between 1 (min) to 10 (max)
  - 1 priority point per 0.1F from setpoint

### Must be idle

<table>
<thead>
<tr>
<th>Load</th>
<th>Temp</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>60.9F</td>
<td>0</td>
</tr>
<tr>
<td>#2</td>
<td>60.7F</td>
<td>0</td>
</tr>
</tbody>
</table>

### May be actuated by need

<table>
<thead>
<tr>
<th>Load</th>
<th>Temp</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>#3</td>
<td>70.8F</td>
<td>8</td>
</tr>
<tr>
<td>#4</td>
<td>70.5F</td>
<td>5</td>
</tr>
<tr>
<td>#5</td>
<td>70.1F</td>
<td>1</td>
</tr>
</tbody>
</table>

### Must be active

<table>
<thead>
<tr>
<th>Load</th>
<th>Temp</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>#6</td>
<td>71.2F</td>
<td>10</td>
</tr>
<tr>
<td>#7</td>
<td>71.5F</td>
<td>10</td>
</tr>
</tbody>
</table>
Workflow

**Agent Thermostat**
- Create thermostat objects
- Poll thermostats
- Post polling data to bus

**Agent Scheduler**
- Log polling data from bus
- Compute control decision
- Post control decisions to bus

**Agent Local Control**
- Log polling data from bus
- Compute control decision
- Post control decisions to bus

**Agent Thermostat**
- Create thermostat objects
- Check operating settings
- Actuate thermostats
Polling the thermostats

`@Core.periodic(poll_period)`

```python
def publish_poll(self):
    poll = self.instance.poll_request()
    headers = {'Zone': self.zonenum}
    for idx, platform in enumerate(self.platforms):
        if idx+1 == self.zonenum:
            try:
                self.vip.pubsub.publish('pubsub', 'poll', headers, poll)
            except Exception:
                Log.error('failed to publish to local bus')
        # connection timeouts (at least 30s before reconnect)
        elif (not platform and (time.time() - self.platform_timeouts[idx]) <= 30):
            continue
        elif (not platform and (time.time() - self.platform_timeouts[idx]) > 30):
            self.remote_setup(idx+1)
        else:
            Log.info('attempting publish to external platforms: Zone ' +
                     str(idx+1) + '/ ' + str(len(self.platforms)))
            with gevent.Timeout(3):
                try:
                    platform.vip.pubsub.publish('pubsub', 'poll', headers, poll)
                except NameError:
                    Log.exception('no data to publish')
                except gevent.Timeout:
                    Log.exception('timeout')
                    self.platform_timeouts[idx] = time.time()
                    self.platforms[idx] = 
                    platform.core.stop()
```

Call method every $t$ seconds
Poll thermostat object
Publish to all interested platforms
Publish to internal message bus
If platform is not connected, attempt to connect
Publish to external platforms
If cannot publish disconnect platform
Agent Thermostat

Checking the Leader

```python
# Subscribe to leader channel heartbeat
@PubSub.subscribe('pubsub','leader')
def leader_check(self, peer, sender, bus, topic, headers, message):
    self.leader[headers["Zone"]-1] = message
    self.timecheck[headers["Zone"]-1] = time.time()

    # To reset leader after time threshold is passed
    for idx, drop_time in enumerate(self.timecheck):
        if time.time() - drop_time > 60:
            self.leader[idx] = 999

    # Order schedulers to move missing to back of list
    self.leader_sorted = sorted(self.leader)

    # If no leader available, switch to local control
    if self.leader_sorted[0] == 999:
        self.local_status = 1
```

- Subscribe to leader channel
- Messages correspond to originating zone (Zone 1 = 1, Zone 2 = 2, etc.)
- Note message posted and time sent
- If leader hasn't posted to channel in over 60s, replace his place on the list
- Sort leader list to move missing schedulers to back of leader list
- If all leaders are missing, instruct thermostat to take control from local controller
Agent Thermostat

Subscribing to Control

```python
### Check for messages posted to lead scheduler's control channel
@PubSub.subscribe('pubsub', 'status')
def pull_control(self, peer, sender, bus, topic, headers, message):
    if topic == 'status/z' + str(self.leader_sorted[0]):
        if headers['Zone'] == self.zonenum:
            if message == 'activate' and self.user_mode == 'COOL':
                if not self.local_control:
                    mode = self.instance.activate()
            elif message == 'shutdown' or self.user_mode == 'OFF':
                if not self.local_control:
                    mode = self.instance.shutdown()

### Check for messages posted to local control channel
@PubSub.subscribe('pubsub', 'local')
def pull_local_control(self, peer, sender, bus, topic, headers, message):
    if headers['Zone'] == self.zonenum:
        if message == 'cool1' and self.user_mode == 'COOL':
            if self.local_control:
                self.instance.set_mode(-1)
        elif message == 'cool2' and self.user_mode == 'COOL':
            if self.local_control:
                self.instance.set_mode(-2)
        elif message == 'off' or self.user_mode == 'OFF':
            if self.local_control:
                self.instance.set_mode(0)
```

- **Subscribe to control channel**
- **Take instructions from lead scheduler and for correct zone**
- **Note the published message**
  - Check whether message should be acted on
  - Act on message
- **Subscribe to local control channel**
- **Note the published message**
  - Check whether message should be acted on
  - Act on message
**Communication**
- Remote Protocol Calls can allow direct interaction between agents
- Agents can publish/subscribe to external VOLTTRON platforms via TCP/IP

**Security**
- VIP holds several authentication steps to allow external platform communication
- Authentication is confirmed by public and private security keys generated by the platforms

**Robustness**
- When network connection lost, revert to local setpoint control

**Interactivity**
- User web interface through Vcentral
- CherryPy server on tstats runs backend

**Communication**
- Remote Protocol Calls can allow direct interaction between agents
- Agents can publish/subscribe to external VOLTTRON platforms via TCP/IP
Priority Based Control – Load Shaping

Real-Time Hourly Prices for April 10th, 2017

The kWh prices above do not include your personal capacity charge.

Day-Ahead Hourly Price | Real-Time Hourly Price

12AM – 5AM

Units that must not run

5AM – 10AM

Units not allowed to run

10AM – 2PM

Units allowed to run

2PM – 7PM

Units that must run
Deployment Plan

- Demonstration of retrofit supervisory controls for buildings/stores
- Deployment in Dollar General Stores
- Open-source solution expandable to other small foot-print supermarkets