Power RAnGeRS:[1]

Guardians of Centertown Energy

[1] **Reliable Analytic General Routing System**

1. Introduction and System Priorities

Catastrophes from the Northeast blackout of 2003 to the 2021 Texas power crisis show how the reliability and fault-tolerance of electric grids are integral to modern life, and how breakdowns in its service can directly lead to insurmountable costs of finances and human life. Moreover, these past experiences of blackout and grid-failure strongly indicate that electric grid system designs must account for edge cases and contingency plans. The goal of this document is to describe Power RAnGeRS, a digital management and control system for Centertown's electric utility service.

Power RAnGeRS integrate several hardware and software modules to deliver three major objectives. Firstly, it enables any solar energy system to share excess power with others in the same microgrid (p3) up to the limit set by each serviced building (p5). Second, through the layered modules of the system, microgrids must remain functional if there is a power outage in the larger system (p3). Lastly, the system must provide steady power supply, enable billing for users, and scale to town needs both in the short and long term future (p3).

Our final system design prioritizes the design principles of *reliability, fault-tolerance,* and *performance.* The primary focus of the system is the residents of Centertown, who are its end users. Reliability means that the system must be able to deliver the aforementioned objectives to the town through normal operation, extreme power demands, and expected outage cases such that residents and municipal services will not be without power. This is accomplished through Power RAnGeRS's layered design, which enables the formation of autonomous microgrids. Fault-tolerance means that the system design assumes frequent failures in both our hardware and software modules, and is intentionally redundant to survive numerous failure cases. Lastly, performance means that Power RAnGeRS responds to changes in demand and network connection in a timely manner.

As Power RAnGeRS's primary clients are Centertown residents, scalability is not prioritized in this system design. though Power RAnGeRS accounts for long-term changes in energy consumption patterns for Centertown's growth and development, the system will not scale beyond Centertown.

This paper is structured as follows: Section 2 describes the details about our design decisions for the communication protocol, congestion control mechanisms, power sharing, data compression, and database accuracy & security. Section 3 analyzes how our system deals with system component failures.

Section 4 discusses potential use cases. Lastly, section 5 details quantitative evaluations for network traffic and the overall expected cost of this project.

2. System Description

Power RAnGeRS consist of several layers of hardware and software modules. At the most granular level, Power RAnGeRS utilizes pre-existing smart meters (SM) to record and track basic usage data for the residential house, apartment, and municipal buildings. Each energy-consuming unit (house, apartment, or municipal building) has two SMs — one that measures incoming energy and another that measures outgoing energy. The next layer up includes the microgrid controllers (MC), which receive data collected from SMs via a LTE network. MCs can command the SM to aggregate data and share energy, depending on the specific command from the central utility (CU). Lastly, MCs pass collected SM data to the CU, which sits at the top layer. The CU produces monthly billing statements for the MCs, purchases additional power if needed, and stores data for long term planning. It communicates with MCs for data aggregation and notifies them if a network fault is detected. Figure 1 and Figure 2 illustrates the primary system design, along with major interactions between main modules and components.

Figure 1: The below diagram shows the connectivity of microgrids and central utility both the network and electrical connections

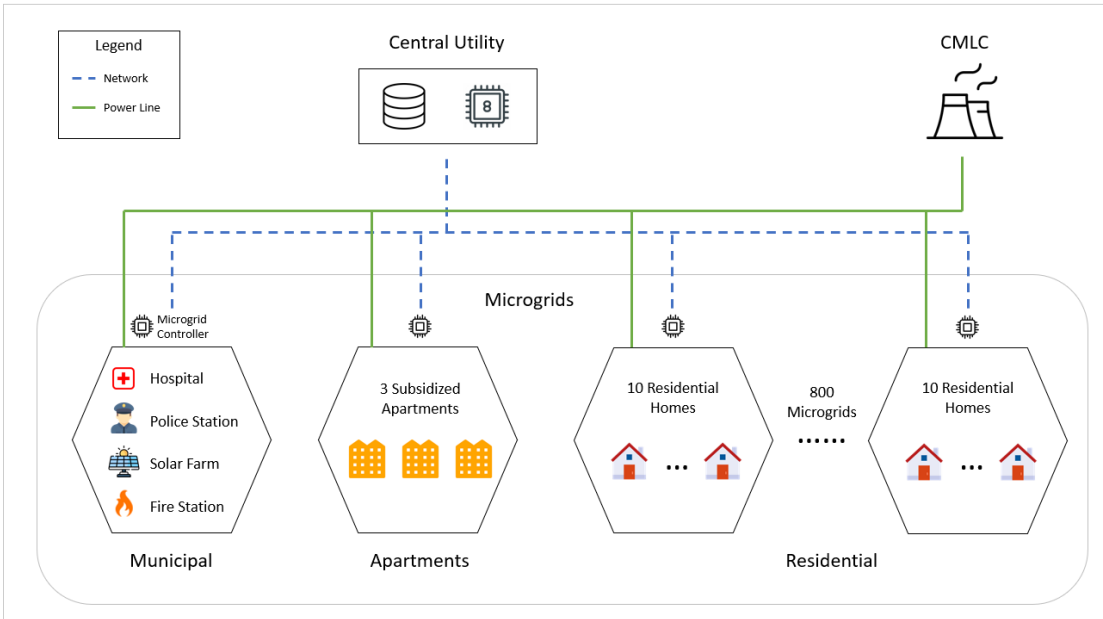
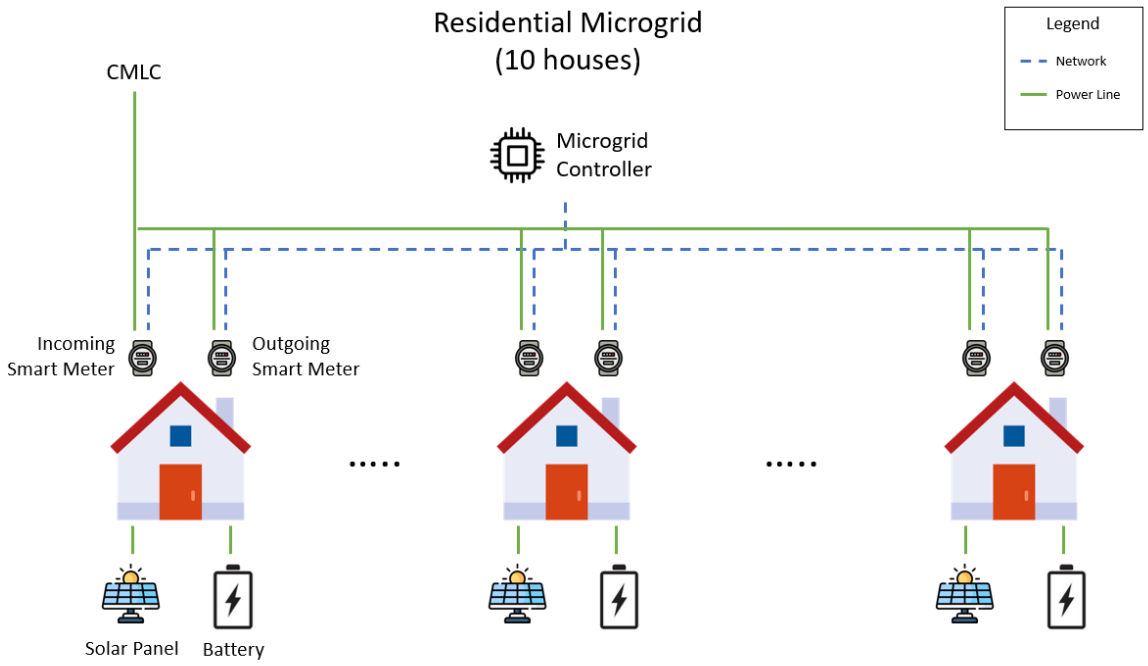


Figure 2: The below diagram shows the connectivity within microgrids - specifically residential microgrids



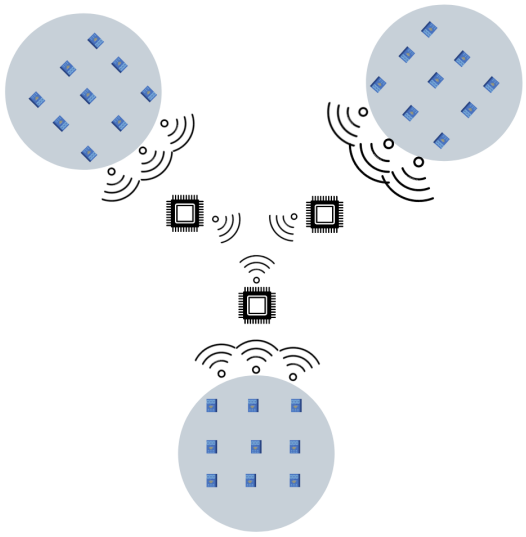
2.1 Design for high availability

As mentioned in the introduction, the goal is to achieve high availability and redundancy to the SMs in order to be able to get the information out to another MC, and this MC will be able to handle the data coming from the SM.

Each SM can take as an input a small set of IP addresses it can forward its data to. We have in total 800 microgrids that have houses, 1 microgrid that has apartments and 1 microgrid that the municipality building are there. We partition the 800 microgrids in groups of 3 and the one group that has only 2 microgrids will get paired up with the microgrid that has only apartments. The microgrid for the municipality buildings is not matched with any other microgrids. We call this group of microgrids a *clique*.

MCs in the same microgrid share the responsibility of handling SMs, in case some other MC within the clique fails. Figure 2 shows in more detail the idea behind the sharing. MCs within the clique ping each other frequently via heartbeat messages. When MC failure is detected, one of the other two MCs is responsible to take over the job of handling its SMs by shortest distance. Neighbors discover when the failed MC recovers when they receive heartbeat messages again from that node. They then stop collecting the data from the failed MC's SMs immediately and let the owner MC handle its SMs.

Figure 3: The below diagram shows the connectivity between MCs within the same clique and their respective SMs (blue are smart meters and black are microgrid controllers).



2.2 Communication Protocol

The goal of our communication protocol reflects the system priority discussed in section 1 and delivers the design goals. To achieve our goals between layers of the network that functions over LTE — a highly constrained environment — we propose a protocol inspired by Constrained Application Protocol (CoAP) over UDP that ensures reliable, secure, and performance-focused communication. UDP is specifically chosen for its low latency, low server-side overhead, and simplicity. However, UDP comes with the necessary tradeoff of packet loss. Moreover, our system model assumes frequent failure and outage in a constrained, distributed, and bandwidth-limited environment. The security guarantees in this proposed protocol improves this shortcoming of UDP and enables the system to deliver key functionality guarantees.

Since performance and data accuracy are prioritized, our proposed protocol ensures secure transport by associating each package with their corresponding acknowledgements and tokens, and achieves packet transmission with bandwidth to spare. Moreover, the acknowledgement system and different packet types defined in the network allows MCs to detect network and hardware failure. This, in turn, enables one of the most important system deliverables: The discovery and automatic formation of autonomous microgrids when the CU is unreachable. Lastly, the communication protocol structure is simple and modular, with specific congestion control mechanisms implementable in very few lines of

code. This modularity ensures that software bugs are unlikely to occur, and that the system is scalable to new functionality updates in the future.

The protocol supports four types of messages: *confirmable, acknowledgement, heartbeat,* and *ping/pong*. Confirmable messages are the typical packets containing user data or command in its payload. Confirmable messages could contain energy consumption information from SMs to MCs, then to the CU. Confirmable messages could also contain commands — aggregate data or power_share_on/off — from the CU to MCs, and from then to SMs. Acknowledgements are sent back after receiving confirmable messages to indicate that a message has been received — without an acknowledgement, the sender retransmits the package within a timeout until an acknowledgement has been received.

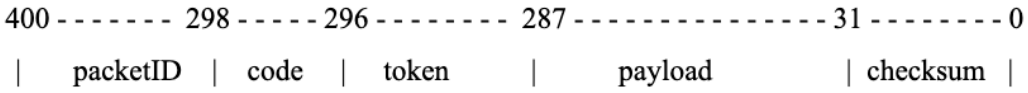
Heartbeat messages are regularly transmitted between all nodes in our network to check network health. If the MC has not received a heartbeat message from the CU after some pre-programmed timeout, it will send a *ping* message to verify the health of the connection. If no *pong* response is received, then the MC assumes failure in either the hardware or network. In response, the MC sends *share_power_on* commands to SMs, enabling the formation of an autonomous microgrid. When power-sharing mode is enabled, MCs will keep sending *pings* until it receives a *pong*, at which point *share_power_off* command is issued to SMs within the microgrid. This effectively disables the microgrid, allowing batteries to recharge and queue data to be sent to the CU.

There are several tradeoffs that accompany this design decision. The most obvious one is between functionality and bandwidth. The extra message types exchanged between nodes in the network impose a greater bandwidth overhead than simply defaulting to only exchanging messages with instructions or payloads. On the other hand, if we discard heartbeat and ping/pong messages, we lose the ability to detect when and where failures occur and resolve in the network. While designs for a more contained and reliable network environment could potentially omit such extra messages, we need the different message types to accommodate for the unreliable network environment and failure detection. As the evaluation will show, however, the extra messages do not exceed the bandwidth limit with optimizations at the SM level. Moreover, UDP allows us to make latency guarantees during normal operation.

Packet Structure

The figure below illustrates specific fields in a packet. *Packet ID* is a concatenation of the sender ID and a counter. *Code* is a 2-bit number ranging from 0 to 3, each mapping to a specific message type. Code 00 indicates Empty type, 01 indicates ping/pong, 10 indicates acknowledgement, and 11 indicates

confirmable. *Token* is a 9-bit number used to compute checksum, and is only included to verify the sender identity for acknowledgement and confirmable message types. Payload contains either the PacketID of the confirmed message in the case of an acknowledgement, or the energy consumption information in the case of a normal, confirmable message. Lastly, the *checksum* field sits at the end for the receiver to verify the identity of the sender.



The token and checksum fields are required because our network model assumes frequent failure and unreliable connection. Failures in the network or hardware could scramble the packet payload even without direct packet loss. Since our system must enable billing, we prioritize the integrity of packet payload. Thus, when a receiver receives a new packet, it recomputes the checksum with the provided token in the packet. If the recomputed checksum equals the checksum sent over by the packet, the receiver knows that the packet payload is intact and sends back an acknowledgement to confirm the reception. On the other hand, if the checksum does not compute, the receiver withholds the acknowledgement, and the sender must retransmit until the checksum computes.

This design choice makes a tradeoff, once again, between bandwidth and functionality. Specifically, the functionality prioritizes billing and command accuracy, which impact our various stakeholders differently. While residents and taxpayers may be indifferent to loss of billing information, the municipal government certainly cares about accurate billing. Researchers and Centertown residents generally would prefer accurate data collection to better accommodate future energy uses. On the other hand, all Centertown energy consumers prioritize accurate microgrid activation commands, as microgrids must function independently in case the CU is cut off from the network. Overall, this breakdown of interests shows that accuracy means different things for different stakeholders, and that some definitions are more important than others for different groups. Ultimately, however, the checksum's accuracy guarantee is needed to ensure that power sharing commands' integrity is maintained, so the key system deliverable could be met.

Congestion Control

Due to the numerous message types flowing in our network, congestion control mechanisms are critical in maintaining system performance and reliability. All nodes in our network send and receive in set time stamps. In contrast to usage patterns from video streaming, bursts from intended usage is highly unlikely. However, the network is also unreliable and prone to packet drops. As a result, our model

assumes that congestion occurs when nodes accrue a long queue from spotty connection and attempt to send at the same time when connection recovers. Moreover, our system deliverable and priorities indicate that microgrid instructions in critical Centraltown utility services should be prioritized: In the event of an unreachable CU and limited bandwidth, the formation of the municipal microgrid should be prioritized over that of the residential microgrids. Because we require both bandwidth-based scheduling and weight balancing depending on node identity, we choose *weighted round robin* as the congestion control mechanism.

To implement *weighted round robin,* we model each MC as a sender with at most 64GB of queue space. In each round, a sender node accrues some $quantum$ and send at most that many bytes of packets in the queue. In the case of the MC responsible for critical utilities, the CU calculates some $\alpha > 1$ such that each round, that MC accrues α * $quantum$ credits instead. The code implementation for a single sender node looks as follows in each round:

```
for each microgrid controller MC:
    MC.credit += MC.quantum
    while MC.credit >= size of next packet p:
        MC.credit = MC.credit - size(p)
        send(p)
```

Besides delivering system requirements, the most obvious benefit to this congestion control scheme is that the implementation is extremely simple and can be completed in very few lines of code. Commercial software typically contains 20-30 bugs in every 1000 lines of code, and the most probable source of failure in our software modules is the software itself. For Centertown's end users, the modularity and simplicity of this scheme means that software bugs are very unlikely to occur, and that the system is simple to implement. Moreover, the modular design scales well for future developments in Centertown — Be it adding more MCs, or changing microgrid layout, or fine-tuning weight priority for each sender node.

Below are key benefits to the chosen communication and congestion control protocol:

Reliability: A key merit of this network protocol is data accuracy, overcoming assumed packet loss. In the event that a packet is dropped by the network, the sender will retransmit that packet until it

receives an acknowledgement. Moreover, in the case that the CU is disconnected from the network, microgrids will form and supply energy to Centertown. As soon as the CU is live again and *ping/pong* requests are sent, microgrids will reconnect and normal function resumes.

Performance: The system bottlenecks are the network health and the manually configured timeout limits under which packets are retransmitted. Although there exists a necessary tradeoff between minimizing packet loss and performance, we assume that energy supply will not be cut off within the timeout. Moreover, our system accommodates for long-term growth and improvement, and assumes that ttl calculations will improve as more data is gathered over time.

Security: Security is maintained by the system because of the ID, token, and checksum fields supported by the protocol. In the case of an unauthorized connection that intercepts the network, packets with unrecognized IDs will be automatically dropped. Moreover, if messages are manipulated during transmission, checksum will not equal the computed sum by the sent token — in that case, the packet will also be dropped.

2.3 Power Sharing & Purchasing Regional Power

This section will discuss how our system handles power sharing and the conditions under which it will decide to purchase regional power. Our macro energy decision-making module prioritizes reliability and performance. We want to ensure that consumers' requests for energy are received and processed swiftly to distribute power where it is needed in an affordable manner.

We make the assumption that the price rate for energy from the regional utility is always static, and that the non-peak hour rate is lower than the peak hour rate. It was mentioned earlier that MCs regularly send GET requests to SMs within its cluster. Part of the response is the battery level of the client. The MC can then use this information to decide whether to turn on power sharing.

Our power sharing implementation ensures that units with battery levels relatively greater than their set minima share as much of their excess power as possible before other houses start sharing. As an example, suppose there are three houses who have all set their minimum threshold to be 50% and H_1 is at 40%, H_2 at 53%, and H_3 at 55%. Under our power sharing algorithm, H_3 would share all its excess to H_1 before H_2 even begins sharing, such that the final power levels would be 48%, 50%, and 50% respectively. In the case that there are multiple houses beneath their minima, the excess power will be

distributed evenly until the unit that is sharing is at its threshold, in which case the next unit with the greater amount of excess power will begin sharing. This algorithm applies to residential and apartment microgrids. The municipal microgrid only has one battery, so it can only be recharged using energy from the CU, which will be discussed next.

The CU does not have any information about a microgrid's overall battery level. It only knows to send power to the microgrid when the respective controller requests for power. Part of the header of the packet is the request's origin, i.e. which microgrid generated that request. This is how the CU will know which microgrid its decisions will affect. Our system does not give priority to one microgrid over another if they are of the same type — the types being residential, apartment, or municipal. However, it does enforce the priority between types as specified in the specs: the municipal buildings have the highest priority, followed by the apartment, and lastly, the residential.

Beyond the microgrid, our power sharing works as follows. During non-peak hours, the CU will purchase power up until its battery reaches a level of 75% to minimize costs. Not charging up the CU's battery to full leaves some room for clients to send excess power, which will be discussed shortly. 75% is an arbitrary threshold to start but should be adjusted accordingly as more data is collected about Centertown's behavior and how much excess energy consumers are expected to supply back to the system. During peak hours, the CU will only distribute power from its battery if its level is greater than 25%. Otherwise, it will purchase just enough power from the regional utility to meet the demands of the microgrids and charge accordingly. The remaining amount is only to be distributed during emergencies, such as when the regional utility is down, and the microgrids are making requests for energy. Similar as before, 25% is arbitrary and should be adjusted as more data about how much energy is expected to be consumed during a worst-case disaster scenario.

When a microgrid has excess power and shares it back to the CU, then the microgrid will be given credit to be split evenly among its units. The credit will total to the same cost as if the same amount of power was purchased from the regional utility the same hour during which the energy was shared back to the CU. On the other hand, if a microgrid requests energy to meet its demands, then the microgrid will receive energy from the CU battery and be charged with the same rate as if purchased from the regional utility at that hour. Earlier, we mentioned the levels of priority of the different types of microgrids when distributing energy. We enforce that in the requests for power. If there is an outstanding request for energy of a microgrid with higher priority, then power will be distributed to said microgrid until it has met its

minimum threshold before responding to other requests. If there are two microgrids of the same type requesting power, then the power will be distributed evenly.

For example, for simplicity, assume that there are three requests, one each from the municipal, an apartment, and a residential microgrid, whose thresholds are all set at 75% with battery levels indicating some level below that value. Then the municipal microgrid will be the only microgrid receiving power. Once its battery level reaches 75%, then power will be distributed only to the apartment microgrid until it reaches 75%. The same idea extends to the residential microgrid. If instead all three requests were from distinct residential microgrids, then power will be distributed evenly to all three simultaneously until at least one reaches its threshold, at which point power will then be distributed evenly to the remaining two. Requests from a higher priority microgrid will always interrupt requests from a lower priority microgrid.

2.4 Data Compression

We noticed that there is a lot of unused space within the packages we send. For instance, 8 bits are used for only a variable that is 1 bit. Therefore, in this subsection, we focus on compressing the data to be able to increase the maximum number of packages that can go through our network.

We're gathering two types of data from SMs event records and history records. In order to maximize the number of records and improve latency we can utilize data compression. This utilization helps us achieve one of our main design principles: *performance* and by making low level optimizations to the data we're sending we're making a lot more room for the higher level modules to utilize more resources.. First we can start by talking about history records.

1. Lossy_aggregation: 8 bits => 1 bit. This entry tells us whether we're using lossy aggregation or not. We can use a 0 to say we're not using it and 1 to say we're using lossy aggregation.
2. Meter_id: 64 bits => 16 bits. We know that there are 8000 residential houses and we estimate that at most we'll have two SMs per house. This means that we need at most 16 bits to represent the different SM IDs since there are definitely less than 2^{16} SMs.
3. Start Time: 32 bits => 28 bits. We can use the standard time representation which needs 32 bits and then trim off a couple of bits for the milliseconds, starting at 1970 and supporting only the next 200 years, since by then other systems will be in place. This saves us 4 bits on start time.
4. Delta: this field will be added to the records to indicate the length of the power usage so it would be start_time - end_time. Since we know that every minute the SMs report their data then we will need at most 6 bits to store the length of time that we used power for since we know that the delta

can be at most 60 seconds so 2^6 is more than enough for that field. This field will replace end_time since we have a delta and a start time.

5. Records Type: 8 bits => 4 bits we can save also another bits on record types since there are a lot less than 2^4 record types and we get that bit to complete our bytes so that we save overall 13 bytes

Secondly, let's talk about event records.

1. Previous & New State: 16 bits => 4 bits this entry tells us the current state of the SM. We need only 2 bits for shutdown, initialize or normal, 1 bit for whether we're connected to the power net, and 1 bit for whether we're aggregating data or not so overall 4 bits
2. Meter_id: 64 bits => 16 bits same idea as above for history records
3. Time: 32 bits => 28 bits same as above for start time and end time.

We can see that overall we save a lot of bytes roughly 36% (36 => 26 bytes) of the data which helps us store a lot more data to be used in both research and billing.

History Records

Field name	**Fields size before compression**	**Field size after compression**
record_type	8 bits	4 bits
lossy_aggregation	8 bits	1 bit
record_id	64 bits	64 bits
meter_id	64 bits	16 bits
account_number	16 bits	16 bits
start_time	32 bits	28 bits
delta	32 bits	6 bits
Reading	64 bits	64 bits
Total:	36 bytes	23 bytes

Event Record

Field name	Fields size before compression	Field size after compression
previous_state	8 bits	4 bits
new_state	8 bits	4 bits
record_id	64 bits	64 bits
meter_id	64 bits	16 bits
time	32 bits	28 bits
Total:	24 bytes	17 bytes

2.5 Data Accuracy & Security

Our system design models the CU as both a *coordinator* and a *database*. This section expands the specific challenges and workarounds regarding the CU as a database in a network constrained, failure-prone, concurrent system. Specifically, this section will introduce the design, then extrapolate challenges arising from these design choices, then explain how the design overcomes such challenges.

Data Accuracy

In the typical system use case, we can model each MC as a user that will try to write and commit to the CU's data storage in predictable, periodic increments. We have 802 MC database users in total — 1 for the municipal utilities, 1 for the apartment, and 1 per every 10 houses. In the most high-demand scenario with no failures, all of these database users will attempt to write to the database simultaneously. If the CU enforced locking on database writes, there would be an incredibly large latency overhead, especially if the batch is large. The implications of high latency overhead is that our system components may not detect failures in time, and the end users' energy demand would not be sufficiently met. As a result, the CU's data storage must enable locking on MC ID, rather than locking on write. Moreover, the database implementation should honor ACID principles of atomicity, consistency, isolation, and durability to deliver the system objectives of enabling energy distribution, data collection, and billing.

With a general-purpose database implementation, it may not be sufficient to allocate locks on unique users due to the potential of lock contention, long term blocking, and deadlock. Locks outlive statements, which means that a transaction may hold lock on a previous statement, even though it has moved on to subsequent actions. This prevents another thread from accessing the lock-protected field, thus imposing additional latency to transactions. Although the delay may be small for limited users and

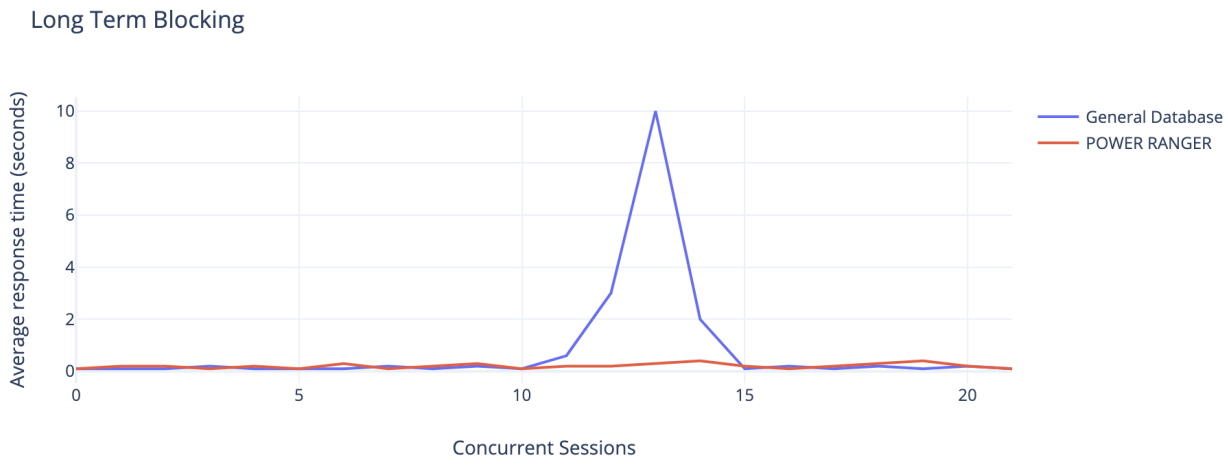
small batches, it builds up unpredictably and exponentially as the system scales. Worse still, locking on specific data fields may result in deadlock in our implementation of bundled microgrids.

Under normal functions of our system, however, *MCs will only write to their own fields*. Even in case of data-sharing cliques, each MC may not have a determined set of smart meters, but they will only write as their own MC ID. The table below illustrates the data fields that could be stored in the CU.

Lock here

Timestampe	SM ID	MC Sender ID	MC Owner ID	Payload
xxxxxxxx	xxxxxx	xxxxxxxxxxx	xxxxxxxxxxxx	xxxxxxxxxxxxxx
xxxxxxxx	xxxxxx	xxxxxxxxxxx	xxxxxxxxxxxx	xxxxxxxxxxxxxx

Where MC Sender ID is the MC ID that delivers the packet to the CU, which is the field that we will impose locking on. MC Owner ID is the MC ID that the smart meter originally belongs to. This field is necessary for research and power-usage prediction. Lastly, the payload is where compressed logs from smart meters are stored. The figure below illustrates potential issues with latency and lock-blocking as a general-purpose system scales:



As a result, we will usually never encounter a situation where MC_A holds MC_B's lock, and vice versa. Moreover, latency overheads from batch writes can only impact a single thread, which is the thread belonging to the MC that holds the lock on its own ID. As a result, usual concerns of deadlock, lock contention, and blocking are generally avoided in our system because of the constrained number of users. On the other hand, while this scheme sufficiently addresses concurrency concerns with trusted actors, it

does not secure our system from *malicious attackers* who could easily impersonate microgrids in our system.

Security

The UDP network that carries all communication in our system is notoriously susceptible to spoofing and DOS attacks. An attacker can easily tap into our network and read MC IDs to impersonate known MCs in our system. Once knowing multiple valid MC IDs, the attacker can then overwhelm the CU by issuing numerous writes with garbage data, thereby starving the legitimate requests from being processed and executed. Under this threat model, then, we must verify both the validity of the MC ID and the request. As a result, the CU must allocate some processing resources to determine what requests are valid and what are not, and drop the garbage as to not starve resources from other valid requests.

To safeguard the security of our database and service, we implement a simple symmetric-key encryption algorithm between MC nodes and the CU. At system-setup time, all MCs and the CU exchange their respective private keys to be used in the encryption and decryption process. Before sending any data of any type, the sender encrypts the message with its private key. The receiver then decrypts the message to perform identity verification and checksum. With this scheme, our data in transit will always be encrypted, and actors in our threat model will never be able to impersonate any nodes in our system, because message content will never exist in plaintext.

There are several tradeoffs in our implementation of both data storage and security guarantees. Firstly, the use of locks on MC ID imposes a tradeoff between scalability and functionality. The number of locks on our database represent the maximum number of threads that the CU supports at a given time. However, if Centertown one day quadruples in size, the CU may not have enough processing power to support this locking scheme. If we do not impose locking on MC ID, however, we would violate ACID properties and the accuracy of stored data — sender A could overwrite sender B's information in the system, resulting in incorrect energy usage and billing information.

Moreover, the encryption scheme imposes an additional tradeoff between performance and security. Though symmetric-key cryptography is relatively lightweight in comparison to other encryption mechanisms, encryption and decrypting the message with every send and receive will impose performance overhead on nodes in our system, especially the MCs with extremely limited processing power. As a result, we implement the cryptographic module in the *hardware* of both MCs and CU to optimize for performance. Although this decision will impose additional financial burden on Centertown,

we believe that the cost of security guarantee greatly outweighs the cost of recovering from a potential ransomware or DDos attack.

3. Fault Tolerance

During the design of our system for the town, we learned that one of the main hardware heroes of our system, the MCs, are quite unreliable and fail pretty often. We also learned that the CU and the SMs are also prone to failures. This gave another dimension to our problem we did not anticipate, however, we decided to act upon it using our main principles.

When the Microgrid fails, the SMs are not able to receive instructions on when to power share and are unable to share and receive excess energy. However, the customers will never be without any energy, since the SM, whenever the energy in the battery goes below the threshold set by the user, will start taking energy from the CU.

Even though, as explained above, the MC failure is not fatal and users can still consume energy, we decided, as one of the goals of our system is to be highly available for our customers, to be reliable and to be efficient with our power and share it when needed to prevent unnecessary purchases from the CU, to add extra complexity in our system to handle these failures. This will not only optimize energy consumption within the microgrids, but it will also reduce the waste from the non renewable waste produced by the main factory. The drawback is that, because of this extra complexity in the system, MIT Researchers will be getting lossy reports from the SMs.

In this section we analyze the system's response to specific failures of the components of the system. Note that we can have 3 different failures: MC, SM and CU failure.

Microgrid Controller Failure

As mentioned in section 4.1, we have designed our system to be redundant when it comes to MC failures. When an MC fails, the other MCs will get to know almost immediately due to the frequent pinging between the MCs. The SMs are broadcasting the data to all 3 MCs, although only one MC is processing the data at a time. As long as the MC is alive, it handles all the microgrid's SM traffic. When it fails, one of the other two MCs takes over. One of the other two MCs has higher priority than the other so which MC takes over is always deterministic (Figure 3)

Figure 3: The below table shows which MC is in charge of which Microgrid in all the cases of failures

MC1	MC2	MC3
M1	M2	M3
✈	M1, M2	M3
M1	✈	M2, M3
M1,M3	M2	✈
✈	✈	M1,M2, M3
✈	M1,M2, M3	✈
M1,M2, M3	✈	✈

Section 6 analyzes the network traffic effect this design choice has on the system.

Central Utility & Smart Meter Failure

The CU is responsible for providing energy to the residents of the house through the power plant. When the CU fails, there is nothing we can do in terms of redundancy. MCs just keep pinging the CU and keep storing all the information from the microgrids to send it to the CU once it is back up, as potentially the SMs will potentially keep getting energy from the power plant, even though the CU is down.

There is not much complexity to the SMs failures. The SMs fail for a very short period of time (10-15 seconds), which is even smaller than the time they take to send packets to the microgrids. As they are expected to fail pretty often, their failures are handled in the MCs logic.

4. Use Cases

In this section we analyze different scenarios that can take place in our system and how the system reacts to them.

Sunny day in Centertown

When it is a sufficiently sunny day in Centertown, houses and apartments are going to be living with the energy they themselves produce. The solar panels are going to be producing enough energy for the houses and everyone will have energy without the need to buy from the central utility. This achieves one of the main goals of our system: to save money for our customers

Two MCs in the house clique fail

When 2 MCs fail, the third MC is taking over for both of them. Now three times as much data has to be handled in that microgrid. This becomes a problem when we are trying to send data to the Central Utility. Since we use the weighted round robin to determine which MC sends how many packages, the extra 'work' that has been added will be smoothed away by that procedure. Eventually the other MCs will come online again and the MC will be relieved from handling all the smart meters.

5. Performance Evaluation

In this section we will provide some evaluation of our system. We go through quantitative analysis of metrics such as network traffic and cost which we think are the most important.

5.1 Network Traffic Calculations

One of the most important factors that affected the design of our system is the network traffic between the components of our system. Our employers constrained us into using a specific plan for the SMs and MCs. Specifically we can only spend 2 GB for communications between components per month. The MC sits in the middle of the SMs and CU and most of the network traffic goes through it. As discussed in section 4.1, we have 3 different types of cliques: the Houses cliques, that consist of houses microgrids, the apartment-house clique that consists of 2 houses microgrids and the apartment microgrid and lastly we have alone the municipality clique. The microgrids in these cliques have different traffic going through them so we analyze them independently. Overall, we have 3 different types of traffic going through the MC:

- Heartbeats / pings between the MC and other MCs in the clique or the CU. In our system we need to know which components have failed or not so we ping frequently (every 5 seconds)
- Smart meter data: this data is essentially power that goes through the SMs, power stored, power generated and a history log if it is an 'outgoing' SM, and power through the SM and history log if it is an 'incoming' SM.
- Smart meter data from MC to CU. The data that was sent to the microgrid has to be sent to the CU for MIT Researchers to process.
- Lastly, we have some messages that are being sent between components for example when the MC tells the SMs to power share. These generally happen pretty rarely in comparison with the other ones.

We are going to provide calculations for only the house clique parts, as the other cliques have very similar calculations.

Note that because we use the protocol introduced in section 3, we have an overhead of around ~50 bytes. Therefore in order to ping, or send an empty package, we need 50 bytes. If we send a ping every x_{ping} minutes, then we have:

$$\frac{50\ bytes}{x_{ping}\ mins} = \frac{50 \cdot 60 \cdot 24 \cdot 30}{x_{ping}} bytes/month = \frac{2.16}{x_{ping}}\ MB/\text{month}$$

As we have heartbeats between Microgrids and CU:

$$D_{ping} = 6 \cdot \frac{2.16}{x_{ping}} MB/month = \frac{13}{x_{ping}}\ MB/month$$

Note that in section 2.4, we reduced the size of the event and history logs from 24 to 17 bytes and from 36 to 23 bytes. Therefore, for the data from the SMs to the MC, we have:

- 10 'outgoing' SMs sending packaged 2 packages of sizes 24 bytes and 36 bytes every x_{SM} minutes.
- 10 'incoming' SMs that send 4 packages; 3 of size 36 and one of size 24 every x_{SM} minutes.

Note that we also have the extra padding that adds 50 bytes, therefore we have:

$$\frac{[3 \cdot (50 + 23) + 1 \cdot (17 + 50) + 1 \cdot (50 + 23) + 1 \cdot (17 + 50)]bytes}{x_{SM}\ mins} = \frac{19}{x_{SM}}\ MB/month$$

for each house. Therefore for the whole microgrid:

$$D_{MC-SM} = 30 \cdot \frac{22}{x_{SM}} MB/month = \frac{552}{x_{SM}} MB/month$$

Because remember we have 30 houses broadcasting to each microgrid instead of 10. But the microgrid will only have to send to the CU only a third of what it receives (on average), therefore:

$$D_{MC-CU} = \frac{D_{MC-SM}}{3} = \frac{184}{x_{SM}} MB/month$$

Lastly, we left extra space for S extra packages every day, for power sharing, aggregating etc. (of around 100 bytes) between the MC and all the SMs and CU. This costs:

$$D_{extra} = 31 \cdot [(100\ bytes) \cdot \frac{S}{day}] = 93 \cdot S\ KB/month$$

In total we have:

$$D_{total} = D_{extra} + D_{MC-CU} + D_{MC-SM} + D_{ping} = (\frac{552}{x_{SM}} + \frac{184}{x_{SM}} + 0.093\,S + \frac{13}{x_{ping}})\,MB/month$$

And we want that to be less than 2000. It remains to tune the parameters and we decide to choose $x_{SM} = 0.5$, $x_{ping} = \frac{1}{12}$ and S = 1000. Note that for the houses we get data every 15 seconds, which is almost optimal (the best being every 15 seconds) and yields the results in the table below:

	House cliques	Apt-house clique	Municipality clique
Pings (MC-MC, MC-CU)	156 MB (pings every 5 secs)	156 MB (pings every 5 seconds)	52 MB (every 5 secs, only MC-CU)
MC - SM	1104 MB (pings every 30 secs)	1126 MB (every 1 min)	85 MB (every 15 secs)
MC - CU	368 MB	375 MB	85 MB
Power sharing, aggregate etc.	93 MB	63 MB	93 MB
Total	1.51 GB / month	1.72 GB/month	0.315 GB / month

5.2 Expected Cost

We can divide the cost into two parts: the cost that comes with the hardware and generally devices, and the cost that comes with the humanpower that will need to be paid in order to make this system possible. As far as the first part goes, we have in total 802 microgrids and 16304 SMs that each have their one plan. We did not buy any extra plans even though we had the opportunity to do so. Since each plan costs 10$, these plans will cost in total **$171,060 per month**.

Next we have to estimate the amount of personnel we will need to hire in order to code up our system. We essentially need people for several independent parts:

- Data Engineer: Receiving packages from SMs/CU/other MCs and storing it appropriately in the microgrid's device. Also has the role padding the data and forwarding them to the CU.
- Software engineer: implements all the procedures that are required of the MC such as handling the data in the storage, sending orders to SMs etc.
- Algorithm Engineer: researches and applies different strategies of sharing energy of all microgrids, as well as energy of the city, in order to optimize and save money to customers.

We expect that in order to be able to implement our idea, we will need at least 2 engineers from each category. We expect to pay a salary around **$5,000 per month** to each of them. The final cost of the

system is around **$200,000 per month**. As there are 8300 users it will be around **$24 per month,** which we think is reasonable.

6. Conclusion

Power RAnGeRS enables excess solar energy sharing with neighbors in the same microgrid, keeps these microgrids running in case of outages and provides steady power supplies while billing users who are mostly centertown residents. The layered system delivers on its three primary goals while prioritizing its design principles. An important part of the system is its resilience and adaptability in unique situations where there is a lack of energy from the CU, downed power lines due to storms, or permanent changes in consumer behavior such as switching to online working environments.

7. Author Contributions

In this DP report, Angelos Assos wrote sections 2.1, 3, 4, and 5. Gary Nguyen wrote 2.3, 2.4, 4, and the system diagrams. Mahmoud Khalifa wrote section 2.4, the conclusion, editing, and formatting, while Yichuan Shi wrote the introduction, 2.2, and 2.5. All team members participated in the design formulation and consensus process.

8. Bibliography and Acknowledgement

Bormann, Carsten, Simon Lemay, Hannes Tschofenig, Klaus Hartke, and Bilhanan Silverajan. “CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets.” Edited by Brian Raymor. Internet Engineering Task Force, December 18, 2017. https://tools.ietf.org/id/draft-ietf-core-coap-tcp-tls-11.html

We also benefited greatly from 6.033 office hour sections with the teaching staff.