OpenADR 3.0 Demand Flexibility for Heat Pump Water Heaters

IEA EBC Annex 96 — Activity A3

Anand Prakash
Carnegie Mellon University
anandkrp@andrew.cmu.edu

Outline

1. **Introduction** — Motivation and goals
2. **Background** — OpenADR 3.0, demand flexibility, HPWHs
3. **Use Case** — Price-responsive HPWH control through OpenADR
4. **Software Architecture** — Components and data flow
5. **Implementation** — LP/heuristic schedulers and CTA-2045 integration
6. **How to Use** — Setup, run, and extend

1. Introduction

Why Demand Flexibility?

- Grid decarbonization requires flexible loads that can **shift consumption**
- Water heating accounts for **~18% of residential energy use** in the US
- Heat pump water heaters (HPWHs) with storage tanks are ideal candidates:
  - Thermal storage enables **load shifting** without impacting comfort
  - Can pre-heat during cheap/clean hours, coast during expensive/dirty hours
- **Challenge:** How do we communicate price signals to devices at scale?

Project Goals

Develop an **open-source software toolkit** so that researchers and practitioners can:

1. Set up an OpenADR 3.0 communication infrastructure (VTN + VEN)
2. Fetch real electricity pricing data and publish it as OpenADR events
3. Run a control algorithm that converts price signals into HPWH schedules
4. Generate CTA-2045 demand response commands for water heaters
5. Test the full pipeline end-to-end on their own machines

> All code, documentation, and quickstart notebooks are publicly available.

2. Background

OpenADR 3.0

Open Automated Demand Response — an open standard for communicating DR signals.

Concept	Description
VTN (Virtual Top Node)	Server — publishes programs, events, price signals
VEN (Virtual End Node)	Client — receives signals, controls devices
Program	Defines a demand response program (e.g., dynamic pricing)
Event	Time-based signal with payload (e.g., hourly prices)
Report	Telemetry data sent from VEN back to VTN

REST API (JSON over HTTP) with OAuth 2.0 authentication

Heat Pump Water Heaters as Flexible Loads

How HPWHs provide flexibility:

- Tank stores thermal energy (50–80 gal)
- Heat pump COP of 3–4x vs resistance
- Can **pre-heat** during low-price hours, **coast** during high-price hours
- No comfort impact if managed well

Parameter	Typical Value
Tank capacity	50–80 gallons
HP output	4–5 kW thermal
COP	2.5–4.5
Thermal storage	8–15 kWh

CTA-2045 and Device Communication

CTA-2045 is a modular communications interface for energy devices, providing standardized demand response commands for water heaters:

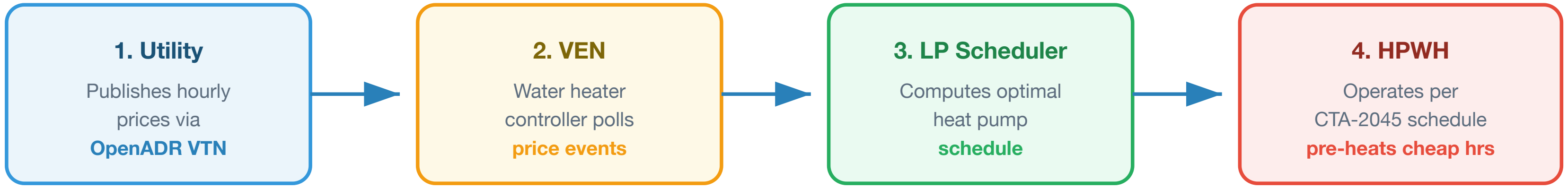
Signal	Code	Water Heater Action
Shed	-1	Lower setpoint, disable HP — coast on stored energy
Normal	0	Default operation
Load Up	1	Raise setpoint, pre-heat the tank
Adv. Load Up	2	Max setpoint, tight deadband

This project covers the full pipeline: **OpenADR** → **Control Algorithm** → **CTA-2045**

3. Use Case

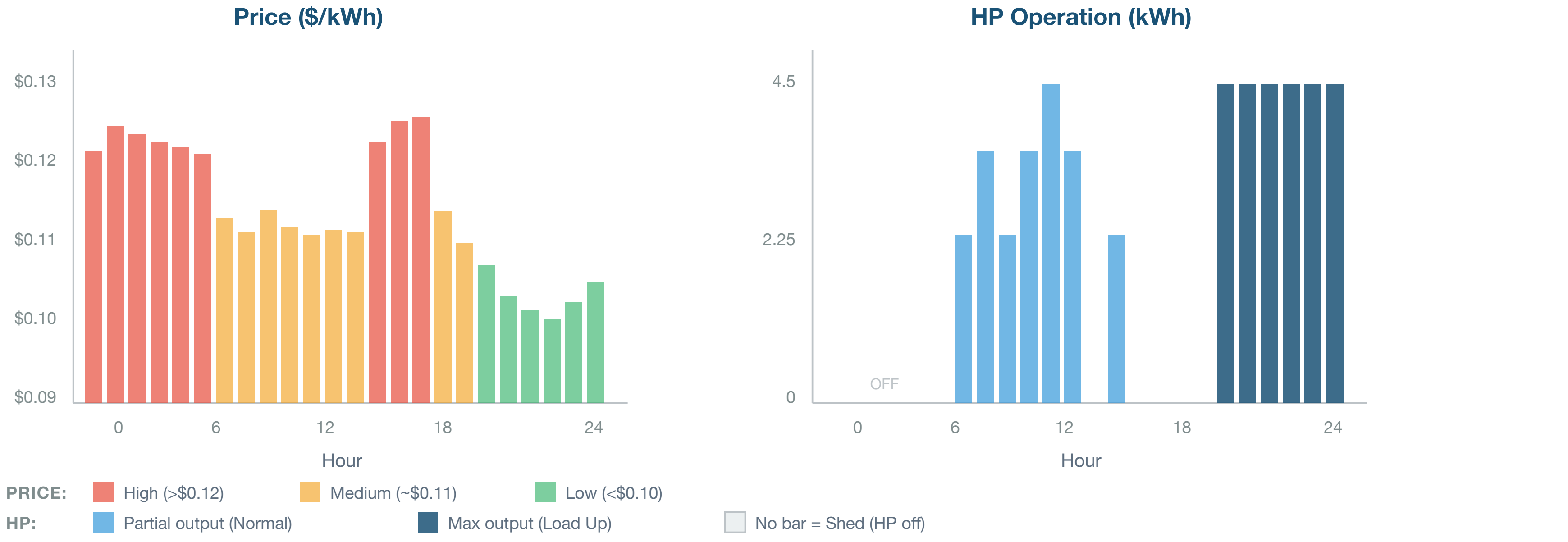
Price-Responsive HPWH Control Through OpenADR

Scenario: A utility publishes dynamic electricity prices. A water heater controller receives these prices and optimizes its operation schedule.



OpenADR 3.0 Price Signal → LP Scheduler → CTA-2045 Commands → Device Action

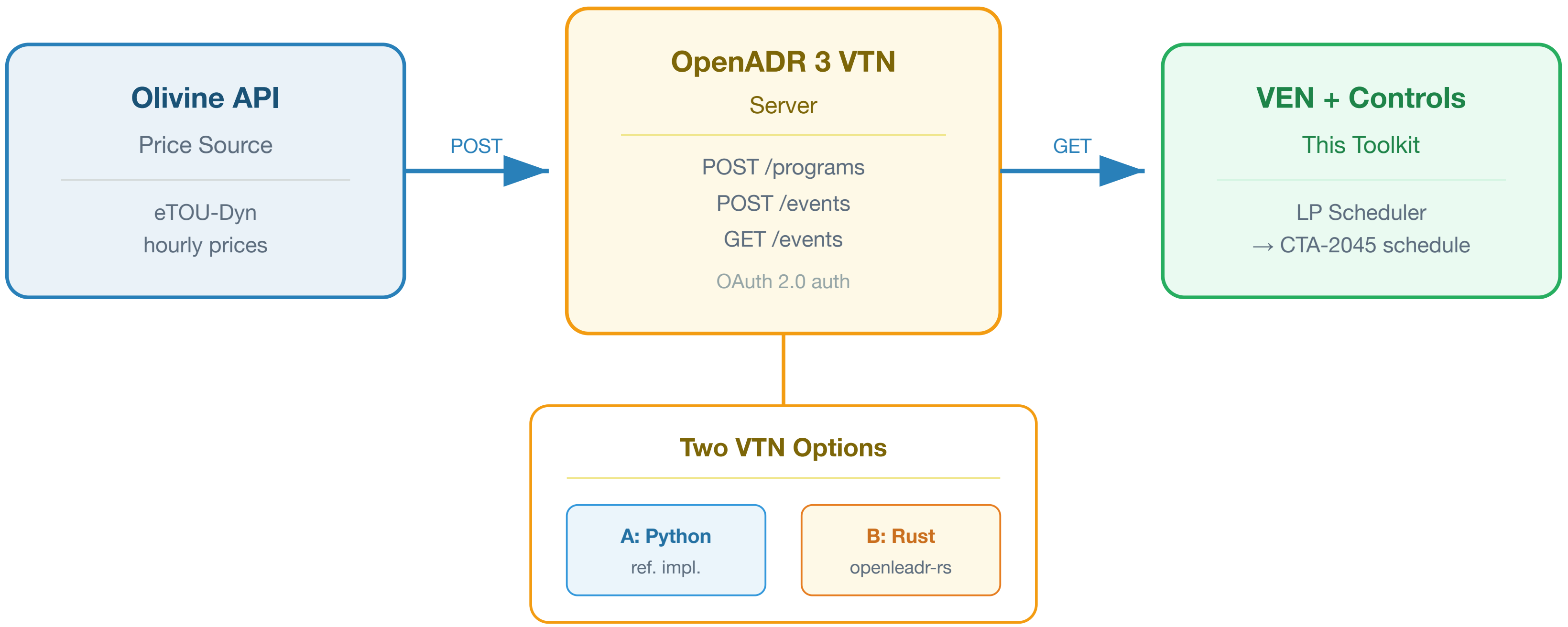
Example: 24-Hour Price Signal and Response



Result: Same hot water delivered at lower cost by shifting to cheap hours.

4. Software Architecture

System Architecture



Repository Structure

```
annex96-a3-hotwater/
├── README.md                          # Project overview
├── requirements.txt                   # Python dependencies
├── instructions.ipynb                 # Setup: Python VTN
├── instructions-openleadr.ipynb       # Setup: openleadr-rs VTN
├── quickstart.ipynb                   # Demo: Python VTN
├── quickstart-openleadr.ipynb         # Demo: openleadr-rs VTN
├── controls/                          # Control algorithms
│   ├── hpwh_load_shift_lp.py          # LP scheduler (globally optimal)
│   ├── hpwh_load_shift_heuristic.py   # Heuristic scheduler (greedy)
│   └── cta2045.py                     # CTA-2045 schedule generation
├── sample_data/                       # Example JSON payloads
└── presentation/                      # This presentation
```

Two VTN Options

	Python VTN Ref. Impl.	openleadr-rs (Rust)
Language	Python (Flask)	Rust (Axum)
Database	In-memory	PostgreSQL (Docker)
Auth	`bl_client/1001`	`any-business`
Base URL	`localhost:8080/openadr3/3.0.1`	`localhost:3000`
Access	Contact for access	Open source
Best for	Quick local testing	Production-like setup

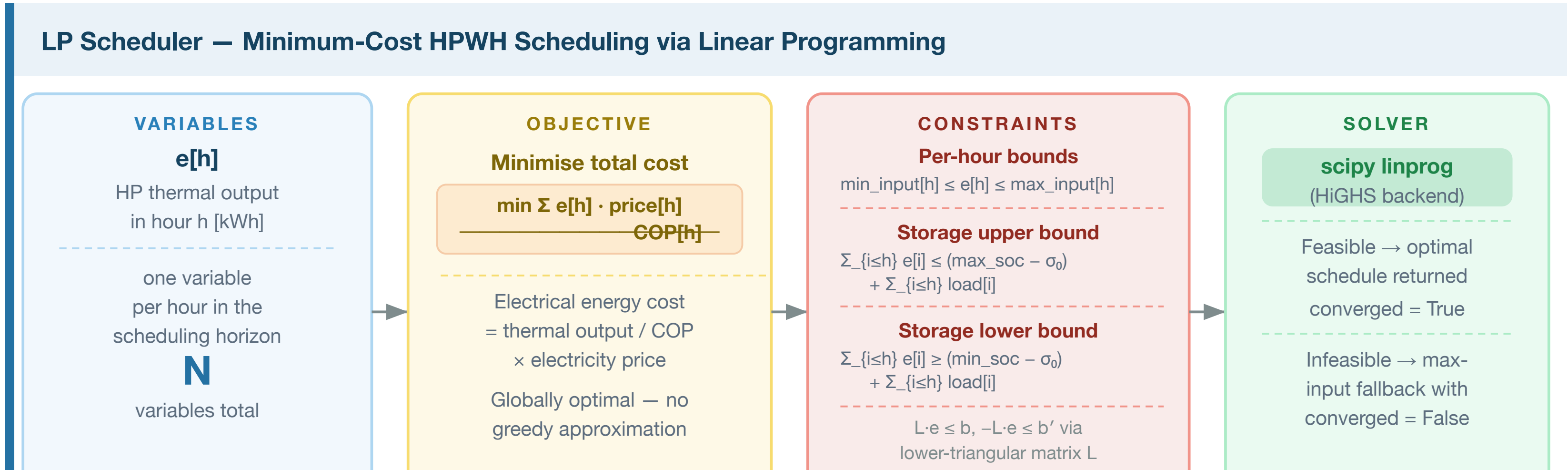
5. Implementation

HPWH Load Shift Scheduler

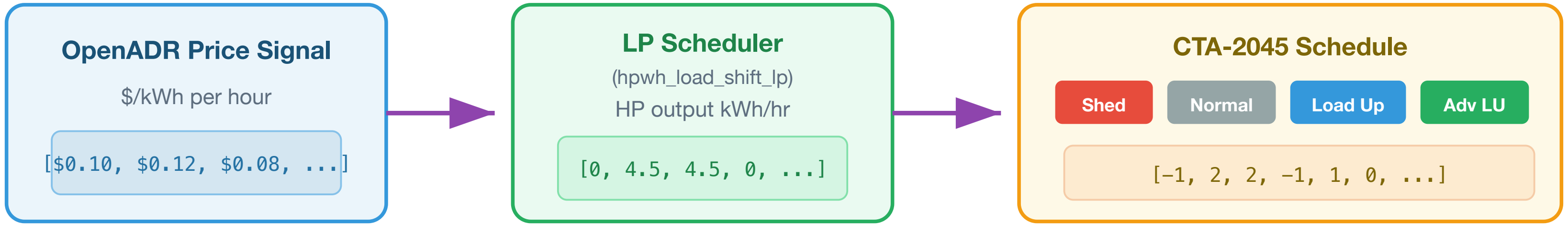
Two interchangeable implementations in `controls/`:

	LP Scheduler (`hpwh_load_shift_lp`)	Heuristic (`hpwh_load_shift_heuristic`)
Method	Linear program (HiGHS via scipy)	Bottom-up greedy
Solution	Globally optimal	Near-optimal
Speed	Fast (milliseconds)	$O(N^2)$ worst case
Dep.	scipy	numpy only

LP formulation: min Σ e[h]·price[h]/COP[h] subject to storage bounds and HP capacity bounds.



CTA-2045 Schedule Generation



Full pipeline: OpenADR price signal → LP-optimal schedule → CTA-2045 commands

> Two approaches: from LP/heuristic scheduler output (uses HP output levels), or directly from prices (uses percentile thresholds).

Quickstart Demo Pipeline

```
# Step 1: Authenticate with VTN
token = get_token("bl_client", "1001")

# Step 2: Fetch prices from Olivine API
olivine_data = requests.get(OLIVINE_PRICING_URL).json()

# Step 3: Create pricing program on VTN
requests.post(f"{VTN}/programs", json=program_data, headers=auth)

# Step 4: Publish price event
requests.post(f"{VTN}/events", json=event_data, headers=auth)

# Step 5: Read events as VEN
events = requests.get(f"{VTN}/events", headers=ven_auth).json()

# Step 6: Run LP Scheduler → Step 7: Generate CTA-2045 schedule
schedule, converged = hpwh_load_shift(params)
cta_schedule = hpwh_load_shift_to_cta2045(schedule, params)
```

Example Output

```
LP status: Optimization terminated successfully (HiGHS Status 7: Optimal)

Hourly schedule (kWh):
  Hour  0: OFF   0.00 kWh  @ $0.12052/kWh   →  Shed
  Hour  1: OFF   0.00 kWh  @ $0.12227/kWh   →  Shed
  ...
  Hour 18:  ON   1.50 kWh  @ $0.11120/kWh   →  Normal
  Hour 19:  ON   1.50 kWh  @ $0.10689/kWh   →  Normal
  Hour 20:  ON   1.50 kWh  @ $0.10519/kWh   →  Normal
  Hour 21:  ON   1.50 kWh  @ $0.10300/kWh   →  Normal
  Hour 22:  ON   1.50 kWh  @ $0.10620/kWh   →  Normal
  Hour 23:  ON   1.50 kWh  @ $0.10930/kWh   →  Normal

Total electricity cost: $0.28741
```

LP finds the globally optimal allocation — charges exactly what is needed at the cheapest hours.

Quickstart: Step by Step

Each step — inputs, what it does, outputs, and what to change for your implementation

Step 1: Setup & Verify VTN Connection

1 Setup & Verify VTN Connection

Exchange credentials for an OAuth token; confirm the VTN is reachable

INPUTS

- **VTN_BASE_URL**
  localhost:8080/openadr3/3.0.1
- **BL_CLIENT_ID / SECRET**
  Business Logic credentials
- **VEN_CLIENT_ID / SECRET**
  VEN read-only credentials

WHAT IT DOES

POST /auth/token

grant_type: client_credentials
→ returns JWT Bearer token

GET /programs

Confirm VTN returns HTTP 200
(raises exception if offline)

OUTPUTS

- **access_token**
  Bearer JWT for API headers
- **bl_headers() helper**
  Authorization header for BL calls
- **ven_headers() helper**
  Authorization header for VEN calls
- **Confirmed VTN is live**
  exception raised if not running

CUSTOMIZE

→ VTN implementation: change VTN_BASE_URL to localhost:3000 (no path prefix) for openleadr-rs
→ Credentials: update CLIENT_ID / SECRET to match your VTN's user fixture (see instructions.ipynb)
→ Token caching: get_token() is called per request; cache the token for higher-throughput use
→ TLS: replace http:// with https:// and pass verify= cert path for non-local deployments

Step 2: Fetch Dynamic Prices from Olivine

2 Fetch Dynamic Prices from Olivine

Fetch eTOU-Dyn dynamic electricity prices from the Olivine API

INPUTS

- **OLIVINE_PRICING_URL**
  api.olivineinc.com/…/etou-dyn
- **Accept: application/json**
  HTTP request header

WHAT IT DOES

GET OLIVINE_PRICING_URL

Parse vtnComment → metadata dict

Parse dtstart → start_time

Extract eiEventSignals intervals

```
price = interval["streamPayloadBase"]
[0]["item"]["value"]
```

Repeats for each interval (N hours)

OUTPUTS

- **prices [$/kWh] × N hrs**
  one price per scheduling hour
- **start_time (datetime)**
  pricing window start (ISO 8601)
- **intervals (raw list)**
  full API response for reference
- **metadata (dict)**
  retailer, rate name, date range

CUSTOMIZE

→ Price source: replace OLIVINE_PRICING_URL with any hourly price feed, RTO/ISO API, or local file
→ Load from file: read a CSV/JSON with [hour, price_$/kWh] columns instead of a live API call
→ Horizon: N = len(intervals); slice to a shorter window for faster re-runs or sub-day scheduling
→ Units: algorithm assumes $/kWh — convert if your data source uses ¢/kWh, mills, or other units

Step 3: Create Pricing Program on VTN

3

Create Pricing Program on VTN

Register a program definition that groups related price events on the VTN

INPUTS

- **OAuth token (Step 1)**
  via bl_headers()
- **programName**
  "etou-dynamic-pricing"
- **payloadDescriptors**
  [{payloadType: PRICE, units: KWH}]

WHAT IT DOES

POST /programs

Register program definition on VTN
VTN assigns a unique program_id

program_id links all future events
to this program (Step 4)
VENs can filter events by program_id
to subscribe to specific tariffs

OUTPUTS

- **program_id**
  "0" (Python VTN) or
  UUID (openleadr-rs)
- **Stored program record**
  persists on VTN until deleted
- **Confirmed HTTP 201**
  creation verified via raise_for_status()

CUSTOMIZE

→ programName: any descriptive string matching your tariff (e.g. "eTOU-Dynamic", "real-time-pricing")
→ Payload type: PRICE (absolute $/kWh) or PRICE_RELATIVE (delta from a baseline price)
→ openleadr-rs extras: add programType, retailerName, country, principalSubdivision fields
→ Reuse: query GET /programs first and skip creation if a matching program already exists

Step 4: Publish Price Signal as an Event

4

Publish Price Signal as an Event

Package the fetched prices as an OpenADR 3 event and post it to the VTN

INPUTS

- **prices + start_time (Step 2)**
  hourly prices and window start
- **program_id (Step 3)**
  links event to program
- **OAuth token (Step 1)**
  via bl_headers()

WHAT IT DOES

Format each hour as an interval:

```
{id: hour, payloads: [{type:
"PRICE", values: [price]}]}
```

POST /events

Event stored on VTN server

Any auth'd VEN can now read it

via GET /events (Step 5)

OUTPUTS

- **event_id**
  stored event identifier on VTN
- **N intervals on VTN**
  queryable by all VENs
- **Linked to program_id**
  VENs can filter by program
- **HTTP 201 confirmed**
  raised if creation fails

CUSTOMIZE

→ Interval duration: "PT1H" (1 hr default) — OpenADR 3 also supports "PT15M" for 15-minute intervals
→ Price rounding: round(price, 5) — adjust decimal places to match utility data precision requirements
→ Multiple events: publish separate events per tariff zone, VEN group, or building type
→ Event expiry: set intervalPeriod.duration to automatically expire stale events on the VTN

Step 5: Read Events as a VEN

5

Read Events as a VEN

Simulate the water heater controller reading price signals from the VTN

INPUTS

- **VEN_CLIENT_ID / SECRET**
  VEN credentials (Step 1)
- **VTN_BASE_URL**
  same server as Steps 3–4
- **(Optional) programID**
  filter events by program

WHAT IT DOES

GET /events (as VEN)

Sort intervals by interval id
for each payload where
payload["type"] == "PRICE":
prices.append(values[0])

Prices now ready for the
LP Scheduler (Step 6)

OUTPUTS

- **prices [$/kWh] × N hrs**
  same prices, from VEN perspective
- **events[] (raw list)**
  full event objects from VTN
- **event metadata**
  eventName, intervalPeriod.start

CUSTOMIZE

→ Filter by program: append ?programID=X to GET /events to subscribe to a specific tariff
→ Multiple programs: loop events[] and select by evt["programID"] for multi-tariff deployments
→ Real deployment: replace this notebook GET with a persistent VEN client process (e.g. openleadr-python)
→ Report-back: POST /reports to send HPWH telemetry (SOC, temperatures, energy use) to the VTN

Step 6: Run LP Scheduler

6

Run LP Scheduler

Solve for the globally optimal HP schedule that minimises cost while meeting load

INPUTS

- **price [$/kWh] × N hrs**
  from Step 5 — prices list
- **max_input / min_input [kWh]**
  HP thermal capacity bounds
- **max/min_storage_capacity**
  tank bounds + initial_soc [kWh]
- **load [kWh/hr] × N hrs**
  hot-water draw profile
- **cop [list, per hour]**
  heat pump efficiency profile

WHAT IT DOES

hpwh_load_shift(params)

① Build LP objective: min Σ e[h]·price[h]/COP[h]
② Per-hour bounds: min_input ≤ e[h] ≤ max_input
③ SOC upper bound via L·e ≤ b_max
④ SOC lower bound via –L·e ≤ b_min
⑤ Solve with scipy linprog (HiGHS)

Infeasible → max-input fallback
converged = False

OUTPUTS

- **control[h] [kWh/hr]**
  optimal HP thermal output per hour
- **cost[h] [$/hr]**
  electricity cost per hour
- **converged (bool)**
  True = optimal found
  False = fallback to max-input

CUSTOMIZE

→ HP capacity: set max_input to your HPWH spec in kW (= kWh at hourly time steps)
→ Tank size: max_storage_capacity ≈ 12 kWh per 80-gal tank; min_storage_capacity = reserve
→ Draw profile: replace [avg_draw]*N with a measured or simulated hourly hot-water draw schedule
→ Swap solver: replace with hpwh_load_shift_heuristic for a greedy alternative (no scipy needed)

Step 7: Generate CTA-2045 Schedule

7 Generate CTA-2045 Schedule

Map the HP schedule to discrete demand-response commands for the water heater

INPUTS

- **schedule dict (Step 6)**
  schedule["control"] per hour
- **params (Step 6)**
  params["max_input"] per hour

WHAT IT DOES

hpwh_load_shift_to_cta2045(sched, params)

fraction = control[h] / max_input[h]

fraction = 0%	Shed (–1)
0% – 30%	Normal (0)
30% – 80%	Load Up (1)
≥ 80%	Adv LU (2)

Repeat for every hour in horizon

OUTPUTS

- **signals per hour**
  list of [-1, 0, 1, 2] values
- **signal_names per hour**
  ["Shed", "Normal", ...]
- **Formatted schedule string**
  via format_schedule() for logging
- **plot_schedule() figure**
  bar chart of signals over time

CUSTOMIZE

→ Thresholds: change 30% and 80% in hpwh_load_shift_to_cta2045() to match your device's response curve
→ Price-based mapping: use prices_to_cta2045(prices) instead — assigns signals by price percentile
→ Physical control: replace format_schedule() print with an API call to your HPWH hardware
→ Fleet deployment: loop over units and call hpwh_load_shift() + hpwh_load_shift_to_cta2045() per device

6. How to Use This Software

Quick Setup

1. Clone and install

```
git clone <repository-url>
cd annex96-a3-hotwater
pip install -r requirements.txt
```

2. Start a VTN (choose one)

Option A: Python VTN (contact anandkrp@andrew.cmu.edu for access)

```
cd openadr3-vtn-reference-implementation
virtualenv venv && source venv/bin/activate
pip install -r requirements.txt && python -m swagger_server
```

Option B: openleadr-rs (open source — see `instructions-openleadr.ipynb`)

Run the Quickstart

3. Launch the notebook

```
jupyter notebook quickstart.ipynb           # for Python VTN
jupyter notebook quickstart-openleadr.ipynb  # for openleadr-rs
```

4. What the notebook does

Step	Action
1–2	Connect to VTN, fetch dynamic prices from Olivine API
3–4	Create pricing program, publish hourly price event
5	Read events as a VEN
6	Run LP Scheduler and plot optimal schedule
7	Generate CTA-2045 command schedule

Extending the Software

- **Customize HPWH parameters** — Edit Step 6: tank size, HP capacity, COP, draw profile
- **Use your own price data** — Replace Olivine API with your own source
- **Swap the scheduler** — `hpwh_load_shift_lp` (optimal) ↔️ `hpwh_load_shift_heuristic` (no scipy needed)
- **Integrate with CTA-2045 hardware** — Connect generated schedules to physical devices
- **Connect to a real VEN** — Replace notebook HTTP calls with a persistent VEN client

Resources and References

- **Repository:** `annex96-a3-hotwater/`
- **OpenADR 3.0.1 Spec:** included in repo, or openadr.org
- **openleadr-rs:** github.com/OpenLEADR/openleadr-rs
- **Olivine API:** `api.olivineinc.com/i/oe/pricing/signal/paced/etou-dyn`
- **scipy linprog / HiGHS:** scipy.org — LP solver used by the scheduler

Thank You

Anand Prakash

anandkrp@andrew.cmu.edu

IEA EBC Annex 96 — Activity A3

Carnegie Mellon University

All source code and documentation available in the `annex96-a3-hotwater` *repository*