Adversarially Adapting Deceptive Views and Reconnaissance Scans on a Software Defined Network

Jonathan Kelly
CSAIL, MIT
Cambridge, USA
jgkelly@mit

Michael DeLaus
CSAIL, MIT
Cambridge, USA
mdelaus@mit.edu

Erik Hemberg
CSAIL, MIT
Cambridge, USA
hembergerik@csail.mit.edu

Una-May O'Reilly
CSAIL, MIT
Cambridge, USA
unamay@csail.mit.edu

***Abstract*—To gain strategic insight into defending against the network reconnaissance stage of advanced persistent threats, we recreate the escalating competition between scans and deceptive views on a Software Defined Network (SDN). Our threat model presumes the defense is a deceptive network view unique for each node on the network. It can be configured in terms of the number of honeypots and subnets, as well as how real nodes are distributed across the subnets. It assumes attacks are NMAP ping scans that can be configured in terms of how many IP addresses are scanned and how they are visited. Higher performing defenses detect the scanner quicker while leaking as little information as possible while higher performing attacks are better at evading detection and discovering real nodes. By using Artificial Intelligence in the form of a competitive coevolutionary genetic algorithm, we can analyze the configurations of high performing static defenses and attacks versus their evolving adversary as well as the optimized configuration of the adversary itself. When attacks and defenses both evolve, we can observe that the extent of evolution influences the best configurations.**

***Index Terms*—Coevolution, Software Defined Networks, Network Deception, Genetic Algorithms**

I. Introduction

Cyber attacks targeting networks can be responsible for significant data breaches [15]. Attackers surreptitiously gain access to a network using phishing, social engineering or some such malicious means [6]. Once they have penetrated the network by compromising a device or endpoint, hence forward called a node, they conduct scanning to check out other nodes [13]. This reconnaissance allows them to learn the topology of the network, the configurations of nodes (that may be vulnerable to penetration), or the location of targeted data [14]. With this information they are able to exfiltrate sensitive information or further their goals, while remaining undetected. Such multi-stage threats are usually called advanced persistent threats – APTs. In this paper we focus on the *scanning stage of APTs*, motivated by the report that roughly 70% of cyber attacks are preceded by some sort of network scan [2].

Many cyber defense strategies aim to prevent adversaries from gaining access to the network. Still others focus upon what to do when a network has been compromised [1]. Herein we adopt the latter strategy by investigating *deceptive network defenses* to counter APT scanning. The idea of a deceptive defense is to represent fake, but operational, views of the network at different nodes so that a scanner will be detected before it is successful. The deceptive network routing information entraps the scan at a decoy node, "honeypot", slows it down to make it useless, or cause it to reveal itself through its (delusional) assumptions about the network.

Implementing deception is challenging on a conventional network, however it is easier on a programmable software defined network (SDN) [7]. The flexibility that SDNs provide can be used to administer a deceptive network view and dispatch it to nodes. A SDN can distort the view of the network for any node and designate decoy nodes on the network.

One existing SDN multi-component deceptive defense system, by Achleitner et al, [1], foils scanning by generating camouflaged versions of the actual network and providing them to nodes when they renew their DHCP leases. We use this system in an emulation capacity. It allows us to explicitly execute and evaluate competing adversarial strategies in the form of attacker scanning and defender configurations. We layer Artificial Intelligence (AI) in the form of a genetic algorithm (GA) on top of the system. With the GA, we are modeling the intelligence of adaptive adversaries. The GA optimizes the configurations of attacks and defense. The specific competitive, coevolutionary nature of the GA supports a continuously adapting population of attackers that each try to defeat a continuously adapting member of a population of defenses [4], see Figure 1. Specifically, our interests are in: *i)* assuming effective attacks or defenses will be intelligently optimized, *ii)* identifying and explaining effective network configurations for deception (defense) and scanning (attack), given this assumption *iii)* confirming whether adversarial Artificial Intelligence of this nature can help to anticipate strategies for defense configurations to better protect against APTs.

Our contributions can be summarized as:

1) We show how AI, here a genetic algorithm with co-evolution, can optimize both scan attacks and deceptive defenses in a network with a SDN network emulator.

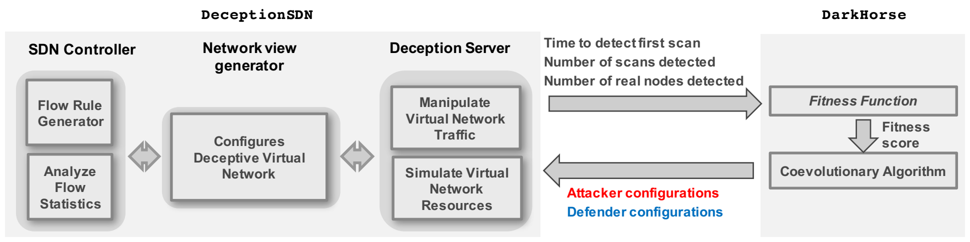

Fig. 1: Overview of SDN system with deception, named `DeceptiveSDN`, and `DarkHorse`. The two systems recreate the evolving competition between scans and deceptive views.

2) We analyze the optimized configurations for both defenders and attackers in a deceptive network subject to static adversaries or when coevolving.
3) Our experiments show that depending on whether the defender or the attacker is adapting, but the adversary is static, crowding real nodes onto one subnet will either be the optimized configuration, or the worst configuration respectively.
4) Our experiments with coevolving adversaries indicate sensitivity to the rates of evolution. When the attacker evolves more (i.e. defender is static temporarily), the attacker's optimized scanning batch sizes are small while the defender's best response is to distribute real nodes randomly. Conversely, when the defender evolves more (i.e. attacker is static temporarily), the attacker's optimized scanning batch size is 20 times larger while the defender's best response is to distribute real nodes evenly, rather than randomly.

Achleitner's deception system [1], called by us `DeceptiveSDN`, is described in Section II. Our threat model and AI system, named `DarkHorse`, are described in Section III. We analyze the experimental results in Section IV. Finally, Section V summarizes the results and future work.

II. Background

Two examples of research on deception are [5] and [1]. Passive deception works by deploying static decoy systems such as honeypots [8], [16], [12]. On a network, honeypots appear real to other nodes on the network, but do not exist in the underlying network. They do not contain any live data or information, but they can contain false information that makes them appear to be high value targets. They can be configured to prevent the intruder from accessing network enclaves and to gather information and signal unauthorized use, see Figure 2.

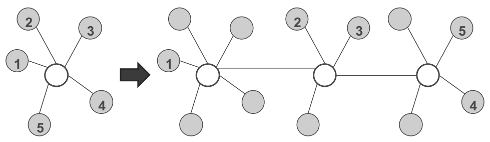

Fig. 2: Example of a virtual network view showing the placement of nodes and honeypots in order to delay adversaries from identifying real nodes

Achleitner's deception system [1], which we call `DeceptiveSDN`, defends against network reconnaissance by simulating virtual network topologies and using passive deception. It relies upon SDN flow tables and the SDN separation of controller and switch. These support agile distribution of virtual, imposter networks that overlay the true one. Each time any node on the network requests a DHCP lease, it internally is assigned a unique deceptive network view. A network view generator sets up these deceptive views so that the overall address space appears falsely large. It places real nodes on virtual subnets to increase the time it takes a scanner to find them and inserts honeypots which it can monitor for illicit activity. Using the dynamic address translation provided by the deception server, only one real node is used, but is simulated to appear as many nodes dispersed throughout the network. To ensure accurate routing, a server handles dynamic address translation between the overlay network, and true network addresses by rewriting the headers in "real time". Packets are transferred to it through flow table logic. Virtual routers simulate virtual paths so that the scan attack cannot infer the actual topology. [1] demonstrates the system against a variety of scan attacks.

As we will detail in the next section, the defensive network views and scanning attacks of `DeceptiveSDN` comprise strategic, parametrically configurable, multi-dimensional spaces. `DeceptiveSDN` serves as an example of a system where it is unknown how to optimally set up an attack or defense configuration – a situation that depends on the adversary. (Note that from now on we refer to the virtual network unless explicitly stated). `DarkHorse` utilizes `DeceptiveSDN` as an experimental platform. `DeceptiveSDN` allows `DarkHorse` to measure the outcome of an "adversarial engagement" between an attack and a defense, each specified by some set of configuration values. Outcome informs the GA about relatively better configurations, given the adversaries' conflicting objectives. We next succinctly describe our threat model and `DarkHorse`.

III. Method

Threat Model: `DarkHorse` assumes that a node on a network has been compromised, and from it there is an attacker scanning the SDN in an attempt to identify vulnerable nodes. The attacker's goals are to evade detection while scanning over as much of the network as possible and discovering the real nodes. The defender's goals are to delay the attacker, waste their resources, and detect a scan as quickly as possible. Both the attacker and defender have strategy parameters through which they can control their respective configurations (details provided ahead). The defender is assumed to generate a deceptive view of the network with fake nodes.

`DarkHorse`: To generate adaptive dynamics, `DarkHorse` runs a two population, competitive coevolutionary GA [10], [11]. One population is attacker scan configurations and the other is defensive `DeceptiveSDN` configurations.

The GA adaptively searches through the parameterized configuration spaces, see Table I. Note that we start with a small search space to validate the method. In addition, the minmax formulation during the coevolutionary search increases the problem complexity.

TABLE I: The actor, attacker (red) and defender (blue), configurations and ranges for values.

Actor	Configuration	Range
Attacker	`NMAP` IP scan batch size	$[10, 50, 100, 200]$
	Total number of IPs to scan	$[200, 300, 400]$
	IP address visit order	*random*, *local*, *seqential*, *local-seq.*
Defender	Number of real nodes (H)	$[20, \ldots, 40]$
	Min honeypots per subnet	$[1, \ldots, 10]$
	Max honeypots per subnet	$[11, \ldots, 20]$
	Number of subnets	$[3, \ldots, 6]$
	Real node distribution in subnets	*random*, *even*, *crowding*

Attackers use one of four IP visit order heuristics. They: *1) random*: uniformly visit at random from all subnets, or *2) local*: randomly visit IP addresses closer to scan with higher likelihood, or *3) seqential*: sequentially visit , starting from a random IP address, or, *4) local-seq.*: sequentially visit, starting at a random nearby IP address. We used the most efficient and the least invasive scan type, a ping scan. This is challenging for a defender to detect so it served our purposes. A defender can configure a different network view for every node. It can distribute real nodes throughout the network *1) even*, evenly, *2) random*, randomly, or *3) crowding*, by crowding them all in one random subnet. Honeypots are evenly distributed throughout subnets.

The GA passes pairs of an attack scan and defense configuration to the `DeceptiveSDN`. On the `DeceptiveSDN`, the scan is run from a randomly picked node after the defense is passed as configuration information to the deception network view generator. The `DeceptiveSDN` passes back scan performance information that the GA uses in its fitness function to calculate a performance score. This score is used by the GA to select better attacker and defender configurations to undergo evolutionary adaptation. Reflecting the objectives of the scans and defenses, the fitness scores have four components: *A)* time for defense to detect a scan (sec.), d, *B)* time to run the scan (sec.), t, *C)* number of scan detections by defender, n, *D)* and the fraction h/H which is the ratio of real nodes that were discovered to total real nodes. The defender fitness function, where α, β, and γ are weight constants for different aspects of the fitness value, is:

$$f = \alpha(1 - \frac{d}{t}) + \min(\beta, n) + \max(0, 50 - \gamma h/H) \quad (1)$$

We use $\alpha = 25, \beta = 25, \gamma = 25$. The attacker and defender fitness scores are inverses. The intuition behind this is that for an attacker the most important thing is to discover vulnerable nodes without being detected, and conversely the most important thing for a defender is to detect the scanner quickly while leaking as little information as possible.

Experimental Setup: Software for Achleitner's `DeceptiveSDN` [1] is available from https://github.com/deceptionsystem/master. We provide a competitive coevolutionary algorithm at https://github.com/flexgp/donkey_ge and the specific configurations at https://github.com/ALFA-group/darkhorseSDN. All experiments are run twice and we present the best.

We use `mininet`, a SDN network emulator [9]. First the `DeceptiveSDN` is started with an IP address space of 255 times the number of subnets. Any other network node that requests a new DHCP lease gets a uniquely generated deceptive network view. A node is chosen at random as the compromised node and `NMAP` [3] is used to conduct the reconnaissance scan on the network. We chose `NMAP` as our scanning tool as it is relatively efficient, very versatile, and would be easy for an adversary to install on a compromised node. Also it is a widely used scanner and provides a good approximation of what a malicious scan would look like. In our setup one simulation of an engagement took around a minute to evaluate, the configurations of the defender and attacker `NMAP` added some variance.

IV. Results and Discussion

A. Evolving Defenders vs a Static Attacker

Our first set of experiments consider an evolving defender, i.e. defenses adapted by the GA and a static attacker, i.e. a scan that does not change. In each of the 20 experiments we aim to learn an high performing defender configuration for a different attacker configuration. The attacker configurations are varied based on IP visit order, batch size and number of IPs to scan. We believe good defender configurations are those with more honeypots(HPs), more subnets, and fewer real nodes. They will also evenly distribute real nodes across subnets. Among the static attacks, we hypothesize that those with a lower batch size and a scan that starts locally will pose the greatest challenge for the evolving defender.

Table II top part (blue/white) presents rows grouped in sets of four, for visual interpretability. Each group represents an attacker configuration that is static except for the IP visit order.

1) Static attacker analysis: In considering the relative attack performance among the different static attacks one thing that stands out is that, on average, an attacker configuration that scans IPs with *local* has the measurments that varies the least against evolved defenders, while an attacker with *local-seq.* is the least. This confirms our hypothesis. This is especially interesting because in the wild, an attacker often starts by scanning their local subnet. Our results validate the merit in that approach by finding that it is the most difficult to evolve a defender against this behavior. Thus they lead to a recommendation that, anytime a node requests a new DHCP lease and is assigned a network view, the `DeceptiveSDN` should create a deceptive view where all other real nodes are kept off of the deceptive subnet that the requesting node is placed on. In this way, the `DeceptiveSDN` will be able to limit the impact from a scanner using a scan heuristic that prefers to scan locally.

Another result is that smaller `NMAP` batch sizes, on average, yield an evolved defender that is slightly less successful than defenders evolved against larger batch sizes. This implies that it could be more difficult to detect multiple small scans, than it is to detect long unbroken scans, and is a way in which malicious nodes can attempt to keep their scans undetected.

A third observation is that, in general, attacks that scan a smaller number of IPs perform better. While this initially seems clear, as scanning fewer IPs means less time for the defender to detect the scanner, the result is actually subtly complex. One portion of a attacker's fitness score depends on whether it evades scan detection, while the rest considers the number of real nodes discovered. Thus, when an attacker increases the number of IP addresses it scans and thereby incurs a greater risk of getting discovered, it also gains greater potential reward should it discover more real nodes. Therefore, the result that increasing the number of IP addresses leads to worse fitness, demonstrates that against an evolved defender, given this fitness function, the risk of being discovered outweighs the reward of discovering more real nodes.

2) Evolved defender analysis: As for the evolved defenders, see Table III, configurations with more honeypots and more subnets almost always are the optimized evolved solutions. This is intuitive, since more honeypots increase the likelihood that the scanner will be discovered, and more subnets increase the sparsity of real nodes, making it harder for the scan to discover them. What is noteworthy is that the average number of real nodes in the evolved defender configurations is 16.4. It seems that it would be obviously better for a defender to have less real nodes in the network, because that means that there are fewer for the attacker to discover. This is not so because the fitness function does not depend on the absolute number of real nodes discovered, but instead depends on the percentage of real nodes discovered. This design choice was made for exactly this reason, so that every defender configuration wouldn't automatically evolve to having the minimum number of real nodes. In future experiments, this is an aspect of the fitness function that could be modified.

TABLE III: Evolved defender configurations. Averaged values and experimental frequency of real node distribution strategies.

Defender Configurations	Avg Evolved Values$\pm$Std	Range
Number of real nodes	16.4 ± 2.87	10-20
Number of subnets	5.5 ± 0.7	4-6
Min honeypots per subnet	6.6 ± 2.3	1-10
Max honeypots per subnet	17.2 ± 2.6	10-20
Real node distribution strategies	*even*: 4, *crowding*: 11 *random*: 5	

B. Evolving Attackers vs a Static Defender

Next we aim to learn a high performing attacker configuration against 24 different defender configurations. We make our predictions with similar rationale to that of our first set of experiments, hypothesizing that defender configurations with more honeypots, more subnets, and fewer real nodes would be more challenging for the attacker to evolve against. Similar to our first round of experiments, we hypothesize that smaller `NMAP` batch sizes will be harder to detect by the SDN controller and thus yield better fitness scores. Table II lower half(red/white) lists the results from the static defender experiments. The rows are grouped in sets of three, each group representing a defender configuration that is constant except for variation according to real node distribution heuristic.

1) Static defender analysis: Similar to the evolved defender, static defender configurations with more honeypots and more subnets are more robust than configurations with fewer honeypots and subnets. This is intuitive since more honeypots increase the chance to detect a scan, and the more subnets spread out the real nodes, they are more difficult to find.

Defender configurations that crowd real nodes into a subnet are consistently less effective, in terms of fitness score, than either even or randomly distributed configurations. This result seems to stand in contrast to the results of the evolved defender, which most frequently evolved to have a crowded distribution of real nodes. While these results at first seem contradictory, after analysis they make sense. For the static defender, it makes sense that a crowded distribution of real nodes does poorly, as we are evolving the attacker. Over the course of the evolution, one of the attacker configurations will likely evolve to scan the space where the crowded distribution occurs, thus discovering many real nodes in one scan. The evolved defender uses the crowded distribution in the precisely opposite way, since the attacker scan is static, during the defender evolution it is able to move the crowded distribution of real nodes to a subnet where the attacker isn't scanning, thus significantly decreasing the number of real nodes that the attacker discovers. In conclusion, depending on whether the defender or the attacker is evolving, a crowded distribution of real nodes will either be the optimized configuration, or the worst configuration.

2) Evolved attacker analysis: Analyzing the evolved attacker configurations corroborates much of the preceding analysis. As evidenced by Table IV, scans with *local* are most often the optimal solutions. Similarly, for the evolved attacker, smaller `NMAP` batch sizes and a smaller number of scanned IP addresses yield better fitness values.

TABLE IV: Frequencies of IP visit order heuristics in the evolved attacker.

IP Visit Order	Frequency of Scan
random	0
local	7
seqential	14
local-seq.	3

C. Coevolution Experiment

Finally, we coevolve attacker and a defender configurations. We use a lockstep algorithm [10] that locks one population static for 3 generations while evolving its adversary, and then releases the locking, allowing the adversary just one generation to evolve. These asymmetric steps are repeated three times and simulate different rates of adaption. Note that these experiments use the percent of the total network space to scan instead of the number of IP addresses.

TABLE V: Lockstepped attackers and defenders. Configuration of best adversary at last iteration is shown.

	Configuration	**High Rate Attack Evolution**	**High Rate Defense Evolution**
Best Attack	Visit order	*seqential*	*seqential*
	`NMAP` batch size	200	10
	Ratio of IPs to scan	0.10	0.10
Best Defense	Real nodes	22	24
	Distribution heuristic	*even*	*crowding*
	Virtual subnets	6	4
	[min-max] HPs/subnet	[8-14]	[7-18]

The results for the best attack at the last generation, see Table V, show that with a faster or slower evolving defender, an evolvable attacker scan that sequentially visits local IPs (*seqential*) and scans a smaller portion of the network, is the most effective. However, depending on whether the defense evolves slower or faster than the attack, the optimized attack uses a `NMAP` batch size of 200, or 10 – twenty times larger. Reading across the Best Defense row, one distinction between fast or slow evolution is how real nodes should be distributed across subnets. *random* is never ideal, whereas *even* is better when the defense can evolve slower, and *crowding* when the defense evolves faster, is better. Another difference is the number of virtual subnets – 6 and 4 respectively for slower and faster defense evolution. A final observation, which we extract from the fitness scores, is that the defender does significantly worse during coevolution than during the static evolution scenarios previously investigated. Illuminated by the average fitness score of the evolving Defender on the static attacker was 83, whereas on the evolving attacker the fitness value was barely above 60. This indicates that an adaptive attacker could pose a threat a `DeceptiveSDN` with adaptive defenses.

V. Conclusions and Future Work

Our results from `DarkHorse` imply that deceptive SDN networks can make it more challenging for an adversary to scan the network, especially without being detected. The different configurations that the defender and the attacker use have impacts on the number of real nodes detected by the attacker, as well as the probability that the attacker is discovered. When setting up the different views to assign to nodes on the network, our experiments recommend dispersing real nodes throughout the deceptive subnets. If a specific node is a target, they indicate to keep other real nodes off of its subnet. They also indicate that a deceptive network view can defend well against many attacks, however, even with a deceptive network view, there exist attack configurations which are better performing than others. Namely, scans that use smaller batch sizes, scan with a local preference, and scan smaller IP ranges are more effective.

While our results indicate that more deceptive subnets and honeypots are better, we have not added their cost to regular network traffic. A cost would result in a balance point between the security of having more virtual subnets, and the increased delay of sending packets through virtual routers.

For future work, improving upon the sophistication of the scanning and defending, e.g. adapting how to allocate honeypots, offers one direction. `DarkHorse` also allows scenarios where more than one real node is compromised. These and different fitness functions scenarios could also be investigated.

References

[1] Stefan Achleitner, Thomas Laporta, and Patrick McDaniel. Cyber deception: Virtual networks to defend insider reconnaissance. *In Proceedings of the 2016 International Workshop on Managing Insider Security Threats*, pages 57–68, 2016.

[2] Panjwani et al. An experimental evaluation to determine if port scans are precursors to an attack. *Dependable Systems and Networks, DSN*, page 8, 2015.

[3] Gordon Lyon. Nmap network scanner. `https://nmap.org/`, 2018. [Online; accessed 6-July-2018].

[4] Erik Hemberg, Jacob Rosen, Geoff Warner, Sanith Wijesinghe, and Una-May OReilly. Detecting tax evasion: a co-evolutionary approach. *Artificial Intelligence and Law*, 24(2):149–182, 2016.

[5] Karel Horák, Quanyan Zhu, and Branislav Bošanský. Manipulating adversary's belief: A dynamic game approach to deception by design for proactive network security. In Stefan Rass, Bo An, Christopher Kiekintveld, Fei Fang, and Stefan Schauer, editors, *Decision and Game Theory for Security*, pages 273–294, Cham, 2017. Springer International Publishing.

[6] Infosec Institute. Advanced persistent threats - attack and defense. `https://resources.infosecinstitute.com/advanced-persistent-threats-attack-and-defense/`. [Online; accessed 15-August-2018].

[7] Keith Kirkpatrick. Software-defined networking. *Communications of the ACM*, 56(9), 2013.

[8] Christian Kreibich and Jon Crowcroft. Honeycomb: Creating intrusion detection signatures using honeypots. *SIGCOMM Comput. Commun. Rev.*, 34(1):51–56, January 2004.

[9] Mininet Team. Mininet - realistic virtual sdn network emulator. `http://mininet.org/`, 2018. [Online; accessed 6-July-2018].

[10] Marcos Pertierra. Investigating coevolutionary algorithms for expensive fitness evaluations in cybersecurity. Master's thesis, Massachusetts Institute of Technology, 2018.

[11] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. Coevolutionary principles. In *Handbook of natural computing*, pages 987–1033. Springer, 2012.

[12] Niels Provos and Thorsten Holz. *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.

[13] Aditya Sood and Richard Enbody. Targeted cyberattacks: a superset of advanced persistent threats. *IEEE Security & Privacy*, 11(1):54–61, 2013.

[14] Symantec. Preparing for a cyber attack. `http://www.symantec.com/content/en/us/enterprise/other_resources/b-preparing-for-a-cyber-attack-interactive-SYM285k_050913.pdf`. [Online; accessed 15-August-2018].

[15] Taylor Armerding. The 17 biggest data breaches of the 21st century. `https://www.csoonline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html`. [Online; accessed 27-August-2018].

[16] A. J. Underbrink. *Effective Cyber Deception*, pages 115–147. Springer International Publishing, Cham, 2016.

TABLE II: Static attack & defense experimental results (color for readability). *Static Attack* (top blue/white rows): The first 3 columns show attacker configurations for static attack (red font), the next 4 show the best evolved defense (blue font), and the last 4 show measurements. *Static Defense* (bottom red/white rows): The first 4 columns show attacker configurations for static attack (blue font), the next 3 show the best evolved defense (red font), and the last 4 show measurements.

Visit Order	Batch Size	Num. IPs	N Real Nodes	Real Node Dist	Subnets	Min-Max HP	Nodes Disc.	Detected Scans	1^{st} Detection(s)	HPs
random	100	500	17	*crowding*	6	6-19	17	19	18.89	98
local	100	500	17	*crowding*	4	10-18	17	25	20.47	66
seqential	100	500	20	*random*	6	6-20	9	26	18.72	77
local-seq.	100	500	19	*crowding*	6	9-12	19	24	16.66	74
random	100	400	17	*crowding*	6	8-18	17	21	18.50	104
local	100	400	20	*crowding*	6	6-16	20	18	30.84	77
seqential	100	400	14	*even*	6	8-20	3	24	16.59	107
local-seq.	100	400	20	*crowding*	6	10-15	17	18	27.11	106
random	5	400	19	*crowding*	6	7-18	9	9	34.62	96
local	5	400	18	*random*	5	8-13	4	10	32.60	77
seqential	5	400	11	*crowding*	5	3-18	3	8	42.55	66
local-seq.	5	400	15	*even*	6	4-17	3	4	59.04	78
random	5	200	13	*random*	6	4-16	6	7	28.61	60
local	5	200	19	*crowding*	5	5-18	11	8	30.53	85
seqential	5	200	11	*random*	5	8-20	2	7	46.64	78
local-seq.	5	200	17	*crowding*	6	1-20	9	6	18.40	42
random	10	200	15	*random*	5	7-19	7	5	34.63	61
local	10	200	13	*even*	4	6-14	4	8	34.57	37
seqential	10	200	17	*even*	6	8-20	4	12	14.37	87
local-seq.	10	200	15	*crowding*	5	7-13	11	8	26.45	60
N Real Nodes	**Real Node Dist**	**Subnets**	**Min-Max HP**	**Visit Order**	**Batch Size**	**Num. IPs**	**Nodes Disc.**	**Detected Scans**	**1^{st} Detection(s)**	**HPs**
20	*even*	4	1-10	*local*	25	300	8	11	22.62	26
20	*crowding*	4	1-10	*seqential*	5	400	12	2	42.74	14
20	*random*	4	1-10	*local*	10	300	4	3	26.61	19
20	*even*	4	10-20	*local*	25	200	5	19	22.54	86
20	*crowding*	4	10-20	*seqential*	50	300	20	19	16.47	73
20	*random*	4	10-20	*seqential*	10	400	8	12	22.43	76
20	*even*	10	1-10	*seqential*	5	400	3	2	24.52	54
20	*crowding*	10	1-10	*seqential*	50	400	20	11	20.50	68
20	*random*	10	1-10	*local*	5	200	2	1	85.35	54
20	*even*	10	10-20	*local*	5	200	1	9	40.72	185
20	*crowding*	10	10-20	*seqential*	50	300	20	22	23.06	214
20	*random*	10	10-20	*local*	5	200	5	9	40.64	170
20	*even*	10	40-80	*seqential*	50	200	2	4	22.13	983
20	*crowding*	10	40-80	*seqential*	50	300	11	18	18.64	683
20	*random*	10	40-80	*seqential*	5	200	2	12	30.84	998
50	*even*	10	1-10	*local-seq.*	25	200	9	13	30.58	36
50	*crowding*	10	1-10	*seqential*	5	300	15	2	68.95	52
50	*random*	10	1-10	*seqential*	5	400	3	2	97.45	48
50	*even*	10	10-20	*seqential*	25	200	7	16	20.42	205
50	*crowding*	10	10-20	*local-seq.*	5	400	20	7	30.76	231
50	*random*	10	10-20	*seqential*	50	200	2	14	22.63	183
20	*even*	20	10-20	*seqential*	25	200	2	17	16.67	379
20	*crowding*	20	10-20	*local*	5	300	12	4	24.81	347
20	*random*	20	10-20	*local-seq.*	25	300	1	18	16.58	389