Link Vulnerability Aware Task Deployment over Edge and Cloud

MALEK ASSAAD, University of Twente, NL

Most applications of IoT are bounded by constraints such as latency, processing capacity and security. When an IoT device cannot meet the constraints of an application, the work may be offloaded to more capable systems such as the cloud or edge nodes. There exist many solutions to this problem, solved with graph theory, machine learning or artificial intelligence but they lack depth when it comes to security while deciding the deployment point. This research aims to provide an algorithm for where to deploy an application, to the edge or the cloud, while respecting the constraints and taking advantage of heterogeneous encryption schemes. It also discusses the benefits and costs of the security aware solution. The paper arrives at a successful solution that utilizes graph theory that allows nodes to change the encryption scheme of data to secure it while it passes over vulnerable links. The solution has compromises, especially in time to compute as networks get larger.

Additional Key Words and Phrases: Internet of Things, encryption, decryption, algorithm, service deployment, edge computing, cloud computing

1 INTRODUCTION

IoT devices are ubiquitous and are becoming increasingly common and more integrated in human lives. As such, applications for Internet of Things (IoT) devices are also becoming increasingly complex and resource demanding [1]. IoT devices are expected to keep up with application requirements which is not always feasible considering their limited resources. These devices may, therefore, need to process their data at a remote location to allow for smarter operation in the short term or for statistical purposes in the long term. Service deployment determines the best location to deploy such task with the goal of successfully completing it considering the constraints of the application, such as latency. There are many factors that affect latency of an application and among them are network bandwidth, deployment point of the application and encryption scheme.

The deployment space of an application is a network of end, edge and cloud nodes connected by data links [2] and an example of which is displayed in Figure 1. The network is not randomly linked, instead, it consists of three main spaces pertaining to the three types of nodes in it, the i) end space, ii) edge space and iii) cloud space. Data links can exist between any two nodes in a space, any one end and edge node, and any one edge and cloud node. Any two nodes in such a network may have different capabilities, such as processing or encrypting/decrypting speeds and any two links may have different values for latency and vulnerability to attacks.

Prior research such as [3][4][5] have investigated the service deployment problem and provided solutions with performance measures. However, the solutions lacked consideration for the vulnerability of links, that is, they do not consider that different links may have different vulnerabilities nor do they consider what encryption scheme to use as part of their service deployment algorithm. As on any real world network, data is susceptible to attacks, such as eavesdropping, when traversing links. Moreover and relatively frequently, applications require the transmission of sensitive data, that is why many of the work done, including papers [4] and [5], assume the use of one all encompassing encryption scheme whereas some, like paper [3], consider some nodes unsafe for sensitive data transmission. Utilizing one encryption scheme limits the number of paths data can take in a network, which reduces the number of valid deployment points for an application or dismisses some nodes for being too vulnerable.

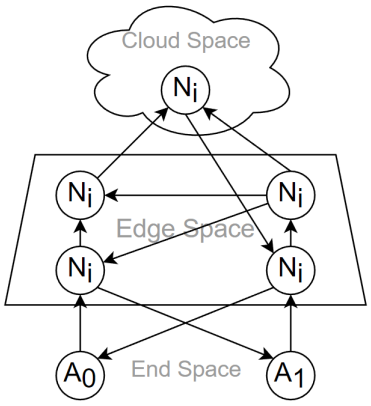

Fig. 1. Typical Network Topology

The aim of the research is to provide a contribution that places particular emphasis on security that also respects all other application constraints. The aim is to create a system model that considers not only the latencies of transmission, encryption, decryption and processing but also link vulnerability. The model will then be used to create a service deployment algorithm that deploys and application on a network such that the latency constraint is respected and the most secure path is selected for data to traverse. The input of the algorithm is the latency constraint of the application, and the output is a path through the network such that the total latency of the links is less than that of the input latency, the vulnerability of the links is minimized and it passes through only one node that process the data. An example of a service deployment is cloud gaming which requires low latency and will therefore need to be deployed close to the IoT device. Finally, the research also includes a section that investigates and compares the proposed solution to the others, highlighting any strengths and weaknesses in its application and suggests possible future research and development. The comparisons will be done by running simulations on a real world scenario application on networks of different sizes and variable values.

Although there exists many research papers on the problem of service deployment, few of them consider how to use security as

Author's address: Malek Assaad, m.h.assaad@student.utwente.nl, University of Twente, PO Box 217, 7500 AE Enschede, Enschede, NL.

a tool to improve it and how to encrypt the data in such a way to maintain data integrity during the deployment an application. The paper will create a model for the problem of service deployment, provide a security aware solution and discuss its costs and benefits in the face of existing solutions. Moreover, the performance of the solution will be measured to compare its performance next to some baseline deployment algorithm. By considering all that has been discussed so far, the following research questions arise. By answering them throughout this research, we can come to a valid and successful solution for service deployment.

(1) How can the problem of service deployment be modeled?
(2) What is a possible solution for service deployment that utilizes different encryption schemes?
(3) How does a security aware algorithm improve upon existing solutions for service deployment in terms of vulnerability and path choices?

The general research question is:

- To what extent is it possible to devise an algorithm that utilizes different encryption schemes to improve upon service deployment?

2 RELATED WORK

Encryption is categorized into symmetric and asymmetric algorithms, most of which differ in how difficult they are to reverse and how fast encryption and decryption process takes. While some like ECC-256 are harder to break at the cost of encryption/decryption time, some, like ECC-128, are faster at encryption at the cost of security [6]. The differences in encryption schemes allows this research to investigate the possibility of using more than one of them when deploying an application to protect links of different vulnerability levels.

Security is not just about encryption. Data protection is a vast field and contains many methods to solve it. Two of the most prominent in IoT are encryption and steganography. Previous research suggests that the use of steganography can only be done for limited applications in case data is heterogeneous [7]. Encryption's ability to secure different data types makes it of more interest for this research. The scheme used for encrypting data can be changed based on circumstances and compensate for high vulnerability, high latency or both as there exists many encryption schemes that can fulfil different requirements. Earlier studies such as [8][9][2] explain the differences through performance measures. The security level of a scheme is measured by the amount of time needed to reverse it, i.e. decrypt it, without explicit permission. While many encryption schemes, like AES and ECC, can compete on that front, some can achieve it with lower time overhead. When data is sensitive, it is more often the case that latency is not as important as security and can thus be encrypted using better schemes and otherwise for nonsensitive data. Prior work, [10], shows that most encryption scheme time overheads follow a linear trend as data size increases and the gradient depends on the processing power of the encrypting machine.

The research carried out on service deployment lacks particular emphasis on security. Most papers, such as [4] and [5], assume a uniform and equal encryption for all applications, and this may limit the number of situations in which an application can be deployed. On the other hand, some papers, like [3], give more consideration to this problem by either encrypting or not based on application requirements. Our paper aims to improve upon this by opening up the possibility of multiple encryption schemes, across any number of nodes.

Existing studies on service deployment have approached the problem using different mathematical techniques including but not limited to machine learning, mixed integer programming, graph theory and artificial intelligence [11]. Most can resourcefully deploy an application, but many things differentiate them; some excel at deployment speed and others excel at memory efficiency for example. Most, however, do not consider how encryption schemes can be utilized to create a better service deployment algorithm. Study [3], on the other hand, does consider a binary approach to security. They deploy security sensitive applications to the edge but never to the cloud and either encrypt or do not encrypt data when transmitting based on their sensitivity. Paper [10] considers the overhead created by some security services, like authentication service, integrity service, and confidentiality service when calculating incurring costs of a deployment but do not consider vulnerability of edges nor use a heterogeneous encryption scheme.

3 SYSTEM MODEL

We consider a network of connected edge, cloud and end nodes as depicted in Figure 2 which we call the abstracted view of the network. Nodes may have different processing capabilities and may also support different subsets of encryption schemes depending on their type. Some nodes may be consumers of data, they are interested in data that is produced by another node, producer, in the network. Producers and consumers in a network are will be referred to as end nodes. The producer and consumer need not be different but when they are, they must be connected by some path in the network.

We define an application as a set of three values, A = <IoT *producer*, IoT *consumer*, A_L>. The producer is an end node which produces the data to be processed whereas the consumer is an end node that consumes and acts on the processed data. The producer and the consumer are not necessarily different or the same. A_L is the maximum latency allowed for data to be processed and traverse a network of nodes from the producer to the consumer. An example of an application is <IoT *producer* = trafficCamera, IoT *consumer* = policeStation, A_L = 2000ms. The producer is a simple camera on a road and the data is a video stream. The stream must be processed to measure vehicle speeds and the processed data is to be sent over a network of nodes to the consumer police station within no more than 2 seconds. The application to be deployed is the input of the service deployment algorithm which is used to determine where to deploy the application and the necessary path data must take, depending on the producer, consumer, link vulnerabilities and the latency constraint.

The output of the algorithm is simply a path through the network, defined as an ordered list of nodes such that data travelling from the given producer to the given consumer is processed only once and its transmission latency abides by the constraints of the application.

Notation	Definition
n_i	An abstracted edge or cloud node in the network
N_A	Set of all nodes in the abstracted network
N_E	Set of all nodes in the expanded network
a_i	An end node (i.e. IoT device) in the network, $a_i \in N_A$
P	A processing node, $P \in N_E$
X_i	An encryption node that encrypts data using scheme i, $X_i \in V$
Y_i	A decryption node that decrypts data using scheme i, $Y_i \in V$
I_i	An input node that accepts a directed edge coming from an output node O_i that outputs data encrypted using scheme i, $I_i \in V$
O_i	An output node that produces a directed edge towards an input node I_i that accepts data encrypted using scheme i, $O_i \in V$
E	Set of all edges in the network
e_i	Edge between two nodes, v and $u \mid v, u \in N_E$
l_i^e	Total latency of an edge e_i
t_i^e	Transmission latency of an edge e_i
v_i^e	Vulnerability cost of an edge e_i
p_i^n	Processing latency of a node n_i
e_i^n	Encryption latency of a node n_i
d_i^n	Decryption latency of a node n_i
K	Number of encryption schemes available to the network
A	The application to be deployed
A_L	The application's max latency
$D(v_0, v_1, , v_n)$	A deployment/path that starts at v_0 and ends at v_n
E_D	The set of edges in the deployment D
$l(D)$	Total latency in a deployment
$v(D)$	Total vulnerability cost of a deployment

Table 1. Variable Definitions

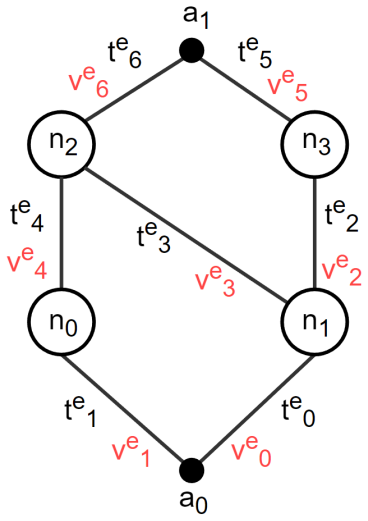

Fig. 2. Abstract view of a network with two end nodes and 4 regular nodes with transmission and vulnerability costs.

The solution's goal is to give the most secure path for data to traverse while respecting applications' latency constraints.

The latency of any link between two nodes in the abstracted network is determined by the bandwidth of the link multiplied by the amount of data being sent across it and is always positive. When we expand the abstracted network in Section 4.2, we will be creating extra links within each node to symbolize processing, encryption and decryption, where the latency of such links will be determined by the process acting on the data. For example, a link going into an encrypt node will have its latency set to the amount of time needed to encrypt data at that node while taking into consideration data size. On the other hand, the vulnerability score is a positive or negative number. A positive vulnerability is given, by an administrator, to the links in the abstracted network and it depends on external factors such as traffic on the link, sensitivity of the data or packet loss percent. The negative vulnerability cost (protection against vulnerability) is given to the new links created by the expansion algorithm and are dependant on the encryption scheme of each node. For example, an ECC-128 encrypt node may have a vulnerability score of -25 while ECC-256 may have -50 because ECC-256 is harder to reverse; The negative score is also determined by the administrator. We refer to the process of increasing vulnerability as vulnerability gain, whereas the decrease of vulnerability as vulnerability loss.

What we refer to as maximum security path is just a reference to some method that extracts the best path to achieve least vulnerability. A path of least vulnerability can be calculated in many different ways and is an optimization problem that is best left for the context at hand. Some methods for finding the path of least vulnerability will be discussed later in Section 4.4.

4 PROPOSED SECURITY-AWARE SERVICE DEPLOYMENT

4.1 The Network

An example of a network is displayed in Figure 2, where the producer, a_0, and consumer, a_1, are different. A successful algorithm must be able to output a path along the edges of the network such that data starts at the end node a_0, travels along directed edges through the network, get processed at one of the nodes n_i and finally be forwarded to its final destination a_1.

Figure 2 is only an abstraction, designed to simplify the input of the algorithm. The algorithm we are exploring must expand the network to accommodate different encryption schemes and their associated costs. An expansion is considered successful only if it allows for 3 different paths through any node. They are:

(1) A node can let data hop through it, without processing or changing its encryption scheme.
(2) A node can change the encryption scheme applied to data passing through it.
(3) A node can process and change (or preserve) the encryption scheme applied to data passing through it.

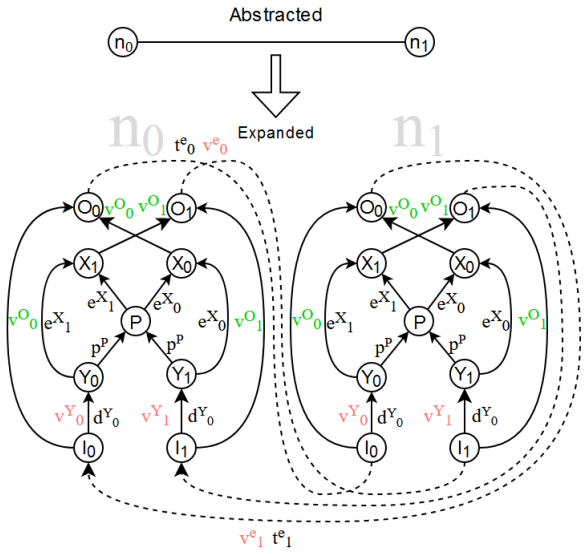

Fig. 3. Expansion processes of an abstracted graph with two connected nodes.

The three different paths are necessary to allow maximum freedom for encryption scheme selection. Figure 3 shows an expansion of an abstracted network. All nodes in an abstracted network are expanded into what we refer to as the node clusters. The cluster of a node n_i is made up of $4K + 1$ separate nodes. For each node cluster, there exists one central processing node P, a set of decryption nodes $\{Y_i\}$, a set of encryption nodes $\{X_i\}$, a set of input nodes $\{I_i\}$, and a set of output nodes $\{O_i\}$, where i is an encryption scheme $\{i|0 < i < K - 1\}$. Between a node's cluster nodes, there exists edges that fulfil the requirements of a successful expansion. An edge outgoing from a node I_i to a node O_i represents a simple hop through the node without change in encryption or processing (1). An outgoing edge from a decryption node Y_i, going into any encryption node X_j, where $i \neq j$, represents only a change in encryption scheme without processing (2) (note that X_i for some $i \leq K - 1$ can also mean no encryption). An outgoing edge from a decryption node Y_i, going into a processing node P, and then into an encryption node X_j, where $i, j < K - 1$, represents data processing and a change or a preservation of encryption scheme (3). This system of directed edges fulfils the three requirements of a successful network expansion and allows data to pass through any case. All different nodes, i.e. edge, end and cloud, are made up of the same structure.

There are some assumptions made about a node cluster. We first assume there are no transmission latencies within an edge node because the data is handled within one storage device. Second, we assume nodes are trusted, i.e. there is no vulnerability cost for data entering, but otherwise is true for traversing edges between two different nodes. We also assume that the vulnerability loss is strictly equal or less than the vulnerability gain of the next coming edge. This last assumption is necessary as to prevent negative cycles.

4.2 Network Expansion Algorithm

Networks are provided to the algorithm in abstract form and are expanded by it to account for multiple encryption schemes and processing capabilities. The pseudo code in Algorithm 1 allows different number of encryption schemes per node and achieves the expansion with worst case complexity of $K^2(|N_A| + |E|) + |N_A|K$ where K is the maximum number of encryption schemes in the network. For any edge, $e_i \in E$, there may exist up to 5 different costs; i) transmission ii) encryption, iii) decryption, iv) processing latencies and v) vulnerability costs and they depend on the target node of the edge. For example, an edge incoming to a decrypt node will have a decryption latency.

Algorithm 1 Network Expander

```
for n in Nodes do
    Create a process node
    for es in n.getEncryptionSchemes() do
        Create an input, output, decrypt and encrypt node
        Create an edge between input and decrypt node
        Create an edge between decrypt and process node
        Create an edge between process and encrypt node
        Create an edge between encrypt and output node
        Create an edge between input and output node
    for dn in n.getDecryptNodes() do
        for en in n.getEncryptNodes() do
            if dn.encryptScheme ≠ en.encryptScheme then
                Create edge between dn and en
for src, target in Edges do
    for o in src.outputNodes do
        for i in src.inputNodes do
            if o.encryptScheme == i.encrptScheme then
                Create an edge between o and i
```

4.3 Valid Deployment

A deployment is simply a path through the network that starts at the consumer and ends at the producer, while passing through one processing node. We denote such deployment as $D(t, p, s)$, where t is the producer, p is the processor and s is the consumer. Such a deployment must respect the constraints of the application to be considered valid. Validity of a deployment, $D(t, p, s)$, can be summarized with the following validation constraints.

i) One process node It must pass through one and only one processing, P, node; denoted as p in $D(t, p, s)$.

ii) Constrained latency It must have latency less than or equal to the application's max latency:

$$l(D) = \sum_{n=0}^{|E_D|} l_n^e$$

$$l(D) \leq A_L \tag{1}$$

iii) Maximum security The deployment path must be the best secured path under the required application latency constraint A_L.

4.4 Optimizing security

There are multiple ways to define security when looking for the best path. Approach 1 could be to simply pick the path with the lowest vulnerability sum. This definition may not be appropriate in some cases because it allows a well encrypted, not so vulnerable edge to make up for a weakly encrypted, very vulnerable edge. Take for example a simple directed graph where node 1 is connected to node 2, which is connected to 3, and so on until 5 (1->2->3->4->5). Assume the first 2 edges are relatively vulnerable and the last 2 are relatively safe. Assuming v_i^e at the incoming edges of any two X_i and X_j $i = j$ are equal, then minimizing vulnerability cannot differentiate between where the first two edges are weakly encrypted and the second two are strongly encrypted and vice versa.

Approach 2 is to find the difference between the vulnerability gain of transferring the data over a link and the absolute vulnerability loss of encrypting data. For example, let's say a packet is encrypted using ECC, which has a vulnerability score of -25, and travels across a link with vulnerability of 50, then the security score of going over said link would be $|50 - | - 25||$. This associates the encryption scheme with the link being traversed which avoids the problem of summing vulnerability.

Both of these approaches are possible to use depending on the requirements of the application, it is not necessarily that one will be better than the other. It is also possible to pick the path with the lowest max vulnerability, or any path which has vulnerability no greater than some number X. The method used to optimize vulnerability cost will highly depend on the nature of the application as well as the size of the network.

4.5 Service Deployment Algorithm

A network of nodes is governed by two main variables; latency and vulnerability. We consider latency to be a hard constraint and security to be elastic, i.e. a deployment must respect the maximum latency constraint but vulnerability cost can simply be optimized. This is the approach the paper will take but it just as applicable for the opposite case.

Assume we are given a set of nodes, among them are v and u. An edge between any two nodes has two costs, latency and vulnerability. We are able to find all paths between v and u with latency less than A_L using breadth first search (BFS) or depth first search (DFS). By keeping track of vulnerability cost as well, we are able to use optimization to pick a path with best security while ensuring that the latency constraint is respected. We use this concept but to an extended capacity to solve our problem of ensuring one processing node is passed.

We first begin by using an exhaustive-search algorithm to find all valid paths between the producer and consumer. We also use Branch-and-Bound to limit our exhaustive search to valid deployments. The algorithm consists of using DFS at the producer and explores outgoing edges, repeating the process for discovered nodes. Our version of DFS does not keep track of visited nodes and that is to ensure that we can find all paths, including any two that may pass through the same node. But we do keep track of visited edges to avoid cycles. Along the way, we limit the search by keeping track of two variables, i) latency budget and ii) whether data has been processed. For every edge the algorithm passes on, it deducts the latency cost from the latency budget and for whenever we pass through a processing node, we set visitedProcessNode to true. The budget variables ensure that the final list of paths extracted adheres to two of our rules of deployment in Section 4.3, i and ii. The pseudo code is provided in Algorithm 2, which tackles exhaustive search using recursion and avoids cycles by keep track of visited edges; it outputs a list of valid unoptimized paths with worst case complexity of $O(2^n)$. Finally, we run our paths through the optimizer and that outputs the most secure valid path in the network.

Algorithm 2 Branch-and-Bound

```
1:  Input expanded graph*, start, end, latencyBudget, boolean visitedProcessNode, List visitedEdges
2:  Output List validPaths
3:  paths = List
4:  for edge in start.outgoingEdges do
5:      node = edge.Target
6:      newLatencyBudget = latencyBudget - edge.Latency;
7:      newVisitedProcessNode = visitedProcessNode;
8:      newVisitedEdges = visitedEdges
9:      if newLatencyBudget < 0 then
10:         continue; // No more latency budget. Prune.
11:     if node is a processNode and visitedProcessNode and node is not end then
12:         continue; // We already processed the data. Prune.
13:     if edge is a link then
14:         if edge is in visitedEdges then
15:             continue; // Prevent cycles. Prune.
16:         newVisitedEdges.add(edge);
17:     path = new Path();
18:     path.addNode(node);
19:     if node is the end node and visitedProcessNode == true then
20:         paths.add(path); // We found a valid path.
21:     else
22:         if n is not the end node then
23:             newPaths = Branch-and-Bound(graph*, node, end, newLatencyBudget, newVisitedProcessNode, newVisitedEdges) // Reoccur
24:             for newPath in newPaths do
25:                 newPath = path + newPath;
26:                 paths.add(newPath);
27: return paths;
```

5 PERFORMANCE

5.1 Baseline

Before it is possible carry out performance measures of the proposed algorithm, it is necessary to define a baseline to which the proposed algorithm will be compared. Ideally, the baseline must be similar to existing solutions, which are not easily generalized. We decided that the baseline will traverse the same expanded network proposed in Section 4.2 using Dijkstra's shortest path algorithm on latency to

find a set of paths from the producer to the consumer through every processing node in the network. Using the set of paths generated by Dijkstra's Algorithm, we pick one at random as the deployment path. Baseline time complexity will therefore be $O(|N_A|V^2)$. The network on which we will test both algorithms will also be randomly generated, first ensuring that it is connected and second, we generate random additional links between nodes while still respecting the topology explained in Section 1.

5.2 Simulation Variables

The networks that will be simulated to measure the performance of the proposed algorithm have many variables. It is not feasible, neither of significant interest, to simulate every variable against another. There are also an infinite number of applications that can be simulated which is not feasible to do. Therefore, the variables were set to certain values, unless otherwise stated, and are highlighted in Table 2. Encryption latencies values are chosen as per paper [10]'s performance tables. Vulnerability losses are chosen such that the better encryption scheme can just mitigate the full vulnerability cost of a minimally vulnerable link (to avoid negative cycles). Link bandwidth is chosen to be in the range of lower average consumer WiFi speeds or average of cellular 4G connection while the number of encryption schemes (ES) was chosen to be 2. The chosen numbers can easily be anything else, as they are highly dependant on many other factors such as country of operation, application, time and budget.

Table 2. Simulation Variable Values

Variable	Value(s)
A_L	15 s
$\|N_A\|$	21
Link bandwidth	4-20 MB/s
Link vulnerability	50-100
Number of encryption schemes	2
ES 1 vulnerability loss	-25
ES 2 vulnerability loss	-50
ES 1 latency at end,edge,cloud	375-475,275-325,40-60 ms/MB
ES 2 latency at end,edge,cloud	700-900,575-625,90-110 ms/MB
Application data size	5MB

5.3 Time Feasibility

An important metric for any algorithm is its time complexity and that is why our algorithm will be compared to the baseline with an increasing network size. Time feasibility of the algorithm depends on many factors. Although its worst case complexity is exponential, variable values can greatly reduce its run time. The variables that affect the run time most are the application's latency constraint, A_L, and the magnitude of link latencies because the algorithm trims the branching process as soon as the latency budget is depleted. We naturally expect that the time needed to find the best path will also increase exponentially as the number of nodes increases. By plotting time taken, we will have a better insight on how feasible the algorithm is.

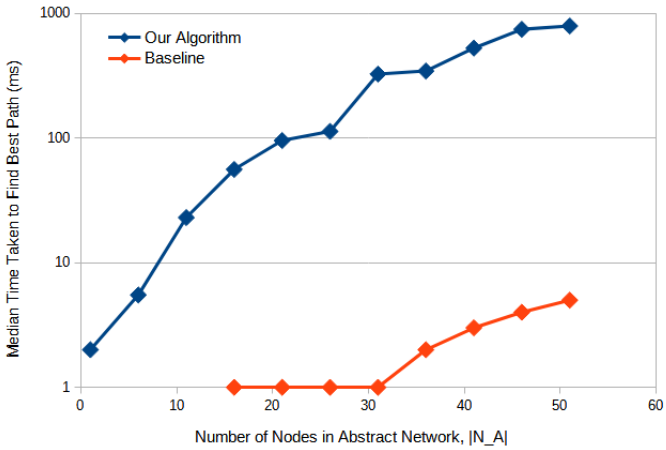

Fig. 4. Average time taken to find the best path for different network sizes. Our (blue) vs baseline algorithms (orange). 100 samples for each $|N_A|$. Logarithmic scale.

Figure 4 shows the results of the simulations with a sample size of 100 for each network size. It shows the median time needed to find the best path using our algorithm (blue) against a baseline algorithm (orange). Indeed, the time complexity of our algorithm is exponential. As the number of nodes increases, so does the number of paths that algorithm has to search and naturally, so does the time it takes. During data collection, we observed relatively large standard deviations in the samples which can be explained by the great variation in path latencies that are caused by the random generation. Although time increases exponentially on average, there may be cases where the algorithm runs relatively faster or slower. On the other hand, the baseline algorithm runs much faster, as expected, due to its relatively more manageable complexity and lack of exhaustive search. The results show that our algorithm is less feasible than the baseline as the network size increases. However, if service deployment is done offline and the deployment algorithm runs on a high-compute node, our proposal can be used.

5.4 Vulnerability

Given the goal of our research, we expect to see an improvement in the vulnerability score of the deployments generated by our algorithm compared to the baseline. Given that the baseline picks paths at random, without consideration for vulnerability, we expect that the greater the difference in vulnerability across the network, the greater the difference is between our algorithm and the baseline. We also expect that as the number of nodes increases, so does the vulnerability difference between our algorithm and the baseline and that is because the longer the paths are, the more our algorithm can select more secure routes and increase the security over the randomness of the baseline. For this section, we consider the vulnerability sum as the vulnerability score which must be minimized.

We first look at the average vulnerability generated by the time feasibility samples in Figure 5 which plots the score against the number of nodes. There is in fact an average difference of 85 that

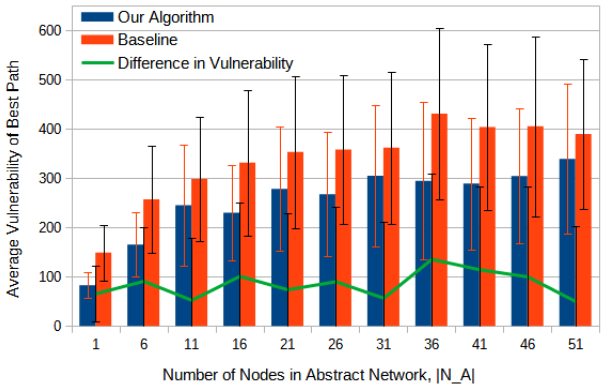

Fig. 5. Average vulnerability of the path outputted against different network sizes. 100 samples for each $|N_A|$. Our algorithm vs baseline.

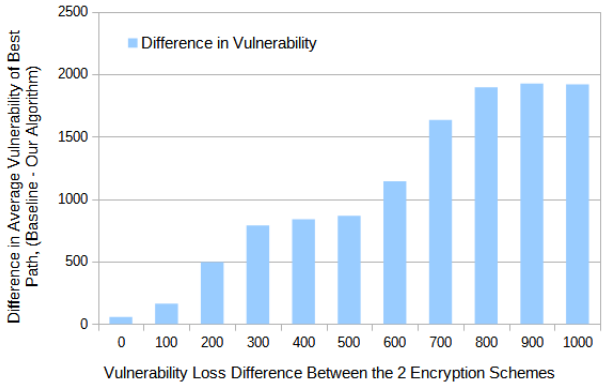

Fig. 6. Average vulnerability difference against the vulnerability loss difference of our two encryption schemes for network of size $|N_A| = 16$. 100 samples per x value.

follows a constant trend as the number of nodes increases. This makes the greatest difference at $|N_A| = 1$ in which our algorithm shows a 45% improvement against the baseline. There could be many reasons to why we see a constant difference but one that stands out could be our small A_L selection. A low latency constraint limits the number of paths our algorithm can find and thus the more likely it is that the path of least vulnerability is in fact not that different than a randomly selected path. However, another variable that could play part is the difference between vulnerability loss of the encryption schemes, that is why we devised another simulation scenario where we test the change in vulnerability over the vulnerability loss difference between encryption schemes.

Figure 6 shows the results of the simulation. The graph shows the difference in the average vulnerability of the paths between the baseline and our algorithm. As expected, the difference in vulnerability increases with greater difference in encrypt schemes. This trend appears because our baseline is a lowest cost path finding algorithm that operates on latency and will naturally prefer routes with lower costs, i.e. faster encryption/decryption times and will therefore always take the faster and less safe encryption scheme path. From this data, we can extrapolate that the greater the difference between encryption schemes, the better our algorithm will perform against the baseline. Including more encryption schemes that extend the difference in vulnerability loss, although makes the algorithm slower, increases the potential of our algorithm.

5.5 Resilience

An advantage of our solution's utilization of graph theory alongside heterogeneous encryption schemes is that it produces a greater count of valid paths between a producer and consumer. Assume some other algorithm that only considers homogeneous encryption. In cases where such algorithm can only find one path in a network that respects the latency constraint, our algorithm can find more through change of encryption schemes. Moreover, our algorithm can transfer data between two end points that do not support the same encryption by means of scheme change along the data path. This allows the proposed algorithm to work on a greater variety of networks and be less susceptible to nodes/links dropping in a network.

6 DISCUSSION

6.1 Measure Limitations

The performance measures were carried out on a consumer grade desktop. Ordinarily, such algorithms are run on data centers and dedicated hardware that allows for greater processing power. In our simulations, the sample size was limited to 100 and the maximum network size to 51 edge nodes, making it difficult to see trends for a greater range of network sizes. In Figure 4, it could be the case that if we were to increase the number of nodes as well as the latency constraint, we could see a different trend in the difference in vulnerability, which was not feasible within the scope of this research. Moreover, the proposed algorithm's code was single threaded and could have allowed for better testing if it utilized multi threaded processing especially given the recursive nature of the algorithm.

In Figures 4 and 6, the error bars were too wide to deduce trend lines through the data, making it hard to conclude with certainty that they follow specific growth patterns. The main contributing factor to this behaviour is the randomization of the network links. We test on randomly generated networks that have variations in variables and number of links, making the standard deviation of the samples too great to draw conclusive best fit lines.

6.2 Algorithm Limitations

During our testing, relatively large networks were able to be simulated when the costs and links were generated in such a way such that A_L was quickly depleted and otherwise took too long to become feasible on a consumer grade desktop. This makes the algorithm unsuitable for large networks with large latency constraints. Moreover, the time it takes for the algorithm to complete is also dependant on the number of nodes (as per Figure 4 in the network, meaning that additional encryption schemes can greatly increase the run time. This means that for our algorithm to improve vulnerability, it needs

to take longer to run, given that our algorithm excels more with a greater difference in encryption schemes.

Our algorithm is not latency optimizing. It does not differentiate between two paths based on their latency score. It only ensures that the latency constraint is respected and security is maximized. This is the major limiting factor of this approach and the baseline out performs it in every case. If latency is to be considered there will need to be a compromise between it and vulnerability and can be investigated in future work.

7 CONCLUSION AND FUTURE WORK

The system model provides an intuitive and complete formalization of the problem of service deployment. It considers all processing, encryption, decryption and transmission latencies, vulnerability costs and the processing capabilities of nodes in a network. It also supports the use of heterogeneous encryption along any network size and shape, allows for heterogeneous producer/consumer and provides an expansion algorithm that allows an administrator to create an abstracted network that is automatically expanded for the service deployment algorithm.

The expansion algorithm takes an abstracted network and expands it to support multiple encryption schemes. The service deployment algorithm uses the expanded network to find a list of all valid paths from a producer to a consumer that adhere to the rules set out in Section 4.3. The algorithm then uses the sum of vulnerability of each paths to find the most secure one in the list. The algorithm opens up the possibility to use any optimization method if the sum vulnerability is not appropriate.

The proposed algorithm provides a measurable increase in the security of a deployment when compared to a baseline algorithm. Although there is no conclusive evidence whether the improvement scales with the number of nodes, there is evidence that it does scale with the vulnerability loss difference across encryption schemes. The algorithm is also more resilient to node failures due to the exhaustively searched list of paths it outputs and its ability to utilize heterogeneous encryption when data traverses a network.

There are limitations within this research, mostly generated by the exponential time complexity associated with the exhaustive search approach. Large network sizes were not tested to a full extent due to exponential increase in paths to search, lack of dedicated hardware and no use of multi threaded programming.

There is still significant amount of work that could be done to follow up this research. To better measure the capabilities of this algorithm, it could be tested and optimized further using hardware and multi threaded programming. On the other hand, it can also be tested on less capable systems to better understand the feasibility of such an exhaustive search algorithm. Along the same lines, the algorithm can be compared to more well researched solutions to understand under what circumstances it is feasible to use the proposed algorithm.

An interesting but important aspect to investigate is how can the algorithm can be further expanded to account for a constantly changing network and also how to account for a shared network among multiple producers and consumers. Finally, formalizing the optimization problem introduced in Section 4.4 and finding a general solution can go a long way to completing this research.

REFERENCES

[1] O. Tsymbal, "Iot trends to drive innovation for business in 2022," Apr 2022. [Online]. Available: https://mobidev.biz/blog/iot-technology-trends
[2] V. Gezer, J. Um, and M. Ruskowski, "An introduction to edge computing and a real-time capable server architecture," *International Journal of Intelligent Systems*, vol. 11, p. 105, 07 2018.
[3] Z. A. Mann, A. Metzger, J. Prade, R. Seidl, and K. Pohl, "Cost-optimized, data-protection-aware offloading between an edge data center and the cloud," *IEEE Transactions on Services Computing*, pp. 1–1, 2022.
[4] K. C. Antharaju, T. R. Reddy, and N. Ramakrishnaiah, "A survey on mobile edge computing: Joint offloading and resource allocation perspective," EasyChair Preprint no. 7002, EasyChair, 2021.
[5] M. Cui, Y. Fei, and Y. Liu, "A survey on secure deployment of mobile services in edge computing," *Security and Communication Networks*, vol. 2021, pp. 1–8, 01 2021.
[6] H. Almajed and A. Almogren, "A secure and efficient ecc-based scheme for edge computing and internet of things," *Sensors*, vol. 20, p. 6158, 10 2020.
[7] M. Amjath and V. Senthooran, "Secure communication using steganography in iot environment," 12 2020, pp. 114–119.
[8] H. Tewari, "A lightweight encryption scheme for iot devices in the fog (to appear in future technologies conference 2022, vancouver, canada)," 11 2021.
[9] A. A. Hasib and A. A. M. M. Haque, "A comparative study of the performance and security issues of aes and rsa cryptography," in *2008 Third International Conference on Convergence and Hybrid Information Technology*, vol. 2, 2008, pp. 505–510.
[10] B. Huang, Z. Li, P. Tang, S. Wang, J. Zhao, H. Hu, W. Li, and V. Chang, "Security modeling and efficient computation offloading for service workflow in mobile edge computing," *Future Generation Computer Systems*, vol. 97, pp. 755–774, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18326773
[11] F. Saeik, M. Avgeris, D. Spatharakis, N. Santi, D. Dechouniotis, J. Violos, A. Leivadeas, N. Athanasopoulos, N. Mitton, and S. Papavassiliou, "Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions," *Computer Networks*, vol. 195, p. 108177, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621002322

8 APPENDIX

Source code (Maven Java Project, JGraphT), including all tests performed: https://github.com/MalekMHA/ServiceDeployer