USING NEURAL NETWORKS TO IDENTIFY CONTROL AND MANAGEMENT PLANE POISON MESSAGES

XIAOJIANG DU, MARK A. SHAYMAN, RONALD SKOOG
Department of Electrical and Computer Eng. University of Maryland, Telcordia Technologies

Abstract: Poison message failure propagation is a mechanism that has been responsible for large scale failures in both telecommunications and IP networks: Some or all of the network elements have a software or protocol 'bug' that is activated on receipt of a certain network control/management message (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. Identifying the responsible message type can permit filters to be configured to block poison message propagation, thereby preventing instability. Since message types have distinctive modes of propagation, the node failure pattern can provide valuable information to help identify the culprit message type. Through extensive simulations, we show that artificial neural networks are effective in isolating the responsible message type.

Key words: Poison message, neural network, node failure pattern, fault management

1. INTRODUCTION

There have been a number of incidents where commercial data and telecommunications networks have collapsed due to their entering an unstable mode of operation. The events were caused by unintentional triggers activating underlying system defects (e.g., software 'bugs,' design flaws, etc.) that create the propagation mechanism for instability. These system defects are generally not known to the

network providers, and new defects are constantly introduced. More importantly, these points of vulnerability can be easily triggered through malicious attack.

Our goal is to develop a fault management framework that can protect networks from unstable behavior when the trigger mechanism and defect causing instability are unknown. This can be accomplished by identifying *generic failure propagation mechanisms* that permit failures to lead to network instability. Once these mechanisms are identified, control techniques can be designed to prevent or interrupt the failure propagation. There are several failure propagation mechanisms that can cause unstable network. Five generic propagation mechanisms have been identified thus far [4]:

- System failure propagation via routing updates;
- System failure propagation from management / control plane 'poison message';
- System failure propagation from data plane 'invalid message';
- Congestion propagation from congestion back pressure;
- Deadlocks created from overload and timeouts.

Here we focus on one of these that we call the 'poison message' failure propagation mechanism.

1.1 The Poison Message Failure Propagation Problem

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 [8]. 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.

In the AT&T incident above, the trigger event is the normal maintenance event that caused the first switch to take itself out of service. The poison message is the ordinary message sent to neighboring switches informing them that it is temporarily going out of service. The poison message creates a state in the neighboring switches in which faulty code may be executed. In this case, an auxiliary event must occur for the faulty code to be executed causing the node to fail, namely the arrival of a pair of closely spaced call setup messages. The dependence on the auxiliary event makes the node failure probabilistic with the probability depending on network load (the rate of call setup messages). While in this particular example, the poison message itself is not flawed, in other examples such as the OSPF case referred to above, the poison message may be malformed or contain fields with abnormal values.

Obviously, the more challenging case is the one in which the message itself is completely normal and is 'poison' only because of a software defect in the router or switch.

There are some discussions about the poison message problem in [8]. One idea to fight the poison message problem is to limit the number of similar type Network Elements managed by the central control function and use a distributed architecture. But the above idea is not practical [8]. It might cause lots of changes to the existing systems. And it also causes software, hardware inconsistency in the system.

Our philosophy is to add some functions to the existing system rather than change it. We want 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 propose using *passive diagnosis* and *active diagnosis* to identify and block the corresponding protocol or message type.

1.2 The Problem Features

This problem has several differences from traditional network fault management problems. Typical network fault management deals with localized failure [5] [7]. For instance, when there is something wrong with a switch, what propagates is not the failure but the consequences of the failure on the data plane (e.g., congestion builds up at upstream nodes). Then multiple alarms are generated that need to be correlated to find the root cause [3] [9]. In our problem, the failure itself propagates, and propagation occurs through messages associated with particular control plane or management plane protocols. It is also different from worms or viruses in that worms and viruses propagate at the application layer.

A message type may have a characteristic pattern of propagation. For example, OSPF uses flooding so a poison message carried by OSPF link state advertisements is passed to all neighbors. In contrast, RSVP path messages follow shortest paths so a poison message carried by RSVP is passed to a sequence of routers along such a path. Consequently, we expect pattern recognition techniques to be useful in helping to infer the responsible message type.

We make the following three assumptions:

1) There is a central controller and a central observer in the network. I.e., 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 poison message type when such failure occurs.

In this problem, we use centralized network management scheme. This is due to the nature of the problem. The failure itself propagates in the whole network. In

order to identify the responsible protocol, event correlation, failure pattern recognition and other techniques are used. They all need global information, correlation and coordination. We also suggest some distributed methods to deal with the poison message problem, such as the Finite State Machine (FSM) method. The FSM in our framework is a distributed method in the sense that it is applied separately to information collected at individual failed nodes, rather than across multiple failed nodes. The details are given in section 2.

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. Filtering may be used to isolate the responsible protocol by blocking one or more message types and observing whether or not failure propagation continues. The filter settings may or may not be chosen to be the same throughout the network.

We suggest combining both passive diagnosis and active diagnosis to find out the poison message. Passive diagnosis includes analyzing protocol events at an individual failed node, correlating protocol events across multiple failed nodes, and classifying the observed pattern of failure propagation. Active diagnosis uses protocol or message type filtering.

2. PASSIVE DIAGNOSIS

Passive diagnosis includes the following methods.

1) Finite State Machine Method: This is a distributed method used at a single failed node. All communication protocols can be modeled as finite state machines [3] [11]. When a node fails, the neighbor of the failed node will retrieve messages belonging to the failed node. From the message sequence for each protocol, we can determine what state a protocol was in immediately prior to failure by checking the FSM model. We can also find out whether those messages match (are consistent with) the FSM. If there are one or more mismatches between the messages and the FSM, that probably means there is something wrong in the protocol.

2) Correlating Messages: Event correlation is an important technique in fault management. Recently exchanged messages are stored prior to node failure. Then we analyze the stored messages from multiple failed nodes. If multiple nodes are failed by the same poison message, there must be some common features in the stored messages. One can compare the stored messages from those failed nodes. If for a protocol, there are no common received messages among the failed nodes, then we can probably rule out this protocol--i.e., this protocol is not responsible for the poison message. On the other hand, if many failed nodes have the same final message in one protocol, we can use Bayes' Rule to calculate the probability of the

final message being the poison one. We have reported the details of this technique in [1].

3) Using Node Failure Pattern: Different message types have different failure propagation patterns. One way to exploit the node failure pattern is to use a neural network classifier. The neural network is trained via simulation. A simulation testbed can be set up for a communication network. The testbed has the same topology and protocol configuration as the real network. Then for each message type used in the network, the poison message failure is simulated. And the simulation is run for the probability of a node failure taking on different values. After the neural network is trained, it is applied using the node failure sequence as input, and a pattern match score is the output. In addition, we can combine the neural network with the sequential decision problem. The details are discussed in Section 4.

The node failure pattern can also be combined with the network management and configuration information. For example, suppose BGP is the responsible protocol carrying the poison message. Since only the BGP speaker routers exchange BGP messages, when we get several failed nodes, we can check if all these nodes are BGP speakers. If any failed node is not a BGP speaker, then we can rule out BGP--i.e., BGP is not the poison protocol.

Generate Probability Distribution: 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.

3. ACTIVE DIAGNOSIS

From passive diagnosis we have an estimated probability distribution over the possible poison protocols or message types. 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 or management 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] [12].

The Sequential Decision Problem

- 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 objective function taking into account:

– degree to which each action will help in isolating the responsible message type;
– urgency of diagnosis under each potential message type. E.g., if a suspect protocol sends its messages to a large number of nodes via flooding, the risk of network instability were it to be poison would be particularly great, and this risk provides additional impetus for filtering such a message type;
– impact of filtering action on network performance. A critical protocol or message type should be blocked only if there is a compelling need to do so.

There are three possible outcomes when message filtering is considered.

1) If message filtering is used and the propagation is stopped within a certain time, then either the poison message type is found (if only one message type is filtered) or the types that are not filtered are ruled out (two or more types are filtered).

2) If message filtering is used but the propagation is not stopped within a certain time, then we did not find the responsible message type this time. The filtered message types are removed from the possible suspect set. Collect more information, update the probability vector and reconfigure the filters.

3) If the current action is not to filter any message types, then we simply take another observation. Several other nodes will fail as the poison message propagates in the network. This information is used to update the probability vector. Based on the updated state, a new action is taken to configure the filters.

The sequential decision problem is modeled as a Markov decision process (MDP) in [2]. Also we proposed a heuristic policy and used a Q-factor approximation algorithm to obtain an improved policy for the MDP problem in [2].

We have proposed passive diagnosis and active diagnosis to identify the poison message. In this paper, we focus on one of the methods in passive diagnosis – *Using Node Failure Pattern*. We use neural networks to explore the node failure pattern of

different poison messages. Also neural networks can be combined with the sequential decision problem in active diagnosis. Details are given in section 4.

4. NEURAL NETWORK SIMULATION RESULTS

We have implemented an OPNET testbed to simulate an MPLS network in which poison messages can be carried by BGP, LDP, or OSPF. The testbed has 14 routers of which 5 are Label Edge Routers and 9 are (non-edge) Label Switching Routers. The topology of the testbed network is shown below.

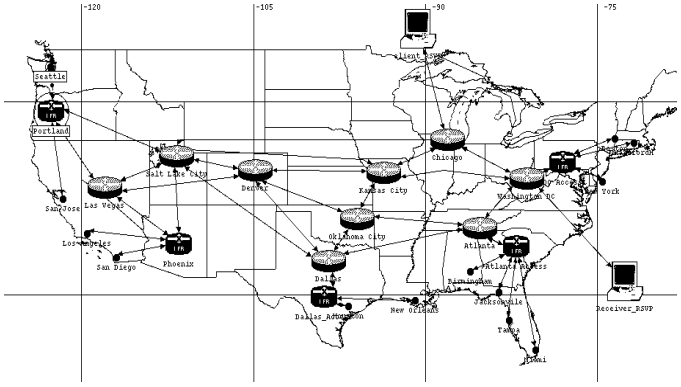

Figure 1. The Topology of OPNET Testbed

We use the Neural Network Toolbox in MATLAB to design, implement, and simulate neural networks. There are several different neural network architectures supported in MATLAB. We implemented two kinds of them in our simulation.

1) Feedforward backpropagation. Standard backpropagation is a gradient descent algorithm. This type of neural network is most commonly used for prediction, pattern recognition, and nonlinear function fitting.

2) Radial basis networks provide an alternative fast method for designing nonlinear feed-forward networks. Variations include generalized regression and probabilistic neural networks. Radial basis networks are particularly useful in classification problems.

Training and Learning Functions are mathematical procedures used to automatically adjust the network's weights and biases. The training function dictates a global algorithm that affects all the weights and biases of a given network. The learning function can be applied to individual weights and biases within a network. The training function we used is: trainb, which is a batch training with weight and bias learning rules.

4.1 Neural Network Structure and Training

We have implemented three feedforward backpropagation neural networks and one radial basis neural network. They have similar structure; all of them have three layers.

1) Input layer with 28 inputs. There are 14 nodes in the communication network. We use a vector to denote the node status in the communication network: $S_k = \left[s_k^1, s_k^2, \ldots, s_k^{14} \right]^T$, where k is the discrete time step, and s_k^m = 0 or 1. (0 means this node is normal, and 1 means this node is failed). The 28 inputs represent node status vectors at two consecutive time steps: S_{k-1}, S_k.

2) Hidden layer in the middle. In the middle of the three layers is the hidden layer. There is a transfer function in the hidden layer. A lot of transfer functions are implemented in MATLAB. In our simulation, we use three kinds of transfer functions in the feedforward backpropagation neural networks: 'tansig', 'logsig', and 'purelin'.

a. tansig - Hyperbolic tangent sigmoid transfer function.

b. purelin - Linear transfer function.

c. logsig - Log sigmoid transfer function.

3) Output layer with 12 outputs. The output is the probability distribution vector of the poison message. 12 outputs represent 12 message types in the OPNET testbed. Each output is the probability of the corresponding message being poison.

From the OPNET simulations, we record the node status vector S_k at time k. Then we use these data to train the neural networks. The input of the neural networks is a 28-element vector representing S_{k-1} & S_k. We use 32 sets of such input to train the neural networks. And the target of training (i.e., the output during training) is a 12-element vector representing the probability distribution vector of the poison message. Since we know what is the poison message during simulations, we know the target vectors. For example, a target vector of the 4th message being poison is [000100000000]. We set the training goal to be: error $<10^{-10}$. And the training epoch number set to be 50. One epoch means that the training data is used once. All neural networks meet the training goal.

4.2 Main Test Results

After training neural networks, we use some new data to test the neural networks. In our simulation we use 17 sets of 28-element vectors to test the four different neural networks. The result is very good. All of the neural networks can output a good probability distribution of the poison message for 14 out of 17 input data. Four sets of outputs are listed in Table 1.

In Table 1, "NN" means Neural Networks. And m1,m2,m3 & m4 represent four different poison messages. The numbers 1,2,3 and 4 in 1st row represent four different neural networks. Number 1,2 and 3 represent three feedforward

backpropagation neural networks with transfer function being: 'tansig', 'purelin' in 1, 'tansig', 'tansig' in 2, and 'tansig', 'logsig' in 3. And 4 represents the radial basis neural network. The outputs of neural networks include both positive and negative numbers. We call the outputs the 'distribution scores'. And the final probability distribution of poison message is an origin shift and normalization from the distribution scores. I.e., $y=a(x+b)$, where x is the distribution score and y is the probability distribution, and a & b are parameters. Since during the neural network training, the output of non-poison message is set to zero, we set the transformation of the smallest (negative) number in the output vector to be zero. I.e., set b to be $-x_0$ where x_0 is the smallest number. And a can be determined from $\sum y=1$. The data in Table 1 is the probability distribution after transformation. The training data for the neural networks in Table 1 only included five different poison messages. That is why the outputs from neural networks 3 and 4 have several zeros.

Table 1. Output of Neural Networks

NN	1	2	3	4	NN	1	2	3	4
	0.0841	**0.1953**	**0.9992**	**0.8212**		0.0140	0.1302	0	0.0796
m1	0.1235	0.1273	0	0.1127	m2	**0.1791**	**0.1925**	0.0067	**0.3986**
	0.1146	0	0	0.0078		0.1148	0.0656	0	0.1599
	0.0427	0.0191	0.0006	0.0322		0	0	0	0.0032
	0	0.1035	0.0002	0.0262		0.1514	0.1919	0.9933	0.3586
	0.0892	0.0794	0	0		0.0937	0.0471	0	0
	0.0898	0.0870	0	0		0.1149	0.0793	0	0
	0.1331	0.0780	0	0		0.1237	0.0375	0	0
	0.1269	0.0805	0	0		0.1242	0.0844	0	0
	0.0446	0.0844	0	0		0.0022	0.0519	0	0
	0.1143	0.0743	0	0		0.0499	0.0488	0	0
	0.0373	0.0711	0	0		0.0320	0.0708	0	0
	0.0348	0.1247	0.0061	0.0475		0.0003	0.0885	0	0.0096
m3	0	0	0	0.0298	m4	0.0132	0	0	0.0032
	0.1677	**0.1290**	**0.7421**	**0.6973**		0.3187	0.0711	0	0.0032
	0.0413	0.1227	0.0000	0.0010		0.0777	**0.1551**	**0.9999**	**0.9735**
	0.1128	0.1281	0.2518	0.2244		0.0388	0.1549	0.0001	0.0105
	0.0745	0.0496	0	0		0.1162	0.0735	0	0
	0.0515	0.0876	0	0		0.1164	0.0819	0	0
	0.0984	0.0373	0	0		0.1452	0.0731	0	0
	0.0919	0.1057	0	0		0.0738	0.0812	0	0
	0.1172	0.0464	0	0		0.0653	0.0785	0	0
	0.1782	0.0646	0	0		0.0345	0.0676	0	0
	0.0317	0.1043	0	0		0	0.0746	0	0

The boldface numbers are the probabilities assigned to the actual poison messages. And the underlined numbers are instances where the neural networks

assign largest probability to wrong message types. Thus, poison messages 1, 3 and 4 are correctly diagnosed by neural networks 2,3,4 but misdiagnosed by 1. Poison message 2 is correctly diagnosed by neural networks 1,2,4 but misdiagnosed by 3. From Table1, we can see that most of the time, the four neural networks can provide a good probability distribution about the poison message--i.e., the neural networks can identify the poison message. Combining all the test results, we find that the radial basis neural network performs the best. Also we find that different neural networks fail for different cases. This suggests that we can combine the outputs from two (or more) neural networks to get better results.

4.3 Serial Test

In the previous tests, we only use S_{k-1} & S_k as input--i.e., we only use the node status at two time steps. Another way to test the neural network is to input a series of node status, e.g., S_1, S_2; S_2, S_3; ... S_{k-1}, S_k . This means we want to input more information to the neural network in the hope of getting better results.

In particular we did the serial tests for the data sets that the neural network failed to correctly diagnose with the original input. The results are encouraging. The neural network can gradually identify the poison message for about 60%~70% of these data sets--i.e., the output gradually changed from a bad probability distribution to good probability distribution. One example of the serial test is given in Table 2. In the example, the neural network is a feedforward backpropagation neural network. We also used radial basis neural network for the serial test and have similar results. In this example, message No. 3 is the poison message. And the data in Table 2 is the probability distribution after origin shift and normalization.

Table 2. Serial Test Result

Input	S_1, S_2	S_2, S_3	S_3, S_4	S_4, S_5	S_5, S_6	S_6, S_7	Ave.
1	0.1296	0.0633	0.1558	0.0339	0.1231	0.0811	0.0865
2	0	0.0056	0.1032	0	0	0.1191	0.0380
3	0.0743	0.1267	0.1101	**0.1635**	**0.1726**	**0.1461**	**0.1415**
4	0.1231	0.1942	0.0725	0.0403	0.1205	0.0412	0.0986
5	0.1258	0.1217	0.1228	0.1100	0.1221	0	0.1004
6	0.0816	0.0888	0.0937	0.0727	0.0570	0.0860	0.0800
7	0.0975	0.1145	0.0774	0.0502	0.0664	0.0866	0.0821
8	0.0569	0.0185	0.0398	0.0960	0.0499	0.1283	0.0649
9	0.0983	0.0491	0	0.0896	0.0822	0.1224	0.0736
10	0.0642	0.1107	0.1337	0.1143	0.0473	0.0431	0.0855
11	0.0652	0	0.0142	0.0961	0.0684	0.1102	0.0590
12	0.0835	0.1070	0.0769	0.1333	0.0906	0.0359	0.0879

In Table 2, the 1st column is list of the 12 message types. And column 2 through column 7 are the probability distribution for different inputs. From Table 2, we can

see that at the beginning, the neural network assigns message No. 1 the largest probability, so it does not find the poison message. When we input S_4, S_5, the neural network finds the poison message – message No. 3. And the neural network continues to find the poison message in the later tests. That shows the outputs of the neural network stabilized after input S_4, S_5. The last column is the average probabilities of the previous 6 columns. The average probabilities can be thought of as the combined result from a series of inputs. From Table 2, we can see that the combined result finds the poison message--i.e., it assigns the largest probability to the poison message.

4.4 Integration With the Sequential Decision Problem

The neural network can provide a good probability distribution about the poison message. But it cannot solve the problem completely since the neural network may assign a large probability to a wrong message type. We still need some actions to confirm that we find the poison message. One action is to use message filtering -- turn off the possible poison message, and see if the failure propagation stops in a certain time. If it stops, then the identity of the poison message is confirmed. On the other hand, if failures continue, then we have new data to input to the neural network. However, we need a neural network that takes into account the knowledge that a particular message type is blocked. Thus, it should output a probability distribution over the remaining message types.

We considered two approaches to combining neural networks with message filtering. In the first approach, we added another 12 inputs to the neural network. Each input represents one message type. When a message is turned off, the corresponding input is set to –1. Also the corresponding output in training is set to –1. The idea is that we use –1 to denote that this message is turned off. We trained the new neural network and tested its performance. But it did not work well for the test data.

In the second approach, we assumed that at most one message is turned off at any time. Then we created 12 additional neural networks, one corresponding to each message type that can be turned off. Each neural network has 28 inputs as before, but only 11 (rather than 12) outputs. If a message is turned off, we remove the output corresponding to that message. We trained these neural networks and the test results show that this method works well. One example of the integration with the sequential decision problem is given in Table 3. In the example, a feedforward backpropagation neural network is used.

In Table 3, the first four columns correspond to the test with 11-output neural networks. Each row is the probability of the corresponding message type being poison. And in the first row, k is the time step in the sequential decision problem. The poison message in this example is message No. 3. In the example, we use the policy of filtering the message type with the highest probability at each time step.

The bold number corresponds to the message being turned off. At time step 1, message No. 4 is turned off. Since it is not the poison message, the failure still propagates. We observe some nodes fail and input that node status to the corresponding neural network with 11 outputs. We list the outputs in column 3 (k=2). From Table 3 we can see that at time step 2, message No. 5 has the highest probability and it is turned off. The poison message is not found yet. So we observe some other node failure, and input the information to the neural network corresponding to message No. 5 turned off. Then at time step 3, we find the poison message -- message No. 3.

Table 3. Integration of Neural Networks With the Sequential Decision Problem

Step	k = 1	k = 2	k = 3	New	k = 1	k = 2	k = 3
p_1	0.1086	0.1475	0.1431	Test	0.1086	0.1669	0.1665
p_2	0	0	0		0	0.1440	0
p_3	0.1280	0.1459	**0.1480**		0.1280	0.1831	**0.1722**
p_4	**0.1650**	/	0.1407		**0.1650**	0	0
p_5	0.1633	**0.1478**	/		0.1633	**0.1881**	0
p_6	0.0386	0.0689	0.0569		0.0386	0.0451	0.0662
p_7	0.0541	0.0803	0.1004		0.0541	0	0.1169
p_8	0.0906	0.0604	0.0428		0.0906	0.0621	0.0498
p_9	0.0553	0.0995	0.1213		0.0553	0.0199	0.1411
p_{10}	0.1194	0.0572	0.0533		0.1194	0.0326	0.0620
p_{11}	0.0671	0.0828	0.0741		0.0671	0.1030	0.0862
p_{12}	0.0099	0.1097	0.1196		0.0099	0.0552	0.1392

Since the identity of the poison message does not change, it follows that if a message type has been filtered and propagation continues, that message type can be ruled out as the poison message. Thus, it would make sense to eliminate it as a possible output from the neural networks applied in subsequent time steps. In the example, when message type 5 is blocked, it would be desirable to use a neural network that had the outputs for both types 4 and 5 removed. However, to apply this approach in general would require training a neural network corresponding to each subset of the set of message types. The number of neural networks would be exponential in the number of message types, which is not practical. So we tried another approach that uses the original neural network with 12 outputs and computes the conditional probabilities given all message types that have been previously ruled out. For the message type that has been ruled out, its probability is set to zero, and the probability distribution is obtained by normalizing the rest of the outputs. The results of the new tests are listed in the last three columns in Table 3, and the underline zeros are those set manually. From table 3, we can see that the new approach works well, and it requires only one neural network.

If different message types have different filtering costs, the policy of blocking the message type with highest probability can be extended to the well known

heuristic policy based on the ratios $E[C_j]/p_j$. I.e., the message type selected for blocking is the one that has the smallest ratio $E[C_j]/p_j$ where $E[C_j]$ is the expected cost (in terms of network performance) associated with blocking message type j for a time step, and p_j is the current estimate of the probability that message type j is the poison one. In our example, we are considering the special case where the costs are all equal.

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. We have proposed passive diagnosis, which includes the FSM method applied at individual failed nodes, correlating protocol events across multiple failed nodes and using node failure pattern recognition. If passive diagnosis cannot solve the problem by itself, it can be augmented by message type filtering, which is formulated as a sequential decision problem. In this paper, we focus on identifying node failure pattern using artificial neural networks. We have implemented and tested four different types of neural networks. Our tests show that neural networks can provide a good probability distribution for the poison message in most cases. We also performed the serial test that works for many of the data sets for which the original test failed. Furthermore, we have combined the neural networks with the sequential decision problem. In the sequential decision problem, the decision as to which message type(s) to filter is based on the current state consisting of the set of failed nodes together with the estimated probability that each message type is poison. The neural networks appear to be effective as a computational tool for updating these probabilities without requiring an explicit model for the transition probabilities in the underlying Markov chain.

ACKNOWLEDGEMENT

This research was partially supported by DARPA under contract N66001-00-C-8037.

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, "Markov Decision Based Filtering to Prevent Network Instability from Control Plane Poison Messages" submitted for publication.

[3] A. Bouloutas, et al, "Fault identification using a finite state machine model with unreliable partially observed data sequences," *IEEE Tran. Communications*, Vol.: 41 Issue: 7, pp: 1074–1083, July 1993.

[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*, University 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] I. Katzela; M. Schwartz, "Schemes for Fault Identification in Communication Networks", *Networking, IEEE/ACM Transactions on* , Vol.: 3. Issue: 6, pp: 753 – 764, Dec. 1995.

[8] 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, pp: 367–376, 1994.

[9] A. Bouloutas, S. Calo, and A. Finkel, "Alarm correlation and fault identification in communication networks", *Communications, IEEE Transactions on*, Vol.: 42, Issue: 2, pp: 523 -533, Feb-Apr 1994.

[10] D.A. Castanon, "Optimal search strategies in dynamic hypothesis testing", *IEEE Trans. Systems, Man and Cybernetics*, Vol.: 25 Issue: 7, July 1995

[11] A. Bouloutas, G.W. Hart and M. Schwartz, "Simple finite-state fault detection for communication networks," *IEEE Trans. Communications*, Vol. 40, Mar. 1992.

[12] 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.

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

[14] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Vols. I and II, Athena Scientific, 2000.

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