ACES: A Cooperative Energy System

6.033 Design Project Report

WRAP Instructor: Brianna Williams

Recitation Instructor: Karen Sollins

May 2nd, 2022

1 Introduction

New technology and environmental awareness have transformed the landscape of energy management. Energy production is becoming increasingly decentralized with rising use of personal solar panels. To take advantage of this paradigm shift, we propose A Cooperative Energy System (ACES), which (1) reliably provides low-cost energy to Centertown customers, (2) enables power sharing within microgrids, (3) calculates customer bills, and (4) collects comprehensive usage data for research. Challenges of creating such a system include real-time monitoring of components to share power when needed, collecting fine-grained data within storage and network constraints, and responding to failures.

The design of ACES prioritizes *reliability* and *adaptability*. Our primary goal is *reliability*, which means accurate and timely delivery of power and data, even during outages. We prioritize reliability because it directly benefits residents and municipal services, who are most impacted by our system. Specifically, we focus on reliable electricity, since power failures disrupt residents' lives by impeding emergency services, heating and air conditioning, Internet communications, and more. Moreover, reliable data collection enables fair billing and system analytics, benefiting researchers who require usage data. Additionally, we prioritize *adaptability*, or the system's configurability by future engineers and ability to adjust to external changes. We aim to minimize the burden of maintenance, which residents would otherwise bear the cost of. Furthermore, adaptability increases ACES's longevity and efficiency as system requirements evolve.

ACES accomplishes these goals by dynamically assigning microgrid controllers to manage energy sharing and transfer data from smart meters to the central utility for billing and export. Combined with request retransmissions and prioritization of electricity distribution, this design provides reliability and adaptability in the face of component, network, and power failures.

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

2 System Description

ACES relies on three layers of components. At the bottom layer, smart meters respond to ANSI commands. Some meters ("meters with batteries") are attached to solar panels. In the middle, microgrid controllers (MGCs) handle energy sharing and collect meter data. Finally, a top-level central utility collects data from the MGCs, provides data extraction APIs, distributes electricity, and handles billing. Figure 1 summarizes this organization.

Notably, the functionality of ACES is divided into two domains: energy management and data collection. Energy management occurs at the microgrid level and aims to effectively deliver power to residents. This domain includes commands for energy sharing (depending on battery levels throughout the microgrid) and reducing power demand. On the other hand, data collection occurs at the smart meter level, with the purpose of transferring data to the central utility. Accordingly, each MGC is assigned control of a set of microgrids for the energy management domain and an unrelated set of smart meters (dispersed across microgrids) for data collection. By introducing this difference in granularity, the system effectively load-balances the storage and network demands of data collection, especially in the high-density apartment microgrid.

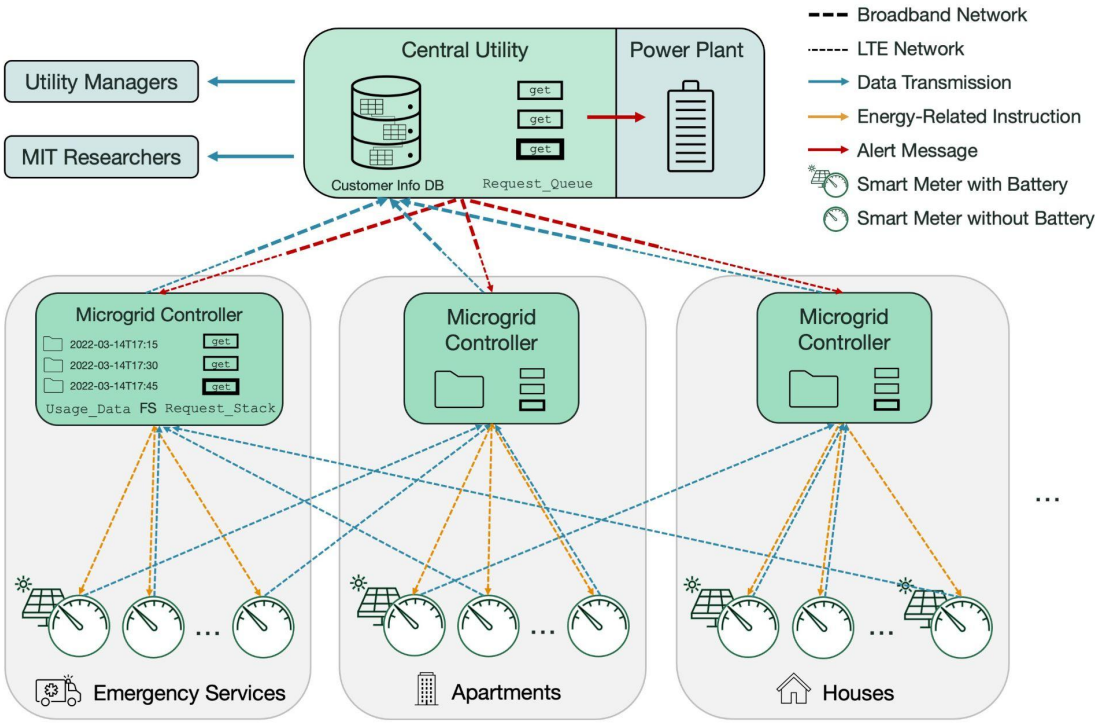
Heartbeats between the central utility and MGCs facilitate detection of component failures. When MGCs fail, their assigned microgrids and smart meters are transferred to other MGCs, enabling reliable energy

and data management. Since MGCs manage energy sharing independently, this functionality persists when the broadband network fails, increasing ACES's reliability.

In terms of storage mechanisms, the MGCs rely on a flat `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). The CIDB offers reliable concurrency and easily adaptable tables, as well as SQL querying and exportability.

For all network communications, ACES employs TCP to achieve reliable, in-order transport. The MGCs and central utility additionally retransmit failed commands for boosted fault tolerance. We use TCP for improved performance, as the transport layer rectifies small amounts of packet loss without necessitating large application-level retransmissions. Since fine-grained usage information enables researchers to improve ACES, the system exclusively uses lossless aggregation for increased adaptability.

Figure 1: ACES Overview



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

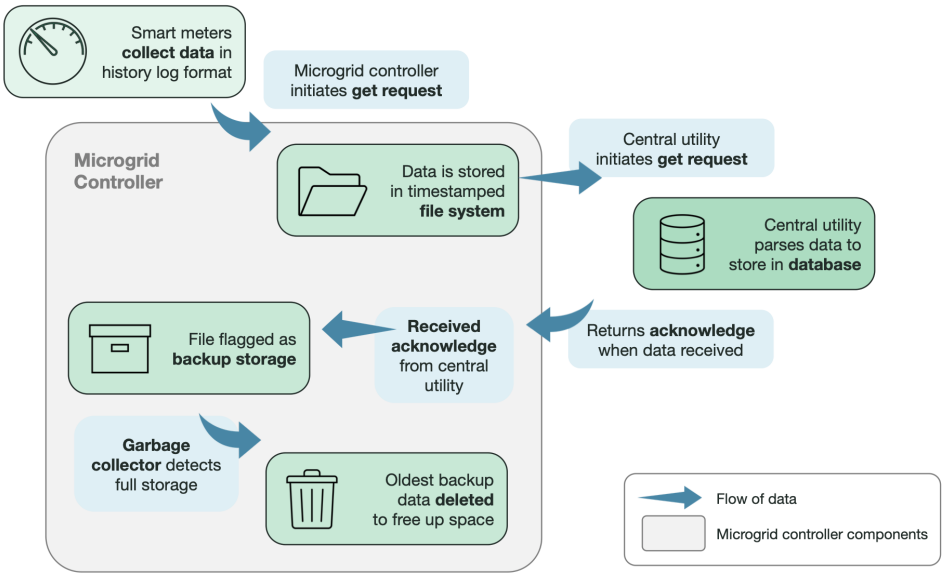
2.1 Microgrid Controllers

The microgrid controllers (MGCs) are responsible for (1) managing energy usage within microgrids and (2) collecting data from smart meters. In each domain, MGCs receive microgrid and smart meter assignment updates from the central utility. Section 2.2.2 details the assignment process. Our system assumes that smart meters always handle incoming commands by replying to the sender's IP address (and direct `put` requests to the IP address that most recently requested its data). In addition, the MGCs track and retransmit unsuccessful commands in case of network failure.

For energy management, every microgrid is assigned to at least one primary and one backup MGC; this redundancy alleviates MGC failures when the central utility cannot update assignments. Under normal conditions, a MGC handles energy management for its primary assignments only, which reduces network usage. However, if the central utility is unreachable (as detected through heartbeats), the MGC also performs energy management for its backup assignments. Since all decisions are deterministic, the system behaves correctly when multiple MGCs control a microgrid simultaneously.

For data collection, each MGC regularly initiates `get` requests to its assigned smart meters. By scheduling `get` requests themselves, the MGCs avoid relying on the central utility to run data collection, improving the system's fault tolerance. In addition, the MGCs retain backups of all data sent to the central utility in case of storage failures. We describe implementation details in the following sections.

Figure 2: ACES Data Flow



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

2.1.1 Storage

The MGCs' storage includes several components, summarized in the table below. All storage is persistent (stored in files) to allow for recovery after failure.

Table 1: MGC Storage

Component	Purpose
`System_Info`	Stores the IDs, IP addresses, microgrid affiliations, and battery thresholds of all smart meters.
`Primary_Microgrids`	Stores the IDs of all microgrids assigned to the MGC for energy management.

`Backup_Microgrids`	Stores IDs of the MGC's backup energy management assignments.
`Data_Assignments`	Stores IDs of smart meters assigned to the MGC for data collection.
`Usage_Data`	Stores data from smart meters in files, with one file for all data from each 15-minute interval (matching the 15-minute query interval from the central utility).
`Request_Stack`	Tracks current commands.

`System_Info` is a static file generated upon initialization. The central utility can rewrite it by command (see Section 2.1.3). Similarly, `Primary_Microgrids, Backup_Microgrids,` and `Data_Assignments` are assigned by the central utility upon initialization and updated by command.

`Usage_Data` stores history logs received from smart meters in files. Each file includes an `is_backup` bit. Once `Usage_Data` exceeds 50GB, the oldest backup files are deleted to maintain storage availability. We store data logs directly without parsing due to limited processing power.

Finally, the `Request_Stack` stores all outstanding commands until they succeed. We chose the stack data type over a queue to prioritize addressing more current requests for power from customers. Each command has an associated expiration, after which it is dropped from the stack if still failing. For instance, `reduce_power_on` expires in a week, as it likely no longer holds relevance after that duration. This mechanism discards stale commands and curbs the size of the `Request_Stack`.

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. Each process runs on a single thread since the MGCs are single-processor machines.

Table 2: MCG Processes

Process Name	Description
`Initialize_meters`	Starts all assigned smart meters using `initialize` and instructs them to perpetually apply lossless aggregation with `aggregate(lossless=true)`.
`Query_meter_data`	Requests and stores data from assigned smart meters.
`Manage_energy_distribution`	Responds to changes in battery statuses within assigned microgrids.
`Send_data_to_utility`	Handles `get` and `acknowledge` commands from the central utility.
`Forward_`	Forwards `reduce_power_on` and `reduce_power_off` from the

commands	central utility to its assigned microgrids.
Garbage_collect	Deletes oldest backup files when close to reaching storage capacity.

Query_meter_data runs every 60 seconds. We chose a one-minute interval so that even with a few retransmissions, the MGC obtains the necessary meter data before the central utility requests it. It sends get requests to all meters in Data_Assignments, adding the requests to the Request_Stack. For the remainder of the 60 seconds, Query_meter_data continues retransmitting commands from the Request_Stack.

Upon receiving data from a smart meter, Query_meter_data stores the data in the current Usage_Data file and removes the completed request from the Request_Stack. If Query_meter_data does not receive even a single record associated with a get request, that request is considered unsuccessful and remains in the Request_Stack. As a result, if a TCP connection fails in the middle of a data transfer, the affected get requests are retransmitted in full.

Send_data_to_utility sends requested data to the central utility. Whenever the central utility acknowledges receipt of a file, Send_data_to_utility marks that file as a backup. Each MGC can back-up at least six months of data (see Section 3.1.2).

Every minute, Manage_energy_distribution queries all smart meters with batteries in the MGC's assigned microgrids for their most recent "power stored" (i.e. battery level) record using get. This minimal data is not acknowledged to allow independent collection in the data collection domain. If any battery is at 0%, Manage_energy_distribution sends share_power_on to all meters above their thresholds. Meters are instructed to share_power_off if they are below their threshold or no battery requires power. Fortunately, each microgrid contains at most 10 smart meters with batteries. Thus, network usage is minimal compared to full data collection, enabling minute-by-minute power sharing instructions. Provided MGC availability exceeds 0.65, each controller remains under the 2GB LTE capacity (see Section 3.1.2). Manage_energy_distribution also directs smart meters to reduce_power_on if the central utility is offline and issues reduce_power_off once the central utility is reachable 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:

- **hello:** sends a heartbeat message
- **modify_energy_assignment:** adds or removes a primary or backup microgrid assignment for energy management
- **modify_data_assignment:** adds or removes a smart meter assigned to the MGC for data collection
- **reduce_power_on / reduce_power_off:** toggles reduction of power demand within microgrid
- **put_billing_rates:** delivers billing rates for current month
- **set_system_info:** updates System_Info file

2.2 Central Utility

The central utility's responsibilities span data management, energy management, and billing. It contributes to ACES's reliability by dynamically assigning microgrids and smart meters to each MGC. These assignments change as MGCs crash and reboot, enabling resiliency to MGC failures and load balancing across controllers. The central utility retransmits failed commands and prioritizes electricity distribution in crises for added reliability. It adapts to changing requirements by employing a 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 threads read and write data, and its contents are highly configurable, facilitating reorganization and addition of data. For example, the CIDB was easily augmented to accommodate dynamic MGC assignments in response to new controller failures. Table 3 lists the CIDB's contents. Each field's subscript indicates its size in bytes.

Table 3: CIDB Tables

* Primary Key

Table Name	Columns
Smart Meters	$Meter\_ID_8$*, $Compressed\_Meter\_ID_4$, $Account\_Number_2$, $IP\_Address_4$, $Function_1$, $Direction_1$, $Microgrid\_ID_2$
Microgrid Controllers	$Microgrid\_ID_2$*, $Location_1$ (Residential/Apartment/Emergency), $IP\_Address_4$, $Current\_Status_1$ (Available/Unavailable)
Energy Management Assignments	$Microgrid\_ID_2$*, $Primary\_Manager\_Microgrid\_ID_2$, $Backup\_Manager\_Microgrid\_ID_2$
Data Collection Assignments	$Compressed\_Meter\_ID_4$*, $Manager\_Microgrid\_ID_2$
Meter Records	[$Compressed\_Meter\_ID_4$, $Record\_Type_1$, $Start\_Time_4$]*, $Record\_Duration_2$, $Lossy\_Aggregation_1$, $Reading_8$
Power Sharing Records	[$Compressed\_Meter\_ID_4$, $State\_Change\_Time_4$]*, $Sharing\_State_1$ (On/Off)
Customer Accounts	$Account\_Number_2$*, $Microgrid\_ID_2$, $Base\_Bill_8$
Billing Rates	$Month\_ID_8$*, $Peak\_Hour\_Rate_8$, $Non\_Peak\_Hour\_Rate_8$, $Starting\_Peak\_Hour_4$, $Ending\_Peak\_Hour_4$
Usage Statistics	[$Month\_ID_8$, $Account\_Number_2$, $Statistic\_Type_1$ (Generation/Consumption/Contribution)]*, $Statistic\_Value_8$

To save storage space, the CIDB uses 4-byte IDs for smart meters (permitting over one billion IDs). The Smart Meters table maps each original 8-byte ID to its "compressed" ID. Billing rates and usage statistics are deleted after five years, since the utility managers only desire data within that time frame. The fine-grained meter and power sharing records, however, are deleted after six months due to storage

limitations. With this configuration, the CIDB occupies up to 1.8TB of the 2TB available in the central utility (see Section 3.1.3).

Unsuccessful commands are chronicled in a `Request_Queue` file and retransmitted for reliable delivery in case TCP, broadband, or an MGC fails. We utilize a queue because older meter data is needed more urgently for billing. Like in the MGCs, each command is dropped from the queue upon expiration.

2.2.2 MGC Assignment System

The central utility maintains mappings from microgrids and smart meters to their assigned MGCs in the CIDB. Each microgrid has a *primary* MGC actively managing its energy-related activities and a *backup* MGC prepared to perform those responsibilities if the primary fails. By pre-assigning backups before failures occur, ACES ensures that if the central utility becomes unreachable, MGCs can promote themselves from backups to primaries. In addition, each smart meter is given an MGC that collects its data. There are no pre-assigned backups for data collection because if the central utility cannot update assignments, it also cannot make use of any collected data.

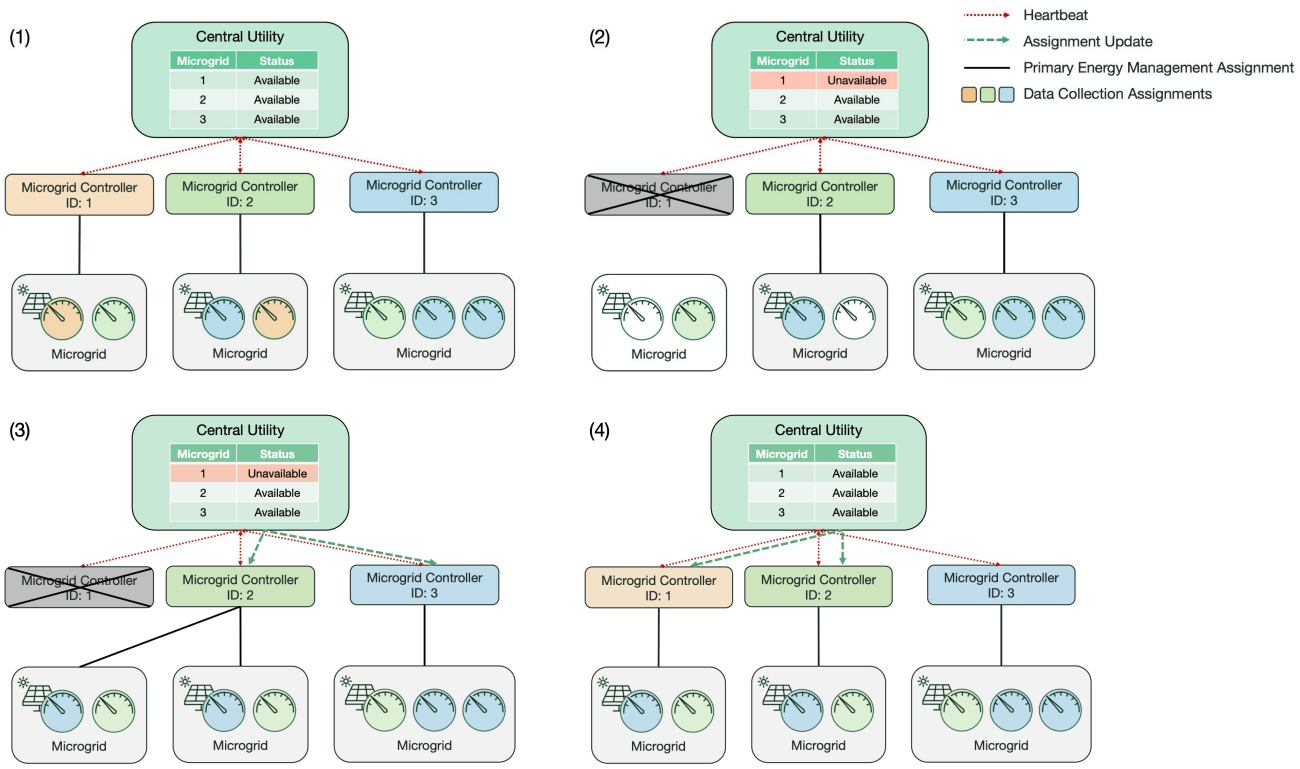
The `Manage_assignments` process communicates assignments to MGCs using `modify_energy_assignment` and `modify_data_assignment`. Upon system initialization, each MGC acts as the primary energy manager of its own microgrid for simplicity and as the backup of another randomly assigned microgrid. Moreover, there are $2(8000) + 303 + 4 = 16307$ smart meters, so each of the 802 MGCs collects data from $\frac{16307}{802} \approx 21$ randomly assigned meters. Physical distance between MGCs and their assignments is not optimized since all infrastructure resides within a single town. Instead, ACES prioritizes load balancing to decrease storage and network usage.

To detect changes in MGCs' availability, the `Update_availability` process pings each controller every second using `hello`. If the MGC acknowledges, it is marked available in the CIDB. If three consecutive heartbeats are not acknowledged (ruling out momentary packet loss), the MGC is marked unavailable. Heartbeats are sent every second to detect MGC failures within a few seconds, but not more often to limit network usage (see Section 3.1.2).

If an MGC with primary microgrid assignments *P* and backup assignments *B* becomes unavailable, `Manage_assignments` promotes the backup MGCs for microgrids in *P* to primaries and assigns new backups for all microgrids in *P* or *B*. It also assigns a new MGC to each smart meter the failed MGC was collecting data from. As a result, functioning MGCs start managing the affected microgrids and smart meters with minimal disruption. Assigning new backups promotes reliability in case many MGCs fail simultaneously. Whenever a microgrid or smart meter requires reassignment, ACES chooses an MGC currently servicing the fewest microgrids or smart meters, respectively, to maintain an even load. Mistakenly believing an MGC is unavailable due to link failures results in duplicative assignments, but network usage remains under capacity even with this redundancy (see Section 3.1.2).

When an MGC becomes available, it is reassigned as the primary of its own microgrid. Once the MGC acknowledges the new assignment, `Manage_assignments` demotes the interim primary of that microgrid to a backup and removes the backup MGC with the highest load. The newly available MGC is not immediately assigned to other microgrids or smart meters. This cautiousness prevents large volumes of changing assignments if a MGC oscillates between appearing available and unavailable. As other controllers fail, the MGC naturally accumulates more assignments, eventually rebalancing the load. Figure 3 illustrates the assignment process.

Figure 3: Dynamic MGC Assignment



Exhibits MGCs' evolving microgrid and meter assignments as an MGC fails and comes online again.

2.2.3 Data Management

To provide the utility managers and MIT researchers with usage information and create residents' bills, a `Query_data` process gathers meter data from the MGCs using periodic polling. The central utility also provides `Recover_data`, `Compute_statistics`, `Export_statistics`, and `Export_all_data` functions, as outlined below.

Every 15 minutes, `Query_data` issues `get` to each MGC, requesting all data collected during an earlier 15-minute interval (at 9:45am, it requests data from 9:15-9:30am). This buffer gives the MGCs time to first obtain the desired data from smart meters. Once again, if any record is not received, the `get` request is retransmitted in full to reliably recover from TCP failures. After performing its new `get` requests, `Query_data` issues previously failed commands from the `Request_Queue` until the end of that 15-minute interval. Successful commands are removed from the queue. `Query_data` uses multiple threads for parallel processing: one thread sends commands over the network, another listens for and processes incoming responses, and a third writes received data records to the CIDB.

We chose a 15-minute interval for data collection so that under normal operations, the central utility's data is at most 45 minutes out of date (see Section 3.1.3). We believe this delay is acceptable since billing only occurs once a month and the utility managers and researchers do not require real-time data. The interval was not shortened further to give `Query_data` ample time to process each request.

In case of a natural disaster, hardware failure, or malicious attack that erases non-volatile storage, `Recover_data` re-requests six months of meter records from the MGCs using `get`, increasing the system's fault tolerance. `Recover_data` refrains from requesting more data due to storage constraints.

Once daily, `Compute_statistics` performs SQL queries on Meter Records and Power Sharing Records to update the total power generated, consumed, and contributed by each property for the current month in Usage Statistics. Generated power is measured directly by smart meters with batteries. ACES estimates consumed power by taking into account incoming power through meters and deltas between battery power generated and power stored. It also considers how long a meter was sharing or shared with based on its battery level and the Power Sharing Records. Similarly, contributed power is measured using outgoing power through meters and estimated intra-microgrid shared power.

Furthermore, the central utility provides an `Export_statistics` API to utility managers, which uses Usage Statistics to compute averages for the past month, quarter, year, and five years, within a day's accuracy. `Export_all_data` exports the entire CIDB, making ACES more adaptable by the MIT researchers, who receive six months of lossless records.

2.2.4 Energy Management

A `Purchase_power` process purchases electricity during non-peak hours and prioritizes distribution when supply is limited. These optimizations keep Centertown's important facilities functioning.

`Purchase_power` purchases from the regional grid to maintain a full battery immediately prior to the start of peak hours, or during peak hours if the battery is critically low. This proactive purchasing means residents may not have the chance to contribute as much power to the central utility to lower their bills, but ACES makes this tradeoff to store more electricity for outages and high demand scenarios, thus prioritizing reliability.

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_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. The emergency services are never deprived of available electricity, as their continuance is prioritized first and foremost; the subsidized housing customers are next in priority. Consequently, ACES reliably provides power to emergency facilities, followed by apartments, by alerting microgrids and rerouting electricity in cases of high demand. Once supply is `NORMAL`, `Purchase_power` issues `reduce_power_off` and routes electricity as usual.

Microgrids are also instructed to `reduce_power_on` when main power lines are down, and `reduce_power_off` afterwards, to limit loss of electricity.

2.2.5 Billing

Finally, the central utility creates bills and sets rates, as shown in Table 5.

Table 4: 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 promote adaptability as well by accommodating varying billing rates and providing microgrids with information for optimizing electricity distribution.

3 Evaluation

Having described its key components, we now evaluate ACES and its behavior under various use cases.

3.1 Quantitative Evaluation

3.1.1 Smart Meters

All meters perpetually apply lossless aggregation, so we anticipate they will never exceed their 64GB of storage. ACES always losslessly aggregates data to reduce storage and network usage, not only in smart meters but throughout the system, without sacrificing accuracy. Regarding network usage, each smart meter exclusively communicates with MGCs, and in section 3.1.2 we demonstrate that each MGC remains under the same 2GB monthly LTE limit in both directions, despite interacting with several smart meters and the central utility at once. An individual smart meter will therefore comfortably remain below the LTE capacity.

3.1.2 Microgrid Controllers

Each MGC's performance depends on how many microgrids and smart meters it services. We thus let E_i be the average (over a month) number of microgrids the MGC with ID i manages energy-related activities for, and let D_i be the number of smart meters (on average over a month) MGC i collects data for. We will show that even a bottleneck MGC servicing $E = \max_i E_i$ microgrids and $D = \max_i D_i$

smart meters respects the constraints of the infrastructure. Table 5 demonstrates how E and D grow as MGC availability decreases.

Table 5: E and D as Function of Availability

Mean MGC Availability A	$E \approx \left\lceil \frac{2}{A} \right\rceil$	$D \approx \left\lceil \frac{16307}{802A} \right\rceil$
1.00	2	21
0.75	3	28
0.50	4	41
0.25	8	82
0.01	200	2,034

The above table presupposes an even load across controllers. In practice, E and D could assume slightly larger values due to delayed load balancing; assuming a lower availability compensates for this difference. The table also assumes that each microgrid's primary and backup MGCs are managing it simultaneously, as this may happen under certain failure circumstances.

Storage in the MGCs is dominated by the history logs in the `Usage_Data` file system. Each smart meter generates at most 3 types of 36-byte records every 15 seconds, yielding

$$\left(\frac{3 \text{ records}}{1 \text{ meter } \cdot 15 \text{ seconds}}\right)\left(\frac{2678400 \text{ seconds}}{1 \text{ month}}\right)\left(\frac{36 \text{ bytes}}{1 \text{ record}}\right) D \text{ meters} \approx \frac{0.02D \text{ GB}}{1 \text{ month}}$$

without aggregation. ACES allocates 50GB for `Usage_Data`; the remaining 14GB easily accommodates the other files, like `System_Info`. Figure 4 graphs the number of months of data the MGCs can store as availability increases.

ACES can adapt to more pervasive and longer MGC failures: even when availability is 0.05, they can store the necessary six months of data. Since mean availability is likely above 0.99, the MGC storage could support over 20 times as many smart meters with the same number of controllers.

For all network usage analyses, we conservatively assume a 5 percent increase in traffic due to TCP retransmissions and an additional 10 percent increase in traffic due to application-level retransmissions[1][2]. We also assume a 40-byte header is added to each packet, and that all control-flow (non–meter-data) packets contain 10 bytes of content[3].

Figure 4: Months of Data Stored in MGCs

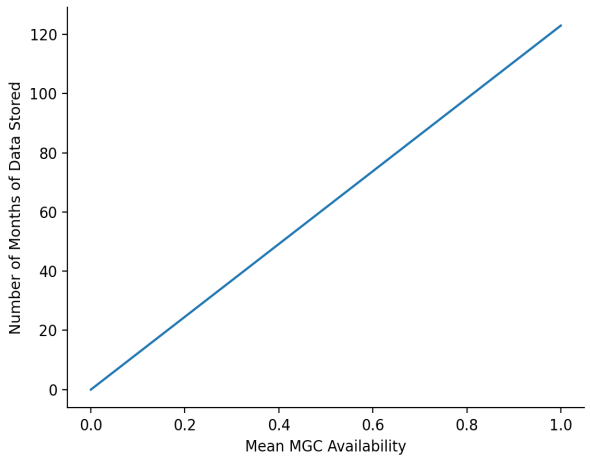
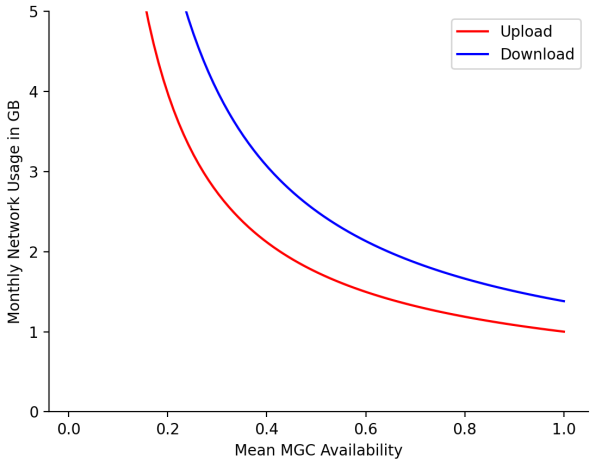


Figure 5: Monthly LTE Network Usage



We now consider monthly LTE network upload and download usage. Due to their infrequency, we simply reserve 0.1GB per month in each direction for commands and acknowledgments involving initialization, aggregation, power demand reduction, billing rates, MGC assignment updates, new system information, and event logs. Event logs are insignificant because a smart meter's state rarely changes in our system. Turning to non-negligible upload traffic, the bottleneck MGC issues up to $10E$ `get` requests, power sharing instructions, and `get` acknowledgments each minute for energy management purposes, one to each meter with a battery in its assigned microgrids (each microgrid has at most 10 meters with batteries). For data management, it sends a `get` request and acknowledgment to each of its D meters each minute. To the central utility, the MGC transmits a `hello` acknowledgement every second and 15 minutes of its D meters' records (at most $15 \cdot 4 \cdot 3$ 36-byte history logs and 15 13-byte power-sharing records each) every 15 minutes. After accounting for retransmission overhead, we obtain a monthly upload usage of

$$0.1 + 0.078E + 0.029D + 0.155 \text{ GB}.$$

For download usage, at most $10E$ smart meters send one 36-byte record (their battery level) and a power sharing acknowledgement each minute to the bottleneck MGC. In addition, the MGC's D assigned meters send a full minute of history logs (at most $4 \cdot 3$ 36-byte records) every minute. We count these history logs twice because some smart meters send this same data using `put`, but ACES ignores those requests for simplicity. The central utility sends the MGC a `hello` every second and one `get` request and acknowledgment every 15 minutes. This traffic produces a monthly download usage of

$$0.1 + 0.65E + 0.049D + 0.155 \text{ GB}.$$

Figure 5 graphs these upload and download usages as a function of availability. Both directions remain under the available 2GB per month when availability is at least 0.65 (which is exceedingly low – roughly corresponding to 20-minute crashes every 40 minutes). Below this point, ACES relies on its lossless data compression to stay under capacity. If, however, availability is at least 0.99, the current network capacity could support a 40 percent increase in the number of smart meters.

ACES's decoupling of energy and data management is key to operating under the constrained LTE capacity. If one MGC collected data from all apartment smart meters, it would exceed 2GB in usage, necessitating lossy aggregation or more capacity. Distributing meters evenly across MGCs significantly

lessens usage at the bottleneck controller. This restructuring also ensures that the network can support additional communications like frequent `hello` pings, MGC assignment updates, and occasional duplicative commands, which increase the reliability of ACES during network and MGC failures.

The main latency-sensitive task MGCs conduct is intra-microgrid energy sharing. Suppose one meter needs charge and another meter is above its threshold. An MGC will query the microgrid's battery levels in at most one minute (unless, for instance, LTE is down). If MGCs or the broadband network fails, the MGC assignment system enables ACES to reliably recover in the span of seconds due to heartbeats. Once an MGC starts querying battery levels, the remaining delay is magnitudes shorter than one minute. We estimate an RTT of around 1.6ms between any two Centertown components[4]. With a 10Mbps connection, sending control-flow packets generates minimal transmission and queueing delay. After the MGC receives the requested battery levels, processing delay is dominated by reading the at most 5MB `System_Info` file to identify battery thresholds, which takes under one second at a reading speed of 35MBps[5].

Consequently, a meter will be instructed to share power in slightly over a minute. If TCP fails, this delay could increase to a few minutes, as the connection would need to be reestablished. We believe a delay on the order of one to five minutes is tolerable because the underlying electrical system likely experiences delays of the same magnitude – it may take several seconds for a meter to physically share electricity over the local grid, so sending power sharing instructions more than once a minute would prove superfluous.

Each MGC performs few computationally intensive operations, as it primarily sends and processes network packets and reads and writes disk files. A single processor is more than sufficient for executing these responsibilities.

3.1.3 Central Utility

The CIDB occupies almost all of the central utility's 2TB of storage. Table 6 bounds the number of rows in each CIDB table.

Table 6: Rows in CIDB Tables

Table	Maximum Number of Rows	Notes
Smart Meters	16,307	16307 smart meters
Microgrid Controllers	802	802 MGCs
Energy Management Assignments	802	802 MGCs meters
Data Collection Assignments	16,307	16307 smart meters
Meter Records	$3.5 \cdot 10^{10}$	(3 record types $\cdot$ 8004 meters with batteries $+$ 1 record type $\cdot$ 8303 meters without batteries) $\cdot$ 1071360 15-second intervals in 6 months

Power Sharing Records	$2.2 \cdot 10^9$	$8004 \text{ meters with batteries} \cdot 267840 \text{ minutes in 6 months}$
Customer Accounts	8,303	8303 Centertown customers
Billing Rates	60	60 months in 5 years
Usage Statistics	$1.5 \cdot 10^6$	$60 \text{ months in 5 years} \cdot 8303 \text{ customers} \cdot 3 \text{ statistic types}$

Using the number of bytes per row as indicated by the field sizes in Table 3, we see that the CIDB stores at most 0.72TB of raw data. We estimate a storage overhead factor of 2.5 associated with using a relational database, based on a proof-of-concept CIDB we created using MySQL with over a million records. We believe that the isolation and adaptability the CIDB provides warrants this overhead. The overhead yields a usage of 1.8TB, leaving an ample 200GB for the `Request_Queue`. The central utility storage is ACES's primary bottleneck. If forced to accommodate more customers or data without additional storage space, the central utility could not store six months of fine-grained data.

There is no capacity associated with the broadband network utilized by the central utility; for reference, its upload and download usage is on the order of hundreds of GB each month.

Under normal operations, the latency associated with transferring a smart meter record to the CIDB is under 45 minutes. The central utility queries MGCs for the record at most 30 minutes after its creation (because of the 15-minute buffer period). We expect the MGCs to possess the record by this point since they query for meter data every minute. The `get` request reaches the MGC storing the record in several milliseconds, after which it reads the appropriate `Usage_Data` file. Even if the MGC were singlehandedly managing all smart meters, this file would occupy at most 70MB, and could be read in 2 seconds at a speed of 35MBps[5]. With a 10Mbps connection, the MGC transmits the file to the central utility in under one minute, likely sooner since each MGC manages only a fraction of the smart meters. The smart meters collectively generate approximately 2 million records every 15 minutes. At a rate of 50,000 inserts per second, the central utility adds these records to the CIDB in less than one minute[6].

Thus, the record in question would reach the CIDB less than three minutes after the central utility sends its `get` requests, or under 45 minutes total. If several application-level retransmissions occur or database inserts are unexpectedly slow, we anticipate that the CIDB would contain the record within a couple hours in the worst case. We deem this delay acceptable since the data is utilized for billing or exported only occasionally.

With its eight cores, the central utility can simultaneously run `Manage_assignments`, `Update_availability`, the three `Query_data` threads, and `Purchase_power`, leaving two cores for other functions. Like the MGCs, the central utility's processes are likely I/O bound, rendering a 2.6GHz processor sufficient.

3.2 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 to 0%, its energy management 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's load balancing across MGCs allows the system to operate under the existing LTE capacity (instead of spending more on higher capacity), also lowering the financial burden on low- and middle-income homeowners and apartment dwellers.

In addition, the system accommodates an *extreme power demand* use case — for instance, a heat wave may drastically increase air-conditioning usage across all microgrids. As per the specification, all solar panels charge for 12 hours per day, which supports 150% of their microgrids' average demands. Residential homes use $1.20(12/24) + 1.05(12/24) = 112.5\%$ of typical demand, leaving 37.5% to store or send back to the central utility. The municipal buildings require 105% of typical demand, with 45% left over. The only microgrid whose consumption exceeds generated solar energy is the apartment microgrid, with 200% of its typical demand (so ~50% must be delivered by the central utility). We expect the central utility to meet the rest of this demand, as there are 8000 homes generating excess power compared to 300 apartments. Even if the homes do not share enough, the apartments can draw from the central utility's proactively fully-charged batteries — which have a capacity of 12 hours of average Centertown consumption.

In the worst case, regional supply may reach critically low levels. This outcome is more likely if demand increases further, available sunlight decreases, or the heat wave fails to subside after a week. We believe that because an increase in life-threatening emergencies is inevitable during a heat wave, it is especially important to keep the municipal services functioning during this time. Thus, the central utility adapts by restricting the flow of electricity to houses and apartments to keep the municipal buildings functioning, as we prioritize hospital patients and those experiencing emergencies over other residents. A drawback of this choice is that homeowners and elderly apartment dwellers with health problems that necessitate air conditioning may preemptively lose power. Subsidized housing residents are especially at risk, as they are more likely to require sustained and affordable power due to factors such as age, medical needs, socioeconomic status, and more. For this reason, apartments' power is only limited after ceasing the flow of electricity to homes. Ideally, the available solar power and early alerts urging residents to lower demand would prevent this outcome altogether.

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; additionally, cloudy weather causes a 10-15% reduction in generated solar energy. The MGCs continue internal operations as usual and ask residents to reduce usage to prevent properties from losing power — otherwise, residents in affected microgrids may lose power 10-15% of the time (since the solar panels in normal weather generate 100% of average demand). As the central utility cannot reassign MGCs because the broadband connection is lost, MGCs manage energy for both their primary and backup microgrids; the redundancy makes it unlikely for a given microgrid to have no functional MGC for energy sharing.

The broadband outage also poses challenges to data collection. The MGCs accumulate meter data during the 12-hour outage, and processing this backlog after the storm may delay billing and data delivery by several hours. Recall from Section 3.1.3 that fewer than three minutes elapse on average between `Query_data` sending its 802 `get` requests and the requested data reaching the CIDB. This processing

rate suggests that `Query_data` could comfortably handle another $4(802)$ `get` requests that accumulated in the `Request_Queue` during the outage in the remainder of each 15-minute interval. The central utility would then obtain all data from the outage period within under four hours. In the worst case, suppose `Query_data` spends one second on average processing a single `get` request. It could consume $15(60) - 802 = 98$ failed `get` requests every 15 minutes, recovering all outage data within 5 days. 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 due to the separation of data collection and energy management. If the storm outage lasted longer than 12 hours, the delay in data delivery to the central utility would increase linearly; for instance, a 24-hour outage may result in a worst-case delay of 10 days.

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 utility managers and researchers receive data in a timely and lossless manner. However, each scenario presents different obstacles to achieving the best case. For example, storm outages make the worst-case scenario where ACES delivers incomplete or lossy data to researchers more likely. Another potentially catastrophic outcome is customers losing electricity, which is more probable during extreme demand circumstances. Losing power negatively impacts customers in many ways, ranging from minor inconveniences to massive health and safety risks. Consequently, while the system places importance on delivering lossless data to researchers, ACES prioritizes energy management over data collection when it comes to timeliness.

3.3 Limitations

The central utility's 2TB storage capacity is the main impediment to ACES's scalability in terms of supporting more customers or data. We chose to store only six months of lossless records as opposed to more lossy records because the researchers requested high accuracy information and can consult the usage statistics for longer-term patterns.

Another limitation of ACES concerns privacy. This data sent to MIT researchers 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 and dynamically assigned MGCs, request tracking, and backup

files. Moreover, ACES *adapts* to changing circumstances by supporting modifications to its tables, energy and data MGC assignments, and billing parameters, as well as producing high quality data for research.

The system is not without areas for improvement — for instance, the collection of fine-grained data benefits researchers, but would compromise Centertown residents' privacy were the data to be accessed by malicious actors. Researchers' data access is also limited in scope to six months. A future improvement for ACES could be to add the ability for researchers to stream meter records from the MGCs through the central utility, as the MGCs collectively have enough storage for five years of data. Additionally, instead of fully charging its battery before peak hours, the central utility could employ a neural network to predict daily usage and purchase strategically to lower costs.

5 Author Contributions

The authors collaborated on conceiving all aspects of the ACES design. In communicating that design, [redacted] focused on analyzing the impact and use cases of the system, [redacted] on the design principles and microgrid controllers, and [redacted] on the central utility and quantitative evaluation.

6 Acknowledgements

We would like to thank Karen Sollins for technical feedback which helped guide our system design, as well as Brianna Williams for feedback on effectively communicating the design. We are also grateful to Katrina LaCurts and the 6.033 staff for their work in preparing and teaching this course.

7 References

[1] Pentikousis, Kostas & Badr, Hussein & Andrade, Asha. (2011). A comparative study of aggregate TCP retransmission rates. International Journal of Computers and Applications. 32. 10.2316/Journal.202.2010.4.202-2660.

[2] Padmanabhan, Venkata & Ramabhadran, Sriram & Agarwal, Sharad & Padhye, Jitendra. (2006). A study of end-to-end web access failures. 15. 10.1145/1368436.1368457.

[3] Kostic, Z. & Qiu, Xiaoxin & Chang, Li. (2000). Impact of TCP/IP header compression on the performance of a cellular system. 1. 281 - 286 vol.1. 10.1109/WCNC.2000.904643.

[4] Nygren, Erik & Sitaraman, Ramesh & Sun, Jennifer. (2010). The Akamai network: a platform for high-performance internet applications. Operating Systems Review. 44. 2-19.

[5] 6.033 Lecture 7, Spring 2022.

[6] Morel, Benjamin. (2017). High-speed inserts with MySQL. Medium. https://medium.com/@benmorel/high-speed-inserts-with-mysql-9d3dcd76f723