ACES: A Cooperative Energy System

6.033 Preliminary Report

[redacted]

March 18th, 2022

1 Introduction

New technology and environmental awareness have changed the landscape of energy management. Energy production is becoming increasingly decentralized as homeowners and public buildings install personal solar panels. To take advantage of this paradigm shift, we propose A Cooperative Energy System (ACES), which will (1) provide low-cost and reliable energy to customers, (2) enable power sharing within microgrids, (3) calculate customer bills, and (4) collect comprehensive usage data for research.

The design of ACES prioritizes *reliability* and *adaptability*. Our primary goal is *reliability*, which means accurate and timely delivery of power and data, even when outages occur. We prioritize reliability because it directly benefits residents and other consumers of electricity, who are most impacted by our system. Specifically, we focus on reliable electricity, since power failures disrupt residents' lives by impeding essential services like emergency care, heating in homes, and Internet communication. Moreover, reliable data collection ensures fair billing and system analytics. Additionally, we prioritize *adaptability*, or the system's ability to adjust to external changes and be configured by future engineers. The residents would likely bear the cost of reworking the system, so we want to minimize the necessity and difficulty of maintenance. Furthermore, adaptability increases the longevity and efficiency of our system in the face of unexpected changes.

In Section 2, we outline the system design and implementation. Afterwards, we evaluate the impact and feasibility of our system in Section 3.

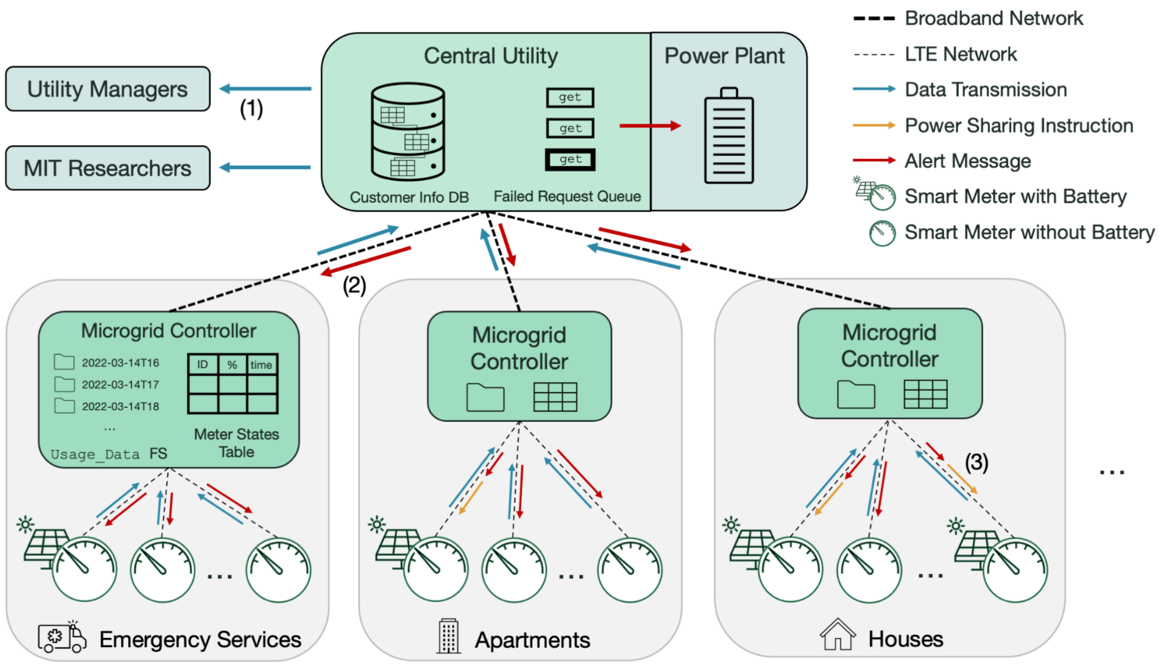
2 System Description

ACES relies on three layers of components. At the bottom layer, smart meters initiate and respond to ANSI commands. Some meters, "meters with batteries," are attached to solar panels. In the middle, the microgrid controllers (MGCs) collect meter data and handle energy sharing. Managing energy sharing in a distributed manner allows sharing to continue during power or network outages elsewhere, increasing ACES's *reliability*. Finally, at the top, the central utility collects data from the MGCs, provides data extraction APIs, distributes electricity, and handles billing. Figure 1 summarizes these components and their interactions.

In terms of storage mechanisms, the MGCs relies on a flat, lightweight `Usage_Data` file system to store meter readings, as they perform little data processing before forwarding information to the central utility. In contrast, the central utility features a relational database, the Customer Information Database (CIDB), which supports SQL querying. The CIDB offers *reliable* concurrency and easily *adaptable* tables, as well as efficient querying and simplicity of export.

For all network communications, ACES employs TCP. We deemed TCP's associated overhead tolerable to achieve *reliable*, in-order transport. Each data collecting component additionally tracks failed requests for boosted fault tolerance. Since fine-grained usage information enables researchers to improve ACES, the system uses lossy aggregation sparingly for increased *adaptability*.

Figure 1: ACES Overview



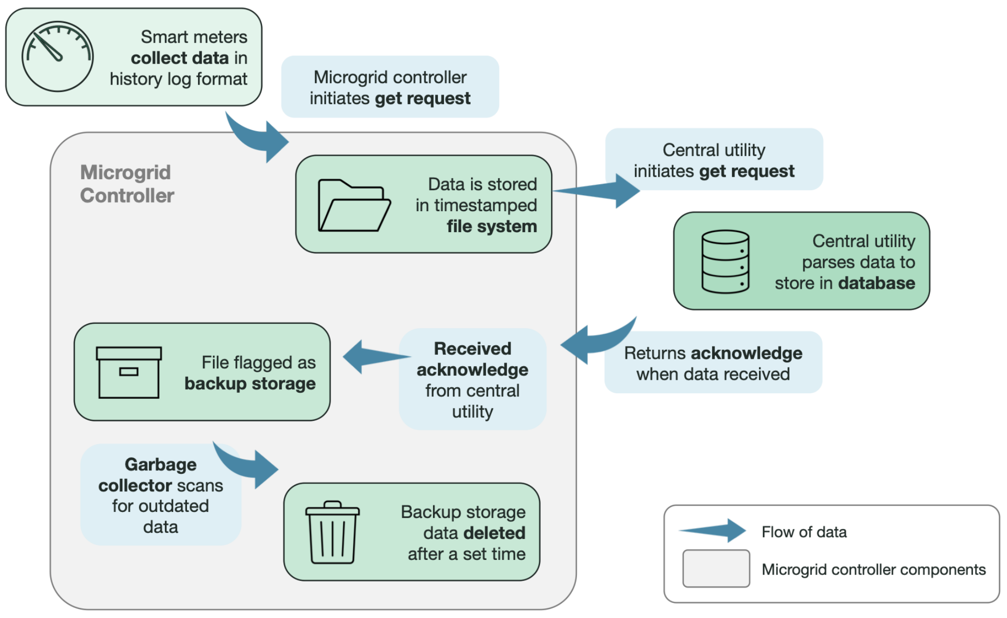
(1) Usage data collected by central utility exported to utility managers and researchers
(2) Alerts communicate critical instructions such as “reduce demand” or “aggregate”
(3) Smart meters with batteries asked to share power with other residents in microgrid

Illustrates the main components of ACES, including storage and network mechanisms.

2.1 Microgrid Controllers

The main purposes of the microgrid controllers (MGCs) are to mediate communications between the central utility and smart meters and enable intra-microgrid energy sharing. For data transmission, we employ a store and forward approach to prioritize *reliable* communication. The MGCs backup all data sent to the central utility for ten weeks so that the central utility can re-request data from the MGCs in case of storage failures. Furthermore, we initiate `get` requests from the MGCs themselves instead of relying on instructions from the central utility to enable independent operation of microgrids. By reducing the MGCs’ reliance on the central utility, we improve the system’s fault tolerance. In addition, MGCs support variable transmission time intervals to *adapt* to changes in network capacity or processing speed. We describe implementation details in the following sections.

Figure 2: ACES Data Flow


Smart meters **collect data** in history log format
Microgrid controller initiates **get request**
Microgrid Controller
Data is stored in timestamped **file system**
Central utility initiates **get request**
Central utility parses data to store in **database**
File flagged as **backup storage**
Received acknowledge from central utility
Returns **acknowledge** when data received
Garbage collector scans for outdated data
Backup storage data **deleted** after a set time
Flow of data
Microgrid controller components

Highlights the flow of data from the smart meters to the central utility.

2.1.1 Storage

The storage for the MGCs are divided into several components, summarized below.

Table 1: MGC Storage

Component	Data Type	Purpose
`Meter_States`	Table	Stores the states of each meter as booleans: `is_on, sharing_power, aggregating, reducing_power_demand`. Also stores the battery threshold and current battery percentage of each meter if applicable.
`Meter_Last_Update`	Table	Stores the last time data was received from each meter.
`Data_Time_Interval`	Variable	Time interval (in seconds) between smart meter data queries; equals 300 by default.
`Usage_Data`	File System	Stores data from smart meters in files organized by one-hour intervals.
`Request_Stack`	Stack	Tracks current `get` requests.

The `Data_Time_Interval` parameter can be modified by a command from the central utility, improving ACES's adaptability to changes in network demand.

Usage_Data stores history logs received from smart meters in files. Each file includes an is_backup bit. We store data logs directly without parsing due to limited processing power in the MGCs.

Finally, the Request_Stack stores all outstanding get requests until the requested data is received. The stack retransmits failed requests to ensure completeness of the data. We chose the stack data type over a queue because customers are affected by the current state of the system. Thus, we prioritize transmitting recent requests so that we can respond to current power demands quickly.

2.1.2 Processes

To monitor energy sharing within the microgrid and collect smart meter data, each MGC runs the processes outlined in Table 2.

Table 2: MCG Processes

Process Name	Description
Initialize_meters	Starts or restarts all smart meters in microgrid by sending initialize and aggregate (lossless=true).
Query_meter_data	Requests and stores data from smart meters.
Send_data_to_utility	Handles get and acknowledge commands from the central utility.
Process_utility_ commands	Forwards relevant commands (e.g. reduce_power_demand_on) from the central utility to its smart meters.
Manage_energy_ distribution	Responds to changes in battery statuses within the microgrid.
Garbage_collect	Deletes out-of-date backup files.

Query_meter_data runs every Data_Time_Interval seconds. It adds get requests to all meters whose last update was greater than Data_Time_Interval seconds ago to the Request_Stack, and proceeds to send get requests from the stack until the next Data_Time_Interval. Upon receiving data from a smart meter, Receive_data stores the data into the current file and updates Meter_Last_Update with the current time, along with the meters' battery statuses. Finally, Query_meter_data removes all completed get requests from the Request_Stack.

Send_data_to_utility sends requested data to the central utility. Whenever it receives an acknowledgement from the central utility that a file was received, Send_data_to_utility marks that file as a backup. We apply lossy aggregation to backup data to cover as much data as possible within storage limitations.

Manage_energy_distribution handles intra-microgrid energy sharing by instructing meters above their thresholds to share_power_on when at least one other battery is below its threshold. Meters are instructed to share_power_off if no battery requires power.

`Manage_energy_distribution` also directs smart meters to `reduce_power_demand_on` if the central utility is offline and issues `reduce_power_demand_off` once the central utility is running again.

2.1.3 Interface with Central Utility
In addition to the default ANSI commands `get, put, acknowledge,` and `aggregate`, we define the following set of commands from the central utility to the MGCs:

- **`put_billing_rates:`** delivers the billing rates for the current month
- **`reduce_power_demand_on / reduce_power_demand_off:`** toggles function to reduce power demand within each microgrid
- **`set_data_time_interval:`** sets the `Data_Time_Interval` parameter

2.2 Central Utility

The central utility's responsibilities span data management, energy management, and billing. Its contributions to ACES's *reliability* include tracking failed data requests and prioritizing electricity distribution in crises. The central utility *adapts* to changing requirements by employing a flexible relational database, collecting high quality data, and supporting configurable billing parameters.

2.2.1 Storage Mechanisms
The central utility stores its information in the Customer Information Database (CIDB). The CIDB provides reliable storage as multiple processes read and write data, and its contents are easily reconfigurable. Table 3 lists the CIDB tables.

Table 3: CIDB Tables

Table Name	Columns	Notes
Smart Meters	Meter_ID*, Account_Number, IP_Address, Function, Direction, Controller_ID	Created upon system initialization.
Microgrid Controllers	Controller_ID*, Location (Residential, Apartment, or Emergency), IP_Address	
Meter Records	Record_ID*, Record_Type, Meter_ID, Lossy_Aggregation, Start_Time, End_Time, Reading	Records deleted after six months.
Customer Accounts	Account_Number*, Controller_ID, Base_Bill	
Billing Rates	Month_ID*, Peak_Hour_Rate, Non_Peak_Hour_Rate, Starting_Peak_Hour, Ending_Peak_Hour	
Usage Statistics	Month_ID*, Controller_ID, Statistic_Type (Generation, Consumption, or Contribution)	

* Primary Key

We note that if the broadband network or TCP fails, unsuccessful `get` requests are chronicled in a Failed Request Queue (FRQ) file and remade later for reliable data collection.

2.2.2 Data Management

To provide the utility managers and MIT researchers with desired usage information and create residents' bills, the central utility gathers meter data from the microgrids using periodic polling. Table 4 shows the relevant processes.

Table 4: Data Management Processes

Process/Function Name	Description
`Query_data`	Issues `get` commands to each microgrid every hour.
`Remake_failed_ requests`	Every second, pops one `get` command from FRQ if non-empty and remakes request.
`Recover_data`	Re-requests data from up to ten weeks ago from the microgrids using `get`.
`Compute_ statistics`	Once daily, updates total generation, consumption, and contribution for current month in Usage Statistics. Accomplished using SQL queries on Meter Records.

Each `get` command issued by `Query_data` requests data from the most recent completed hour. If `Query_data` receives all the requested data, it is stored in Meter Records. If the request fails, it is appended to the FRQ. Multiple `Query_data` threads run concurrently to reduce latency. Notably, no data aggregation occurs between the central utility and MGCs, making ACES more adaptable by researchers, who receive fine-grained data. `Query_data` only polls for data each hour because none of its applications require near real-time data.

`Remake_failed_requests` operates similarly to `Query_data`, reliably bringing the central utility up to date within several hours after a network outage. In case of a natural disaster, hardware failure, or malicious attack that erases data in storage, `Recover_data` salvages backups from the MGCs, increasing the system's fault tolerance.

Furthermore, the central utility provides two APIs, `Export_statistics` and `Export_all_data` to the utility managers and MIT researchers, respectively. `Export_statistics` uses Usage Statistics to compute averages for the past month, quarter, year, and five years, within a day's accuracy. We can easily expand upon the gathered statistics due to the CIDB's adaptability. `Export_all_data` exports the entire CIDB, giving researchers access to six months of fine-grained records (likely entirely lossless), thus aiding in their efforts to improve ACES.

2.2.3 Energy Management

The central utility purchases electricity during non-peak hours and prioritizes distribution when supply is limited. These optimizations help minimize residents' costs and keep Centertown's important facilities functioning.

The `Purchase_power` process handles energy management. It purchases from the regional grid to maintain a full battery during non-peak hours, or during peak hours if the battery is critically low. This proactive purchasing lowers customers' bills.

We assume the New England regional grid responds to each electricity request with an update regarding their current supply (`NORMAL`, `LOW`, `CRITICAL`, or `EMPTY`). If the regional grid states that supply is not `NORMAL`, `Purchase_power` tells all microgrids to `reduce_power_demand_on`. If supply is `CRITICAL`, the flow of electricity to houses ceases and 50% less is routed to apartments. If supply is `EMPTY`, no electricity is routed to apartments either. Consequently, ACES reliably provides power to emergency facilities, followed by subsidized apartments, by alerting microgrids and rerouting electricity in cases of high demand. Once supply becomes `NORMAL`, `Purchase_power` issues `reduce_power_demand_off` and routes electricity as usual.

2.2.4 Billing

The central utility meets billing requirements by creating bills and setting rates, as shown in Table 5.

Table 5: Billing Functions

Function Name	Description
`Create_bills`	The first day of each month, creates customers' bills based on usage the previous month.
`Set_billing_ rates`	Once a month, adds new month's rates to Billing Rates.
`Send_billing_ rates`	Once a month, sends current rates to MGCs using `put_billing_rates`.

`Create_bills` uses SQL to construct bills. For every customer, it starts with the base bill from Customer Accounts. By filtering through Meter Records, it charges based on time of day for incoming power and credits customers using the non-peak rate for outgoing power (prorated if necessary). If the bill is negative, it is adjusted to $0 and the customer's base bill is reduced by 25%. If the bill is positive, their base bill increases by 25% (capped at the initial value). By adjusting base bills, ACES adapts to changes in the electricity contributions of residents.

The other functions contribute to ACES's adaptability as well by accommodating varying billing rates and providing microgrids with valuable information for optimizing electricity distribution.

3 System Impact

Having described the key components of ACES, we now discuss its best and worst case behavior under various use cases and evaluate the system.

3.1 Use Cases

First, we consider the *normal operations* use case, with typical demand levels and enough sunlight to power all microgrids. If a property's battery drops below its sharing threshold, its MGC learns this within minutes, and instructs other smart meters to share power. By facilitating energy sharing, ACES reduces costs to low-income or retired residents with limited savings. ACES also lowers residents' base bills by operating under the existing LTE capacity to avoid higher network costs. As a consequence, MGCs request data less frequently, but we believe a delay on the order of minutes in the best case is permissible to lower the financial burden on low- and middle-income homeowners and apartment dwellers.

In addition, the system accommodates an *extreme power demand* use case, where demand increases for all microgrids — for instance, a heat wave may drastically increase air-conditioning usage. ACES sends additional power to the apartment microgrid to meet this demand; this power comes from the central utility's proactively purchased charge, as well as extra generated power from the other microgrids (assuming 12 hours of sunlight). If regional supply is critically low, the central utility adapts by restricting the flow of electricity to houses and some apartments to keep the emergency facilities functioning. We believe that because an increase in life-threatening emergencies is inevitable during a heat wave, Centertown must remain able to reliably respond to these incidents. A drawback of this choice is that homeowners and elderly apartment dwellers with health problems that necessitate air conditioning may preemptively lose power. However, in the best case, this would not happen because of the available solar power and early alerts urging residents to lower demand.

Next, in the *storm outages* use case, the central utility is taken offline and half of the residential microgrids are isolated from the town lines. The MGCs continue internal operations as usual and ask residents to reduce demand to prevent properties from losing power. This scenario poses challenges to usual data collection since the broadband network fails; hence, ACES has microgrids accumulate data during the 12-hour outage. Processing this backlog after the storm may delay billing and data delivery by only several hours in the best case. Since the MIT researchers desire large volumes of accurate data rather than instantaneous updates, ACES prioritizes obtaining lossless data eventually over fast — but lossy — retrieval. Notably, this tradeoff does not hinder the delivery of power to residents.

Finally, we consider *long-term changes*, where demand shifts suddenly due to circumstances such as the pandemic. If peak hours change, the central utility updates when it proactively charges its battery. In the worst case, demand may increase enough that a full charge of the central utility's battery cannot last Centertown the entirety of peak hours, forcing it to buy power at a higher rate and add to customers' bills; however, ACES is constrained by the battery's capacity. Due to the often necessary and even life-saving role electricity plays for customers, we prioritize reliable service over cost-effectiveness — the system buys from the New England power grid during peak hours if it must, but minimizes the need to do so.

In all use cases, ACES aims to *reliably* provide affordable energy and collect fine-grained data. Some of the best outcomes for ACES are when residents receive steady power, energy sharing reduces customers' bills, and data is transmitted across the system in a timely and lossless manner for utility managers and researchers. However, each scenario presents different obstacles to achieving the best case. Regarding data collection, for example, storm outages may make the worst-case scenario where ACES delivers incomplete, lossy, or no data to researchers more likely. Another potentially catastrophic outcome occurs when customers lose access to electricity, which is more likely in the extreme demand use case. Losing power would negatively impact customers in many ways, ranging from minor inconveniences to massive health and safety risks. Subsidized housing residents are especially at risk, as they may be more likely to require sustained and affordable power due to factors such as age, medical needs, socioeconomic status, and more.

3.2 Evaluation

ACES respects the storage and network constraints of its infrastructure. Since we leverage lossless aggregation, smart meters never run out of storage. At the microgrid level, the bottleneck is the apartment MGC, since it services over 300 smart meters — an order of magnitude higher than any other microgrid. The data stored in the apartments' MGC occupies roughly 50GB. Across all microgrids, we can only store six months of fine-grained records in the central utility, occupying about 1.5TB of the 2TB total.

In terms of network bandwidth, the apartment MGC could need to download up to 2.7GB of meter data monthly, but we anticipate at least the necessary 30% reduction from lossless aggregation. Regardless, a potential fairness issue with ACES as it stands is that the apartments' microgrid is most susceptible to network congestion. In the worst case, its data could undergo lossy aggregation for part of the month. As a result, researchers may have more difficulty optimizing the system for apartment dwellers, leading to inequity.

Another limitation of ACES concerns privacy. MIT researchers receive fine-grained smart meter data that comprehensively logs when each property consumed electricity. Anyone with (possibly unauthorized) access to this data could discern, for example, when residents are away from home. One of ACES's goals is to reliably collect and send usage data to researchers, but this comes with the tradeoff of risking Centertown customers' privacy.

4 Conclusion

Our proposed system, ACES, provides low-cost, reliable energy by enabling intra-microgrid sharing. Additionally, it manages customer billing and collects fine-grained data for researchers. The design provides *reliability* through decentralized microgrids, request tracking, and backup files. Moreover, ACES *adapts* to changing circumstances by supporting modifications to its tables, data collection patterns, and billing parameters.