Using Reinforcement Learning for Proactive Network Fault Management

Qiming He, Mark A. Shayman,

Dept. of Electrical and Computer Engineering,
University of Maryland, College Park, MD 20742
Fax: 010-301-314-9281, Email:qiminghe,shayman@eng.umd.edu

Abstract

For high-speed networks, it is important that fault management be proactive–i.e., detect, diagnose, and mitigate problems before they result in severe degradation of network performance. Proactive fault management depends on monitoring the network to obtain the data on which to base manager decisions. However, monitoring introduces additional overhead that may itself degrade network performance especially when the network is in a stressed state. Thus, a tradeoff must be made between the amount of data collected and transferred on one hand, and the speed and accuracy of fault detection and diagnosis on the other hand. Such a tradeoff can be naturally formulated as a Partially Observable Markov Decision Process (POMDP) whose solution can be used to construct decision-rule for both centralized and distributed intelligent agents. Since the exact solution of POMDP's for a realistic number of states is computationally prohibitive, we develop a reinforcement-learning-based fast algorithm which learns the decision-rule in an approximate network simulator and makes it fast deployable to the real network. Simulation results are given to diagnose a switch fault in an ATM network.

Keywords
Network Fault Management, Proactive Diagnosis, Intelligent Agents, Reinforcement Learning, POMDP, ATM.

1 Introduction

Traditionally network fault management (NFM) has focused on event correlation [22] [28]. More recently, there is an increasing emphasis on proactive fault management [11] [26]. Proactive fault management requires that the network be monitored in order to detect and diagnose faults at an early stage. However, monitoring itself can introduce significant communication and computational overhead [13]. Thus, it is desirable to determine monitoring policies that represent optimal tradeoff between the amount of monitoring and the speed and accuracy with which potential faults can be detected and diagnosed.

Proactive fault management demands intelligent management actions be taken by central manager or remote agents. (We use manager/agent to refer to either entity). The intelligence of a manager/agent can be defined as taking optimal management actions (polling, repairing, etc.) and minimizing operation costs while observing network status. More specifically, when anomalies arise in the network, intelligent NFM requires both manager and agents to take approporiate actions based on network states and observations (e.g., alarms) which appear sequentially. For example, on the manager's side, polling devices randomly (or in round-robin) may mean a waste of both bandwidth and time. On the agent's side, sending false alarms diminishes the effectiveness of the manager. It is desirable for the manager to choose polling sequences which build up confidence in a fault hypothesis by focusing more on those network elements which might be directly related to the root-cause of the alarms. For an intelligent agent, who either resides in the network element (NE) or acts as a stand-alone proxy, should react to the changing network environment intelligently instead of solely providing MIB entries when being polled.

The problems of choosing optimal control actions in a stochastic environment have been studied extensively by both control and AI communities [3] [24]. Recently, there has been considerable interest in using Markov Decision Prceosses (MDP) to unify such efforts [16]. MDP views the world as a controlled Markov chain. State transition probabilities depend on control actions. For each individual transition, there is an associated immediate reward. Costs may be treated as negative rewards. The optimal policy is a control law that yields minimal costs in a finite or infinite horizon.

MDP is an elegent and flexible model of choosing optimal actions in a stochastic environment. When the model describing the environment is known, i.e., a model-based problem, the optimal policy can be obtained by such algorithms as Dynamic Programming (DP) [12]. However, when the model is not accurately known, or for a model-free problem, reinforcement learning (RL) can be used for solving MDP's [25]. In the context of NFM, MDP models have been discussed in [14] and both MDP's and reinforcement learning in [1].

In this paper, we suggest that the behavior of NFM manager/agent, i.e., tradoffs among monitoring, diagnosis and mitigation can be formulated as a Partially Observable MDP (POMDP). Partial observability implies that states cannot be completely observed and hence must be estimated by sequential observations. For instance, if we regard the switches up-and-down as states, they are hidden to all end-node observers. Even if system states can be completely monitored, observations might be missing or delayed in the communication network. POMDP may serve well as the theoretical basis of general diagnosis without exact knowledge of states and has found many applications in recent years [5] [8]. In this paper, the solution of the POMDP is used to construct the decision rule for the intelligent managers and agents.

Various exact solution algorithms for POMDP's have been proposed [17]. The majority of them are model-based. None of them can accommodate a realistic number of states for network fault management. In this paper, we develop a fast reinforcement learning algorithm to obtain approximate solutions to POMDP's with large numbers of states and thus make such a model pratically applicable to engineering applications. Firstly, a preliminary policy is learned in a simulator in which the states are completely observable. Secondly, the learned policy is then fine-tuned by applying an analogous learning scheme in the real world where the states are incompletely observed.

This paper is organized in 5 sections. Section 2 is a brief introduction to MDP, POMDP and their solutions. Section 3 presents our fast algorithm. Some simulation examples of centralized and decentralized fault diagnosis schemes are given in section 4. Section 5 contains the conclusion of this paper.

2 Basics on MDP/POMDP

MDP is a framework for sequential decision-making under uncertainty. Fault management gives rise to such decision problems. For example, if a device is polled and returns back an abnormal signal, we may not be sure if it is a real-alarm or a false-alarm. If we then poll it repeatedly and always get alarms, then it is probably a real-alarm, but we still may not be sure of the root-cause. Proceedingly, we might poll its neighbors. If this yields an alarm, then it is less likely the failure of the first device being polled and we might check the connecting devices (e.g., hub or router or the same server they both access). The decisions that need to be made are "poll it again?", "when to stop?", "which node needs to be polled next?", "when to take some recovery actions?" All these actions are closely related with time and cost. The goal is to do it as quickly as possible while minimizing the total cost.

2.1 MDP

Under MDP umbrella, control actions are assumed to be performed by an agent placed in a stochastic environment which can be described by a 4-tuple $<\mathcal{S},\mathcal{A},\mathcal{T},\mathcal{R}>$. $\mathcal{S}$ is a finite set of states. $\mathcal{A}$ is a finite set of actions. $\mathcal{T}$ is a mapping $\mathcal{T}:\mathcal{S}\times\mathcal{A}\rightarrow Pr(\mathcal{S})$. $\mathcal{R}$ is a reward function $\mathcal{R}:\mathcal{S}\times\mathcal{A}\rightarrow\Re^1$. Agent and environment interact with a sequence discrete steps $t=0,1,2,3...$ (as shown in Figure 1). The dynamics of the environment are stationary and Markov, i.e., state and action at one time step $s_t\in\mathcal{S}$, $a_t\in\mathcal{A}$ determines the distribution of state and reward $s_{t+1}\in\mathcal{S}, r_{t+1}\in\mathcal{R}$ next time step.

$$P(s,a,s')=Pr[s_{t+1}=s'|s_t=s,a_t=a] \quad (1)$$

$$R(s,a,s')=E[r_{t+1}|s_t=s,a_t=a,s_{t+1}=s'] \quad (2)$$

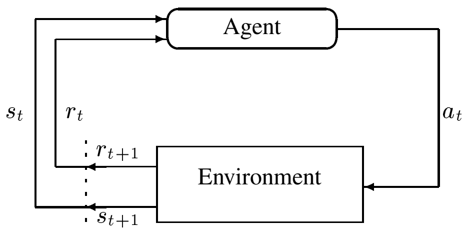

Figure 1: Intelligent Agent interacts with the environment

Agent's policy is a mapping from state to action. A deterministic policy gives the action to be taken for each state, $\pi_t:\mathcal{S}\rightarrow\mathcal{A}$. The agent's objective is to choose actions to maximize subsequent reward

$$R_t=\sum_{k=0}^{\infty}\gamma^k r_{t+k} \quad (3)$$

where $0<\gamma<1$ is a discount rate. The state-value given a policy is formally the expected reward starting that state

$$V^\pi(s)=E[R_t|s_t=s,\pi] \quad (4)$$

and the state-value given by the optimal policy is the one superior to all others

$$V^*(s)=\sup_\pi V^\pi(s) \quad (5)$$

There always exists an optimal policy π^* that archieves this maximum [3].

If the model is known, the optimal policy can be determined from the optimal state-value function. However, if the model is not accurately known, it is not feasible to derive optimal policy from V^*; instead we need $Q^*(s,a)$, the optimal state-action value function. In general, state-action value function is defined as

$$Q^{\pi}(s,a) = E[R_t | s_t = s, a_t = a, \pi] \tag{6}$$

which gives the expected reward when action a is taken in state s after which actions are chosen according to the policy π. Similarly,

$$Q^*(s,a) = \sup_{\pi} Q^{\pi}(s,a) \tag{7}$$

The optimal control action performed by the agent is $\pi^*(s) = argmax_a Q^*(s,a)$.

MDP was originally studied under the assumption that $P(s,a,s')$ and $R(s,a,s')$ are completely known. There are various methods to determine the optimal policies for MDP, e.g., value iteration [2], policy iteration [12]. When the system dynamics are unknown, reinforcement learning can be applied [25], e.g., the pioneering one-step off-policy Q-learning algorithm [27] which iterates the Q-value based on the resulting next state and immediate reward:

$$\begin{aligned} Q^{t+1}(s,a) =& (1-\alpha)Q^t(s_t,a_t) + \\ & \alpha[r_t + \max_{a_{t+1}} Q^t(s_{t+1},a_{t+1})] \end{aligned} \tag{8}$$

where $0 < \alpha < 1$ is a learning parameter. It can be proved Q^t under minimal technical conditions, converges to Q^* asymptotically as $t \to \infty$ [27]. The optimal policy can then be determined from Q^*.

Solution algorithms for MDP's can handle problems with thousands of states. Unfortunately, in many real-life applications, states cannot be observed completely. In a communication network, if we denote the up-down states of NE, the states can only be estimated by polling performance metrics of related NE. We perform management operations based on our confidence of the states' status. If we are pretty sure what device is down, then we will take replace-action. If we observe alarms indicating some kind of congestion or cell-loss, it might be due to the nature of the statistical switching or it might be a fault. In this case, we may have to take preventive-action such as collecting more information of related NE's or re-routing traffic around the suspicious NE to maintain QoS. Such decisions are made sequentially in order to reduce long-run costs (or equivalently, to maximize long-run rewards).

2.2 POMDP

POMDP is a framework to handle observation uncertainty in the MDP. A model-based POMDP has 6-tuple $< \mathcal{S}, \mathcal{T}, \mathcal{A}, \mathcal{O}, \Pi, \mathcal{R} >$ where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$ and $\mathcal{R}$ follow the definitions in MDP. $\mathcal{O}$ is a set of observations and Π is an observation function $\Pi : \mathcal{S} \times \mathcal{A} \to Pr(\mathcal{O})$. Solution to POMDP refers to a control policy that will yield greatest reward over some finite or infinite horizons ([17]).

In MDP (or Completely Observable MDP), current action only depends on the current state, previous states and actions are irrelevant. In the POMDP, the current action may depend on the entire sequence of past actions and resulting observations. However, it is equivalent to have the current action depend on a sufficient statistic called *belief state*, which gives the probability for each possible value for the current state given all the past actions and observations. By doing such a conversion, a POMDP problem over state space can be treated as an MDP problem over belief space and Markov property holds. There is an additional element $\mathcal{I}$ defining the problem-specific initial belief state.

In principle, solutions to MDP can be used directly to solve POMDP assuming POMDP is an MDP over belief states. Unfortunately, the infinite number of belief states makes the direct use impractical. However, due to Sondik's work ([23]), the optimal value function for any POMDP can be approximated arbitrarily well by piecewise-linear convex function (PWLC), thus the optimal action is [19]

$$\pi(b) = argmax_{\{a \in \mathcal{A} | l_a \in L\}} l_a \cdot b \tag{9}$$

where b is a belief state, $\mathcal{L}$ is a collection of solution vectors of $| \mathcal{S} |$-dimension.

The discussion of exact solution is beyond the scope of this paper. Basically, numerous exact algorithms to POMDP have their different ways of finding solution vectors set in (9) [29] [4]. Most of them start by constructing a finite representation of PWLC function over belief state, then iteratively updating this representation, expanding horizon and truncating it at some desired depth. Generally, such truncated exact solution works well for small problems with less than 100 states. For larger problem, it becomes computationally intractable. Some proposed memoryless approximate methods [18] [20] can handle a large numbers of states but may result in a poor policy. The goal of the following section is to develop an approximate algorithm that generally results in policies with good performance for NFM problems with large numbers of states.

3 Fast Algorithm

In order to use POMDP as a framework for constructing intelligent decision system, we need to solve POMDP problem without the exact model of the real world. The dynamics of our network are hard to obtain explicitly. A practical approach is to build a simulator that closely resembles the real-world model. The model of the simulator is easy to identify. Then we pretend the real-world model is the same as that of the simulator. Agent brings the optimal policy from the simulator to the real-world and keeps learning to resolve the model inconsistency, if any. Furthermore, the real-life problems often end up with huge number of states, actions and observations. Therefore, it is crucial to use function approximation to parameterize value functions. Our on-policy linear approximate learning algorithm is drived from $Sarsa(\lambda)$ [25] which is originally applied in MDP. The idea is we assign Q-value to belief-action pair instead of state-action pair and use linear function to approximate it.

$$Q_t(b_t, a) = \sum_{s \in \mathcal{S}} Q_t(s, a) b_t(s) \tag{10}$$

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha \delta_t e_t(s, a) \qquad \forall s \tag{11}$$

where

$$\delta_t = r_{t+1} + \gamma Q_t(b_{t+1}, a) - Q_t(b_t, a) \tag{12}$$

$$\begin{aligned} e_t(s, a) &= \gamma \lambda e_{t-1}(s, a) + \nabla_{Q_t(s,a)} Q_t(b_t, a) \\ &= \gamma \lambda e_{t-1}(s, a) + b_t(s) \qquad \forall s \end{aligned} \tag{13}$$

(11) (12) (13) is the $Sarsa(\lambda)$ updating rule applied to parameters, i.e., the Q-value at the vertex of the belief space. $e_t(s, a)$ is the eligibility trace with $e_0(s, a) = 0, \forall s$. $0 < \lambda < 1$ is the trace-decay parameter. $0 < \alpha < 1$ is the learning step. $0 < \gamma < 1$ is the discount factor.

We begin with the learning phase assuming the states are completely observed in the simulator. State transitions are jumping at vertexes of the belief space and Q-values are updated accordingly. After the learning phase, we have not only identified an explicit model (for the simulator) but also the policy for the simulator which can be used as a jump-starter to supply the next phase. Then we bring the learned policy to the execution phase where states are partially observed. Using the model estimated, we can perform belief updating inside the belief space and fine-tune our policy so far (as shown in Fig 2). The psedocode of our algorithm can be found in [9]. The performance of this algorithm is reported in [10].

4 Modeling and Simulation

We have used an $OPNET$-based [21] testbed to simulate the fault diagnosis process in an ATM network. Our testbed (as shown in Fig 3) consists of ATM-switches and end-nodes. Each end-node has several applications (e.g. email, ftp, database) installed. Some applications act as clients while others act as servers. Traffic is delivered between clients and servers. We assume a fault occurs in an ATM-switch where it is hard to be diagnosed directly yet can be estimated by using polling to obtain performance data from end-nodes. We trigger a delay-fault in one of the switches which decreases the switch processing rate. OAM (Operation and Maintenance) cells [15] are implemented to measure end-to-end performance, e.g. delay, cell-loss-ratio. Faulty cells will be observed, if not dropped due to enforcement of QoS violation before reaching destination. A threshold is defined for each performance metric to convert the metric into a binary-valued variable that takes the values $normal$ and $abnormal$. We include the possibility that the reported value of the variable is incorrect. The goal of the manager is to find a polling sequence that accurately determines the root cause while minimizing the amount of polling actions required.

4.1 Centralized NMS

Assuming only one switch can be in fault at any time, the system-state viewed by central manager consists of mutually exclusive sub-state $\mathcal{S} = \mathcal{S}^0 \cup \mathcal{S}^1$, $\mathcal{S}^0 = \{s_0\}$, $\mathcal{S}^1 = \{s_1, s_2, ...s_N\}$ in which s_0 means all switches are normal and s_i means merely i^{th} switch is in fault. Actions taken by manager consist of repair and pollings, $\mathcal{A} = \mathcal{A}^r \cup \mathcal{A}^p$, $\mathcal{A}^r = \{a_1^r, ..a_i^r..a_N^r\}$, $\mathcal{A}^p = \{a_1^p, ..a_j^p..a_N^p\}$ in which a_i^r is the action to repair i^{th} switch, a_j^p is the action to poll j^{th} end-node. Observation at each end-node is $\mathcal{O} = \{normal, abnormal\}$. *Normal* means the OAM cell is QoS-complaint; otherwise it is *abnormal*. In simulator, state transition matrix

$$P(s' = s_k | s = s_i, a = a_j^p) = \delta_{ik} \tag{14}$$

$$P(s' = s_k | s = s_i, a = a_j^r) = \begin{cases} \delta_{ik} & i \neq j \\ \delta_{0k} & i = j \end{cases} \tag{15}$$

$$P(o | s_i, a = a_j^p) = \begin{cases} q_j^i & o\text{=abnormal}, i \neq 0 \\ 1 - q_j^i & o\text{=normal}, i \neq 0 \\ \mu & o\text{=abnormal}, i = 0 \\ 1 - \mu & o\text{=normal}, i = 0 \end{cases} \tag{16}$$

in which $\mu \ll 1$ is the probability of false alarm, q_j^i is the fraction of shortest paths originating from j^{th} end-node that pass through i^{th} switch. Reward function is

$$R(s_i, a_j) = \begin{cases} C & i = j, a_j \in \mathcal{A}^r \\ C' & i \neq j, a_j \in \mathcal{A}^r \\ c & a_j \in \mathcal{A}^p \end{cases} \qquad (17)$$

in which $C \gg c \gg C'$.

4.2 Distributed NMS

In centralized NMS, manager takes full responsibility to make decisions; agents only supply measurements when being polled. When the network size grows larger, increased state and observation number will slow down manager's response time and polling traffic will consume an undesirable amount of bandwidth. An alternative is to migrate intelligence from manager to agents as in the "management-by-delegation" paradigm [7].

POMDP framework can be easily adapted from manager to agents. Each agent only cares about its domain where only related elements are included. For example in Fig 3, the state set of C_rl can be reduced to $\{SW1, SW2, SW3\}$ since its traffic will only pass these switches. State of $\{SW4, SW5, SW6\}$ will not likely affect observations in this end-node.

4.3 Experiment and Simulation

Suppose we trigger a fault in $SW1$ to simulate extra processing delay, i.e. all ATM cells will take much longer than usual passing this switch. Costs are chosen as $C = 100, c = -1, C' = -100$.

4.3.1 An example in centralized NMS

Suppose the manager starts from the initial belief {0.7,0.05,0.05,0.05, 0.05,0.05, 0.05 } which means the central manager originally believe that the network is more likely in the normal condition. One of the instance of belief updating and actions chosen by POMDP policy is shown in Table 1.

An intuitive explanation is that the manager starts by choosing an end-node (i.e., 4^{th} node C_rl) to poll and gets $normal$ observation (i.e., QoS-compliant) which means $SW2$ is not likely in fault. Then it updates its belief and chooses to poll 2^{nd} node (i.e., $S\_rl\_em\_db\_ftp$) and gets an $abnormal$ observation. To make sure it is not false alarm, multiple pollings of this node are made and belief is updated consequently. Then it comes to poll 8^{th} node (C_custom) and gets $normal$ observation. Since C_custom has to access $S\_em\_x\_ftp\_custom$ server via $SW4$ and $SW5$, it rules out the possibility that $SW4$ or $SW5$ are in fault. The final action is taken to repair $SW1$, the faulty switch that has been correctly diagnosed.

4.3.2 An example in distributed NMS

As discussed in section 4.2, some of the manager's decision role can be delegated to agents. For example, an intelligent agent is installed on end-node c_rl. It may take actions to poll related end-nodes such as c_em_ftp, $S\_rl\_em\_db\_ftp$ and $S\_x\_rl\_db\_em$. Following are two instances of agent's policy assuming fault is still triggered in $SW1$ and observations come from simulator traces. Initial belief is (0.97, 0.01, 0.01, 0.01).

Policy-1: Choose action Polling-$S\_x\_rl\_db\_em$ and get $abnormal$. Update belief to (0.43 0.44 0.11 0.004). Poll again and get $abnormal$. Update belief to (0.001 0.93 0.06 0.0001). Send suspicion "$SW1$ in fault" to manager.

Policy-2: Choose action polling-C_em_ftp and get $abnormal$. Update belief to (0.328 0.202 0.33 0.135). Poll it again and get $nomral$. Update belief to (0.66 0.165 0.007 0.165). Poll it again and get $normal$. Update belief to (0.037 0.555 0.037 0.37). Send suspicion "$SW1$ in fault" to manager.

5 Conclusion

In the networked environment, fault management needs non-trivial techniques to deal with not only catastrophic failure but also partial failures, especially when processors, memory, file systems, and applications tend to be distributed. Partial failures may be difficult to detect at an early stage, yet may eventually lead to serious degradation of network performance. Proactive diagnosis of such faults cannot be achieved without consideration of the dynamics of the network and its monitoring processes. The POMDP framework takes this information into account and has the promise of producing effective monitoring and control policies for intelligent manager/agent. To overcome model uncertainty and the large number of states, we have proposed a fast algorithm, based on reinforcement learning, for obtaining approximate solutions to the relevant POMDP's.

6 Acknowledgement

This research was partially supported by DoD contract MDA 90497C3015 and NSF grant ECS 9626399

References

[1] Baras J., Li H., Mykoniatis G. (1998), *Integrated, Distributed Fault Management for Communication Networks*, ISR technical report TR 98-21.

[2] Bellman R.E., (1957), *Dynamic Programming*, Princeton University Press.

[3] Bertsekas, D. P., (1995), *Dynamic Programming and Optimal Control*, Athena Scientific.

[4] Cassandra, A. R., *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes* http://www.cs.brown.edu/research/ai/pomdp/index.htm Ph.D. Thesis. Brown University, Department of Computer Science. 1998.

[5] Cassandra A. R. *A Survey of POMDP Applications*, 1998. http:// www.cs.duke.edu/ mlittman/talks/fs98-pomdp/working-notes.html

[6] Goldszmidt G., Yemini, (1995), *Distributed management by delegation*, 15th International Conference on Distributed Computing.

[7] Hauskrecht M., Fraser H. (1998), *Modeling Treatment of Ischemic Heart Disease with Partially Observable Markov Decision Processes*, In Proceedings of American Medical Informatics Association annual symposium on Computer Applications in Health Care, Orlando, Florida, pp. 538-542.

[8] He Q., Shayman M. A., (1999), *Using POMDP as Modeling Framework for Network Fault Management*, ISR Technical Report.

[9] He Q., Shayman M. A., (1999), *Solving POMDP by On-line Linear Approximate Learning Algorithm*, ISR Technical Report.

[10] Hood C.S., Ji Chuanyi, (1997), *Automated Proactive Anomaly Detection*, Integrated Network Management V, Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management, p. 688-699.

[11] Howard, R., *Dynamic Programming and Markov Processes*, MIT Press, Cambridge, MA. 196.

[12] HP Solution Note (1997), *Testing Operation and Maintenance Implementation for ATM*, ATM/Broadband Testing seminar, Hewlett-Packard Company.

[13] Huard J. F., Lazar A. A., (1996), *Fault Isolation based on Decision-Theoretic Troubleshooting*, TR442-96-08, COMET Group, Columbia University.

[14] ITU-T Recommendation, (1995), *I.610 B-ISDN Operation and Maintenance: Principles and Functions*.

[15] Kaelbling L. P., Littman M. L. Moore, A. W., (1996), *Reinforcement Learning: A Survey*, Journal of Artificial Intelligence Research, Volume 4.

[16] Kaelbling L. P., Littman M. L., Cassandra, A. R. (1998), *Planning and Acting in Partially Observable Stochastic Domains*, Artificial Intelligence, Vol. 101.

[17] Kimura, H., Miyazaki, K., and Kobayashi, S., *Reinforcement Learning in POMDPs with Function Approximation*, Proceedings of the 14th International Conference on Machine Learning, pp.152–160. 1997.

[18] Littman, M. L., Cassandra, A. R., Kaelbling L. P., (1998), *Learning policies for partially observable environments: Scaling up*, Proceedings of the Twelfth International Conference on Machine Learning, p362–370.

[19] Loch J, Singh S., *Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes*, ICML98.

[20] MIL3 Inc. (1998) *OPNET 4.0 User-Manual*.

[21] Nygate Y. A., (1995), *Event Correlation using Rule Based and Object Based Techniques*, Integrated Network Management IV, Proceedings of fourth international symposium on integrated network management, ed. by A.S. Sethi.

[22] Sondik E. J. (1971), *The optimal control of partially observable Markov Processes*, Ph.D. Thesis Stanford University.

[23] Stuart J. Russell, Peter Norvig, *Artifical Intelligence: A Modern Appraoch*, Prentice-Hall, Englewood Cliffs, New Jersey. 1994.

[24] Sutton R. S., (1998), *Reinforcement Learning: An introduction*, MIT Press.

[25] Thottan M., Ji Chuanyi, (1998), *Proactive Anomaly Detection Using Distributed Intelligent Agents*, IEEE Networks Sept/Oct p21-27.

[26] Watkins C. J. C. H., Dayan P. (1992), *Technical note: Q-learning*, Machine Learning, 8(3/4) p.272-292.

[27] Yemini, A., et al. (1996) *High Speed and Robust Event Correlation*, IEEE Communication Magazine, p82-90,May 1996.

[28] Zhang N., Liu W., (1997), *A Model Approximate Scheme for Planning in Partially Observable Stochastic Domains*, Journal of Artificial Intelligence, 7, p.199-230.

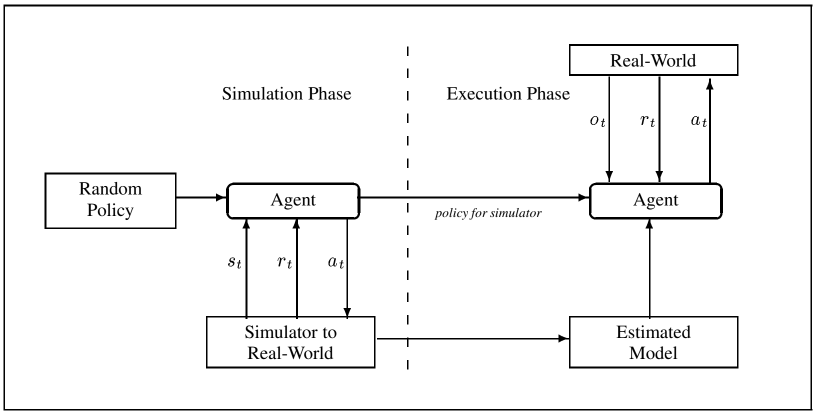

Figure 2: Two-phase Learning Algorithm

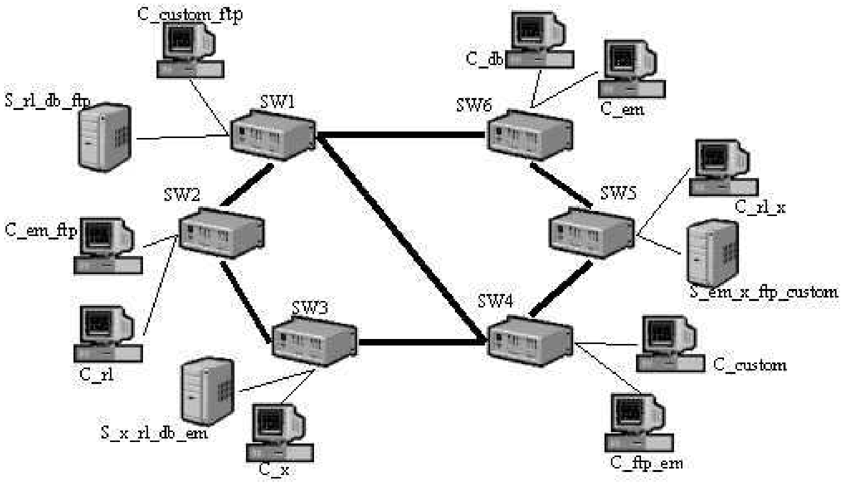

Figure 3: The Topology of ATM Simulation Testbed for Fault Management

normal	sw1	sw2	sw3	sw4	sw5	sw6	obs & actions
0.7	0.05	0.05	0.05	0.05	0.05	0.05	$a_4^p \leftarrow$ normal
0.779	0.028	0.005	0.028	0.055	0.055	0.055	$a_2^p \leftarrow$abnormal
0.098	0.35	0.002	0.003	0.156	0.156	0.234	$a_2^p \leftarrow$abnormal
0.002	0.701	0.002	0.0	0.007	0.157	0.157	$a_2^p \leftarrow$abnormal
0.0	0.89	0.0	0.02	0.02	0.057	0.0672	$a_2^p \leftarrow$abnormal
0.0	0.965	0.0	0.0	0.005	0.005	0.0245	$a_8^p \leftarrow$normal
0.0	0.997	0.0	0.0	0.003	0.0	0.0013	a_1^r

Table 1: Polling sequence determined by exact solution of POMDP