Incentivisation of Outsourced Network Testing: View from Platform Perspective

Sultan Alasmari[1], Weichao Wang[1] and Yu Wang[2]
[1]*College of Computing and Informatics, University of North Carolina Charlotte, NC, U.S.A.*
[2]*Department of Computer and Information Sciences, Temple University, PA, U.S.A.*

Keywords: Incentivisation of Network Testers, Outsourced Network Testing.

Abstract: With the development of Security as a Service (SAAS), many companies outsource their network security functionality to security service providers. To guarantee the execution and quality of such services, a third party can help the end customer verify the enforcement of the security service level agreement (SSLA). Since individual testers often lack the capability and trustworthiness to attract many customers, a platform is needed to bridge the gap between the customers and the testers. In this paper, we investigate the incentivisation of outsourced network testing from the platform perspective. We first define the problem of cost/benefit model of the platform and identify the restriction factors. We describe multiple testing task assignment scenarios and prove that they are NP problems. Next we design heuristic algorithms for the problem. Our simulation results examine the performance of the heuristic approaches.

1 INTRODUCTION

With fast development and wide adoption of Security as a Service (SaaS) (Hawedi et al., 2018), more and more corporations start to depend on third party companies to protect their networks. However, the security services must be verified periodically to make sure that the service provider does not violate any of the service level agreements (SLA) (de Carvalho et al., 2017).

While the requirement on periodic verification sounds reasonable, it is quite hard to enforce in real life if the company depends on one or two nodes to conduct such test since their identities can be easily recognized and remembered by the service provider. In (Alasmari et al., 2020), the authors propose an approach similar to the crowd-sensing system: a platform serves as the middleman to connect the customers who need their security properties to be tested and the nodes who can conduct such tests for them. The authors investigate the expected properties of the outsourced verification services and the mechanisms to prevent either testers or customers from cheating.

While the approach in (Alasmari et al., 2020) presents an overview of the platform, it lacks the discussion on an important problem: incentive models for the platform and network testers to participate in the verification procedure. Note that to conduct network testing, a tester often needs to send out some packets that will be considered 'suspicious' or even 'malicious' under some cases. For example, to assess whether or not the network security service provider will react promptly enough to identify some malicious payload, the tester may need to send out some packets that match to the malware signature. These packets, however, may be labeled by the tester's internet service provider (ISP) and cause negative consequence (e.g. limiting the network bandwidth of the tester). Therefore, a cost-benefit model must be established and analyzed before such a platform can be deployed.

To solve this problem, in this paper we will establish a cost-benefit model from the platform point of view. We will first classify the network testing traffic based on the probability that they will be labeled as suspicious by ISP. We will then establish a model for the problem as a linear programming problem. We analyze the complexity of the problem and prove that it is NP hard. We will then design heuristic algorithms to solve the incentive model under different situations. Finally, we apply the algorithm to multiple cost/benefit scenarios of outsourced network testing and evaluate its performance.

Our paper has the following unique contributions. First, we establish an incentive model for outsourced network security testing. In our model the testers try to maximize their profit while staying under the radar

Alasmari, S., Wang, W. and Wang, Y.
Incentivisation of Outsourced Network Testing: View from Platform Perspective.
DOI: 10.5220/0010897300003120
In *Proceedings of the 8th International Conference on Information Systems Security and Privacy (ICISSP 2022)*, pages 499-506
ISBN: 978-989-758-553-1; ISSN: 2184-4356

of network security monitors. Second, we model the problem as a linear programming problem and show that it is equivalent to variations of the knapsack problem. We then design heuristic algorithms to solve the problem. Third, we apply our algorithms to multiple network testing scenarios and show that the outputs of the algorithms can explain the potential policies of the platform very well.

The remainder of the paper is organized as follows. In Section 2, we describe related work. In Section 3, we first discuss the differences between our incentive problem and several problems that also demand incentive models. We then use a linear programming model to characterize the problem and prove that it is an NP problem. We present several heuristic algorithms to solve it. Section 4 presents the quantitative evaluation results. Finally, Section 5 concludes the paper.

2 RELATED WORK

In this part, we will describe the state-of-the-art research in several directions from which we can benefit. We are especially interested in the outsourced security services, their enforcement, and incentive models in other domains that we can refer to. For Security-as-a-Service (SaaS), researchers have conducted a lot of efforts to use Service Level Agreement (SLA) to define the criteria of evaluation. For example, the EU researchers built the framework SPECS (Rak et al., 2013) and the project MUSA (Rios et al., 2016) that allowed users to prepare, negotiate, implement, and remediate security SLAs. The efforts in (Casola et al., 2020) try to embed security into the system from the design phase. In (Hawedi et al., 2018), the authors designed a different approach. They embed a lightweight IDS system at the cloud provider and allow the tenants to configure their own rules in the IDS for their VM. This approach provides a certain level of flexibility to users who have security expertise.

There are efforts in which the security provider allows end customers to participate in the configuration of security measures. For example, in (Casola et al., 2017), end users can propose security objectives and the provider will determine and allocate resources to satisfy the needs. Example approaches are built to measure parameters for network security (Wonjiga et al., 2019a) and data integrity (Wonjiga et al., 2019b). A similar approach is to standardize the interfaces to the network security functions (NSF) in network function virtualization environments (Hyun et al., 2018). The definition and enforcement of security SLAs also benefit from the advances in the new techniques such as AI and smart contract. For example, in (Wonjiga et al., 2019b), researchers applied blockchain to data integrity protection so that both end users and cloud providers can verify the results.

Incentive models have been widely used to promote participation in the activities such as crowdsourcing, crowdsensing (Khan et al., 2019), and next generation wireless networks. Compared to these models, our application scenario has some unique properties. Sending out 'suspicious traffic' to conduct network security tests may lead to identification and disconnection by ISP, thus leading to long term negative impacts. The only scenarios we can find that share the properties are the financial incentives for clinical trials of treatment for infectious diseases (Paul et al., 2021). Here the volunteers will receive cash rewards for being contacting with infectious diseases and testing new medicines. They trade the short term incentives for the potential of long term diseases.

3 INCENTIVISATION OF OUTSOURCED NETWORK TESTING

In this section, we will present the details of the incentive model the platform can use to attract more nodes to participate in the network security services as testers. We will first discuss the assumed scenarios and the functionality of different parties in the system. We will then present the cost and payment from both the platform's and tester's point of view. The incentivization model will then be formally defined as a linear programming problem with constraints. Our analysis will show that the problem is NP hard and heuristic approaches must be designed.

3.1 System Assumptions

Figure 1 shows the application scenario. We assume that an end user u uses the network security services provided by s. u wants to use a third party to verify whether or not s is satisfying the security service level agreement (SSLA). Therefore, it resolves to the network security service testing platform P. To simplify the scenario, we assume that u carefully crafts a group of packets and asks P to conduct the test. To prevent s from recognizing the source address of P and the testing traffic, P can recruit a group of *testers* to conduct the test. The test results can then be shared with u.

From the platform perspective, the overall operations must be profitable to make the business sustain-

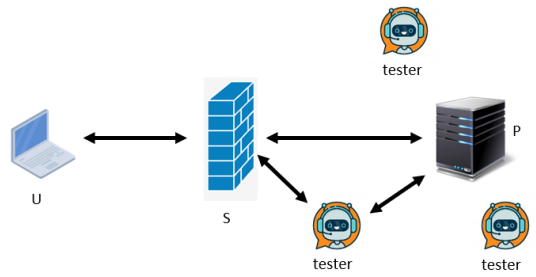

Figure 1: Application scenarios of the proposed approach.

able. For example, P will charge service fees from the user u and have to pay the testers correspondingly. Note that different testers may charge different fees to P because of the network service cost and security setup. While it is not necessarily true for P to make a profit on every request, in the long term it needs to make sure that ($income - $cost) is positive.

Different from the crowd-sensing or crowd-sourcing scenarios (Khan et al., 2019; Zhao et al., 2021) in which the participants use valuable resources such as time and battery power to accomplish tasks, the costs of network testers are usually caused by other factors. For example, many ISPs will monitor the network traffic for potential attack detection. Once a malicious or suspicious node is detected, the ISP may restrict its network bandwidth or sometimes directly disable the services. From this point of view, the testers are risking their network availability to participate in the services.

Because of the complexity of the ISP network monitoring and restriction policies, in this paper we adopt the model presented in (Desai, 2012). Here we classify the testing packets into three categories based on their sensitivity to security policies: low, middle, and high (our model could support more fine grained classification). For each node i, within a time period, the node can send out at most $R_{i,low}$, $R_{i,mid}$, and $R_{i,hi}$ packets at the low, medium, and high sensitivity levels, respectively, if it wants to avoid any restrictions by the ISP. Therefore, P must consider such restrictions when assigning tasks to the tester. Table 1 summarizes the symbols we use in this paper.

3.2 Working Procedure and the Cost Model

In this section, we will describe the working procedure of the platform and establish the cost model so that we can analyze the complexity of the overall problem.

Step 1: User u_i will submit a network testing request to platform P that consists of (H_{ui} high, M_{ui} middle, and L_{ui} low) sensitivity packets. The user agrees to pay $(H_{ui} * BP_h + M_{ui} * BP_m + L_{ui} * BP_l)$ to the platform;

Step 2: Platform P will assign the testing request to one or multiple testers. Note that different criteria and restrictions may need to be considered during this procedure. Here the platform needs to consider both its base price charge upon the user u_i and the price that it needs to pay to the tester to balance the book;

Step 3: The network testing will be conducted. Once the testing operations are accomplished, each party will pay the fee as the agreed amount. This request is complete.

3.2.1 Task Assignment Method 1

In this task assignment method, we assume that each testing request needs to be assigned to a single tester. Therefore, we can formally define the problem as:

$$\begin{aligned}
\text{maximize:} \quad & \sum_{i=1}^{|G|}\sum_{j=1}^{|J|} X_{i,j}(H_i * (BP_h - P_{j,h}) + \\
& M_i * (BP_m - P_{j,m}) + L_i * (BP_l - P_{j,l})) \\
\text{subject to:} \quad & \sum_{i=1}^{|G|} X_{i,j} * H_i \leq R_{j,h},\ X_{i,j} \in \{0,1\},\ \ j = 1\ to\ |J| \\
& \sum_{i=1}^{|G|} X_{i,j} * M_i \leq R_{j,m},\ X_{i,j} \in \{0,1\},\ \ j = 1\ to\ |J| \\
& \sum_{i=1}^{|G|} X_{i,j} * L_i \leq R_{j,l},\ X_{i,j} \in \{0,1\},\ \ j = 1\ to\ |J| \\
& \sum_{j=1}^{|J|} X_{i,j} = 1,\ \ i = 1\ to\ |G|
\end{aligned}$$

Here the objective is to maximize the profit of the platform by assigning the testing requests to testers. While P has uniform price rules for different types of testing traffic, each tester could charge different prices for the same type of packets since they need to consider the network costs. The profit P collects is determined by the difference between the two prices and the number of packets. At the same time, since a testing request can be assigned to only one tester, the total numbers of high, middle, and low sensitivity packets assigned to each tester must be within its limits. Here $X_{i,j}$ represents whether or not the requests G_i is assigned to tester j. The last constraint shows that each task is assigned to only one tester.

Complexity Analysis

From the definition of the problem, we can see that it is related to the knapsack problem. Here each tester's packet limits are the knapsacks' capacity while the testing requests are the items. We have proven the following theorem.

Theorem 1: Assignment method 1 is an NP-complete problem.

Proof: A special case of this problem is equivalent to the subset knapsack problem. ■

Table 1: Symbols used in the paper.

P	platform that provides testing services
T_j	jth tester that conducts testing services
U_i	ith user that demands testing services
BP_h, BP_m, BP_l	base price P charges for each high, mid, and low sensitivity packet
$G_i(H_i, M_i, L_i)$	ith network testing request with H_i high, M_i middle, and L_i low packets
$P_{j,l}$, $P_{j,m}$, and $P_{j,h}$	price tester j charges for sending a low, middle and high sensitivity packet
$R_{j,l}$, $R_{j,m}$, and $R_{j,h}$	the limits of low, middle, and high sensitivity packet tester j can send out
$X_{i,j}$	whether or not we assign request G_i to tester j
$Y_{i,j,H}$	# of high sensitivity packets of G_i assigned to T_j (same for mid and low)
G	the set of network testing requests
J	the set of network testers

3.2.2 Task Assignment Method 2

In this task assignment method, we assume that a testing request can be assigned to multiple testers to jointly accomplish the task. Note that the whole request must be satisfied. In other words, we will not serve only a part of the request. Under this case, the assignment procedure could become more complicated since we can draw the packet capacity from different sources. As long as the remaining capacity of high, middle, and low sensitivity packets is enough for the request, we can satisfy the request.

From the first sight, it seems that this problem can be solved with a greedy algorithm, e.g., always assign the request to the tester that charges the lowest price for a category of the packets. The greedy approach, however, may leave a part of the testing capacity unusable, thus leading to a low profit for P.

Below we show the formal definition of the problem. Here we can assign a request to multiple testers as long as all of the testing packets are covered. $Y_{i,j,H}$ represents the number of high sensitivity packets of request G_i that are assigned to tester T_j. Another restriction is the total number of high (middle, or low) sensitivity packets that are assigned to a tester is within her limit.

$$\text{maximize:} \quad \sum_{i=1}^{|G|}\sum_{j=1}^{|J|}(Y_{i,j,H}*(BP_h-P_{j,h})+Y_{i,j,M}*(BP_m-P_{j,m})+Y_{i,j,L}*(BP_l-P_{j,l}))$$

$$\text{subject to:} \quad \sum_{i=1}^{|G|} Y_{i,j,H} \le R_{j,h}, \quad j=1\ to\ |J|$$

$$\sum_{i=1}^{|G|} Y_{i,j,M} \le R_{j,m}, \quad j=1\ to\ |J|$$

$$\sum_{i=1}^{|G|} Y_{i,j,L} \le R_{j,l}, \quad j=1\ to\ |J|$$

$$\sum_{j=1}^{|J|} Y_{i,j,H} = H_i, \quad i=1\ to\ |G|$$

$$\sum_{j=1}^{|J|} Y_{i,j,M} = M_i, \quad i=1\ to\ |G|$$

$$\sum_{j=1}^{|J|} Y_{i,j,L} = L_i, \quad i=1\ to\ |G|$$

Complexity Analysis

Since now a single testing request can be assigned to multiple testers, it is different from the scenario that we discussed in Section 3.2.1. Now the testing capacity of different nodes can actually be merged together to form a large pool. Please note that since each tester may have different capacities in high, middle, and low sensitivity packets, their sums are also different. At the same time, each tester has its own price policy of the testing packets.

Since the testing capacity can be merged to form a large pool, we have a variant of the multi-dimensional knapsack problem (Laabadi et al., 2018). Here the profit of satisfying a single testing request is not a constant since each tester has its own price policy. We have proven that this variant is still an NP-hard problem.

Theorem 2: Assignment method 2 is an NP-hard problem.

Proof: A special case of this problem is equivalent to the general multidimensional knapsack problem. ■

3.3 Heuristic Algorithms

In this section we will discuss some heuristic algorithms for the task assignment problem. Depending on whether or not a testing request can be assigned to multiple testers, we investigate two different approaches.

The first heuristic algorithm we present is a variation of the Primal Effective Capacity Heuristic (PECH) mechanism for the general MDKP (Akcay et al., 2007) problem. Specifically, it is a greedy algorithm for the Task Assignment Method 1. Here we assume that a task must be assigned to a single tester. Since each tester adopts its own price model, the profit that P can collect from accomplishing the

task depends on the assigned tester. Since the platform needs to select one task from all the unassigned network testing requests, the selection method could directly impact the final result. Several examples of the selection criteria include: (1) first come first serve; (2) fit as many requests as possible to the testing capacity; or (3) randomly choose a task.

Another criteria of tester assignment is the packet capacity usage of different categories. Since we assign a whole request to one tester, we try to use the packet capacities in a balanced way to avoid the situations in which a certain type of packet capacity is used up while for other types a plenty of capacity is remained. Under this case, we will assign a request to a tester that will create the least imbalance in its remaining capacities after satisfying the request. Below we provide an example. Assume that we have two testers T_1 and T_2 who have used their capacities from high to low as follows: $T_1(71\%, 68\%, 72\%)$ and $T_2(66\%, 69\%, 67\%)$. The imbalance is defined as the largest difference between capacity usage in different categories. So the imbalance of T_1 is $72\% - 68\% = 4\%$, and for T_2 is 3%. Now assume that a request contains only high sensitivity packets. Because of the difference in capacities of testers, it will use 2% of T_1's capacity or 3% of T_2's capacity. Therefore, if we assign it to T_1, the new imbalance value will be ((71% + 2%) - 68% = 5%). While for T_2 the new value is (69% - 67% = 2%). Therefore, to reduce imbalance at testers after assignment, we will give the task to T_2.

The Task Assignment Method 2 is a little bit different since we can assign the packets to multiple testers. Therefore, a greedy algorithm will try to assign each single testing packet to the tester who charges the lowest price. Once that tester's capacity is reached, we can move on to the next cheapest tester. Note that this approach tries to maximize the profit from the current request for the platform. If the first-come-first-serve method is always adopted, it is possible that a certain type of packet capacity is used up first, thus preventing us from admitting new requests. For example, if all testing capacity of the middle level sensitivity packets is used up, we will not be able to admit any request that contains middle sensitivity packets since we do not allow a request to be partially satisfied.

To prevent this scenario from happening, we can manage the remaining capacity of different types of packets and try to maintain a balance. For example, during the request assignment procedure we can set up a threshold of the imbalance value between the remaining capacities of different categories. We will not accept any request that will break the threshold. Below we provide an example. Assume that we set the imbalance threshold at 5%. Before admitting a request, the capacity usage are 45% (low), 42% (middle), and 47% (high), respectively. Now if a task requests 1% of middle sensitivity packet capacity and 2% of high, we will not admit it since the ending capacity usage will be 43% (middle) and 49% (high) which will exceed the imbalance threshold 5%.

4 QUANTITATIVE RESULTS

In this part, we will present some quantitative results of the proposed approaches. The experiments focus on the achieved profit of different approaches, and the practicability of the task assignment models.

4.1 Achievable Profits

Based on the discussion in previous sessions, we can see that the task assignment problem is an NP problem. Therefore, in this section, we will compare the maximum profit under some scenarios to the achievable profit of the heuristic approaches. Restricted by the search space size and required computation power, we will experiment with some small scale questions.

We assume that the prices that the platform charges for each high, middle, and low sensitivity packet are \$12, \$10, and \$8, respectively. For each tester, the capacities of high, middle, and low sensitivity packets follow uniform distribution in the ranges of (900, 1100), (1800, 2200), (900, 1100), respectively. The size of the network test requests also follows uniform distribution around the expected values. They are divided into two groups. The first group have the sizes that range from 20% to 90% of the testers' capacities, while the second group range from 10% to 45%. The charging prices of the testers uniformly distribute between 92% to 100% of the platform prices. To calculate the maximum profit of the assignment, we search for all possible combinations.

In this group of experiments, we consider three task assignment mechanisms: first come first serve (FCFS), random assignment, and exclusive search (maximum profit). Here the FCFS mechanism will try to satisfy the tasks based on their arriving order. The random assignment mechanism picks from the pool of unsatisfied tasks and assign it to the tester that will generate the highest profit. In Figure 2, we show the ratio between the profits of the heuristic mechanisms and the maximum profit.

From the figure, we can see that the size of the network testing requests has an large impact on the achievable profit. For example, when the sizes of

the requests are comparable to the capacities of the testers (60% to 90%), very frequently we can fit only one or two requests into the tester's capacity. Therefore, the difference between the maximum profit and the achievable profits of the heuristic mechanisms is not large. As the size of the requests decreases (40% and 50%), we can assign multiple requests to a tester. Therefore, the selection of requests and testers could impact the profit a lot since a poorly designed assignment mechanism can often use low percentage of the capacity, thus causing increased differences between the maximum profit and achievable profits of the heuristic methods. When the size of the requests further decreases, we can fit many requests into the capacity of a single tester, and the percentage usage goes up again. From this point of view, after the platform learns the distribution of the request sizes, they can recruit testers with favorable capacity sizes that can help the platform to increase its profit. From another aspect, we can see that the FCFS and Random assignment mechanisms demonstrate similar performance.

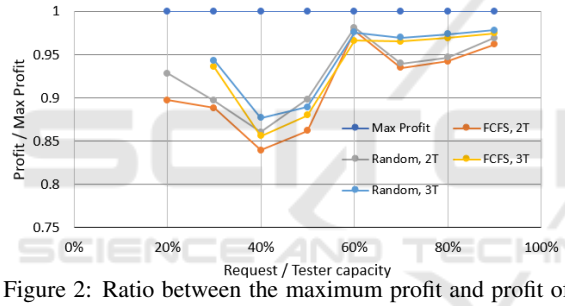

Figure 2: Ratio between the maximum profit and profit of different mechanisms.

4.2 Comparison of Heuristic Approaches

In the second group of experiments, we will compare multiple heuristic approaches. Specifically, we study 4 heuristic mechanisms: (1) FCFS/Greedy: in this mechanism, the platform adopts the first-come-first-serve policy. A single request must be assigned to one tester who can generate the largest profit through accomplishing the request. (2) Random/Greedy: in this method, we assume that all request information is available to the platform. We will then randomly pick a request from the pool and assign it to a tester who can generate the largest profit. We will continue this procedure until all testers use up their capacities, or all requests are satisfied. From this point of view, it is similar to FCFS but we choose requests randomly instead of based on the arriving order.

The third mechanism that we experiment with is called "balanced capacity usage". In this method, we try to assign a request to a tester to minimize the difference between the used capacity in high, middle, and low sensitivity packets at the tester. Specifically, for all tester T_j, we try to achieve *min* (*max capacity usage difference*). The objective is to use the capacities in a balanced way so that we can assign more requests to a single tester.

Last but not least, in the 'merged' mechanism, we allow a request to be assigned to multiple testers and each accomplishes only a part of the task. In this method, the capacities of the high, middle, and low sensitivity packets of different testers are merged into their own pools. For each request, we will choose the tester who charges the lowest price to the corresponding packet type. Since different testers may charge different prices, we may assign the high sensitivity packets to one tester and the middle sensitivity packets to another tester. In this way, we can usually achieve higher profit since we will use the tester capacities more effectively.

In this group of experiments, we assume that the platform recruits 50 testers and there are 1500 requests submitted by end users. The meanings of the parameters are similar to those discussed in Section 4.1. Since we do not have the maximum profit value, in the following figures we will illustrate the absolute profit (in $).

Figure 3 shows the quantitative results. On the X-axis, we have the ratio between the size of the network testing requests and the tester capacity. On the Y-axis, we have the profit value. We have several observations in the figure. First, for the 'merged' mechanism, since we put all the capacities of the testers into the pools of their own categories, we can continue to satisfy requests until at least one of the categories can no longer fit any request. Therefore, we can see that the profit of the 'merged' mechanism is not largely impacted by the request size. On the contrary, for the other three mechanisms, their profit will slowly decrease as the size of the requests increases.

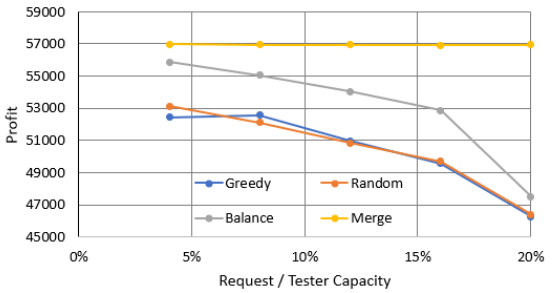

Figure 3: Relationship between the achieved profit and the size of testing requests.

Compared to 'FCFS/Greedy' and 'Random/Greedy', the 'Balanced' mechanism tries

to use the capacity in different categories in a strategic manner so that we can avoid the scenario that one category is exhausted while a large portion of capacities of other categories are still unused. From the figure, we can see that the 'Balanced' mechanism generates higher profit than the other two methods. As the size of the requests increases, more capacity at the testers could not be used. Therefore, the overall profit will decrease. Similar to the reason we discussed in Section 4.1, the 'FCFS/Greedy' and 'Random/Greedy' mechanisms do not demonstrate noticeable differences.

In Figure 4, we study how the price model of the testers impacts the profit of the platform. Specifically, we adjust the average price difference between the testers and the platform. In the X-direction, the average price difference increases from 2% to 10% of the platform price. On the Y-direction we show the profit. From the figure we can see that as the price difference changes, the profit increases almost linearly.

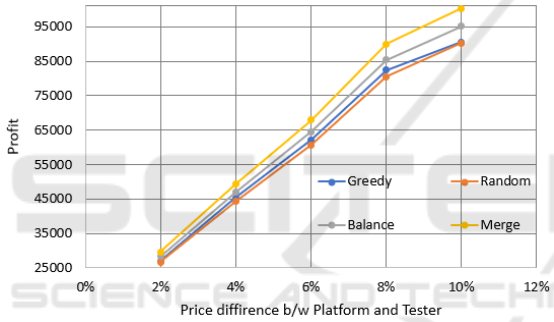

Figure 4: Relationship between the achieved profit and the prices at testers.

4.3 Future Extensions

In this part, we will discuss potential extensions to our approach.

Impacts of Price Model
This paper can be viewed as our exploration of the practicability of the outsourced network testing services. Specifically, we want to see how various factors impact the sustainability and profit of the platform. To simplify the discussion and simulation, we assume a uniform distribution of the price difference between the testers and platform and the prices do not change. In real life, both platform and testers could adopt more complicated price model. For example, they can dynamically adjust the price based on the number of requests, the number of testers, and the remaining capacity. From this point of view, maximization of the platform profit will become a more challenging question.

Privacy of Security Policies
Another concern of the outsourced network testing is the leakage of the network security policies of the end customers. Based on the construction and contents of the network testing packets, a tester may derive out the network security policies that the end user adopts. Therefore, should a tester turn malicious, it has the capability to design attacks based on the knowledge of the policies. To defend against such attacks, a potential mechanism is to distribute the testing traffic to multiple testers so that each party holds only a portion of the knowledge. The end user could also embed deceitful traffic into the request. Subsequent research is needed in this direction.

5 CONCLUSION

In this paper we investigate the problem of outsourced network test. Specifically, we focus on the working model of the platform and the possible mechanisms of task assignment. Our analysis shows that the task assignment problem is an NP problem and we need to design heuristic algorithms to assist the platform to generate higher profit. We conduct simulation to investigate the maximum profit and the achievable profits of different approaches. Our simulation shows that maintaining a balance between the remaining testing capacities of different categories will increase profit of the platform.

When we put the research problem in a larger view, the goal is to allow the platform to run the network testing services in a sustainable way. Therefore, it needs to attract enough number of end users as well as network testers to the platform. At the same time, it needs to protect the privacy of the end users. Different from the service level agreements that focus on resource usage such as CPU cycles, outsourcing of security related SLAs deserves more careful execution since a balance between safety and effectiveness of the approach must be maintained.

REFERENCES

Akcay, Y., Li, H., and Xu, S. (2007). Greedy algorithm for the general multidimensional knapsack problem. In *Annals of Operations Research*, volume 150, pages 17–29.

Alasmari, S., Wang, W., and Wang, Y. (2020). Proof of network security services: Enforcement of security sla through outsourced network testing. In *International Conference on Communication and Network Security (ICCNS)*, pages 52–59.

Casola, V., De Benedictis, A., Eraşcu, M., Modic, J., and Rak, M. (2017). Automatically enforcing security slas in the cloud. *IEEE Transactions on Services Computing*, 10(5):741–755.

Casola, V., De Benedictis, A., Rak, M., and Villano, U. (2020). A novel security-by-design methodology: Modeling and assessing security by slas with a quantitative approach. *Journal of Systems and Software*, 163:110537.

de Carvalho, C., de Andrade, R., de Castro, M., Coutinho, E., and Agoulmine, N. (2017). State of the art and challenges of security sla for cloud computing. *Computers & Electrical Engineering*, 59:141–152.

Desai, V. R. (2012). Techniques for detection of malicious packet drops in networks. Master's thesis, University of Massachusetts Amherst.

Hawedi, M., Talhi, C., and Boucheneb, H. (2018). Security as a service for public cloud tenants(saas). *Procedia Computer Science*, 130:1025–1030.

Hyun, S., Kim, J., Kim, H., Jeong, J., Hares, S., Dunbar, L., and Farrel, A. (2018). Interface to network security functions for cloud-based security services. *IEEE Communications Magazine*, 56(1):171–178.

Khan, F., Ur Rehman, A., Zheng, J., Jan, M. A., and Alam, M. (2019). Mobile crowdsensing: A survey on privacy-preservation, task management, assignment models, and incentives mechanisms. *Future Generation Computer Systems*, 100:456–472.

Laabadi, S., Naimi, M., El Amri, H., and Achchab, B. (2018). The 0/1 multidimensional knapsack problem and its variants: A survey of practical models and heuristic approaches. *American Journal of Operations Research*, (8):395–439.

Paul, M., Harbarth, S., Huttner, A., Thwaites, G., Theuretzbacher, U., Bonten, M., and Leibovici, L. (2021). Investigator-initiated randomized controlled trials in infectious diseases: Better value for money for registration trials of new antimicrobials. *Clinical Infectious Diseases*, 72(7):1259—1264.

Rak, M., Suri, N., Luna, J., Petcu, D., Casola, V., and Villano, U. (2013). Security as a service using an sla-based approach via specs. In *Proceedings of IEEE International Conference on Cloud Computing Technology and Science, (CloudComp)*, pages 1—6.

Rios, E., Mallouli, W., Rak, M., Casola, V., and Ortiz, A. M. (2016). Sla-driven monitoring of multi-cloud application components using the musa framework. In *IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 55–60.

Wonjiga, A. T., Rilling, L., and Morin, C. (2019a). Defining security monitoring slas in iaas clouds: the example of a network ids. Research Report RR-9263, Inria Rennes Bretagne Atlantique, pages 1–37.

Wonjiga, A. T., Rilling, L., and Morin, C. (2019b). Security monitoring sla verification in clouds: the case of data integrity. Research Report RR-9267, Inria Rennes - Bretagne Atlantique, pages 1–29.

Zhao, B., Tang, S., Liu, X., and Zhang, X. (2021). Pace: Privacy-preserving and quality-aware incentive mechanism for mobile crowdsensing. *IEEE Transactions on Mobile Computing*, 20(5):1924–1939.