Design Project Preliminary Report
The CHEESE System

Hector Carrillo, Helen Propson, Michael Gerovitch
hectormc@mit.edu, hpropson@mit.edu, mgerov@mit.edu

May 3rd 2022

Recitation Instructor: Henry Corrigan-Gibbs

1 Introduction

The city of Centertown has made massive strides in its renewable energy initiative. The infrastructure is there but now they desire a system that can truly bring out its fullest potential. To that end, we propose CHEESE, the Centertown Housing Electrical Energy SystEm. CHEESE strives to bring affordable and reliable electricity to all Centertown Residents.

CHEESE manages power sharing and power purchasing through the help of a Reinforcement Learning Algorithm by the name of UCRL2, future proofing CHEESE for years to come. CHEESE divides the city into regions to helps ensure the system runs optimally despite data capacity bottlenecks. To demonstrate the balance of these priorities in our system, we begin with a brief overview.

2 CHEESE Overview

Our system has three main modules: the central utility, microgrid controllers, and smart meters. We add an additional level to the hierarchy: the region [see figure 1 below]. This is a group of 10 house microgrids or three apartment complexes, of which one is assigned to be the main. The main is swapped once every three days with whichever MC has the most amount of data left to help distribute the added load. Most communication to the CU is as a region, not a microgrid, therefore the main microgrid controller can pre-process this data. In addition, any instruction the central Utility wishes to impart onto the region is distributed by the main microgrid controller. This cuts down on CU communication by roughly a factor of 10 on its own.

The distributed layout of the system allows for all the desired functionality related to data collection; data from SMs is sent to MCs and then to CUs. Since all the logs end up being stored in the CU, the CU can determine accurate billing, provide data for the town's analysts, and store data for use by MIT researchers.

Meanwhile, the CU will perform necessary calculations to adjust when and how much power it buys from the region and informs the MCs of upcoming weather abnormalities.

We split up the towns operation into three times (cycled through daily): off-peak, mid-peak, and peak. Additionally, we define three possible states for the operation type: normal, indoor/increased sustained power consumption, and storm. These discrete states help the learning model, since training on discrete values is much more effective than on continuous values. Additionally, including operation states like increased power consumption and storm allow the system to better adapt to various use cases, ensuring that residents of Centertown are able to have reliable power as much as possible.

3 Regions

Our system requires the town to divide its 800 houses microgrids into 80 regions of 10 houses each. This modularity reduces overhead of communication between MCs and the CU. Each region has a main MC that acts as a middleman between the other MCs in the region and the CU.

If the CU were to send instructions, receive battery and power usage data, and send/receive learning model updates to/from each MC, the CU's 2GB monthly data limit would be used up very quickly and inefficiently. We require that the housing microgrids be grouped into regions of ten microgrids in order to take the burden of distributed instructions, packet overhead, and machine learning data off the central utility; the central utility delegates some responsibility to the

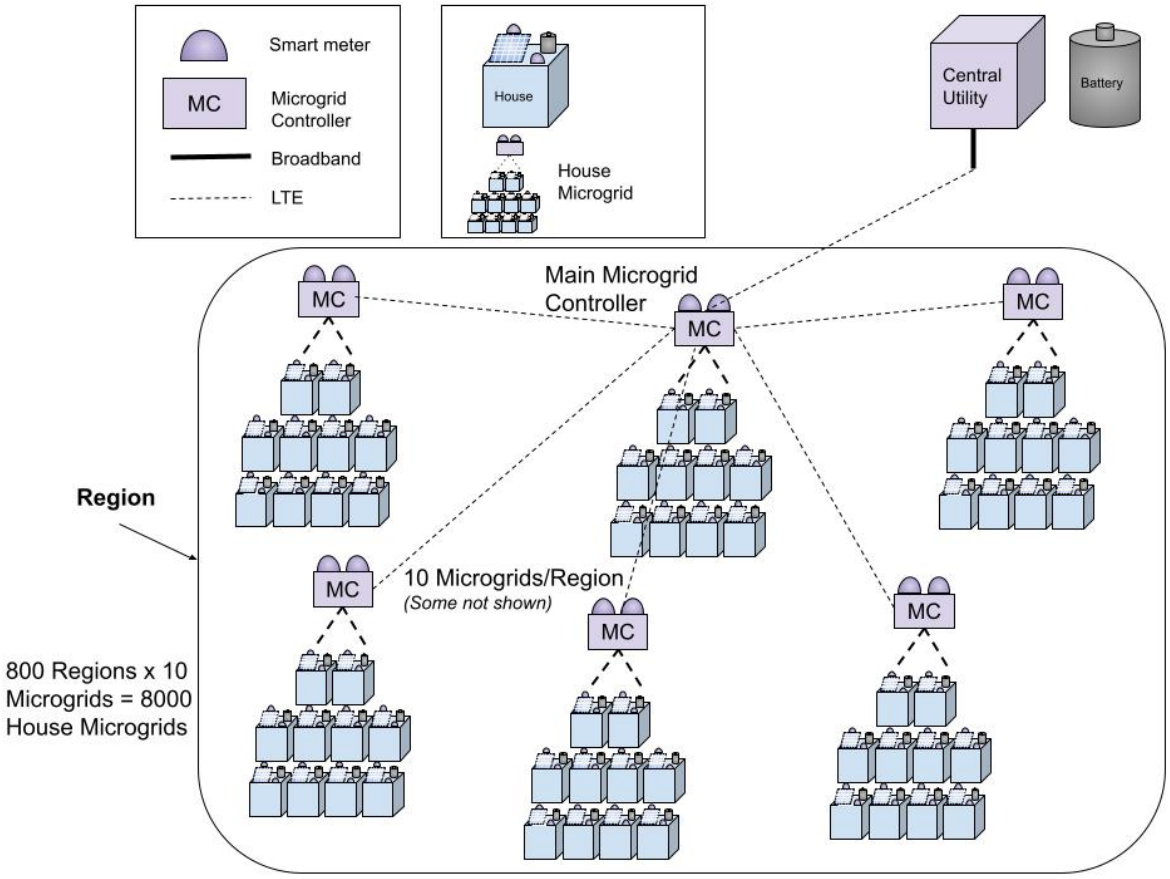


Figure 1: A simplified overview of the CHEESE

region's main microgrid controller. Even though our implementation of these microgrid regions requires that data and instructions are sent and received more than once within a region (creating inefficiency), we evaluate that the monthly data limit of the CU is a bigger bottleneck compared to the monthly data limits of each MC, justifying our choice to delegate communication to the regions.

3.1 Selecting a Main MC

Each region designates one of its MCs as the main MC in charge of communication with the CU on behalf of the rest of all ten microgrids in the region. This selection process is done every three days by the CU. The central utility chooses the MC that has the least data usage this month. This requires each microgrid to store its own monthly data usage, and send it through the main MC to the CU every three days. Once the CU determines the new main MC, it sends an instruction to the old main MC, which informs all MCs in the region of the main MC change.

The reason our system selects a new main MC every three days is that given a random data usage among the MCs, each MC will have a chance to be the main MC since there are roughly $3 \times 10 = 30$ days every month. However, in the worst case, if the same MC is the main MC for a whole month, in the evaluation section we determine that it will take up around 700MB of data per month, which is well below the 2GB limit.

Since this happens relatively infrequently – every three days – it makes sense for the CU to perform this operation, since it is simpler than each region performing this selection process on its own. The CU should distribute this workload over the full three days, doing $800/3$ selections every day and $800/(3 \times 24)$ or roughly 11 every hour, to make sure that the network is never throttled.

3.2 MC to MC Communication

Implementing regions in our system requires communication between MCs. When the system is initiated, the CU goes through each region and sends each of the ten MCs a table with the static IPs of the 9 other MCs in its region, indicating which of the MCs is initialized as the main one. This allows the MCs in a region to communicate with one another over LTE.

During operation of the system, the following information will be send between MCs:

- Smart meter logs
- Instructions sent from CU and distributed by main MC
- Machine learning model data/updates
- Monthly data usage

3.3 Main MC to CU Communication

The main MC within a region will have the following responsibilities to the CU:

- Sending SM logs from all ten microgrids in the region
- Receiving instructions from CU and distributing these instructions to other MCs in the region
- Sending and receiving machine learning model updates/data to and from the other MCs in the region

3.4 Storage and Communications for Region Implementation

The CU contains a table with which MCs are in which region. CU also keeps track of how much data each MC uses each month, updating every three days during the changing of the main MC.

Each MC in a region will have a table with the static IPs of every MC in its region. Additionally, each MC will record how much data it uses every month.

The following commands can be send from the CU to any main MC.

Commands from CU to MC

Function Name	Description
`isolate`	Region is to operate without power from the CU
`set_power_reduce`	Command MC to tell SMs to reduce power
`get_power_usage`	Requests logs from MC
`get_data_usage`	Requests data usage table
`ping`	Allows the CU to check if a MC is running

Each of these instructions can be sent with a flag to distribute to all MCs in the region. If a main MC receives a command with a distribute flag, it will forward this command to all MCs in the region. In response to the request to a `get...` command, the main MC would wait until getting all data from the region's other MCs before sending (or until a timeout). If a MC times out, the main MC reports this to the CU so it can replace it (see later section).

4 Central Utility

Everyday the Central Utility searches the internet for the weather forecast for the day. It will then distribute this knowledege to the microgrids which will inform which machine learning model to use during which times slot.

4.1 Storm

How the system behaves during normal and indoor weather is largely determined by our learning algorithm which we will describe shortly. However, machine learning requires iteration and data, something we cannot afford when a storm is on the horizon. During storms, houses are not allowed to share power and Central Utility is to buy power until the battery is at least 75% full. We do not want power to go out into the city and aggressive power storage helps ensure reliable service in people's time of need.

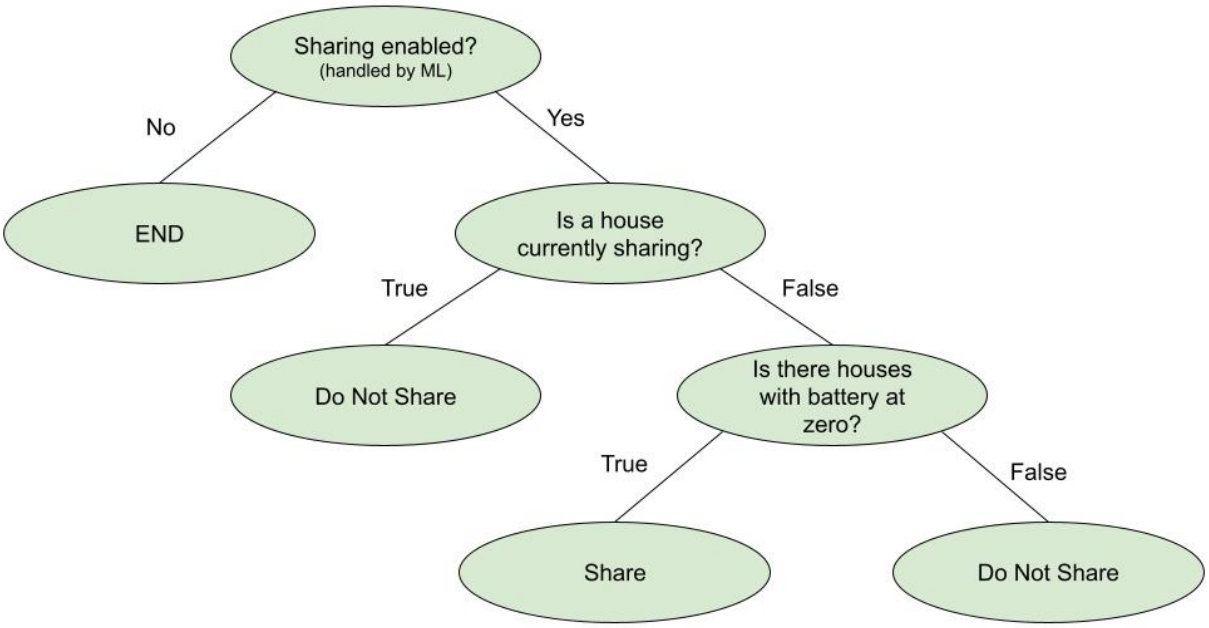
4.1.1 Critical Services Prioritization

Should our preparation not be enough, we are prepared to shut off power to most microgrids in order to keep our hospitals alive, fire department, and police station alive. The Central Utility will induce rolling blackouts starting at 20% battery power to all non-critical and non-apartment microgrids. Once power reaches below 10%, even the apartment microgrids will be shut down. This gives the hospital multiple days worth of electricity to give time for the city to recuperate.

5 Microgrid Controller

5.1 Sharing power

We utilize a fairly simple algorithm for sharing power. It is described by the following tree diagram:



As you can see, the algorithm is quite simple it simply describes the process of recognizing that there exists a house that can receive power. The main complexity comes in the very top node, the 'sharing enabled' which is determined by the model active at the time in which we go into more detail in a later section.

When sharing has been recognized as a possibility, the microgrid controller searches for the house that has shared the least and highest the highest battery and makes them share. They share until the battery is above 0. There is a buffer period where no house can share for 15 seconds to account for noise and oscillations.

5.2 Aggregation of logs

Smart meters create a lot of data, more data than the Central Utility receive, therefore we need to ration the Central Utility's LTE data as best we can. We decided to allocate 1GB of data for log transfer. This is half of its monthly limit but it is vital not only for billing but also for the Central Utility to make informed decisions about when to buy power. With this 1GB limit, we have found we can send logs every 40 minutes form each region. Logs are aggregated as follows: power through meter is an average is a simple average, power generated is a sum, and power

stored is sent as final and initial throughout the 40 minutes. The math behind these calculations can be found at [section 14.2-14.3].

6 Continuous Optimization

The recent pandemic caused a massive shift towards working from home. We propose the usage Online Reinforcement Learning in our system, so that our system evolves how much and when it buys and sells power to account for power usage trends of the town's residents.

The system has two main functions: it determines how to power share and when to buy power. These are a bit too open ended for a machine learning model. Instead, our model is to enable the sharing of power between houses and whether to stop or start buying for the next couple seconds. It is formalized in this way because models tend to like binary outputs. This reduces the risk of our model deciding to buy all power in the world because of some strange extrapolation it performed. This also allows us to more easily construct safeguards to ensure the model doesn't actively makes things worse than no system at all.

The setup is simple. Each microgrid will have a model, each microgrid within a region will have the exact same model, the central utility will have a unique model, and the central utility is the one distributing information as to how to update the model. Microgrids handle when to enable power sharing between houses and Central Utility handles when to buy power from the outer grid.

With the basics out of the way, let us now describe which arguments the model uses to make decisions, which we find helpful to describe from a human perspective.

6.1 Making decisions

If I were to be an AI controlling our microgrid, what data would I need in order to decide whether houses should be allowed to share power or not? Generally, only three things: how full is the battery level in my microgrid, how much power are we generating, and how much power are we consuming. That's it. Give our model these three numbers and it will decide whether or not to enable the sharing of power.

Central Utility's model is a very similar story with a couple minor tweaks. The model has an additional argument in its own battery level, want to differentiate between the city's and the main. Instead of a single microgrid's worth of average battery level, Central Utility averages the entire city for their average battery level. Aside from that, that is all. However, how does the Central Utility know the city's battery level? Well logs are sent to the Central Utility every 40 minutes which contain this data, so the Central Utility will be working with stale data but it will guess the city's based on the arguments described above.

Now that we know how our model makes decisions, we now need to talk about how we change the way our model makes those decisions.

6.2 Online Reinforcement Learning Overview

Typically in machine learning we are creating a model that, when fed in input, the model can accurately guess which bin the input belongs in; for example, is a cat in this picture? Much of that holds true, except in our case, we are interfacing directly with the real world and this brings along some complications. Specifically, we cannot asses our model immediately after creating it. We must let it run for some amount of time and then see how well it performed.

This closely resembles a sect of machine learning called *Online Reinforcement Learning*. *Online* refers to the fact that we can only asses our machine's performance only after trying the model out and *Reinforcement Learning* is typically used when the model will be present in some *environment* instead of being fed inputs like pictures.

Most Online Reinforcement Learning algorithms make an assumptions that we cannot satisfy; for contrast, we will use a robot looking for food in a grid. First, the model is in a static environment. For the robot, it knows that food will be present at block k after it takes n steps because the world is static. This roughly corresponds to the idea that a specific model is just as good now as it will be in 50 years, which it won't be.

This limitation is ultimately what lead us to decide to use the UCRL2 algorithm. It's main claim to fame is its lack of distinction between *know/unknown* configurations. Instead, it recomputes which models are plausible and then determines which one to go to via a 'regret' function which uses the data of previous objective functions as its input [section 13.1].

However, before we go into detail about our objective function, we need to talk about the general structure of our model first.

6.3 Implementation

As discussed in our system overview, we will have three time states and three weather states. The third weather state – storm – will not have a model as it is too infrequent to gather any meaningful amount of data. Each combination of time and weather will have its own model that the system will swap between. The purpose for this is so that we can have more granular *objective functions*, ensuring we prioritize affordability when we want affordability and emphasis reliability when we want reliability. Having the Central utility update 800 models daily is too much to ask for; instead each region will have equivalent models to reduce load on the Central Utility. There isn't much of a trade-off here; since regions are quite close physically, we can expect each microgrid in the region to behave in a similar manner.

We are also aware that behavior is significantly different at the beginning of a time slot versus the end of a time slot, for example most people are waking up at the beginning of mid-peak and work is at full-throttle by the end. Only one model will be active throughout this entire time-slot and its reward is dependent on its performance throughout the slot. This now begs the question, how are we rewarding our models which segues neatly into objective functions.

6.4 Objective Function

A quick note is that objective functions are usually expressed in terms of loss though we found it easier to reason about in terms of reward. Simply take the negation of our objective function to return to a world of loss.

Our objective function is our way of telling our model how well it did. Well, we as a team highly prioritize reliability and affordability and a way we can formalize this is through the use of battery level and cost respectively. We do have to be careful about how we approach this. We don't want to penalize models that existed when times were rough nor do we want to reward models who existed when the sun was extra bright. What we can do to account for this is to measure our system's performance in relation to the expected performance where no system was present at all. In addition, we would like for the reward to be bounded to minimize the probability of expected behaviors. Let's first take a look at the formalization of reliability:

$$(\text{final battery level} - \text{initial battery level}) - \frac{\text{power generated - power consumed}}{\text{max battery level in joules}} \tag{1}$$

Battery level is defined as how full – in percent – the batteries below your hierarchy are. In other words, Central Utility cares about full every battery is, whereas each microgrid controller only cares about its ten houses. Power generated/consumed is calculated over the eight hours this model is active in its time-slot.

This is essentially the formalization of actual battery shift – in percent – minus expected battery shift. Because we are working in percentages, this function is bounded between -1 and 1. An additional benefit of this bounding is that this reward functions scales with hardware upgrades and adequately progresses to the future.

$$\frac{\text{amount spent}}{(\text{power consumed - power generated}) \times \text{regional price per watt}} \tag{2}$$

Essentially our current expenditure on power – only measured in the time the model was active – versus expected expenditure on power. This function is bounded between 0 and 1.5. Why 1.5? Regional price per watt is defined as non-peak price, therefore dissuading the system from buying power during peak times. Unlike equation (1), we want to minimize this value, so we will be taking the negation in the objective function, though we would like to keep the clauses positive, so instead we will subtract this function by 1.5. The final general objective function is as follows:

$$J() = a \cdot \left[(\text{bat}_f - \text{bat}_0) - \frac{\text{gen} + \text{consumed}}{\text{max bat level}} \right] + b \cdot \left[1.5 - \frac{\text{amount spent}}{(\text{consumed - generate}) \times \text{regional price}} \right] \tag{3}$$

Often you find regularization terms in objective functions to prevent overfitting and getting outlandish outputs once you leave the realm of your training data. However, both of our terms in our objective functions *are* our regularization terms. Regularization is often used because we use a proxy for our objective rather than what we are actually seeking. In our function however, we use the direct metrics we care about, eliminating the need for a standard regularization term.

We have the form, now we have to attach coefficients to each clause to reflect how much we prioritize affordability and reliability. We can't just set numbers that sound right, we must make them more rigorous and make sure they make sense in context. In order to do that, we must first define what exactly we are looking for from our 6 models.

6.4.1 Desired Outcomes

As stated, we have six models per microgrid controller; the cross product of 3 time-states and 2 weather states. Let us define what we desire from each state independently and then as a cross product.

From peak-time, we would like to optimize for affordability as it is too costly to purchase power at this time. In both mid-peak and off-peak we would like to emphasize reliability over affordability, though mid-peak still has some emphasis on affordability because of considerable usage.

In nominal weather, affordability is prioritized, because of all the great sun shining upon us and we are somewhat more flexible in these times. Conversely, indoor-weather would prioritize reliability, because we do not want power to fail when people need it most.

These are just numbers though, they do not lack meaning behind any context. So what we will do is walk you through how we decided on the coefficients for the objective functions.

6.4.2 Scenario walk-through

Let us set up a general scenario in which we can plug in different states and times to get a desired outcome. These scenarios will consider only the region models, not the Central Utility models though it turns out similar numbers can be used for both models and achieve good results. Most relevant scenarios are the ones with low battery because share power is useful when a battery is empty.

Consider a model during a time slot where the generation was net negative. In fact, during the span of 8 hours, a net 17% worth of battery was consumed during this time. The battery initiated at 10% for the region. Let us consider three possible battery level endings. Note, a net 7% was lost from the region, therefore at least that much must have come from the Central Utility.

Final battery level	Objective Function
0%	$a((0-.1)-(-.07))+b(7/17)$
2%	$a((.02-.1)-(-.07))+b(9/17)$
5%	$a((.05-.1)-(-.07))+b(12/17)$

If we let $a = b = 1$, then the system marginally prefers 0% battery level, but if we wanted it to incentivize it reliability, i.e. high battery, then $a = 5, b = 1$ would achieve this result. In peak the preference for 0% increases drastically and it takes $a = 9, b = 1$ for 5% to be preferred.

We repeated this type of case analysis and ultimately determined these to be fitting coefficients:

States	a	b
Normal + peak	1	3
Normal + mid-peak	3	2
Normal + off-peak	5	2
Indoor + peak	3	5
Indoor + mid-peak	7	3
Indoor + off-peak	9	2

This is great and all, but how can we be sure that our model doesn't go haywire and accidentally pours all of energy down the local loop drain? For that, we have added a couple safeguards to prevent this from happening.

6.5 AI Safety

Machine learning algorithms aggressively optimize the objective you set out for them, which may result in models behaving in ways you absolutely do not want but technically optimizes the objective. To protect against this, we added a clause to our objective function of the microgrid controllers and a simple restriction to our Central Utility's functionality.

Beginning with the objective function of the microgrid controllers, the model will automatically receive a score of negative infinity should it end with empty battery and greater production than consumption. This is inexcusable and only possible if all power was sent to the local loop and no one used it.

For the Central Utility, it may not purchase power when above 95 percent capacity; this helps safeguard against wasteful purchasing. Should the Central Utility detect that the model was trying to purchase power at this threshold, the model will receive a 20 percent reward reduction; this helps the algorithm dissuade the algorithm from approaching this barrier.

6.6 Communication Evaluation

It is known that machine learning uses a lot of data, which is concerning when we consider the meager 2GB of bandwidth the Central Utility can use with Microgrid communication. However, it is not necessary to send all the data to the Central Utility; all the model cares about is how it is performing. Instead of the Microgrid Controllers passing along all the logs to the Central Utility, the microgrid controllers can instead pre-compute the objective function and simply pass along the number. All in all, our learning model uses 400KB of the Central Utility's data and 5 KB of the main microgrid data to report how well the model performed. It takes roughly equivalent amount of data to update the models in each region, which means that this algorithms takes a meager 1MB of data to do its job[section 14.1].

7 Robustness

7.1 Centralized vs Distributed Decision

The CU decides which microgrid controller is responsible for which microgrid at any given time. The CU is best suited to make this decision, as it already has all the information about the location and IP address of each MC and SM and any data that the new MC would need about the unsupervised microgrid to begin sending the microgrid instructions.

Additionally, if this was a distributed decision, it could involve a MC having to constantly ping all other MCs, flooding the network with messages and creating more opportunities for failure due to the increased complexity. Furthermore, having the CU be the sole decision maker ensures that as our system grows, we will not need to update all other microgrid controllers about new microgrids and MCs. Therefore, we have chosen the CU as the central authority for the decision of which MC to choose as the replacement MC.

7.2 Algorithm

The CU will maintain a view table mapping MCs to the microgrids they are in control of. If a MC goes down, the CU will pick a replacement MC controller that is in control of the least amount of microgrids to reduce strain on the MC's processing, network, and memory limitations. The CU will first look to other MCs within a microgrid's region to find a replacement, but if none of the region's intact MCs are managing less than 3 microgrids, it will look outside of the unsupervised microgrid's region for a replacement MC. If no MC outside the unsupervised microgrid's region is managing less than the MC's within the region, the CU will pick the MC within the region that is managing the least microgrids.

Once the CU has found a replacement, it will update its view table and send a prepare message to the replacement MC. Upon receiving an acknowledgement from the replacement

MC, the CU will then commit this change and send out a confirmation message letting the MC know that this change is now in effect.

The replacement MC will then request data from all smart meters within the microgrid so that it has the energy storage information it needs to determine if a share power command is necessary (the MC can get the IPs of the smart meters via the CU). It will also store all data that the smart meters produce in the time that the original microgrid controller is down.

After the original MC gets back after reboot, we would like it to resume being responsible for the microgrid, so the CU will send a prepare message to both MCs. Once the CU has received an acknowledgement from the original MC, the CU will commit the change and control will be transferred back to the original MC.

The newly-restored MC will not receive data from the interim MC. Instead, the interim MC will send the data it collected while the original MC was down to the CU for storage.

Commands from CU to MC

Function Name	Description
`prepare`	Informs MC that it will soon be instructed to take over a new microgrid
`commit_change`	Commands MC to oversee microgrid; includes IPs of SMs

Commands from MC to CU

Function Name	Description
`ack`	Acknowledges that the MC is ready to take over new microgrid

7.3 Justification

If at any point during the control transfer a smart meter sends the wrong MC data, the MC can reject the data. We are ok with this happening because the smart meter won't delete the data until it receives an ACK from the MC.

Additionally, we recognize that we are trading off complexity for fault tolerance and the guarantee that power sharing functionality is always enabled. However we are ok making this tradeoff because we want Centertown residents to have access to a reliable source of energy whenever possible.

Furthermore, having a MC responsible for more than one microgrid at a time will not strain the system for the following reasons. Firstly, every 15 seconds, the smart meters of a residential microgrid generate 1440 bytes of data (without any aggregation being preformed), so the bandwidth of the LTE connection will allow for a single MC to receive data from 13020 microgrids. Second, if a MC stores all data generated without aggregation applied to it from a single microgrid for a year before sending it to CU (in which case it could potentially be deleted), it will only be using 4.7 percent of its 64 GB storage capacity, so storing the data of other microgrids for 20 minutes will be relatively negligible at this scale.

8 Evaluation

8.1 Communication and Tradeoffs

We noticed that the biggest bottleneck in our system is the monthly data limit of the CU. We are able to reduce the network load of the CU almost ten times by implementing regions for the

house microgrids, only sacrificing up to 90% more data load on each MC, up to around 700MB per month, which is well below the 2GB limit.

Regions also sacrifice latency, since adding an the main MC as an intermediary can double the time it takes instructions and data to travel in the network. However, our system ensures complete data collection for billing and research even though latency is increased.

The evaluation of the ML's communication is described in section 6.6. Math for the calculations can be found in the appendix.

8.2 Use Cases

8.3 Normal Operation

Normal operation as described in this paper makes sure that all microgrids can adjust for minor spikes in energy usage. During normal operations, battery levels will remain around 75% which means that all residents will have enough power. In the extreme case that individuals burn through all of their battery's power, their neighbors can share power with them. The expectation is that the system will not have to buy any power during this time, since it takes merely 8 hours for a solar panel to fully charge a battery (even in overcast weather, this number goes up by at most 10%).

Data collection will continue as described in the paper. The primary focus during this period of operation will be saving cost for the users. As long as the central utility limit the amount of energy it purchases from the region, the towns residents monthly bills will be minimal.

8.4 Scaling

Since the current bottleneck is 700MB/2GB being used, we could likely double the number of houses before we would reach data capacity. Increasing data network plan would allow the town to accommodate an even greater population.

8.4.1 Extreme Power Demand

Centertown may experience periods of extreme power use. To ensure that residents have enough power to operate essential equipment (refrigerating various cheeses) and air-conditioning, particularly those in the subsidized apartments, the CU purchases an adequate supply from the regional grid prior to peak hours (3pm). The CU will purchase enough energy to fills its battery to 75% minus the amount predicted to be generated by microgrids. The CU may also look at its records to see how much energy was used on days with similar whether conditions in the past to aid this calculation. The idea is that as time goes on, these are parameters than can be tuned by the system's learning algorithm.

This period experiences increases around 20% more energy usage. Barring hardware failure, the system should continue to operate with minimal power from the region, since the solar panels will be able to charge up the batteries during the day. In some cases, residents may need to reduce their power usage to minimize cost. Otherwise, the CU will buy power from the region.

8.4.2 Storm Outages

Similar to the extreme power demand case, the CU preforms energy generation estimations (though these are likely smaller for storms than in the case of high temperatures), and energy usage estimations, and purchases the corresponding amount of net energy demand from the

regional utility. However, unlike high temperatures, the CU likely does not have the ability to determine when power will be restored. Furthermore, it is possible that the CU will not be able to purchase power from the regional utility. Since reliable energy provision to critical services is a top priority, the CU purchases a higher amount of energy than it estimates will be needed for a full day.

Additionally, in preparation for a storm, CU can inform MCs to turn off power sharing even when houses fall below their thresholds. This allows microgrids to maximize the amount of power on their local loops in preparation for the storm, in case of power shortages or power line failure during the storm.

For an outage of 12 hours, as long as batteries are half full, a house can provide power throughout a storm. Further, a full battery could sustain a house for a complete day.

8.4.3 Long Term Changes

In the case of changes to peak hours, the CU purchases more energy from the New England regional company prior to the start of these hours. Additionally, when looking at previous data to make estimations, the CU uses the most recent relevant records to inform its decisions. For instance, to get estimations of energy usage for extreme power demand, it looks at the most recent day with similar weather decisions. As a result, the CU adapts to recent changes within Centertown almost instantaneously.

The learning algorithm will be best suited for making this adjustment since it is specifically being trained to optimize for cost and sustainability during various hours of the day. Since these are long term changes, the system will adjust to the new peak hours relatively quickly. Thus, we work to ensure that residents pay as little extra as possible while still having access to a reliable power supply.

9 Conclusion

Centertown is in need of a reliable, affordable, and adaptable system to manage its electricity. CHEESE addresses this need by creating a hierarchical framework that buys and distributes power throughout the town. CHEESE defines a minimalistic, layered communication network between the town's central utility, regions of microgrids, and smart meters that allows the system to learn from power usage data and predict future power demand. It ensures that a microgrid will always have a MC overseeing it to enable power sharing, so long as there is a MC available. This ensures a fault tolerant system that provides reliable power to the town's residents and services while storing power usage data and keeping costs low. Recommendations for further actions include improving the system's robustness to attacks. The coefficients of the objective functions are also open for further investigation before implementation.

10 Author Contributions

Hector focuses on the central utility and reinforcement learning. Helen is primarily responsible for fault tolerance and microgrid controllers. Michael concentrates on regions and use cases as well as the system diagram.

11 Acknowledgments

We would like to acknowledge Katrina LaCurt's 6.033 lectures and lecture slides for inspiring many of our protocols, including lecture 11 and lecture 20. We also acknowledge Henry Corrigan-Gibbs, whom we consulted in developing our design.

12 All Commands

Central Unit to Microgrid Controller

Function Name	Description
`isolate`	MC is to operate without power from the CU
`adjust_sharing`	Changes under what criteria the MC enables power sharing
`adjust_time`	Tells MC to change when off/mid-peak hours start/end
`set_power_reduce`	Command MC to tell SMs to reduce power
`get_data`	Requests data about the microgrid
`get_monthly_usage`	Requests how much energy each house in the microgrid used
`ping`	Allows the CU to check if a MC is running
`prepare`	Informs MC that it will soon be instructed to take over a new microgrid
`commit_change`	Commands MC to oversee microgrid; includes IPs of SMs

Microgrid to Central Unit

Function Name	Description
`request_more`	Request increase in power on regional line
`set_warning_state`	Informs CU if power levels are depleting
`ping`	Lets the MC know communication is still up
`ack`	Acknowledges that the MC is ready to take over new microgrid

13 Sources

13.1 UCRL2

https://umangja.github.io/posts/2021/02/ucrl2/
https://ambujtewari.github.io/stats701-winter2021/slides/UCRL.pdf
https://www.youtube.com/watch?v=0_u-6NrWIu8

14 Math Appendix

constants:

1. LTE/IP header + TCP header = 40 + 20 = 60 bytes

14.1 Machine Learning LTE Overhead

The data in each package generated by a microgrid controller is 48 bytes; 6 models times 8 bytes (long). 60 twice from packet overhead, sending and acknowledgement. 81 regions as there is no model for the medical facility (no one to share power with).

Non-main MC Sends/receives 108/60 bytes per day to main MC.
168 bytes total per day, 5040 bytes per month.

Main MC Symmetric: uses 5040 bytes per month from non-main MC communication.
Sends/receives 108/60 bytes per day to CU.
168 bytes total per day, 5040 per month.

CU Uses 5040 per month for each of the 81 regions, total 413280 bytes or about 400KB.

Model updates are roughly equal in size and put an equal amount of burden on the communication lines.

14.2 Sending logs to Central Utility

Central Utility burden: Each log is $60(\text{header}) + 60(\textit{header} - \textit{ack}) + 40 \cdot 3 \cdot 100 = 12\text{KB}$. 40 as each log is 36 bytes and we're compressing 10 houses, adding 4 bytes to not lose data, 3 logs, and 100 houses per region. 120 bytes in header from sending and receiving acknowledgement. Each log sent to Central Utility is 12KB.

Each region gets 1GB $/82 = 13\text{MB}$ allowance per month for log sending (82 regions). After a bit of time conversion, this works out to 436.5KB per day. Dividing by the 12KB, each region is able to send logs every 40 seconds. This comes out to 64800 logs sent per region (i.e per house).

Main microgrid burden: Main receives 64800 logs from each microgrid. Each log is $60 + 60 + 36 \cdot 3 \cdot 10 = 1200\text{B}$ per microgrid. $1200 \cdot 64800 \cdot 9 = 700\text{MB}$. In addition to communication with Central Utility, main microgrid uses 713MB per month.

14.3 Sending logs to Microgrid Controller

Number of bytes per update to a microgrid controller from residential smart meters: $10\frac{\text{houses}}{\text{microgrid}} \cdot 4\frac{\text{logs}}{\text{house}} \cdot 36\frac{\text{bytes}}{\text{log}} = 1440\frac{\text{bytes}}{\text{update}}$

Number of updates a MC receives a year: $3.154 \cdot 10^7 \frac{\text{seconds}}{\text{year}} \div 15\frac{\text{seconds}}{\text{update}} = 2102666\frac{\text{updates}}{\text{year}}$

Storage used by MC to stores every update from a single microgrid for a year before sending it to CU: $2102666\frac{\text{updates}}{\text{year}} \cdot 1440\frac{\text{bytes}}{\text{update}} = 3027839040\frac{\text{bytes}}{\text{year}} = 3.02783904\frac{\text{GB}}{\text{update}}$

Percentage of its storage capacity used: $3027839040\frac{\text{bytes}}{\text{year}} \div 6.4 \cdot 10^{10}$ bytes (i.e. 64 GB) = 4.7 percent