

Markov Decision Based Filtering to Prevent Network Instability from Control Plane Poison Messages*

Xiaojiang Du, Mark A. Shayman

*Department of Electrical and Computer Engineering, University of Maryland, College Park, MD
{dxj, shayman}@eng.umd.edu*

Ronald A. Skoog

*Telcordia Technologies, Red Bank, NJ
rskoog@research.telcordia.com*

Abstract: Poison message failure propagation is a mechanism that has been responsible for large scale failures in both telecommunications and IP networks. We propose a combination of passive diagnosis and active diagnosis to deal with the poison message problem. This paper focuses on the active diagnosis problem in which filters are dynamically configured to block suspect protocols or message types. While message filtering can potentially interrupt failure propagation and aid in identifying the poison message type, it can negatively impact the control and management of the network. We formulate this tradeoff as a Markov decision process (MDP) whose solution is the optimal policy for determining which message types to filter at each time step. The size of the state space makes it impractical to use traditional techniques to solve the MDP. Consequently we use a combination of reinforcement learning and feature-based Q-factor approximation to obtain a suboptimal policy. Simulation based experiments indicate that this policy performs significantly better than a heuristic policy that is well known in fault management applications.

1. Introduction

There have been a number of incidents where commercial data and telecommunication networks have collapsed due to their entering an unstable mode of operation. Several failure propagation mechanisms can cause unstable network [1] [4]. Here we focus on one of these that we call the ‘poison message’ failure propagation mechanism: A trigger event causes a particular network control/management message (the poison message) to be sent to other network elements. Some or all of the network elements have a software or protocol ‘bug’ that is activated on receipt of the poison message. This activated ‘bug’ will cause the node to fail with some probability. If the network control or management is such that this

message is persistently passed among the network nodes, and if the node failure probability is sufficiently high, large-scale instability can result. Several such incidents have occurred in telecommunication and other networks, such as an AT&T telephone switching network incident in 1990 [7]. In this case, the poison message was an ordinary control/management message passed between switches; there was nothing abnormal about the message itself. It was a software defect in the nodes that caused the message to have a ‘poison’ effect. We are also aware of an incident in which malformed OSPF packets functioned as poison messages and caused failure of the routers in an entire routing area for an Internet Service Provider.

Our task is to design a fault management framework that can identify the message type, or at least the protocol, carrying the poison message, and block the propagation of the poison message until the network is stabilized. We make the following three assumptions:

1. We use centralized network management.
2. The recent communication history (messages exchanged) of each node in a communication network can be recorded.
3. Because the probability of two message types carrying poison messages at the same time is very small, we assume there is only one protocol carrying the poison message when such failure occurs.

There are several ways to record the communication history. Here we assume we can put a link box at each link of the network. The link box can be used to record messages exchanged recently in the link. The link box can also be configured to block certain message types or all messages belonging to a protocol. We refer to the blocking as message or protocol filtering.

We suggest combining both passive diagnosis and active diagnosis to find out the poison message.

* Research supported by DARPA under contract N66001-00-C-8037.

Passive diagnosis includes Finite State Machine method, Correlating Message method [1] and Using Node Failure Pattern method [2]. The output of passive diagnosis will be a probability distribution that indicates for each protocol (or message type) an estimated probability that it is responsible for the poison message.

2. Active diagnosis

In active diagnosis, filters are dynamically configured to block suspect protocols or message types. Message filtering can be a valuable tool in helping to identify the culprit message type. For example, if a single message type is blocked and the failure propagation stops, this provides strong evidence that the blocked message type is the poison message. On the other hand, if the propagation continues, that message type can be ruled out.

In addition to its use as a diagnostic tool, filtering offers the possibility of interrupting failure propagation while the culprit message type is being identified. For example, all suspect message types can be initially blocked stopping failure propagation. Then message types can be turned on one-by-one until propagation resumes. While this approach may be attractive in preventing additional node failures during the diagnostic process, disabling a large number of control messages may result in unacceptable degradation of network performance. Consequently, the decision making for filter configuration must take into account tradeoffs involving the time to complete diagnosis, the degradation of network performance due to poison message propagation for each of the suspect message types, and the cost to network performance of disabling each of those message types. Each decision on filter configuration leads to further observations, which may call for changing the configuration of the filters. This suggests that policies for dynamic filter configuration may be obtained by formulating and solving a *sequential decision problem* [6] [8].

- At each stage, the state consists of a probability distribution vector with a component for each message type potentially carrying the poison message, and the recent history of the node failures.
- Based on the current state, a decision (action) is made as to how to configure filters.
- When new node failures are observed, the state is updated based on the current state, action and new observation.

Actions are chosen according to a policy that is computed off-line based on optimizing an appropriate objective function. We modeled the sequential decision problem as a Markov Decision Problem. The MDP model is given in the following.

2.1 The Markov decision problem

There are N control message types in the network, which are indexed as $[1, 2, \dots, N]$. One of the types is faulty. We want to find an optimal policy to identify the faulty message type and minimize the expected overall costs.

Possible actions (controls) at each time step are: (1). Turning off one message type; (2). Turning off several message types; (3) Do nothing. (The actions may also include some tests that have less cost than turning off a message type or protocol.)

Given: A probability vector P_0 (from passive diagnosis) whose components give the belief probabilities for each message type being poison, and current node status as the initial state.

We assume a node will either fail in a short time after it receives a poison message or not fail this time. I.e., we exclude the possibility that a node becomes 'infected' by a poison message but fails much later. Then a node status can be only one of two states (normal or failed), which means the node status can be fully observed. For each node in the network, an integer number (0 or 1) is used to denote its status. 0: normal; 1: failed.

We formulate the problem as a stochastic shortest path problem [3]. The horizon is in effect finite, but its length is random and may be affected by the policy being used. There are cost-free termination states.

Termination states: There are more than one terminal states. They are the states where there is some message type j such that the belief probability p_t^j that j is poison is at least $1 - \epsilon$ (ϵ is a given threshold close to 0), and all other probabilities are close to 0, which mean the poison message is found with a desired confidence level. Terminal cost: $J_t = 0$, for the poison message is found.

We want to find a stationary policy μ that minimizes an expected total cost. The total cost (or the objective function) is given later.

2.2 The state space

First we give some definitions. A vector is used to denote the status of each node in the network. In the following, k is the discrete time index.

$$S_k = [s_k^1, s_k^2, \dots, s_k^M]^T, \text{ where } M \text{ is the}$$

number of nodes in the network, $s_k^m = 0$ or 1 .

Let d_k be the random variable representing the identity of the poison message type. $d_k = j$ means the poison message type is j at time k , where $j = 1, \dots,$

N , and N is the number of message types used in the network.

Since the poison message type does not change, we have: $P(d_{k+1} = j | d_k = i) = 1$, only when $j=i$; otherwise, $P(d_{k+1} = j | d_k = i) = 0$ (1)

Define the history process up to time k as $I_k = (Z_0, \dots, Z_k, U_0, \dots, U_{k-1})$, where U_k, Z_k are the action and observation at time k respectively. Z_k is the set of newly failed nodes at time k .

$P_k = P(d_k | I_k)$ is the conditional probability vector of poison message type.

$P_k = [p_k^1, p_k^2, \dots, p_k^N]^T$, where p_k^j is the conditional probability of message type j being the poison one given the observed history up to time k , i.e., $p_k^j = P(d_k = j | I_k)$. Then P_k is a sufficient statistic that contains all information (except node status information) embedded in the history process for control [3].

We make the simplifying assumption that the order of failed nodes does not matter in determining which nodes will fail at next time step. If we choose the system state as $[S_k, d_k]$, and the action is given above, then it is a Partially Observed Markov Decision Process (POMDP), since part of the state d_k is not observable.

Instead, we reformulate the problem as a Markov Decision Process by using the sufficient statistic P_k :

State: The system state is $X_k = [S_k, P_k]$.

Action: $U_k = [u_k^1, u_k^2, \dots, u_k^N]^T$. If protocol j is turned off at time k , $u_k^j = 1$. Otherwise $u_k^j = 0$.

2.3 The cost function

There are three kinds of costs.

1. There is a cost associated with turning off each message type j – call it g_1^j . If the message type turned off is the poison one, the cost is zero, otherwise the cost is g_1^j .

2. There is a cost associated with a node i being down--call it g_2^i . This cost may be different for different nodes. We assume that the total cost is additive over the failed nodes. It would be more realistic, but computationally more difficult, to allow the cost to depend in a more complex way on the set of failed nodes. We do however take into account

whether the set of failed nodes results in a disconnected network.

3. There is a cost when the network is disconnected by the failed nodes--call it g_3 .

Define $N_k^t = \{j | u_k^j = 1, 1 \leq j \leq N\}$ as the index set of all the message types turned off at time k .

The cost function for each time step k is $g(X_k, U_k) = g(S_k, P_k, U_k) = f_1(P_k, U_k) + f_2(S_k) + f_3(S_k)$ where f_1 is the cost of blocking one or more message types, f_2 is the cost of failed nodes, and f_3 is the cost of network disconnectivity.

A simplified cost function is given as: (note $s_k^i = 1$ means node i is down at time k)

$$g(X_k, U_k) = \sum_{j \in N_k^t} (1 - p_k^j) \times g_1^j + \sum_{i=1}^M s_k^i \times g_2^i + g_3 \times L_k$$

where $L_k = 1$, if the network is disconnected.

Otherwise $L_k = 0$. Note that the cost of filtering a particular message type j is weighted by the current conditional probability that it is not the poison one.

2.4 The objective function

We want to find a stationary policy μ that minimizes an expected total cost:

$$J^\mu(X_0) = \lim_{N \rightarrow \infty} E \left\{ \sum_{k=0}^{N-1} g_k(X_k, \mu(X_k)) \right\}.$$

This is the expected cost to reach a terminal state--i.e., a state in which the poison message type has been identified with required degree of confidence--starting at the initial state X_0 .

2.5 The state transition probability

We want to obtain the state transition probability:

$$P(X_{k+1} | X_k, U_k) = P(S_{k+1}, P_{k+1} | S_k, P_k, U_k)$$

Update of S_k

Assume there is no reboot¹ of failed nodes. We make the simplifying assumption that what happens to different nodes at the same time k are independent of each other. Then the update of S_k is given as follows.

For each node i , if $s_k^i = 1$,

$$P(s_{k+1}^i = 1 | s_k^i = 1) = 1, \quad P(s_{k+1}^i = 0 | s_k^i = 1) = 0;$$

If $s_k^i = 0$,

$$P(s_{k+1}^i = 1 | S_k, P_k, U_k)$$

¹ We also have a model that considers node reboot.

$$\begin{aligned}
&= \sum_{j=1, j \in N_k^i}^N \{P(\text{node } i \text{ fails} \mid \text{poison message is } j, \\
&\quad \text{node status } S_k) * P(\text{poison message is } j)\} \\
&= \sum_{j=1, j \in N_k^i}^N p_i^j(S_k) \times P_k^j
\end{aligned}$$

And, $P(S_{k+1}^i = 0 \mid X_k, U_k) = 1 - P(S_{k+1}^i = 1 \mid X_k, U_k)$

Note: $p_i^j(S_k)$ can be determined from the simulation and online estimation.

Update of P_k

For each message type j , ($j=1, \dots, N$)

$$\begin{aligned}
p_{k+1}^j &= P(d_{k+1} = j \mid I_{k+1}) \\
&= P(d_{k+1} = j \mid Z_0, \dots, Z_k, Z_{k+1}, U_0, \dots, U_{k-1}, U_k) \\
&= P(d_{k+1} = j \mid S_0, \dots, S_k, S_{k+1}, U_0, \dots, U_{k-1}, U_k) \\
&= P(d_{k+1} = j \mid I_k, S_{k+1}, U_k) \\
&= P(d_{k+1} = j, S_{k+1} \mid I_k, U_k) / P(S_{k+1} \mid I_k, U_k) \\
&= \sum_{i=1}^N \{P(d_k = i \mid I_k) \times P(d_{k+1} = j \mid d_k = i, U_k) \\
&\quad \times P(S_{k+1} \mid I_k, U_k, d_k = i, d_{k+1} = j)\} / P(S_{k+1} \mid I_k, U_k)
\end{aligned}$$

(from (1), only when $i=j$,

$$\begin{aligned}
&P(d_{k+1} = j \mid d_k = i, U_k) = 1; \text{ otherwise it is } 0.) \\
&= P(d_k = j \mid I_k) \times P(S_{k+1} \mid I_k, U_k, d_{k+1} = j) \\
&\quad / P(S_{k+1} \mid I_k, U_k)
\end{aligned}$$

$$= \partial_2 \times p_k^j \times P(S_{k+1} \mid I_k, U_k, d_{k+1} = j)$$

where $\partial_2 = 1/P(S_{k+1} \mid I_k, U_k)$ is a normalization constant independent of j .

So we have,

$$p_{k+1}^j = \partial_2 \times p_k^j \times P(S_{k+1} \mid I_k, U_k, d_{k+1} = j)$$

Note: $P(S_{k+1} \mid I_k, U_k, d_{k+1} = j)$ can be approximately determined from simulation

3. Experimental results

Since the state space is very large for the sequential decision problem, it is impossible to solve it by using classic MDP solution techniques such as value iteration or policy iteration. One heuristic policy that we have considered is to block a single message type during each decision time step. The message type selected for blocking is the one that has the smallest ratio $E[C]/p$ where the $E[C]$ is the expected cost (in terms of network performance) associated with blocking the message type for a time step, and p is the current estimate of the probability that the message type is the poison one. The use of these ratios to order diagnostic tests for network fault

management has been discussed in [5] [6] [8]. The performance of the heuristic policy is compared with another policy obtained by using reinforcement learning in conjunction with Q-factor approximation. Detailed descriptions of this and related techniques can be found in [9] [10].

The (optimal) Q-factor at state i with action u is: $Q(i, u) = g(i, u) + E_j \{J(j)\}$, where $J(j)$ is the

minimum achievable cost to reach termination starting from state j —i.e., cost to go from state j under an optimal policy. (Note that j represents a state here, not a message type as before.) The interpretation is that $Q(i, u)$ is the expected cost of starting in state i , initially taking the action u , and thereafter choosing actions according to an optimal policy. Given the Q-factor, $J(i)$ is obtained by minimizing $Q(i, u)$ over all admissible u . Furthermore, the minimizing u (if unique) gives the action that would be taken in state i by an optimal policy. Thus, determining $Q(i, u)$ for all state-action pairs (i, u) implicitly specifies the optimal policy.

In our problem, the state i consists of the status (up or down) of each node together with the probability vector whose components give the belief probabilities for each possible message type being poison. The number of values of the node status vector is exponential in the number of nodes, while the number of values of the probability vector (assuming the space of probabilities is discretized) is exponential in the number of message types. Thus, the problem suffers from the well known ‘curse of dimensionality’ for dynamic programming.

To overcome this computational problem, we use parametrized Q-factors. Instead of associating a value with each state, one uses a parametric form to approximate the optimal Q-factor. The parametric form we consider is known as a *linear approximation architecture*: $\tilde{Q}(i, u, r) = \phi(i, u)^T r$

The Q-factor approximation is expressed as a linear combination of so-called features. Each feature is a (possibly nonlinear) function of (i, u) . The coefficients in the linear combination are the components of a parameter vector r . A gradient descent type algorithm is used to tune the parameter values to attempt to get a good approximation for the actual Q-factor. It is known that the success of such approximations depends on how the feature vector $\phi(i, u)$ is chosen. The feature vector $\phi(i, u)$ is meant to capture those ‘features’ of the state i and action u that are considered most relevant to the decision making process. Usually, the feature vector is handcrafted based on available insights on the nature of the problem, prior experience with similar

problems, or experimentation with simple versions of the problem.

3.1 TD (λ) estimation

We use online gradient-based Temporal Difference TD (λ) method for approximating the optimal Q-factor $Q(i, u)$ by optimizing the parameter r .

The algorithm is stated below, where z_k is called the eligibility trace [9] [10].

1. Initialize r and $z_0 = 0$.
2. Repeat:

At state i_k , take action $u_k = \arg \min_{u \in U_k} \tilde{Q}(i_k, u, r)$ that is ϵ -greedy to the current Q-factor. I.e., most of the time, we choose the greedy action, the action that minimizes the current Q-factor approximation for the current state. But with some small probability ϵ , we choose a non-greedy (e.g., random) action. This is to insure sufficient exploration of the state and action space. Then observe one step cost $g(i_k, u_k)$, and next state i_{k+1} . Choose next action: u_{k+1} that is ϵ -greedy with respect to $\tilde{Q}(i_{k+1}, u, r)$.

Calculate TD:

$$d_k = g(i_k, u_k) + \tilde{Q}(i_{k+1}, u_{k+1}, r) - \tilde{Q}(i_k, u_k, r).$$

Update z_k :

$$z_k = \lambda z_{k-1} + \nabla_r \tilde{Q}(i_k, u_k, r)$$

$$= \lambda z_{k-1} + \phi(i_k, u_k)$$

$$\text{or } z_k = \sum_{m=0}^k \lambda^{k-m} \phi(i_m, u_m)$$

Update r : $r := r + d_k z_k$

Until terminal state.

3.2 Test results

First we need to determine how the feature functions should be defined. Once the features have been chosen, we use the algorithm above to tune the parameters in the vector r to obtain a good approximation for the optimal Q-factor. This approximate optimal Q-factor in turn determines a policy. This is the policy that is greedy with respect to the approximate Q-factor; i.e., for a state i , it chooses the action u that minimizes the approximate Q-factor. We then compare the performance of the greedy policy with that of the heuristic policy. We perform two types of comparisons. In the first comparison, we fix a typical initial state and use

Monte Carlo simulation to estimate the expected cost to go under each policy. In the second comparison, we allow the initial state to vary. For each choice of initial state, we generate ‘parallel’ trajectories following each of the two policies. For each initial state, we compare the sample path cost to go under the two policies. To simplify the computations in this section, we restrict the action space for the MDP to include only actions that block a single message. I.e., we exclude actions that block multiple message types simultaneously.

3.2.1 Q-factor structure

To complete the definition of the linear approximation architecture, we must select the feature functions. We first tried linear functions of S_k, P_k & U_k (defined in Section II), but they do not provide good approximation for the Q-factor. Then we tried quadratic functions of S_k, P_k & U_k , and obtained much better results. All the test data is from our OPNET testbed that simulates an MPLS network in which poison messages can be carried by BGP, LDP, or OSPF [1]. There are 14 routers ($M=14$) and 6 different message types ($N=6$) in the testbed. After considerable trial and error, we find a good approximation architecture for Q-factor to be

$$\tilde{Q}(i_k, u_k, r) = \tilde{Q}(S_k, P_k, u_k, r)$$

$$= r_0 + \sum_{l=1}^M s_k^l r_l + \sum_{j=1}^N p_k^j r_{j+M} + \sum_{f=1}^N \{u_f^k r_{f+M+N}$$

$$+ u_f^k p_k^f r_{f+M+2N} + u_f^k g(f) r_{f+M+3N}\}$$

So there are totally $1+M+N+3N=1+14+6+6*3=39$ elements in the parameter vector r .

3.2.2 Comparison of Q-factor

Before applying the learning algorithm to tune the components in the parameter vector r , we must define the cost functions g_1, g_2 & g_3 mentioned in Section II. Ideally, one could extend our control plane simulation model to include the data plane as well. Then we could use it to evaluate the actual cost to network performance (e.g., throughput) resulting from blocking message types, node failures and network disconnection. Since our current simulator does not include the data plane, we simply assigned cost to each node according to the location, functionality of the node and the network topology in the testbed. And we assigned the cost of turning off a message type according to the importance of the message type.

Monte-Carlo simulations are used to estimate the Q-factor under the heuristic policy at some state

action pair (i,u) . After using the learning algorithm to tune r , we also use Monte-Carlo method to estimate the Q-factor under the policy which determines u in state i by minimizing $\bar{Q}(i,u,r)$ over u . Then we compare the Q-factors and cost to go under the two policies.

For the heuristic policy $\bar{\mu}$, the Q-factor is:

$Q^\mu(i_1, u) = g(i_1, u) + E_{i_2} \{J^\mu(i_2)\}$, and it is evaluated by Monte-Carlo method. This is the expected cost to go when action u is taken in the initial state i_1 and the remaining actions are chosen according to the heuristic policy.

We compare the heuristic policy with the policy $\bar{\mu} = \arg \min_u \bar{Q}(i, u, r)$, which is greedy with respect to the optimal Q-factor approximation $\bar{Q}(i, u, r)$. The Q-factor for the greedy policy is given by $Q^{\bar{\mu}}(i_1, u) = g(i_1, u) + E_{i_2} \{J^{\bar{\mu}}(i_2)\}$ and is also evaluated by Monte-Carlo method. This is the expected cost to go when action u is taken in the initial state i_1 and the remaining actions are chosen according to the policy $\bar{\mu}$. We call $\bar{\mu}$ the greedy policy in the following discussion.

To calculate the Q-factors by Monte-Carlo simulations, we need to update the state $(S_k \& P_k$ in Section 2) in the simulations. To do this, the transition probabilities are needed. For the update of S_k , we obtain the probabilities $p_{h,i}^j$ (in Section II) through extensive simulations. For the update of P_k , the probability $P(S_{k+1} | I_k, U_k, d_{k+1} = j)$ is also obtained from simulation.

We start from a typical state $i_1 = \{S_1 = [1, 7, 8, 12]; P_1 = [1/6, 1/6, 1/6, 1/6, 1/6, 1/6]\}$, which is a state with four failed nodes: node 1, 7, 8 and 12, and initially the probability distribution is uniform. And we estimate the Q-factor for each policy for state i_1 with 6 different actions. Each Q-factor value is estimated over 20 different simulation trajectories starting from state action pair (i_1, u) . The results are shown in Table I. In Table I, the 1st row is the action u being 6 different values. $u=1$ means that the 1st message type is turned off, and the rest is similar. The 2nd row gives the value of the Q-factors with the first action being u and then following the heuristic policy $\bar{\mu}$. The 3rd row gives the values of the Q-factors corresponding to the greedy policy $\bar{\mu}$. The

last row is the ratio between the Q-factors under two policies. We can see from Table I, the Q-factor at the same state action pair (i_1, u) under the greedy policy $\bar{\mu}$ is much less than that under the heuristic policy.

As the last row shows, $Q^{\bar{\mu}}$ is only about 50%~70% of Q^μ . That means the policy greedy to the approximated optimum Q-factor is much better than the heuristic policy. So we do find a better policy by Q-factor approximation. We also compare the value function $J(i_1)$ at state i_1 under the heuristic policy and the greedy policy. The value function $J^{\bar{\mu}}(i_1)$ under the greedy policy is only about 50% of the value function $J^\mu(i_1)$ under the heuristic policy.

Table 1. Comparison of Q-factors

Action u	1	2	3	4	5	6
$Q^\mu(i_1, u)$	2248	2367	2433	2079	2127	2254
$Q^{\bar{\mu}}(i_1, u)$	1444	1454	1247	1448	1390	1269
$Q^{\bar{\mu}}/Q^\mu$ (%)	64.2	61.4	51.3	69.6	65.4	56.3

3.2.3 Comparison of two policies

We run parallel simulations from the same initial states. One simulation uses the heuristic policy, the other uses the approximately optimal policy (the greedy policy). And we do this for many initial states. The simulations are constructed to be ‘coupled’ in that unless one policy blocks propagation and the other does not, exactly the same thing happens in the two simulations.

We tested for eight different initial states, and the results are shown in Table II. In Table II, the 1st row is the identifying number of 8 different initial states. Each number represents a different initial state. The 2nd row is the action sequence taken by following the heuristic policy $\bar{\mu}$. For example, corresponding to case 1, the action sequence is **635412**. It means by following the heuristic policy $\bar{\mu}$, the first action is 6, which is turn off message type 6. And then turn off message type 3, message type 5, message type 4, message type 1 and message type 2. The bold number corresponds to the poison message in the simulation. E.g., message type 2 is the poison message in case 1. And the 3rd row is the action sequence under the greedy policy $\bar{\mu}$. As we can see from Table II, the greedy policy $\bar{\mu}$ performs much better than the heuristic policy $\bar{\mu}$. It can find out the poison message in 2 or 3 steps, while the heuristic policy usually takes more steps. According

to the eight cases we tested, the heuristic policy μ averaged 4 steps to find out the poison message. But the greedy policy only takes 2.6 steps on average.

Table 2. Policy comparison

Case	1	2	3	4
μ	635412	352	35246	342
$\bar{\mu}$	42	32	216	32

Case	5	6	7	8	Ave steps
μ	3541	341	351	35241	4.000
$\bar{\mu}$	321	21	5321	321	2.625

5. Summary

We have discussed a particular failure propagation mechanism--poison message failure propagation, and provided a framework to identify the responsible protocol or message type. In this paper, we focus on the active diagnosis. For the sequential decision problem, we explored the use of reinforcement learning with function approximation. We use a feature-based approximation structure for the optimum Q-factor, and we get a greedy policy by minimizing the Q-factor at each state. We also compare the performance of the greedy policy with a heuristic policy. Our tests show that the greedy policy is much better than the heuristic policy. This demonstrates that the feature-based function approximation technique is promising in solving the large-state sequential decision problem in the poison message problem.

6. References

- [1] X. Du, M.A. Shayman and R. Skoog, "Preventing Network Instability Caused by Control Plane Poison Messages" *IEEE MILCOM 2002*, Anaheim, CA, Oct. 2002.
- [2] X. Du, M.A. Shayman and R. Skoog, "Using Neural Networks to Identify Control and Management Plane Poison Messages", *IEEE IM 2003*, Colorado Spring, Colorado, March 2003, to appear.
- [3] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Vols. I and II, Athena Scientific, 2000.
- [4] R. Skoog et al., "Network management and control mechanisms to prevent maliciously induced network instability," *Network Operations and Management Symposium*, Florence, Italy, April 2002.
- [5] H. Li and J. S. Baras, "A framework for supporting intelligent fault and performance management for communication networks", *Technical Report, CSHCN TR 2001-13*, Univ. of Maryland, 2001.
- [6] M.A. Shayman and E. Fernandez-Gaucherand, "Fault management in communication networks: Test scheduling with a risk-sensitive criterion and precedence constraints," *Proceedings of the IEEE Conference on Decision and Control*, Sidney, Australia, Dec. 2000.
- [7] D. J. Houck, K. S. Meier-Hellstern, and R. A. Skoog, "Failure and congestion propagation through signaling controls". *In Proc. 14th Intl. Teletraffic Congress*, Amsterdam: Elsevier, 367-376, 1994.
- [8] J-F. Huard and A.A. Lazar. "Fault isolation based on decision-theoretic troubleshooting", *Technical Report 442-96-08*, Center for

Telecommunications Research, Columbia University, New York, NY, 1996.

[9] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press 1998.

[10] D. Bertsekas and J. Tsitsiklis. *Neurodynamic Programming*, Athena Scientific, 1996.