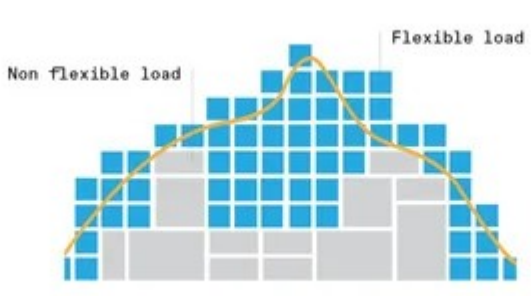
PEMT-CoSim

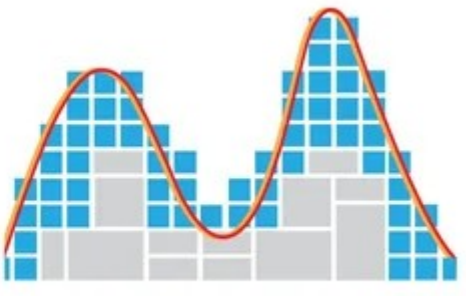
Packetized Energy Management (PEM)

- Packetization concept from communications, applied to energy supply
- Split energy requirements into discrete packets, e.g. 1500W for 5 minutes
- Requests from consumers for packets can be accepted or rejected by a coordinator
- → Demand is shifted to times of higher availability

BEFORE PACKETIZATION



AFTER PACKETIZATION


AFTER RANDOMIZATION

Packetized Energy Trading (PET)

- Packetization of the energy *market* – a *transactive energy* approach
- Energy packets are bought and sold, both from the utility grid and from nearby DERs
- Trading can involve on fixed/regulated pricing schemes, or various types of auction
- Integration of PEM with financial dynamics could benefit both producers and consumers

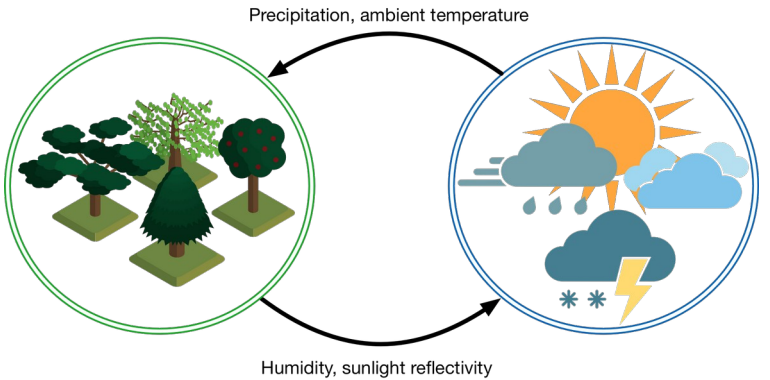
Background: Transactive Energy Simulation Platform (TESP)

- Simulation of bulk-scale and distribution-scale power systems have traditionally been separate
- Transactive energy requires coordination between the two
- TESP uses *cosimulation* to tie these two domains together
- Signals from distribution system simulation available to bulk simulation, and vice versa
- Built on HELICS, integrating *federates* built around existing simulation systems

MATPOWER

Background: HELICS

- Cosimulation framework, funded by US DoE
- “A multi-language, cross-platform library that enables different simulators to easily exchange data and stay synchronized in time”
- Each simulator is a *federate*, federates can *subscribe* to value *publications* from other federates, cosimulation is coordinated by *broker*
- Federate loop:
  - Request a time *t* to simulate up to (e.g. NOW+10s) from broker
  - When ready, broker grants a time $t_g < t$
  - Check subscriptions + update internal values
  - Run simulation up to t_g
  - Submit publications

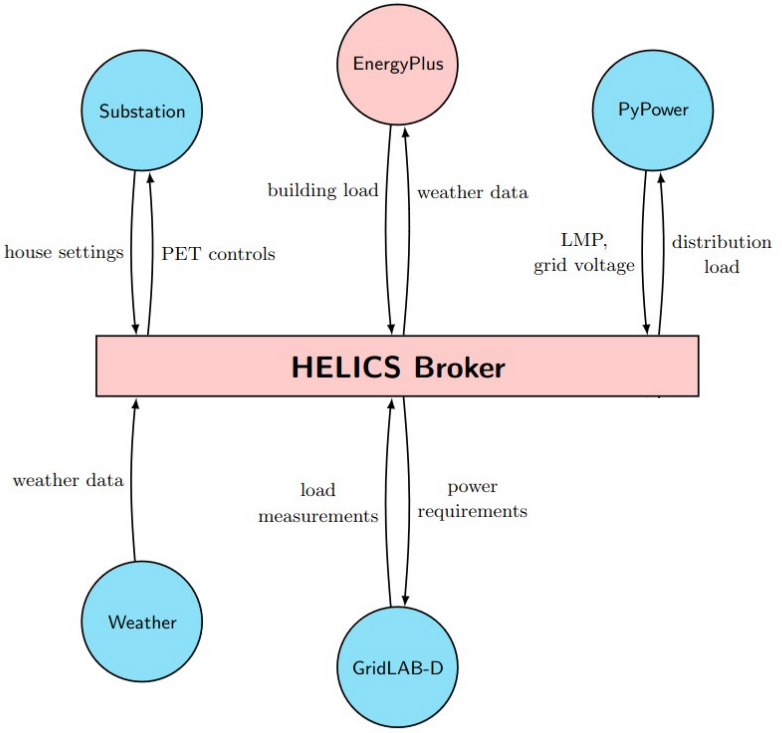


PEMT-CoSim: Principle

- PEMT-CoSim extends TESP by implementing PET in the distribution system
- Simulated microgrid of houses, each with appliances, HVAC, maybe solar PV
- Prosumers bid to buy/sell power packets at a microgrid market
  - e.g. surplus power packets from a solar PV installation might be sold to meet high HVAC load elsewhere
  - Double auction establishes a single clearing price and clears matching orders
- HVAC load is *responsive* – adjusted by each house according to market outcome

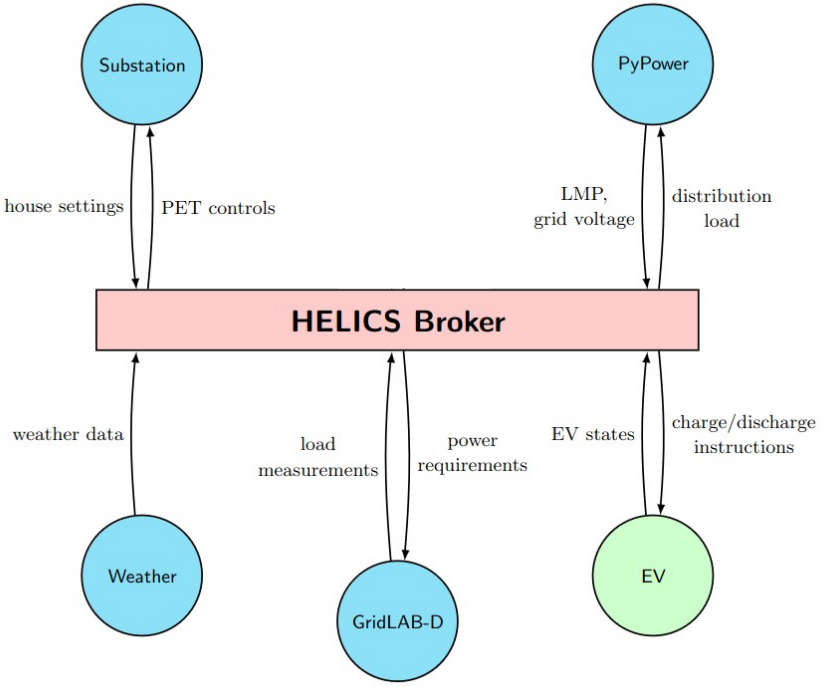
PEMT-CoSim: Architecture

- PEM & PET business logic is all in the **substation federate**
- GridLAB-D is the most important simulator, models house loads (appliances, HVAC), DERs (solar), and house temperature
- PyPower provides utility local marginal price according to grid load
- Weather just publishes from a big CSV
- EnergyPlus seems unused at the moment
- A bit messy and tricky to get running



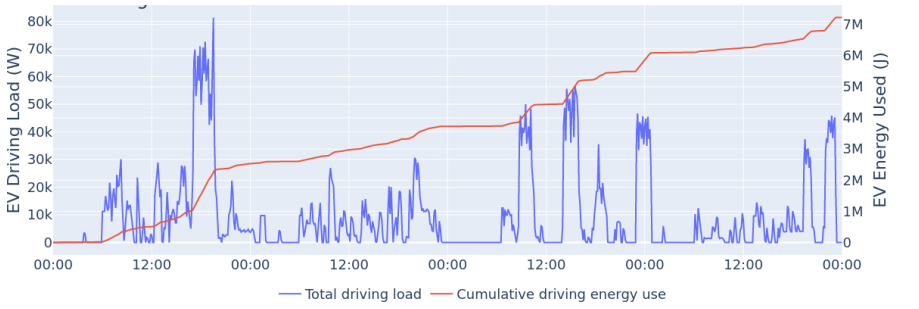
PEMT-CoSim with EVs

- Work done when I was on placement here, exploring V2G EVs as intermittently-available energy storage
- Extended PEMT-CoSim to include an EV federate
- Also changed the auction mechanics, did a bunch of refactoring and modernisation, trimmed some unused bits, and hopefully made it easier to run



EV Federate

- Models a collection of EVs, each belonging to a particular house
- Subscribes to charge/discharge rate instructions from substation federate
- Models battery state of charge, incorporating EV mobility and power patterns from `emobpy` library
- Publishes available range of power flow into/out of EV battery to substation federate
  - e.g. 0 if the EV isn't at a charging point, otherwise dependent on battery state of charge



Running It

- Project now has a functional Dockerfile and dev containers config, containerizing the dependencies
- Can be run locally on a fast machine, or over SSH to a faster server. It's pretty resource intensive.
- If you just want to run the code as-is, you can just download the code, build the image, and run it with a mount of your copy of the code
- This process is explained in the README

```
$
$ docker build -t pemtcosim -f docker/Dockerfile .
Sending build context to Docker daemon  860.9MB
Step 1/29 : FROM ubuntu:jammy
 ---> 7af9ba4f0a47
```

…

```
$
$ docker run -v /home/fred/PEMT-CoSim:/PEMT-CoSim pemtcosim --help
usage: python3 generate_case.py [-h] [-a NAME] [-n NUM_HOUSES] [-e NUM_EV]
                                [-p NUM_PV] [-g GRID_CAP] [-l LENGTH]
                                [-f FIGURE_PERIOD] [-b EV_BUY_IQR_RATIO]
                                [-i INPUT_FILE] [-r]

Generate a PET scenario for simulation

options:
  -h, --help            show this help message and exit
  -a NAME, --name NAME  scenario name
  -n NUM_HOUSES, --num_houses NUM_HOUSES
                        number of houses
  -e NUM_EV, --num_ev NUM_EV
                        number of EVs
  -p NUM_PV, --num_pv NUM_PV
                        number of PVs
  -g GRID_CAP, --grid_cap GRID_CAP
                        grid power capacity (W)
  -l LENGTH, --length LENGTH
                        simulation length in hours
  -f FIGURE_PERIOD, --figure_period FIGURE_PERIOD
                        figure drawing period (seconds)
  -b EV_BUY_IQR_RATIO, --ev_buy_iqr_ratio EV_BUY_IQR_RATIO
                        EV buy IQR ratio
  -i INPUT_FILE, --input_file INPUT_FILE
                        scenario file to use
```

Running It

- An one-time initial step of generating EV mobility profiles is necessary if you want EVs in your simulation
- After the cosimulation has finished, another script `make_figures.py` can read back the saved metrics to produce some graphs
- Again, these steps are explained in detail in the README

Editing It

- As-is, the parameters of the cosimulation allow configuration of various options (number of houses, number of EVs, PVs, grid cap etc)
- If you want to change the business logic or other aspects of the cosimulation, you can work on the code inside the container
- Easiest way to do this is with VSCode Dev Containers and Remote Development extension
- Open your (optionally remote) copy of the code in VSCode, then run “Dev Containers: Reopen in Container”
- You should then have a working python environment with all dependencies needed