Autonomic Computer Network Defence Using Risk State and Reinforcement Learning

Luc BEAUDOIN[a], Nathalie JAPKOWICZ[b] and Stan MATWIN[b]
[a]*Defense Research and Development Canada*
[b]*University of Ottawa*

Abstract. Computer Network Defence is concerned with the active protection of information technology infrastructure against malicious and accidental incidents. Given the growing complexity of IT systems and the speed at which automated attacks can be launched, implementing timely and efficient network incident mitigating actions, whether proactive or reactive, is a great challenge. We refer to the automation of action selection and implementation in this domain as Autonomic Computer Network Defence. In this work, we suggest that Autonomic Computer Network Defence can be achieved using Reinforcement Learning and dynamic risk assessment to learn the optimal action sequence, or policy, to recover from given computer network risk situations. Such a policy could then be used by commercial network management and security products to implement selected mitigating actions automatically, as risk states are sensed.

Keywords. Autonomic Computer Network Defence, Reinforcement Learning, risk, simulation.

Introduction

Autonomic Computer Network Defence (CND) aims to provide a self-protection capability of information technology (IT) networks in order to limit the risk caused by malicious and accidental events. This requires an automated controller with a policy, which selects the most appropriate action in any undesired network state. Due to the complexity and constant evolution of the CND environment, a-priori design for an automated controller is not effective. A solution for generating and continuously improving CND decision policies is needed.

A system capable of achieving autonomic CND must be able to iterate through the CND decision cycle in an automated manner. This cycle typically involves the following steps: sensing network changes, analyzing their impact, selecting an appropriate mitigation action, and implementing this action back onto the network. This process forms a control loop which employs available resources to continuously protect the IT infrastructure. Various commercial products and research prototypes support individual steps of this control loop. However, the search for an adaptive controller design capable of steering IT networks towards an acceptable and stable equilibrium in the face of security events is in its infancy. Related research areas

include autonomic computing, autonomic networking and automated security policy management [1] [2].

In our work, we investigated the suitability of different Reinforcement Learning algorithms paired with dynamic risk assessment to form the basis of an autonomic CND controller. We developed an experimental framework, which includes Discrete Event Dynamics System (DEDS) simulation and graph models of the environment, to iterate through CND policies in various scenarios and attempt to minimize business risk. We show that Reinforcement Learning algorithms can learn efficient CND policies. We also show that the difference between policies becomes less significant as the resources available to implement CND actions are increased.

1. Risk in CND Decision Making: The Proactive-Reactive Dilemma

The North Atlantic Treaty Organization (NATO) has defined Computer Network Defence as: "Actions taken through the use of computer networks to protect, monitor, analyze, detect and respond to unauthorized activity within information systems and computer networks"[1]. Generally, CND actions can be implemented either proactively or reactively, and are triggered by metrics such as up/down asset status, security alerts, disclosure of new vulnerabilities, or capacity engineering. An important difference between proactive and reactive actions is the notion of risk. A proactive action aims to reduce the probability **p** of occurrence of a damaging event (**p<1**), whereas a reactive action typically aims to reduce the damage of an actual event (**p=1**). Often, there are situations of conflicting priorities, where resources must be rationed between proactive and reactive actions. An example of such proactive-reactive dilemma would be Network Operations Centre Staff (NOC Staff) needing to decide between first patching a vulnerable system, or fixing a simultaneous outage on another system. To solve this proactive-reactive dilemma, we need a single risk metric which can account for both potential (**p<1**) and actual (**p=1**) incidents. This risk metric would also need to account for the combined effects of time, new security event arrivals and the mitigation actions selected.

This leads to a combinatorial problem whereby risk for every network state, action and event timing is path-dependent. This means that the overall risk exposure of a given situation depends on the entire sequence of actions taken to recover from it.

To illustrate this important concept, a decision tree for a simple scenario with three assets, only one of which being exposed to the Internet, is shown in Figure 1. The first decision is triggered by a new vulnerability on asset 1, which is exposed to external exploit sources. From the resulting decision tree, we show a sample of seven potential risk outcomes, depending on the timing of the first exploit event, the duration of each action, and the mitigation strategy used. Deciding to immediately patch asset 1's vulnerability (node 1) may lead to an optimal path (green arrow starting at node 2, which means that low risk is assumed from this point on). However, the same decision may also lead to the worst path, in the case an exploit occurs before the patching action is completed, and spreads to the two other assets. For this reason, isolating the vulnerable host first, prior to patching it, may be an advantageous decision path. Although isolating asset 1 results in self-denial of service, it also prevents any exploits

[1] NATO publication 3000 TI-3/TT-1162

from using this host to spread to other hosts. Once exploited, an asset has to be "fixed", which typically means restoring a clean disk image on the host.

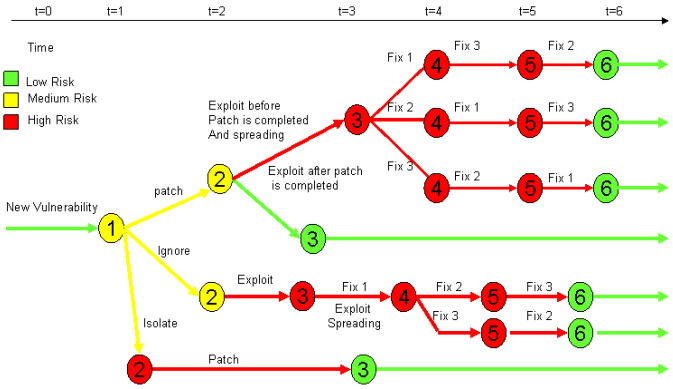

Figure 1. Example of a simple CND decision tree for a new vulnerability event

In more complex scenarios, new vulnerabilities, exploits and outages can occur at any point in time, resulting in new decision branches for each arrival, many of which can be followed concurrently, or serviced through a queue, depending on the number of available resources. If we also consider that assets may have varying levels of business importance, the number of potential outcomes quickly becomes unmanageable. This situation captures the problem we are interested in: given vulnerability, exploit and outage events, an IT network, stated business needs for IT services and limited resources, how can we decide which is the next action to take amongst options of fixing, patching, isolating risk-exposed assets, or simply waiting?

2. Reinforcement Learning for Autonomic Computer Network Defence

In a decision tree such as the one previously introduced, assessing the cumulative risk for every possible branch, in a greedy way, is not a practical strategy. We need to be able to sample this state-action space and steer exploration towards the most promising action selection strategy. This strategy is also known as a policy, and the optimization goal we seek is to minimize business risk over a given time horizon. The search for such an optimal policy is referred to as the Generalized Policy Iteration (GPI) problem, and different approaches have been used to solve it, including Reinforcement Learning [3]. Reinforcement Learning has been successfully used in complex control loop systems, namely in automated packet routing problems [4] and automated server resources allocation problems [1]. One of Reinforcement Learning's benefits is that it can be used online (adaptation to situations as they occur), offline (planning for anticipated situations), on-policy and off-policy. The two last terms refer to whether the learned policy is used to influence exploration (on-policy), or whether the policy iteration is controlled through another mechanism independently, such as random selection, human control, or heuristic rules (off-policy). Finally, Reinforcement Learning can make use of state generalization methods such as function approximators, to scale in continuous state space applications [5].

In Reinforcement Learning, an agent is rewarded after reaching a goal, and this reward is discounted to each preceding action through exploration and policy iterations (also known as *epochs*). Since our objective is to find a policy leading to the minimum risk exposure, we considered both immediate risk reduction, $\Delta R(t)$, and the integral of $R(t)$ over the full simulation period, as potential reward functions. We then experimented with various Reinforcement Learning algorithms such as Q-learning, and parameters such as eligibility traces, learning rates, discount factors and random exploration thresholds, to attempt to converge to a desirable CND policy.

3. Autonomic CND Experimentation Framework Architecture

Our experimentation framework is presented in Figure 2. It is broken down into seven main modules, shown in dark blue, and which we describe in this Section.

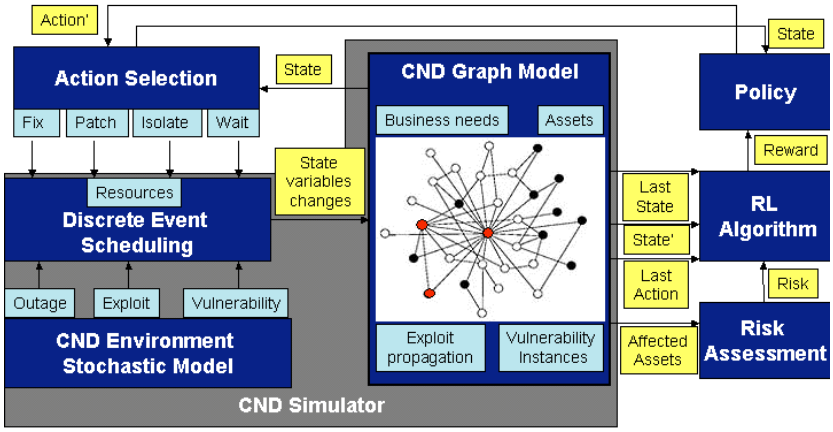

Figure 2. Autonomic CND Experimentation Framework Architecture

Action Selection: For every CND environment state change, the *Action Selection* module searches the *policy* for the best next action, *action'*, to implement given the current *state*. This is performed either through a greedy search or using a *softmax* Boltzman probability driven choice. The resulting *action'* (*fix, patch, isolate* an *asset*, or *wait*) is than passed to a *resource* and scheduled using *Discrete Event Scheduling*.

Discrete Event Scheduling: The scheduling module generates event duration and puts the *action* event in a queue, ordered by time. In some scenarios, it also receives exogenous events from the *CND Environment Stochastic Model*. The scheduling module then advances the simulation clock to the next event in the queue and communicates the associated *State variables changes* to the *CND Graph model*. We implemented this module using the DEDS Java library called ABCMod from University of Ottawa [6], which also supports random seed management and sample dataset collection.

CND Environment Stochastic Model: This module generates arrival times for *outages, vulnerabilities* and *exploits* according to predetermined probability distributions. Leaving the details of these distributions to other forums, we used

Poisson distributions, implemented using the CERN Colt Java library[2], with the following means parameter λ values: 0.0134 vulnerabilities per hour, 0.0093 exploits per hour per vulnerable/exposed host, and 0.00036 outages per hour per host (which includes maintenance related outages). These values were derived from a simple statistical and empirical analysis of incident reports[3] and public data sources[4]. Details can be found in [7] and [8].

CND Graph model: The *CND Graph model* keeps track of the status of each *assets*, their interdependencies, and their support to the *business processes* (needs). This module also enforces the rules for asset status changes (OK, vulnerable, outage or exploited), considering safeguards, actions and exogenous events. This model was implemented using the JGraphT[5] Java library.

Risk Assessment: The *Risk Assessment* module queries the *CND graph Model* for the list of *affected assets*, computes the instantaneous *risk* R(t) and its integral over the simulation run, then passes these scalars to the *RL algorithm*. The risk is updated periodically and considers cumulative effects of potential and actual damages. The dynamic risk assessment algorithm is shown in Eq. (1). Its details are kept to other forums [].

$$R(t) = \sum_{i=1}^{n} \sum_{j=1}^{m} d_i(t) * p_j(t) \tag{1}$$

Where:

- **n** is the number of affected assets;
- **m** is the number of events;
- $\mathbf{d_i(t)}$ is the damage function incurred by the business at time **t** for asset **i;**
- $\mathbf{p_j(t)}$ is the likelihood of occurrence of an exploit for a vulnerability event **j** at time **t**; 1 otherwise.

RL algorithm: Before each *action* implementation is completed, the *RL algorithm* queries the *CND Graph Model* for the current *state*. After the *action* implementation, the RL module queries the Graph Model for the new *state'* and updates the *policy* with these quantities and the associated *reward* ΔR(t), or the integral of R(t), depending on the training strategy. This module is implemented using the QConnectionist [9] Java package[6], and University of New South Whales Reinforcement Learning Java package by Time Eden, Anthony Knittel and Raphael van Uffelen[7] .

Policy: The *policy* updates its state-action map with *rewards* and *states* received from the RL algorithm. It also provides the *action selection* module with the preferred action for a given state. This was implemented using the neural network provided in the QConnectionist framework [9] for the state generalization policy, and standard Java vectors objects for the table policies.

---

[2] The Colt Java library is available at http://acs.lbl.gov/~hoschek/colt/
[3] Trouble Tickets recorded at the Canadian Forces Network Operations Centre
[4] Primarily from the National Institute of Standards and Technology (NIST)
[5] JGraphT is an open source project available at http://jgrapht.sourceforge.net/
[6] The QConnectionist source code is available at http://www.elsy.gdan.pl/
[7] Source available at: http://www.cse.unsw.edu.au/~cs9417ml/RL1/applet.html

4. Experiment

We implemented a simple CND environment model, as shown in Figure 3. It includes eleven nodes regrouped under four interconnected sites in order to create multiple service paths. One site hosts a DNS server, one provides access to the internet gateway (Router-4), and two other sites host each an email server and a client. The business needs include email communications between Email-2 and Email-5, as well as browsing the internet from Browser-3. Functional dependencies exist between the email clients and the email servers, as well as between the browser and the DNS. These business needs and functional dependencies form logical dependencies between assets, shown by orange arcs. Using this model, we ran five simulation experiments: fixing four concurrent outages, fixing eleven concurrent outages, patching eleven concurrent vulnerabilities, patching a vulnerability or fixing exploits, and patching-fixing-isolating-waiting in a continuous event arrival simulation. We used five different policies for each experiment: reinforcement learning with a table and a neural network, as well as heuristic policies including *fixing or patching assets randomly*, *fix or patch the asset with the highest value first*, and doing nothing.

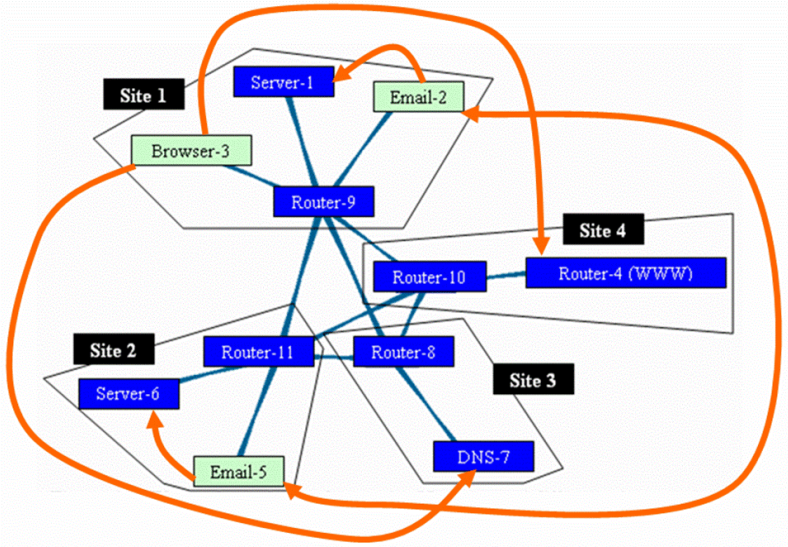

Figure 3. Simple CND environment with eleven inter-dependant assets

For each simulation, random number generator seeds for DEDS event timings were managed to assure independence of results, avoid over fitting local timing conditions and allow policy comparisons.

5. Results and Discussion

Prior to running simulations, we computed the contribution of each asset to the stated business needs. This was accomplished using an asset valuation algorithm based on the ratio of business-enabling network paths supported by each asset described in [8]. In

our simple CND environment, we found sixty-six paths through greedy, depth-first, search. The resulting asset values, presented in Table 1, were later used as damage metrics by the dynamic risk assessment module.

Asset name	Asset Value	Stated business needs
Server-1	0.7	
Email-2	0.7	0.3
Browser-3	0.3	0.3
Router-4 (WWW)	0.3	
Email-5	0.7	0.4
Server-6	0.7	
DNS-7	0.3	
Router-8	0.72	
Router-9	1.0	
Router-10	0.72	
Router-11	0.88	

Table 1. CND environment asset value results

We then conducted simulation runs for each scenarios, which we repeated in the form of epochs to train our RL policies. In the first case, the learning task was to fix four concurrent asset outages in an optimal sequence to minimize risk (risk equal damages in this case, since **p=1** for outages). Both RL policies converged to the same risk performance, which was lower than the three other heuristic policies, as shown in Table 2.

Policy	Integral of R(t)
Let risk grow	41.4
Fix random	31.73438
Fix highest	17.5375
Q-Connectionist	16.3825
Q-Learning RL Table	16.3825

Table 2. Risk integral results of different policies for fixing four concurrent outages.

A sample of the exploration of the solution space by the RL agent can be seen in Figure 4. The agent initially explored, rather randomly, various actions leading sparsely distributed risk results, than converged to action sequences optimizing its reward (minimizing risk in this case). We notice a trend around 32, which is the result for the random action sequence used for exploration. Any results larger than 32 were caused by choosing "wait" actions. These choices were punished through the risk reward and became less frequent as training progressed. We can also observe a secondary periodic value after convergence at around 17. This value corresponds to *fix or patch the asset with the highest value first*, which is an expected equilibrium since asset values contribute largely to the risk rewards for this scenario. The upper bound of this graph is approximately 41 and corresponds to *waiting* for the entire simulation period.

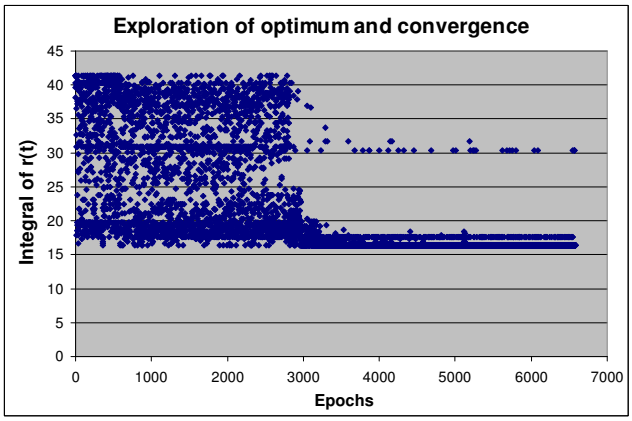

Figure 4. Exploration and convergence pattern for a neural network Reinforcement Learning agent

The other scenarios trialed had significantly larger solution spaces and statistical analysis was required to analyze the results. For this purpose we used Chi-Square metric with the Student distribution and we adjusted the number of runs to maintain our results within approximately 10%, as a quality measure. In all scenarios, the policy learned by the Reinforcement Learning agents achieved lower risk in a statistically significant way when compared to the random policy. These results are shown in Table 3.

Scenario	Random (avg risk)	Q-Learning Table policy (avg risk)	Improvement (avg risk)	QConnectionist Neural Network policy (avg risk)	Improvement (avg risk)
1. Fix 4 outages or wait.	31.73	16.38	**-15.35**	16.38	**-15.35**
2. Fix 11 outages	17.38	16.45	**-0.93**	13.42	**-3.96**
3. Patch 11 vulnerabilities	1.87	1.60	**-0.27**	1.77	**-0.10**
4. Patch 1 vulnerability or fix exploit	1.88	1.61	**-0.27**	1.81	**-0.07**
5. Patch, fix, isolate, or wait with continuous event arrivals	18933	18402	**-531**	18327	**-606**

Table 3. Reinforcement Learning policies results compared against the random policy.

In Figure 5, we show a sample simulation run for scenario 2, where 11 concurrent outages had to be fixed in an optimal sequence. The graph presents various decision points and the risk function R(t) for the five trialed policies. Because the random seed is the same for all actions timing in this case, we observed that some policies were more effective at finding root-cause outages early, hence lowering their overall risk score.

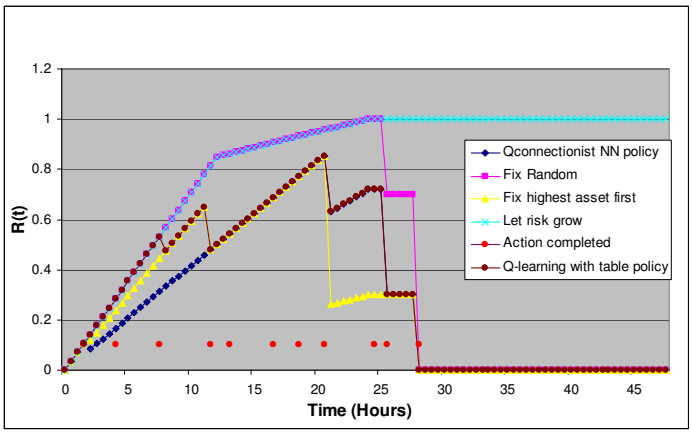

Table 5. Policy comparison for fixing eleven outages in a single simulation run.

Over a 10-year continuous simulation, we observed that policies may be locally optimal for risk, but globally very poor. Figure 6 shows the last 15 days of such a simulation run. The turquoise fill represents the random policy area under R(t). Although, the random policy had the highest risk over the full simulation period, it achieved local optimums in regions marked by the red dashed ovals in the graph.

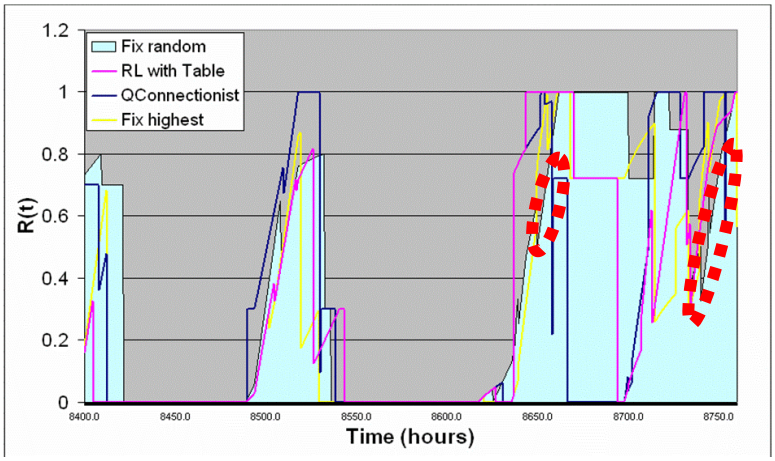

Figure 6. Comparison of policies over fifteen days of a 10-year continuous simulation.

In our experiments, the Reinforcement Learning agents seemed capable of learning generally good policies, but they could not account for all possible situations caused by the various event arrivals and action implementation delays. Even for our simple CND environment, in order to achieve globally optimal policies, more information may have been required (using a different CND state representation, as an example) as well as different learning strategies (more epochs and different parameters).

We were finally interested in evaluating the effect of resources on risk and policies. We conducted a number of 1-year simulation runs, with increasing numbers of NOC

staff resources, for two different policies: *fixing or patching assets randomly* and *fixing or patching the highest asset value first*. The averaged results are shown in Figure 7.

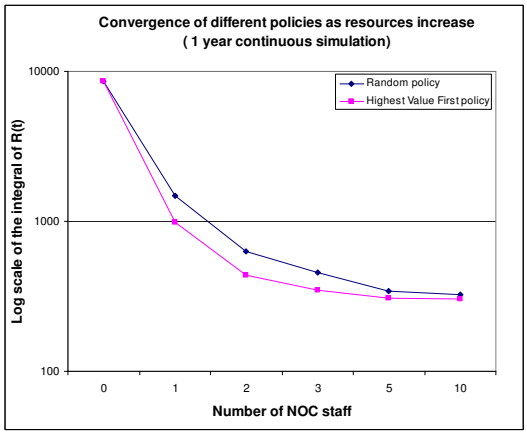

Figure 7. The effect of resource availability on overall risk.

We can observe that as the response capacity (number of NOC staff) is increased, not only is the integral of R(t) decreasing, but so is the difference between both policies[8]. Indeed, if there are no resource constraints, there is no need to prioritize responses since there is no event queue, and decision-making becomes essentially trivial. This observation clearly supports the value of automation in CND.

6. Conclusion and Future Work

In this research effort, we applied Reinforcement Learning to the problem of finding an optimal policy for Autonomic Computer Network Defence. We argued the need for a controller able to dynamically iterate through various policies and retain the best performing one. We have shown that risk maybe a good metric to steer such a controller, as long as it accounts for actual and potential events. We presented our experimentation framework and validated our concept using a simple CND environment and five scenarios. Our results show that Autonomic CND using risk states and Reinforcement Learning is possible, but that policies obtained, although generally good, did not represent global optimums.

As future work, we propose investigating further dynamic risk assessment algorithms and methodologies. We also suggest investigating different Computer Network Defence state representations, for use in conjunction with Reinforcement Learning agents herein tested, to see if better policies can be obtained. Namely, we propose investigating text mining techniques, including feature extraction, and consider modeling the CND environment as a "*bag-of-words*" to leverage these techniques. Finally, we suggest looking into scalability issues, namely investigating distributed

[8] Note that the upper bound of Figure 7 represents having no response capacity, which makes all policies equivalent by default.

policies and the use of Collaborative Reinforcement Learning to achieve superior risk results and stability.

References

[1] Tesauro, *Reinforcement Learning in Autonomic Computing*, IBM T.J. Watson Research Center, IEEE 2007.

[2] Benjamin, Pal, Webber, Atighetchi, Ruber, *Automating Cyber-Defense Management*, ACM Workshop on Recent Advances in Intrusion Tolerant Systems, 2008.

[3] Sutton, Barto, *Reinforcement Learning:An Introduction*, MIT press, 1998.

[4] Boyan, Littman, *Packet Routing in Dynamically Changing Networks: an RL approach*, Advances in Neural Information Processing Systems, Morgan Kaufmann, San Francisco CA (1993), volume 6, 671-678.

[5] Sutton, McAllester, Singh, Mansour, *Policy Gradient Methods for Reinforcement Learning with Function Approximation*, Advances in Neural Information Processing Systems 12, 2000.

[6] Birta, Arbez, *Foundation on Modeling and Simulation*, University of Ottawa, 2006.

[7] Alhazmi, Malaiya, *Quantitative Vulnerability Assessment of Systems Software*, Reliability and Maintainability Symposium, 2005.

[8] Beaudoin, Japkowizc, Matwin, *Autonomic Computer Network Defence Using Risk States and Reinforcement Learning*, Thesis manuscript to be submitted, University of Ottawa, 2009.

[9] Kuzmin, *Connectionist Q-Learning in Robot Control Task*, Riga Technical University, 2002.

[10] Cao, *From Perturbation Analysis to Markov Decision Processes and Reinforcement Learning*, DEDS: Theory and Application, 2003.

[11] Chairman of the Joint Chiefs of Staff Instruction, *Information Assurance and Computer Network Defense*, US DoD, 2004.

[12] Dobson, Denazis, Fenandez, Gaiti, Gelenbe, Massacci, Nixon, Saffre, Schmidt, Zabonelli, *A survey of Autonomic Communications*, ACM Autonomous and Adaptive Systems, Vol. 1, No. 2, 2006.

[13] Kotenko, *Multi-agent Modelling and Simulation of Cyber-Attacks and Cyber-Defense for Homeland Security*, IEEE International Workshop of Intelligent Data Acquisition and Advanced Computing Systems, 2007.

[14] Lefebvre, Grégoire, Froh, Beaudoin, *Computer Network Defence Situation Awareness Information Requirements*, MILCOM 2006.

[15] Moitra, Konda, *The Survivability of Network Systems: An Empirical Analysis*, CMU SEI, 2000.

[16] Ryan, *iWar: A new threat, its convenience – and our increasing vulnerability*, NATO review, 2007.