Secure and Scalable Communications Protocol for Data Collection with Time Minimization in the Smart Grid

Suleyman Uludag, King-Shan Lui, Wenyu Ren, and Klara Nahrstedt

1Department of Computer Science, University of Michigan - Flint, MI, USA
2Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong
3Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Abstract—Deployment of data generation devices, such as sensors and smart meters, has been accelerating towards the vision of Smart Grid. With insufficiencies of the legacy power grid communications protocols, increased data generation and communications bring about new challenges in collecting the data securely, efficiently and in a scalable fashion. Power operators are increasingly concerned about the data collection and processing to ensure smooth and effective operations. In this paper, we present a secure, and scalable data communications protocol for Smart Grid data collection. Under a hierarchical architecture, relay nodes (aka data collectors) collect and convey the data securely from measurement devices to the power operator. While the data collectors can verify the integrity, they are not given access to the content, which may pave the way for third party providers to deliver value-added services or even the data collection itself. We further present optimization solutions for minimizing the total data collection time. An optimal, but intractable solution is complemented with two fast heuristics. Extensive simulations show that heuristics provide a good approximation to the optimal formulation with fast running times.

I. INTRODUCTION

In the Smart Grid, massive number of sensors or measurement devices will be installed to collect real-time information for monitoring. The generated data should be collected in a secure and scalable manner. A hierarchical data collection framework is usually adopted to make data collection scalable. For example, in Advanced Meter Infrastructure (AMI), smart meters first report data to data concentrators [1]–[3]. Data concentrators can then pre-process the data, if necessary, before reporting to the power operator. In this way, the power operator does not have to maintain a separate connection with each smart meter, which is extremely expensive. Besides, data concentrators can aggregate the data, reported by the smart meters, to further reduce the message size. Apart from data collection, this hierarchical communication structure should also allow a delivery of a command or an instruction, issued by the power operator to be delivered to a meter or measurement device in an efficient and secure manner. In this paper, we develop a comprehensive protocol that allows a power operator to collect data, as well as send commands to measurement devices in a secure, scalable, and efficient manner under a hierarchical data collection model.

Fig. 1 presents the data collection architecture considered in this paper. The Measurement Devices (MDs) are sensors or smart meters that generate power-grid specific data. They are small telemetric devices and computationally constrained. Each MD is connected to at least one Data Collector (DC), and each DC may connect to multiple MDs. The Power Operator (PO) has a direct connection with each DC. PO and DCs are relatively more powerful than MDs. The data are reported to Power Operator (PO) via a set of Data Collectors (DCs). PO may also issue commands to the MDs via the DCs. Theoretically, a DC is trustworthy if it is within the security domain of the PO.

However, due to the massive number of MDs and their dispersion over a large area, it may not be appropriate to assume DCs can be completely trusted. In addition, one of the seven actors identified by the NIST in the SG Framework [4] is third party service providers which are to furnish third party value-added services. We assume honest-but-curious model for DCs. Thus, the data collection tasks may be outsourced to third party service providers [5]. Besides, the benefits of cloud computing [6] may be accrued for storage and processing of the data collected. Data sharing to others to provide services like energy management services can be facilitated as well.

In some other applications [7], DCs are mobile and the connections between DCs and MDs are dynamic. Therefore, it would be desirable for MDs to encrypt their data in a way that DCs do not have access to them. In other words, each MD should encrypt its data using an appropriate key to keep its data private to DCs and other possible adversaries. On the
other hand, due to limitation in memory and computational capability, the encryption algorithm used should be efficient. PO should also protect its commands appropriately. Apart from ensuring the security of these commands, it is also crucial to deliver these commands promptly because fast actions of MDs are necessary to maintain the stability and health of the smart grid. In this work, we develop a customized key establishment scheme and data collection protocol to protect the data and commands sent between PO and MDs via DCs in a scalable and efficient manner. In particular, our protocol has the following features:

1) Data reported by a certain MD can be accessed by the PO only, even the message is transported by a DC.
2) The protocol is light-weight in the sense that MDs do not have to perform expensive operations to report data and it does not take a lot of memory to remember key information.
3) The protocol allows commands and urgent data to be delivered promptly and securely.

One or more DCs can be designated to collect data from a certain MD. However, the cost, the delay, and the security of data collection may differ among different DCs. The PO should select DCs according to the performance requirements. Different optimization objectives can be developed. In this paper, we study the time needed for the PO to collect all data from the MDs via the DCs. We study how to assign MDs to DCs such that the time of data collection can be minimized.

The rest of the paper is organized as follows: Section II describes existing efforts on data collection in smart grids. We provide the system and protocol overview in Section III. The details of the protocol are described in Sections IV and V. In Section VI, we analyze the time performance of our mechanism and present the DC-MD assignment problem as an optimization problem. We also propose two heuristic methods to solve the NP-hard assignment problem. We then evaluate the performance of the heuristic solutions in Section VII. We conclude our paper in Section VIII.

II. RELATED WORKS

Security and privacy issues in the Smart Grid are presented in [8]. Data integrity and confidentiality are the major security concerns. End-to-end data protection has been studied extensively in the Internet. However, most schemes, such as TLS [9], assume the devices have abundant memory and computational power to perform expensive cryptographic operations. In smart grids, on the other hand, reporting devices have limited memory with a slow CPU. Traditional Internet security protocols are thus not suitable for data collection in smart grids. DNP3 [10] is a standard communication protocol used in SCADA (Supervisory Control And Data Acquisition). It assumes all components are within the security perimeter of the operator and is not designed to protect data forwarded by the DC as in our situation. A more recent standard for substation automation is the IP-based IEC 61850 [11]. Yet, IEC 61850 was also initially designed without security mechanisms [12]. It is thus generally agreed by the experts that new security protocols for data collection and command delivery need to be developed.

Some efforts have been put on key management of smart meters and sensors in smart grid. [13] describes a key management scheme for secure communication in smart grid. The scheme develops keys for unicast, multicast, and broadcast. Nodes are arranged as binary trees and the secret key of parent is the hash of the children keys. How different parties process or encrypt the data is not discussed. It is not clear whether the data reported by a certain meter can be hidden from the data collector. [14] also considers the key management problem for a massive number of smart meters. Key graph is used to manage unicast, multicast, and broadcast keys. Nevertheless, as DC is not considered in the architecture, the key management scheme cannot be applied in our scenario. The SAKE protocol [15] allows two neighboring sensor nodes to establish keys using hash chains. However, the authors assume the attackers are of limited computational capability as another sensor. The authors in [16] apply the elliptic curve public key technique to perform key management. Mutual authentication between different entities is studied. Nevertheless, there is no discussion on how to protect the data reported by a sensor.

Some protocols have been developed to establish shared keys when the two parties can establish direct communication. [17] describes how to establish keys and secure unicast and multicast communications. The authors suggest keys to be established by direct connection between the two entities that need shared keys. [18] describes how to apply the Diffie-Hellman mechanism to establish a shared key for data authentication between two parties. [19], on the other hand, relies on identity-based cryptography. All these mechanisms cannot be applied in the hierarchical data collection model because the PO and the MDs cannot establish a direct connection. The authors in [20] describe how a device establishes shared keys with different controllers at different hierarchical levels. However, it is assumed that a shared key exists between two adjacent controllers.

Some efforts have been put in studying the transport protocol for data collection among a massive number of MDs. [21] studies how to reduce the storage needed when the control center needs to establish multiple sessions with the MDs. MDs are configured with a long-term shared key with the control center upon manufacturing. A function is used to generate this key. Thus, the control center does not have to remember a lot of keys but can derive the key when needed. Nevertheless, the key developed this way is not very secure. Besides, the protocol is not suitable for the hierarchical data collection architecture. Data collection through a data collector is considered in [22]. The authors propose to maintain two separate TCP connections: one between the control center and the data collector, and another between the MD and the data collector. The two connections can be protected using different mechanisms independently. Nevertheless, the data collector is assumed to be trustworthy that it can read the data sent by the MD.

[23] studies how data generators report data to a honest-but-
curious storage center for a user to retrieve later. To the best of our knowledge, the data collection trust model assumed in this paper is the most related to our scenario. The storage center is similar to the DC in our model that it is semi-trusted, and data should be hidden from it. MDs in our model are the data generators, while PO is a user in their model. However, the paper suggests to use expensive identity-based and public key encryption to protect data to incorporate policy consideration. The experimental computational time for a decryption on a message of size less than 1000 bits in a low-end smart meter (TinyPBC library on a 32-bit ARM XScale PXA271 processor) is around 140ms, while the encryption is supposed to be a few times more expensive. Our protocol, on the other hand, encrypts data using the much more light-weighted symmetric key cryptography. We also perform experiments to study the time performance of our mechanism.

As for the security of the data collection task, there are two major approaches: One is to ensure the protection of the data content directly without regard to the data semantics. An approach presented in [24] is based on symmetric cryptography to provide data confidentiality and authentication between sensors and the base station. [25] describes a protocol for DC to collect data from an MD, but direct communication between DC and MD is assumed. Another category for providing security exploits the aggregate statistics of the sensed data, such as summation, average, minimum, maximum, etc. These approaches take advantage of in-network data processing (also referred to as aggregation) to induce some obfuscating operations on the transmitted data [3], [26]–[33]. Examples of this category include cluster-based data aggregation [26] and its integrity enhanced version [27], secret perturbation [28], k-indistinguishable privacy-preserving data aggregation [29], a centralized authentication server based on-network aggregation for AMI [30], [31], homomorphic encryption-based aggregation [32], a secure architecture for distributed aggregation of additive data [3], and a network coding-based encryption between smart meters and aggregators [33]. Our problem formulation does not assume any statistical property for in-network processing.

None of the papers mentioned above studies how to assign MDs to DCs. We formulate this assignment problem, provide an optimal ILP formulation, show its complexity as NP-Hard, and then develop two fast heuristic algorithms.

III. SYSTEM AND PROTOCOL OVERVIEW

A. Operations and their requirements

As mentioned in Section I, our communication architecture supports MDs to report data and PO to deliver commands in a timely and secure manner. Table I describes each operation. Op 1 is a regular call-for-data from the PO which is performed periodically. Op 2 is performed when PO detects something abnormal and would like a data report from a particular MD. Time is more critical than a regular data reporting. Op 3 is done when MD detects something abnormal and would like to report to the PO. OP 4 is issued when PO needs a group of MDs to perform a certain action as soon as possible.

We develop our protocol to be secure from outsider attacks such as eavesdropping, impersonation, and message tampering, etc. There are three types of insiders in the protocol: PO, DCs, and MDs. We assume the PO is always trustworthy because it is the control of the whole system. The DCs, on the other hand, are honest-but-curious that they would follow the protocol as specified but would like to read the data and share with others if they could. That is, they would not impersonate another entity in the system, nor actively tamper the data, but would like to learn as much as possible based on the information they can access according to the normal operation of the protocol. As the MDs are devices located in the field (for example, on power grid poles), they are not likely to be in a very secure physical environment. We thus assume the MDs may be compromised after installation. In other words, an attacker takes over the MD and is able to read the key information kept in the device. In this situation, the attacker can report fake data to the PO on behalf of the MD. Our protocol cannot identify whether the data reported using a legitimate key is generated by an attacker, but our protocol ensures this compromised MD cannot impersonate others based on the key information it has. To detect whether a certain MD is compromised, intrusion detection techniques can be used, which is beyond the scope of this paper.

B. System Parameters

Before any communication, PO, DCs, and MDs are equipped with a set of system parameters. We assume necessary parameters are configured in a DC or MD before they are installed in the field.

1) Long-term keys: We assume there is a key server that can generate a set of public and private keys for each entity in the system. The public/private key pair is configured into a DC or MD before it is installed in the field. PO, on the other hand, apart from keeping its own key pair, it also remembers the public keys of all MDs and DCs in the system. We denote the public key and private key of A as $A^+$ and $A^-$, respectively. Under normal circumstances, PO would not publish the public keys of DCs and MDs to the general public. However, our protocol is secure even if the attackers know the public key information of any DC or MD they want to attack.

2) Diffie-Hellman (DH) parameters: We adopt the Diffie-Hellman key exchange mechanism to develop shared keys. The DH key exchange works as follows: When Alice and Bob want to generate a shared key using DH, they first each generate a random number and keep it as a secret. Let $a$ and $b$ be the secret, also called the secret DH half key, of Alice and Bob, respectively. They then exchange $g^{a\mod p}$ and $g^{b\mod p}$, where $p$ is prime and $g$ is a primitive root mod $p$. $g^{a\mod p}$ and $g^{b\mod p}$ is called the public half key or public DH key of Alice (Bob). When Alice receives the public half key $g^{b\mod p}$ sent by Bob, she can compute the shared key $g^\text{ab}$ by $(g^{a\mod p})^{b\mod p}$. Similarly, Bob can compute the shared key by $(g^{b\mod p})^{a\mod p}$. Although eavesdroppers can
overhear $g^a \mod p$ and $g^b \mod p$, they cannot compute $g^m \mod p$. Therefore, the shared key is secure. Through forgetting half keys and shared keys appropriately, DH keys also support \textit{perfect forward secrecy}.

Before using the DH mechanism, the PO, DCs, and MDs have to agree on the $g$ and $p$ to be used. We assume PO picks $g$ and $p$ and configures them into DCs and MDs before installation. To make the discussion concise, we drop $\mod p$ when the context is clear in the rest of the paper.

### C. Cryptographic functions

To provide authentication, confidentiality, integrity, and other security protections, messages have to be encrypted, hashed, or signed. We assume the PO selects appropriate cryptographic algorithms for the purposes, and these functions are installed in the DCs and MDs. For example, PO may use AES for symmetric key encryption and SHA-256 for hash computation. Table II summarizes the functions used in the protocol. Function $\text{GENKEY}(X, Y)$ is used when we need a key generated from two numbers $X$ and $Y$. This function is very computationally inexpensive and the time needed is negligible when compared with any cryptographic function.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{PKE}(K, M)$</td>
<td>apply public key encryption on $M$ using $K$</td>
</tr>
<tr>
<td>$\text{PKD}(K, C)$</td>
<td>apply public key decryption on $C$ using $K$</td>
</tr>
<tr>
<td>$\text{SKE}(K, M)$</td>
<td>apply symmetric key encryption on $M$ using $K$</td>
</tr>
<tr>
<td>$\text{SKD}(K, C)$</td>
<td>apply symmetric key decryption on $C$ using $K$</td>
</tr>
<tr>
<td>$\text{SIGN}(A, M)$</td>
<td>verify signature of $M$ signed by $A$ (created using $A$)</td>
</tr>
<tr>
<td>$\text{HASH}(K, M)$</td>
<td>compute keyed-hash of $M$ using key $K$</td>
</tr>
<tr>
<td>$\text{GENKEY}(X, Y)$</td>
<td>generate a key based on $X$ and $Y$</td>
</tr>
</tbody>
</table>

### TABLE II

**SYSTEM FUNCTIONS**

Some cryptographic functions run much slower than others. As some smart grid operations are time sensitive, it is very crucial to identify efficient cryptographic functions appropriately to protect the communication. To further understand the computational time of the cryptographic functions on computationally constrained devices, we measure the time needed to execute some representative cryptographic functions on Raspberry Pi. Raspberry Pi is a tiny computer with a size similar to a credit card. The CPU is 700MHz and the memory available is 512MB.

Fig. 2 shows the computational times of AES with a 256-bit key and SHA-256 of messages of different sizes. We have measured the times needed using other key sizes and the results are similar. Generally speaking, the amounts of time needed for symmetric key operations and hashes grow linearly with the size of the messages, while the growth rate of encryption/decryption is higher. The times for AES encryption and decryption on a message of 128 bytes (1024 bits) are less than 130 $\mu$s. The measured encryption and decryption times of 2048-bit RSA on a 1024-bit message on Raspberry Pi are around 7000 $\mu$s and 306 ms, respectively. We thus can conclude that data protection through symmetric key cryptographic functions allows a much faster data collection process.

<table>
<thead>
<tr>
<th>Message Size (bits)</th>
<th>Sign. (ms)</th>
<th>Ver. (ms)</th>
<th>sign/ver ratio</th>
<th>Sign. (ms)</th>
<th>Ver. (ms)</th>
<th>sign/ver ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>0.47</td>
<td>3.91</td>
<td>15.12</td>
<td>1048.33</td>
<td>11.49</td>
<td>91.23</td>
</tr>
<tr>
<td>256</td>
<td>0.97</td>
<td>4.01</td>
<td>16.19</td>
<td>1033.46</td>
<td>11.65</td>
<td>88.68</td>
</tr>
<tr>
<td>512</td>
<td>2.27</td>
<td>4.00</td>
<td>16.08</td>
<td>1047.98</td>
<td>11.69</td>
<td>89.67</td>
</tr>
</tbody>
</table>

### TABLE III

**RSA COMPUTATIONAL TIME**

Table III presents the time needed to create an RSA signature and verify an RSA signature using different key sizes. The time spent on encrypting a message using public key is similar to the time needed in verifying a signature. The time needed on decrypting a message using private key is similar...
to the time needed on signature creation. It can be observed that the time needed does not grow with message size but with key size. Column \textit{ratio} in the table gives the time ratio of signature computation. In an RSA key pair, the public key is usually much smaller than the private key \cite{34}. Therefore, the time spent on a private key operation (signing a message) is much longer than that on a public key operation (verifying a signature). The difference becomes more prominent when a longer RSA key is used. For example, when the 1024-bit key is used, the time ratio of signature verification of signature computation is about 16, but the ratio rises to 90 when the key size is 3072. Because of the high complexity of RSA private key operations, an efficient protocol should not require MDs to sign a lot of messages.

\begin{table}[h]
\begin{center}
\begin{tabular}{|c|c|c|c|c|}
\hline
\text{Time in microseconds} & 1024 bits & 2048 bits & 3072 bits \\
\hline
Private Key Generation & 347 & 408 & 468 \\
Public Key Generation & 23884 & 76996 & 239600 \\
Shared Key Generation & 30583 & 92115 & 261791 \\
\hline
\end{tabular}
\end{center}
\caption{DIFFIE-HELLMAN COMPUTATIONAL TIME} \label{table:diffie-hellman}
\end{table}

We also measured the time needed to generate different Diffie-Hellman keys with different key sizes (Table \ref{table:diffie-hellman}). It is worth noting that a DH shared key generation is more expensive than an RSA signature verification. It implies that it may not be appropriate to re-generate DH shared key for each data collection instance. By adopting different cryptographic functions and techniques carefully based on their security features and computational complexities, our protocol facilitates efficient and secure data collection.

\section{D. Protocol Overview}

Because encrypting data using public key cryptography is very expensive, before any data collection, we should first develop shared keys among PO, DCs, and MDs for data protection. To ensure data reported by a certain MD can be decrypted by the PO only, we need to establish a key that is known by PO and that MD. We call a key that is known by exactly two parties a \textit{pairwise shared key}. PO and each DC should also develop a pairwise shared key to protect their conversations. The same applies to DC with each MD it will talk to. Apart from pairwise keys, to facilitate a certain command or instruction to be delivered to a group of MDs in a secure and efficient manner, we also develop a set of \textit{group keys} that each group key is shared between the PO, a DC, and the MDs that connect to that DC. The group keys will also be used to update the pairwise shared keys efficiently. We will describe the details in Section IV-B.

The PO initiates the \textit{Shared Key Generation Process} to establish the necessary pairwise shared keys and group keys. We adopt the Diffie-Hellman key exchange mechanism to develop all pairwise shared keys. We authenticate the DH half keys using the long-term public keys to avoid the man-in-the-middle attack. Once the pairwise shared keys and group keys are established, they will be used for data collection and command delivery.

As shown in Table \ref{table:diffie-hellman}, DH operations are expensive. We should not re-generate the DH shared keys for every data collection. However, it may not be very secure if we use the same shared keys to encrypt data collected at different times. To strike a balance of computational complexity and security, the data encryption key for each data collection instance depends on both the DH shared key and the timestamp. As the timestamp changes for every data collection instance, the data encryption key will be changed even though we do not re-generate the DH shared key. In the following, we will first describe the Shared Key Generation process in Section IV. The detailed message exchanges of the four operations mentioned in Section III-A will be provided in Section V.

\section{IV. Shared Key Generation}

Let the set of MDs be $\mathbb{MD}$ and the set of DCs be $\mathbb{DC}$. Before the PO initiates the process, PO has to assign a set of MDs for DC to connect to. We let $\mathbb{MDLIST}_j \subseteq \mathbb{MD}$ be the set of MDs that are assigned to $\mathbb{DC}_j$. Definitely, $\bigcup_{\mathbb{DC}_j \in \mathbb{DC}} \mathbb{MDLIST}_j = \mathbb{MD}$. However, $\mathbb{MDLIST}_i \cap \mathbb{MDLIST}_j$, where $i \neq j$, may not necessarily be $\emptyset$. It is possible that PO would like multiple DCs to collect data from the same MD to enhance reliability. In fact, different assignments between MDs and DCs would differ in data security, cost, and data collection time. In Section VI, we will formulate the assignment problem to minimize the data collection time.

In the rest of this paper, for the ease of discussion, we use \textit{shared key} to refer to \textit{pairwise shared key}. We further denote $K_B^A$ as the shared key between $A$ and $B$. We refer to the set $\{PO, DC_i\} \cup \mathbb{MDLIST}_i$ as group $G_i$, and the group key of $G_i$ is $GK_i$. We use $M1 | M2$ to represent concatenating messages $M1$ and $M2$. The definitions of the functions used can be found in Table \ref{table:shared-key-functions}.

\subsection{A. Initial Shared Key Generation}

Figure 3 presents a summary of the initial shared key generation process. When the procedure starts, the only keys an MD or a DC knows are its own public/private keys and the public key of the PO. After the procedure, $MD_j$ should have established $K_{MDj}^{PO}$, $K_{MDj}^{DC}$, and $GK_j$ if $MD_j \in \mathbb{MDLIST}_j$. Through the procedure, $DC_j$ knows $GK_i$, $K_{DCj}^{PO}$ and $K_{MDj}^{DC}$ for all $MD_j \in \mathbb{MDLIST}_j$. The detailed procedure is as follows:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig3.pdf}
\caption{Initial Shared Key Generation.}
\end{figure}
1) PO starts the key generation process. It first generates a DH secret $a$ to talk to the DCs. It is possible for PO to use different $a$’s for different DCs. If so, PO has to remember the $a$ used for each DC. Apart from $a$, PO also captures the current timestamp $T1$ and sends the following message to $DC_i$.

$$PO \rightarrow DC_i; \text{PKE}(\text{DC}^+_{i}, g^a||T1), \text{SIGN}(\text{PO}, g^a||T1)$$

This message is secure from message tampering and impersonating because an attacker cannot sign $g^a||T1$ correctly.

2) When $DC_i$ receives the message, it retrieves $g^a$ and $T1$ using $DC^-_i$ and $PO^+$. It verifies whether $T1$ is reasonable. If so, it generates its DH secret $b$ and computes $K$ as $g^{ab}\text{mod } p$. $K$ is then the shared key between $PO$ and $DC_i$ ($K^P_{DC_i}$). It also replies $PO$ its public DH key.

$$DC_i \rightarrow PO; \text{PKE}(PO^+, g^b||T1), \text{SKE}(K, T1)$$

An attacker, who does not know $DC^-_i$, cannot develop $K$ and produce a correct message.

3) When PO receives the message, it can retrieve $g^b$ using $PO^-$ to compute $K$. It also verifies $\text{SKE}(K, T1)$ to ensure it was $DC_i$ who sent the message. It then sends $DC_i$ the list of MDs, together with the MDs’ public keys, that it assigns $DC_i$ to talk to. It also creates $C$ for $DC_i$ to talk to the MDs in the list. $C$ contains $g^c$, which is used for establishing shared keys between PO and MDs, and $GK_i$, which is the group key of $G_i$. The public keys of the MDs should also be sent (We assume they are included in $MDLIST_i$ in Figure 3). It is worth noting that PO also sends $\text{SIGN}(\text{PO}, C||DC^+_i)$ to $DC_i$. This signature is to avoid messages from being tampered.

$$PO \rightarrow DC_i; \text{SKE}(K, MDLIST_i||C||\text{SIGN}(\text{PO}, C||DC^+_i)) \quad \text{where } C = g^c||GK_i$$

4) After verifying $\text{SIGN}(\text{PO}, C||DC^+_i)$ to ensure the message has not been tampered, $DC_i$ can then generate its DH half key, $g^e_i$ for establishing shared keys with the MDs. $DC_i$ also captures the current timestamp $T2$ and sends the information to $MD_j$ in $MDLIST_i$ using the public keys provided. $DC_i$ also needs to send its public key. To protect $DC^+_i$ and $C||g^e_i||T2$ from being tampered, $\text{SIGN}(PO, C||DC^+_i)$ and $\text{SIGN}(DC_i, g^e_i||T2)$ are sent as well.

$$DC_i \rightarrow MD_j; DC^+_i, \text{PKE}(MD^+_{j}, C||g^e_i||T2), \text{SIGN}(PO, C||DC^+_i), \text{SIGN}(DC_i, g^e_i||T2)$$

5) Upon receiving the message, $MD_j$ retrieves $C||g^e_i||T2$ and verifies the signatures. It then generates a DH secret key $d$ to establish the shared key between itself and PO ($K^P_{MD_j}$), which is $g^{ed}$, and the shared key with $DC_i$ ($K^{DC}_{MD_j}$), which is $g^{ed}$. It sends the information of $g^d$ to $DC_i$. As $DC_i$ sends the same $g^e_i$ and $T2$ to all other MDs in $MDLIST_i$, $MD_j$ has to sign $g^d$ to authenticate the reply.

$$MD_j \rightarrow DC_i; \text{PKE}(DC^-_{i}, g^d), \text{SIGN}(MD_j, g^d||T2)$$

6) When $DC_i$ receives the message, it retrieves $g^d$ and verifies the signature. If so, it sends $PO$ the key information.

$$DC_i \rightarrow PO; \text{SKE}(K, g^d||T2), \text{SIGN}(MD_j, g^d||T2)$$

7) If $g^d||T2$ encrypted using $K$ and signed by $MD_j$ are the same, the message has not been tampered. PO can then compute the $K^P_{MD_j}$ to be $g^{ed}$. Note that as $DC_i$ can only read $g^e$ and $g^d$ but neither $c$ nor $d$, it cannot compute $g^{ed}$. $g^{ed}$ is thus a key shared by PO and $MD_j$ only.

We now analyze the memory needed for each entity to keep the shared keys. The PO needs to keep a shared key for each DC, a shared key for each MD, and a group key for each group. The total number of keys is $2x|\text{DC}| + |\text{MD}|$. PO has to keep $K^P_{DC_i}$, a shared key with each MD belongs to its group, and a group key. The total is $2 + |MDLIST_i|$. For $MD_j$, for each group $G_i$ it belongs to, it has to keep a shared key with $DC_i$ and the group key $GK_i$. It is worth noting that $MD_j$ can establish different shared keys with PO through different DCs. If PO provides different $g^e$’s for different DCs, the shared keys developed via different DCs must be different. Even when PO provides the same $g^e$ through different DCs, $MD_j$ can also establish different shared keys by replying different $g^d$’s for different DCs. Therefore, $MD_j$ has to keep at most $3 x$ number of groups it belongs to keys in total. PO decides how many groups an MD is associated with and can thus establish keys according to the memory available in different MDs.

B. Shared Key Update

The shared keys generated are expected to be used in the subsequent data collections and command deliveries. As this shared key generation process would not be launched for every single data collection, time and computational complexities are thus not a very major concern, especially for the first time the shared keys are generated. On the other hand, a secure system should periodically change the shared keys used. If the keys have to be frequently changed, the procedure in Fig. 3 might be too expensive as an MD has to handle several expensive public key and DH operations. To reduce the complexity, if the shared keys being used currently are still secure (remain secret to attackers), we can replace the public key encryptions and signatures by symmetric key encryptions as shown in Fig. 4.

In the communication between PO and $DC_i$, both PO and $DC_i$ now encrypt their DH half keys using the current $K^P_{DC_i}$ instead of the public keys. In Step 4, $DC_i$ uses the current group key $GK_i$ to encrypt $C||g^e_i||T2$ so that it can be sent to all $MD_j \in MDLIST_i$. To avoid a malicious MD who knows $GK_i$ from impersonating $DC_i$, $DC_i$ sends also $\text{SKE}(K^P_{MD_j}, T2)$ to authenticate the message to $MD_j$. In Step 5, $MD_j$ only needs to encrypt its new DH half key using symmetric key. Thus, $MD_j$ does not need to perform any expensive public key operations but only needs to perform the three necessary DH operations to compute $g^d$, $g^{ed}$, and $g^{ed}$ for changing keys. To understand how much time we could save by using the simplified shared key generation, we measure the times needed for an MD to process the message from the DC and prepare the reply when different RSA and DH key sizes are used. The results are presented in Fig. 5. More time is saved when a longer RSA key is used. It implies that the shared key update process allows a longer RSA key to be used to enhance security.
It is worth noting that in the key update procedure, it is possible for a malicious DC to trick an MD to accept a fake C created by the DC. This should not happen if all DCs are honest-but-curious. In case it is necessary to protect a previously honest but now malicious DC to issue the attack, the PO can sign C. That is, instead of sending $SKE(GK, C)$, PO should send $SIGN(PO, C)$ to the MDs via the DC.

PO can start collect data after the shared keys are established. We will describe the details of each operation in the next section.

V. DATA COLLECTION AND COMMAND DELIVERY

A. PO initiates Data Collection of a Group of or All MDs

It is a regular data collection initiated by the PO. To ensure the data reported can be read by the PO only, $MD_j$ should encrypt the data using $K_{MD}^{PO}$. The protocol should not request MDs to perform a lot of expensive operations as well. To reduce the total time needed, we should reduce the number of messages PO or DC has to create. In the following, we first present the data collection procedure in a step by step manner. Fig. 6 shows the whole process. In the figure, $K_1$, $K_2$, and $K_3$ are $K_{DC}^{PO}$, $K_{MD}^{DC}$, and $K_{MD}$, respectively.

1) PO first identifies all the DCs that it needs to talk to according to a certain optimization criterion. It captures the current timestamp $T$ and sends a message to $DC_i$. Note that it is possible that PO do not want to collect data from some MDs in $MDLIST_i$. If so, PO should also include the list of intended MDs. We omit that in our protocol to simplify the discussion.

$$PO \rightarrow DC_i: SKE(K_{DC}^{PO}, T || SIGN(PO, T))$$

2) $DC_i$ verifies the signature and checks whether $T$ is reasonable. It then sends $T$ to $MD_j \in MDLIST_i$ (or only the MDs PO wants to collect data from).

$$DC_i \rightarrow MD_j: SKE(GK_j, T || SIGN(PO, T))$$

By encrypting the message using the group key $GK_j$, $DC_i$ only needs to create a single message for all MDs in its group. However, the group key cannot authenticate it was PO who requested the data collection because it is a key shared by many entities. We thus need to include a signature of PO to facilitate authentication. This message should work fine if $DC_i$ has to collect data from every MD in $MDLIST_i$. However, when some MDs are not supposed to report data, those are not reporting can also read $T$ in the message. If this is a serious concern, $DC_i$ can send $SKE(K_{MD}^{DC}, T || SIGN(PO, T))$ to the involved MDs instead. The disadvantage of this approach is $DC_i$ needs to create a different message for different MD and possibly incurs more delay in the data collection process.

3) After verifying $T$, $MD_j$ generates a key $MK = GENKEY(K_{MD}^{PO}, T)$. An encryption key and an integrity key developed based on $MK$ are used to protect the data. The protected data is denoted as $PRODATA$. As $MK$ depends on $T$, different $MK$’s will be used for different data collection instances even $K_{PO}^{MD}$ is not changed. $MD_j$ also generates $DK = GENKEY(K_{MD}^{DC}, T)$. The hash of $PRODATA$ using $DK$ is computed and sent to $DC_i$.

$$MD_j \rightarrow DC_i: PRODATA, HASH(DK, PRODATA)$$

4) $DC_i$ verifies the hash to ensure $PRODATA$ was generated by $MD_j$ even it cannot decrypt $PRODATA$. It then forwards $PRODATA$ to $PO$.

$$DC_i \rightarrow PO: SKE(GENKEY(K_{DC}^{PO}, T), PRODATA)$$

Alternatively, $DC_i$ can encrypt all the replies from $MDs$ in a single message. In this case, only a single symmetric key encryption is needed, but $PO$ may receive some data later.

5) Finally, $PO$ develops $MK$ on its own to extract the data from $PRODATA$. 

Fig. 4. Shared Key Update.

Fig. 5. Time Comparison of Initial Key Generation and Key Update.

Fig. 6. Data Collection.
We now analyze the complexity of the whole process of data collection from the perspective of each entity. We first develop a summary of the cryptographic functions needed in each step in Table V. In the table, we use the function names in Table II to represent a function.

In Step 1, PO only needs to sign \( T \) once as the signature included in the messages to all DCs is the same. After the signature is created, it uses the shared key with different DCs to encrypt the message using symmetric key cryptography. In Step 2, DC needs to decrypt the message and verify the signature. It then encrypts the message to the MDs in MDLIST\(_i\) using the group key. In Step 3, 1 decryption and 1 signature verification are first needed. In generating PRODATA, 1 hash and 1 symmetric key encryption operations are applied. Another hash operation is used to compute HASH(DK, PRODATA). In Step 4, for each message received, DC has to verify the hash and encrypt PRODATA again. DC receives \(|MDLIST|\) messages in total. Finally in Step 5, for each report from MD\(_j\), PO first decrypts using the shared key with DC. It then decrypts and verifies the hash of PRODATA. Therefore, for each reply of MD, it takes 2 symmetric key encryption operations and 1 hash operations. There are altogether \(|MD|\) replies.

It can be observed that each MD, each DC, and the PO need to perform one public key operation only no matter how many messages it has to handle. Besides, the signature verification that MDs and DCs have to perform is not very expensive when compared with signature creation. Our protocol is thus very light-weight and scalable.

**B. PO requests data from MD\(_j\)**

1) PO first identifies a certain DC\(_i\) such that MD\(_j\) \( \in G_i \). \( T \) is the timestamp. Apart from signing the timestamp, PO also encrypts the timestamp using \( K_{MDj}^{PO} \).

\[
PO \rightarrow DC_i: \text{SKE}(K_{DC_i}^{PO}, T) || \text{SIGN}(PO, T) || \text{SKE}(K_{MDj}^{PO}, T)
\]

2) DC\(_i\) sends the information to MD\(_j\) after verifying the signature on \( T \).

\[
DC_i \rightarrow MD_j: \text{SKE}(K_{MDj}^{DC_i}, T) || \text{SKE}(K_{MDj}^{PO}, T)
\]

Steps 3 - 5 are the same as in Section V-A.

**C. MD\(_j\) initiates an urgent data report**

1) MD\(_j\) first identifies a certain DC\(_i\) to relay the message and records the current timestamp \( T \). PRODATA and DK are generated as in Step 3 in Section V-A.

\[
MD_j \rightarrow DC_i: \text{SKE}(K_{MDj}^{DC_i}, T) || \text{SKE}(PRODATA, HASH(DK, PRODATA))
\]

2) DC\(_i\) verifies the hash and forwards PRODATA to PO.

\[
DC_i \rightarrow PO: \text{SKE}(K_{PO}^{DC_i}, T) || PRODATA
\]

3) PO can then extract \( T \) using \( K_{PO}^{DC_i} \) to develop the appropriate keys to decrypt PRODATA.

In reporting emergency information, latency and reliability are very important. In the protocol, MD\(_j\) does not need to perform any expensive public key operation before sending the data report. The latency is thus very small. To enhance reliability, MD\(_j\) can send the data to PO via multiple DCs. It has to compute HASH(DK, PRODATA) and encrypt \( T ||\text{PRODATA} || \text{HASH(DK, PRODATA)} \) using different keys for different DCs in Step 1. As both operations are not expensive, MD\(_j\) can send out the reports promptly.

**D. PO issues an urgent command to a group of MDs**

1) Similar to requesting data, PO should first identify the DCs that cover all the MDs that it wants to send the urgent command to. Let the command be COMD. MDLIST\(_i\) contains the MDs that DC\(_i\) should talk to.

\[
PO \rightarrow DC_i: \text{SKE}(K_{DC_i}^{PO}, SIGN(PO, COMD)) || MDLIST || \text{COMD}
\]

2) DC\(_i\) sends to each MD\(_j\) in MDLIST\(_i\) the urgent command.

\[
DC_i \rightarrow MD_j: \text{SKE}(K_{MDj}, \text{SIGN}(PO, COMD)) || \text{COMD}
\]

The signature of the command by the PO provides authentication check to all MDs and DCs. By using a group key in Step 2, we share the same issue as in Step 2 of Section V-A. The administrator can thus select the most appropriate way to strike a balance of security and efficiency.

**VI. GROUPING OPTIMIZATION**

**A. Preliminaries**

When an MD’s data may be collected by more than one DC, the problem of deciding which DC to allocate to MDs arises. An umbrella definition for a set of problems to associate DCs with MDs for collecting data is given below: 

**Definition 1 (DC to MD Association (DC2MD) Problem):** Given a set \( M \) of \( |M| = m \) MDs with data to report and a set of \( D \) of \( |D| = d \) DCs to collect the aforementioned data, find a feasible association to ensure that all the data from MDs will be collected by the DCs.

The DC2MD problem can be expressed as an instance of a family of combinatorial optimization problems modeled by set covering, packing, and partitioning formulations [35]. Very often, we want to identify a feasible association that gives the best performance among all possible associations. Best performance can be the minimization of the cost of data collection, the total time of data collection, etc. This genre
of problems are collectively referred to as the Family of Set Problems (FSP) [36].

A more generic formulation of the FSP takes on the following structure [35]:

\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad Ax \geq e \\
& \quad x_i \in \{0, 1\} \quad \text{for } i = 1, 2, \ldots, d,
\end{align*}
\]

where \( c \) is a column vector of \( n \) costs or weights, \( x \) is a column vector of \( n \) decision variables, \( e \) is a column vector of \( k \) ones, and \( A \) is a \( k \times n \) matrix of zeros and ones. In the following, we will study how to optimize the time for data collection in details.

\[\text{B. Optimizing the Time for Data Collection}\]

We now consider how to minimize the time to perform data collection from a group of MDs by selecting a single appropriate DC to collect data from each MD. To compute the total time needed for PO to collect the data, we first define some notations to represent the time needed to perform a single cryptographic operation defined in Table II. Theoretically speaking, the time needed for a cryptographic operation depends on the size of the message. As we only perform public key operations on small-sized messages, we ignore this factor and denote \( T^p(\text{OP, A}) \) as the time needed for \( A \) to execute public key cryptographic operation \( \text{PKE, PKD, SIGN, and SIGV} \). For example, the time for PO to sign a message is \( T^p(\text{SIGN, PO}) \). To capture the effect of message size on the computational time of symmetric key and hash operations, we denote the time needed as \( T^s(\text{OP, A, SIZE}) \). As symmetric key encryption and decryption take roughly the same time, we use \( \text{SK} \) to represent both \( \text{SKE} \) and \( \text{SKD} \). We also use \( \text{HASH} \) to denote both hash computation and verification. To simplify our discussion, we assume the size of \( \text{SK} \) to be 1 unit. That is, the time needed for \( \text{DC} \) to develop message \( \text{SKE}(GK_i, T||\text{SIGN}(PO, T)) \) is \( T^s(\text{SK}, DC_i, 1) \). The one-way network delay between \( A \) and \( B \) is \( T^n(A, B) \). We also let \( x_{ij} = 1 \) if \( MD_j \) belongs to \( G_i \).

Table V presents a summary of the cryptographic operations each entity has in perform every step of the data collection process. To simplify our discussion, we use \( M1, M2, M3, \) and \( M4 \) to represent the four messages exchanged between PO, DCs, and MDs as shown in Fig. 6. We only consider the situation where a DC reports all data collected in a single message to PO. To illustrate the process of time analysis, we present Fig. 7 to explain the different time components in the whole data collection process. In the picture, we assume there are only two MDs.

We first develop the time needed for \( \text{DC} \), after having prepared \( M2 \), to send message \( M2 = \text{SKE}(GK_i, T||\text{SIGN}(PO, T)) \) to \( MD_j \) and verify the hash of \( MD_j \)’s reply, which is denoted as \( T_{ij} \). \( T_{ij} \) is the sum of the following components:

1) round-trip network delay between \( DC_i \) and \( MD_j \):
\[2T^n(\text{DC}_i, MD_j)\]

2) time needed for \( MD_j \) to generate reply \( M3 \) (Step 3):
\[T^s(\text{SK, MD}_j, 1) + T^p(\text{SIGV, MD}_j) + T^s(\text{SK, MD}_j, \text{size}) + 2T^s(\text{HASH, MD}_j, \text{size})\]

3) time needed for \( \text{DC}_i \) to verify the hash:
\[T^s(\text{HASH, DC}_i, \text{size})\]

Before \( \text{DC}_i \) can send message \( M2 = \text{SKE}(GK_i, T||\text{SIGN}(PO, T)) \) to \( MD_j \), \( DC_i \) needs to decrypt \( M1 \) and prepare \( M2 \). As described in Step 2 in Section V-A, \( DC_i \) has to spend \( 2T^s(\text{SK, DC}_i, 1) + T^p(\text{SIGV, DC}_i) \) time to prepare \( M2 \). We now study the time needed for \( DC_i \) to prepare the reply (\( M4 \)) to PO after verifying the hashes of the replies from all MDs. Let \( N_i \) be \( \sum_j x_{ij} \). That is, \( N_i \) is the number of MDs in \( G_i \). The total amount of data received by \( DC_i \) is \( N_i \times \text{size} \). The time to prepare \( M4 \) is \( T^s(\text{SK, DC}_i, N_i \times \text{size}) \).

Therefore, the total time needed for \( \text{DC}_i \) from the moment it receives \( M1 \) from \( PO \) to the moment it sends out \( M4 \) to \( PO \) is:
\[T_{\text{DC}_i} = 2T^s(\text{SK, DC}_i, 1) + T^p(\text{SIGV, DC}_i) + \max_j \{x_{ij}T_{ij}\} + T^s(\text{SK, DC}_i, N_i \times \text{size})\]

We now study the time from the moment that \( PO \) sends \( M1 \) until the moment that \( PO \) successfully decrypts and verifies the data carried in \( M4 \) sent by \( DC_i \). We denote this time as \( T_{\text{DC}_i} \).

\[T_{\text{DC}_i} = 2T^n(PO, DC_i) + T^p(\text{PKD, PK}) + 2T^s(\text{SK, PO, N}_i \times \text{size}) + T^s(\text{HASH, PO, N}_i \times \text{size})\]

where
\[f(i) = 2T^s(\text{SK, DC}_i, 1) + T^p(\text{SIGV, DC}_i) + 2T^n(PO, DC_i) + 2T^s(\text{SK, PO, N}_i \times \text{size}) + T^s(\text{HASH, PO, N}_i \times \text{size})\]
C. Problem Formulation

When PO wants to collect all data as soon as possible, we should assign each MD to an appropriate DC such that the maximum $T_{DC}$ over all $i \in \mathbb{D}$ is minimized. Such an objective leads to what is known in the literature as a minimax problem. From Equation 4, we can simplify the terms into two major categories for the minimax optimization: One is the maximizing component ($\max_j \{x_{ij}t_{ij}\}$) and the other is the summative part ($f(i)$). The former consists mostly of the network delay whose maximum value will determine the total completion time for data collection by a DC. The latter includes the processing time, including the cryptographic computation, whose total time will be a summation operation. In what follows, we will ignore the maximization components, as it is rather trivial to address alone, and concentrate on the summative part. Under a realistic data collection scenario, as it is rather similar to the problem from MD $Q$, objective function is denoted by $\alpha$, $\beta$, and $\gamma$, respectively. The summative part of our problem can only work on one job at a time. The well-established 3-field classification introduced in [39] uses notation, where job, machine, and scheduling characteristics are denoted by $\alpha$, $\beta$, and $\gamma$, respectively. The summative part of our objective function is denoted by $Q|C_{max}$, where arbitrary number of machines operating at different speeds must be used to complete a given set of tasks with the minimum makespan objective. This problem setting is also known in the literature as identical parallel machines [40]. In our problem, machines are DCs, and tasks are MDs whose data need to be collected.

The Integer Linear Programming (ILP) formulation for our summative part may be formulated as follows:

$$\min \ max \ \sum_{j} x_{ij}t_{ij}$$ (5)

s.t.  $\sum_{j} x_{ij} = 1, \ \forall i \in \mathbb{D}$ (6)

$$x_{ij} \in \{0,1\}, \ \forall i \in \mathbb{D}, \forall j \in \mathbb{M}$$ (7)

where $x_{ij}$ represents whether DC $i$ is assigned to collect data from MD $j$, and $t_{ij}$ is the amount of the summative part of the total data collection time of MD $j$'s data by DC $i$.

When we set $C_{max}$ represent the maximum data collection time, the above formulation can be rewritten in a standard form as follows:

$$\min \ C_{max}$$ (8)

s.t.  $\sum_{j} x_{ij} = 1, \ \forall i \in \mathbb{D}$ (9)

$$\sum_{j} x_{ij}t_{ij} \leq C_{max}, \ \forall i \in \mathbb{D}$$ (10)

$$x_{ij} \in \{0,1\}, \ \forall i \in \mathbb{D}, \forall j \in \mathbb{M}$$ (11)

The above problem can be shown to be strongly NP-Hard [41], [42] by a reduction from a 3-Partition problem [43]. Also note that this problem is a kind of the dual of the bin packing problem as part of the FSP from Equations 1-3 [41], [44].

This intractability of the problem of minimum makespan makes it unlikely that a polynomial algorithm exists. Thus, we have to resort to heuristic algorithms to address it.

D. Heuristic Approaches

When the machines are assumed to be identical, that is with same speed, then the problem is known as identical parallel machines, and denoted as $P|C_{max}$ [38]. Even for this relaxation, it has been proven that it is in the domain of NP-hard problems [39], [45].

A traditional way to deal with scheduling problems starts with a simple greedy combinatorial algorithm, referred to as List Scheduling (LS) [46]. Jobs are picked from a pre-specified list and fed into the machines as they become available. For $P|C_{max}$, if the job list can be sorted in descending order of processing times, the resulting algorithm is called Largest Processing Time (LPT) [47]. Another algorithm, called multiftit, for $P||C_{max}$ is proposed in [44]. Note that our problem, as stated in Equations (5)-(7), is based on $Q|C_{max}$, a more involved scheduling problem than $P|C_{max}$. We devise the following two heuristics to tackle $Q|C_{max}$ by assuming that total time of data collection by DCs from MDs is summative.

1) Greedy Algorithm: Longest to Least Loaded First (L3F): Longest job to the least loaded DC first (L3F) is a greedy algorithm. We find the largest time for data collection for any (DC, MD) pair, say $\delta, \mu$. We assign MD $\mu$ to a DC that will complete in the least time. Next, we pick the next largest time and assign it to the least loaded MD for the corresponding MD. We iterate until we deplete unassigned MDs. It is obvious that the complexity of the algorithm is $O(d)$.

Algorithm 1: Longest to Least Loaded First (L3F).

1: procedure L3F(X)
2: $\mu \leftarrow 0$ $>$ makespan
3: for $\forall j \in \mathbb{M}$ do
4: $\gamma \leftarrow$ Smallest $t_{ij}$
5: Find least loaded DC that can collect $\gamma$’s data
6: update
7: $\delta \leftarrow$ least loaded DC that can take $\gamma$ without exceeding $\omega$
8: update $\mu$
9: end for
10: end procedure

2) Augmented Multifit Algorithm (AMFA): The basic idea of the multifit approach [44] for $P|C_{max}$ is to sort the jobs and then assign them to the machines in ascending order of load. We adopt multifit and augment it for our problem as shown in Algorithm 2. Input to Algorithm 2 are $\Theta = \{t_{ij}, \forall i \in$
Algorithm 2 Augmented Multifit Algorithm

1: procedure AMFA(Θ, k)
2: Compute upper and lower bounds of the feasible objective function value (U, L)
3: while r <= k do
4: \( \omega \leftarrow (U + L)/2 \)
5: if FFD(\( \omega \)) then
6: \( U \leftarrow \omega \)
7: else
8: \( L = \omega \)
9: end if
10: end while
11: return U
12: end procedure

Algorithm 3 First Fit Decreasing (FFD).

1: procedure FFD(\( \Theta, \omega \))
2: for \( \forall j \in M \) do
3: \( \gamma \leftarrow \text{Largest } t_{ij} \) column
4: \( \sigma \leftarrow \text{smallest in } \gamma \)
5: \( \delta \leftarrow \text{least loaded DC that can take } \gamma \) without exceeding \( \omega \)
6: if \( \delta = \text{NULL} \) then
7: return false
8: else
9: assign \( \sigma \)'s MD to DC \( \delta \)
10: end if
11: end for
12: end procedure

Fig. 8. Searching for approximation solution in AMFA.

\( \mathbb{D}, \forall j \in M \) and \( k \) is the number of iterations, for which values 7 to 19 have given good results in simulations, whose details will be given in Section VII. The basic idea of this algorithm is to choose a lower bound, \( L \), smaller than which \( C_{max} \) will be infeasible, and upper bound, \( U \), above which \( C_{max} \) is guaranteed to be feasible, as shown in Figure 8. As noted in Figure 8, there will be no solution to our problem formulation when \( C_{max} \leq L \). Similarly, for all values of \( C_{max} > U \), solution can be obtained. The area of interest is the set of values, marked as Search Area in Figure 8, where pinpoint the exact point of demarcation between feasible and infeasible is not clear. Our objective in AMFA from Algorithm 2 is to seek an approximate value for this demarcation point where the optimal value is located at. In effect, we are using a binary search within the search area while iterating through the AMFA algorithm. Starting off with the mid-point between \( U \) and \( L \), we use an oracle, called FFD, and to be described below, to check if the our problem with that \( C_{max} \) is feasible. If so, we reduce the search area between from \([L, U]\) to \([L, \frac{U+L}{2}]\). Otherwise, we reduce the search area to \([\frac{U+L}{2}, U]\). We settle an approximate value after a small number of iterations, \( k \).

As alluded above, we use a kind of an oracle, as in the sense of the theory of computation, to decide whether a solution is feasible given \( C_{max} \) and \( \Theta \). The idea of this oracle, called First Fit Decreasing \( \text{FFD} \) algorithm is to find a near-optimum answer to the question [44]. As noted earlier, we extend \( \text{FFD} \) and adopt it for our problem \( Q||C_{max} \) as shown in Algorithm 3.

VII. PERFORMANCE EVALUATION

In order to assess the performance of our approaches, we have used IBM’s CPLEX optimization package to solve the ILP formulations and implemented our approaches in C++.

A. ILP Optimal solution versus heuristics

First, we would like to check how our approaches fare with the optimal solutions provided by the CPLEX to the extend possible. Since the problem is NP-Hard, the ILP formulation that can be solved by CPLEX hit a wall rather quickly: After about 70 number of MDs and 35 DCs, CPLEX started taking very long to yield any results. Thus, we have run some simulations up to 70 MDs and 35 DCs each with 30 runs to get an idea of the comparative performance results. The time for collecting data from MDs by DCs are randomly generated from a uniform probability distribution in the range of 10 to 100. The number of DCs took the values of 10, 20, and 30 while the number of MDs were assigned 25, 40, 55, and 70. All possible combinations were run for 30 times for statistical significance. Figure 9 shows the performance of ILP, AMFA, and \( L^3F \) for all 12 combinations of the number of MDs and DCs. It plots the total time values returned by the ILP from CPLEX as the reference point and hence shows it as a straight line on bottom. Dashed line in the middle with circles is the results of the AMFA approximation algorithm, showing the discrepancy with the optimal ILP. As shown, AMFA is pretty

![Graph showing performance comparison](image)

Fig. 9. Ratio of total data collection time for AMFA and \( L^3F \) to Optimal ILP.
close, whose mean distance was about 1.34. In other words, AMFA appeared to be 1.34 times of the optimum value. L^3F, being a greedy algorithm, performed worse with an average distance ratio to the optimum of approximately 1.96. Note that L^3F showed a higher level of variability than AMFA. Figure 10 shows the box plots with mean values in squares for the AMFA and L^3F algorithms. Upper and lower values of the box shows the 75th and 25th percentiles, the line median and the small square point the mean. Again, the reference point is the optimum values produced by the ILP, and the y axis shows the ratio of the distance of AMFA and L^3F. As shown, AMFA has a much narrower distribution than L^3F with significantly better expected approximation values. The variability, as mentioned before, is also corroborated by these box plots.

**B. Extensive simulations of the heuristics**

As indicated above, due the intractability of our problem, CPLEX cannot yield results after a rather small numbers of MDs and DCs. Thus, in order to evaluate the performance of the AMFA and L^3F algorithms with larger configurations, we had to drop the ILP from the simulations.

We have run extensive simulations with the number of MDs going up to 1000 in increments of 50 starting from 50 and number of DCs at 25, 50, 75, 100. We had a total of 80 unique (MD, DC) pairs. Again, in order to attain statistical significance, each combination pair was run 30 times. The time values for the data collection from MDs by DCs were generated using a uniform density function in the range of 10 to 100. Figure 11 displays the total time of data collection for AMFA over the number of MDs from 50 to 1000 for 25, 50, 75, and 100 DCs as separate lines. Except for when the number of DCs was equal to 25, the total time increases with respect to larger number of MDs is with moderate slope. When DC is equal to 25, the increase is rather steep but still linear. This behavior might indicate that when there is significant imbalance between the number of DCs and MDs the total time to collect data may adversely affected. This point of operating overload is hard to have a threshold value to associate with

Figure 12 plots the notion as Figure 11 but for L^3F. While similar behavior for DC-MD imbalance is still observed here, the overall pattern is much less stable than AMFA. The magnitude of values are also greater. In order to more clearly depict the AMFA versus L^3F behavior, we plotted the 25 and 100 DC lines in Figure 13, where the aforementioned smoothness and magnitude differences are more obvious.

Figure 14 and Figure 15 show the overall performance of AMFA and L^3F over all 80 combinations of number of DC and MDs. AMFA's smoothness across the range of (MD, DC) pairs is evident while L^3F suffers from wide oscillations.

Figure 16 shows the ratio difference of performance for L^3F with respect to AMFA over 80 different MD-DC combinations. L^3F is the zigzag line at the top with the middle showing the normalized performance of AMFA. Performance of L^3F can get as high as 2.3 times the AMFA's with the average distance of 1.6062 times. We also plotted the extrapolated performance line for the ILP based on the average ratio of distance from
AMFA and ILP from our simulation results in Section VII-A.

VIII. CONCLUSION

The bidirectional power and information flow of the Smart Grid vision has led to the proliferation of nodes, such as measurement devices, sensors, smart meters, etc. These devices generate unprecedented amounts of data. The existing, legacy protocols are not capable of addressing this new phenomenon. In order to address this challenge, we propose a comprehensive and secure communications protocol to enable a power operator to collect data from measurement devices in a practical, scalable, and efficient manner under a hierarchical data collection model. Intermediary nodes relay are assumed to follow the honest-but-curious model in relaying the data. Thus, our protocol paves the way for third party service provisioning, as envisioned by the NIST Smart Grid Framework. Examples of such third party services include outsourcing data collection by 3rd party DCs, utilizing cloud computing services for data storage and processing, etc. We formulate an optimization problem for associating the intermediary relay nodes with measurement devices for data collection in order to minimize the total data collection time. The problem is intractable and thus we present two fast approximation heuristics. Extensive simulations results show that heuristics provide a very good approximation with fast convergence.

REFERENCES