Telecom Alarm Prioritization using Neural Networks

Stefan Wallin

Lulea University of Technology LTU Skelleftea
SE-931 87 Skelleftea Sweden
Email: stefan.wallin@dataductus.se

Leif Landén

Data Ductus Nord AB
Torget 6
SE-931 30 Skelleftea Sweden
Email: leif.landen@dataductus.se

Abstract—Telecom Service Providers are faced with an overwhelming flow of alarms. Network administrators need to judge which alarms to resolve in order to maintain the service quality. The problem is that it is hard to pick the most important alarms. Which alarms have the highest priority? A solution that automatically assigns priorities to alarms would increase the efficiency of Network Management Centers. We have prototyped a solution that uses neural networks to assign alarm priority. The neural network learns from network administrators by using the manually assigned priorities in trouble-tickets. Our tests are based on live-data from a large mobile service provider and we show that neural networks can learn to assign relevant priorities to 75% of the alarms.

Index Terms—Communication system operations and management, Neural network applications, Alarm Systems.

I. INTRODUCTION

The operational activities at a service provider's Network Management Center are focused on managing a constant flow of alarms [18]. A primary goal is to resolve the most important problems as quickly as possible. A simplified process description for lowering error impact is:

- Group alarms that are related to the same problem.
- Associate them with a trouble-ticket to manage the problem resolution process.
- Assign a priority to the trouble ticket.
- Analyze and fix the problem.

The starting point for our work is the lack of priorities in alarm systems [21]. Due to the number of alarms and tickets, prioritization is of vital importance: *Which alarms are most critical to resolve? How can we we prioritize the tickets in order to optimize the work-flows?* Alarms in a telecom environment adhere to the X.733 [7] standard which defines a severity field. One might believe that this field could be used as alarm and ticket priority. However, our studies indicate that there is weak correlation between alarm severity and the corresponding assigned ticket priority, see section II. The main reason is probably that the alarm severity is assigned by the equipment provider at "design-time". Priorities are based on using contextual information such as redundancy, topology or SLAs that is dynamic information available at "run-time".

Prioritization of alarms and trouble-tickets is mostly performed *manually* by network administrators. They use their experience and support systems such as inventory and SLA management systems to determine the priority. This manual

process makes the operator organization dependent on a few individual experts [18]. Furthermore, the priority information is only available in the trouble-ticket system and not in the alarm system.

Current solutions for *automatic* alarm prioritization are mainly based on service impact tools and expert systems which uses information from external systems to modify alarms. This enables a prioritization algorithm to decide upon priorities since the alarm is bound to topology, service and business impact [19]. This type of solutions have some intrinsic problems [16]:

- *Maintenance of service models*: formal models of network and service topology need to be maintained which is complex and costly. Also, the change-rate of network topology and service structures is challenging to handle.
- *Maintenance of impact rules*: correlation rules are typically expressed using Rete-based expert systems [5], which require extensive programming.
- *Capturing operators knowledge*: in order to write the rules for the expert system, developers need to have extensive input from the network administrators. However they are critical resources in the organization and can not be allocated time to formalize rules.

The hypothesis studied in this paper is that a learning neural network could suggest relevant priorities by capturing network administrators knowledge. As stated by Pernido et. al: [13]

Experienced decision makers do not rely on formal models of decision making, but rather on their previous experience. They use their expertise to adapt solutions to problems to the current situation.

As data for the analysis and input for the testing we have used a database of alarms with associated trouble-tickets. The database is an export from a large mobile service provider. It contains original alarm data with severity and associated trouble-ticket with manually set priority. In this paper we present the results of the data analysis and a solution using neural networks.

- In section II we present the analysis of the alarm and associated trouble tickets. We show that there is a weak correlation between the severity assigned by the equipment versus the priority assigned by the expert network administrator.

- We present an architecture for assigning priorities using neural networks that adapts to the operator knowledge in section III-A
- Finally the results of a prototype implementation is presented in section III-C. We compare the priorities set by the neural network with those set manually by operators and find that 72% of the alarms are given correct or close to correct priority by our solution.

II. ANALYSIS OF ALARM SEVERITY VERSUS PRIORITY

The trouble-tickets and alarms in our test database consist of operational data from a mobile network operator. It consists of 260973 alarms associated to 85814 trouble-tickets. The alarms have an original severity from the reporting equipment according to the X.733 [7] standard:

- indeterminate(0)
- critical(1)
- major(2)
- minor(3)
- warning(4)
- indeterminate(5)

The mobile operator we studied used a priority scale in the trouble-ticket system of 1-6. Priority 1 is the most urgent and indicates a problem that needs to be resolved within hours, whereas priority 6 has no deadline.

The number of alarms associated with a trouble-ticket varies from 1 to 1161. The average number of alarms that is associated to a specific trouble ticket is 5.2 with a standard deviation of 17. This is an indication that the relationship between alarms and tickets is complex. In an ideal world alarms should indicate a problem and not individual symptoms, if this was true, the fan-out between tickets and alarms would have been much lower.

In Figure 1 we can see that the distribution of alarm severities associated with a trouble-ticket is low. If for example a trouble-ticket have both *major*(2) and *warning*(4) alarms associated, it has a difference of two steps, as illustrated by the third bar in Figure 1. The associated alarms have the same severity in more than 50% of the cases as seen by the first bar.

Figure 2 shows the mapping between alarm severities versus the manually assigned priorities in trouble-tickets. The diagram has one bar for each alarm severity and the bar is divided into fields showing the priority distribution. It clearly illustrates that alarm severities do not correspond to the actual *priorities* of the alarm being handled. For example, we see that alarms with severity *warning*, the rightmost bar, are almost evenly spread among the different priorities.

We illustrate the contrast between priority and alarm severity with two corner cases. ‘Obstruction light’-alarms are reported as critical alarms from base stations. However they are assigned the lowest priority (6) in the trouble ticket since the mobile service is not affected. On the other hand we can look at ‘NbpDedicated_RncRbsControlLinkLossOfRedundancy’-alarms, reported as warning. It is an indication that the control link between RBS and RNC in the 3G network has lost one

Fig. 1. Y-axis: number of tickets, X-axis: difference between max and min alarm-severity

Fig. 2. Ticket priorities per alarm severity.

of the redundant links. These alarms are assigned the highest priority in the trouble tickets since if all links are lost it will severely affect the 3G service.

Discussions with the network operator led to the hypothesis that if we only look at the maximum alarm severity associated to the trouble ticket we would get a better correlation. Figure 3 shows the analysis of this assumption. It clearly illustrates that we still do not have a mapping between alarm severity and corresponding priority. For example we see that priority 4 is distributed across all severities, and it is largest in the *warning* and *critical* severity.

The noticeable difference between Figure 2 and Figure 3 shows us one of the problems the neural network have to overcome. The priority of an alarm depends on its grouping with other alarms. The grouping in this case is performed by the network administrators when performing manual alarm correlation and creating the trouble-ticket. The neural network on the other hand will analyze alarms one by one and assign a priority. If we compare the priorities in indeterminate alarms

Fig. 3. Ticket priorities per maximum severity.

Fig. 4. Alarm severities mapped to priorities.

in figure 2 and figure 3 we see a big difference in priority distribution.

After studying the weak correlation between alarm severity and ticket priority we looked for another correlation: alarm type¹ versus priority. We observe a strong correlation between some alarm types and their priority, but for other alarm types there is no correlation at all. So again, there is no direct and naive alarm versus priority algorithm.

In order to summarize the analysis we illustrate how severities match priorities, see figure 4. A correct match would for example be that critical alarms always were assigned priority one. In order to generate this graph we normalized the 6 priorities versus 5 severities. We can see that only in 17% of the cases we have an exact mapping between severity and priority. In 30% of the cases the mapping is one priority step wrong. This will be compared to the results of the automatic priorities by the neural network later in this paper.

All in all, we can conclude that we have no usable correlation between alarm severities and trouble-ticket priorities

¹X733: EventType, ProbableCause, SpecificProblem

Fig. 5. Architecture Overview

III. USING NEURAL NETWORKS FOR PRIORITIZING ALARMS

A. The architecture

The goal of our work is to set priorities into alarms at the time of reception. We *do not* intend to assign automatic priorities in the trouble-ticket system.

Previous section showed that severities can not be used as priority. Our solution will however give network administrators a proposed priority. The priorities should be based on the experience of network administrators. To achieve this, we have developed an architecture based on neural networks. We have integrated this into an alarm system and a trouble-ticket system. The neural network uses the manually assigned trouble-ticket priorities and associated alarms as learning data. When the alarm system receives an alarm, it interrogates the trained neural network which generates a suggested priority put into the alarm information.

The architecture consists of following main components:

- Neural Network based on lib2f2n2 [10].
- OpenView TeMIP [17] as the alarm handling system
- Trouble Ticket system, ARS Remedy [14]
- Priority trainer
- Alarm Prioritization Module

A neural network is an algorithm capable of advanced pattern recognition. The strength of neural networks is their ability to recognize patterns even with added noise [2]. In this context we use the pattern recognition capabilities to find alarm-priority patterns. One of the problems with using a neural network instead of rule-based AI-solutions is that we can not show what “rule” the neural network follows when prioritizing the alarms [2]. In rule-based systems we can describe the rule in plain text, in neural networks the rule is a matrix of values. We used an open-source *neural network-engine* named libF2N2 [10]. The lib2f2n2 library uses linear activation

$$f(x) = x \quad (1)$$

for the input layer and the logistic function

$$f(x) = 1/(1 + e^{-x}) \quad (2)$$

for all successive layers. The neural network uses back-propagation [15] as the learning mechanism. It only supports

Raw - [6]					
Severity	Prio	Event Type	Probable Cause	Managed Object	Additional Text
Minor	2	Communications...	LossOfFrame	BSC_demo.bsc3	Radio frame lost
Major	3	QualityofService...	ThresholdCr...	BTS_demo.bts31	availability degradation
Major	3	QualityofService...	ThresholdCr...	BTS_demo.bts31	availability degradation
Major	1	QualityofService...	ThresholdCr...	BTS_demo.bts33	availability degradation
Major	6	QualityofService...	ThresholdCr...	BTS_demo.bts31	availability degradation
Major	4	QualityofService...	ThresholdCr...	BTS_demo.bts32	availability degradation

Fig. 6. New column in TeMIP for Alarm Priority

iterative back-propagation, not batch back-propagation.

Two variables play a special role during learning, this is the *learning rate* and the *momentum* of the neural network. Learning rate indicates what portion of the error should be considered when updating the weights of the network. Momentum indicates how much of the last update that should be reused in this change. Momentum is an effective way to move a network towards a good generalization, but can also be a problem if the momentum moves us away from the optimum weights.

The architecture is independent of the actual *Alarm Handling System*. The only requirement is that a new field is needed to handle the priority information in the database and the user interface. We used HP TeMIP extended with a python interface [1]. Figure 6 shows the alarm user interface with a new column indicating priority.

The *Priority Trainer* exports tickets with priority and alarm attributes to the neural network for the training process. The export contains alarm attributes and their manually set priority in the corresponding ticket. We used the following alarm fields to construct the input to the neural network:

- Managed Object: the resource emitting the alarm
- Specific Problem: alarm type/id
- Additional Text: free text field with alarm information
- Perceived Severity: original severity as reported from the equipment

The selection of the above alarm attributes were based on discussions with network administrators to find the most significant attributes used for manual correlation and prioritization. Additional text and Specific problem is encoded using the soundex algorithm to remove the influence of numbers and convert the strings to equal length. During the learning process it prints the mean square error of the output to a file in order to monitor the learning progress.

The *Alarm Prioritization Module* interrogates the neural network for every alarm in order to have a suggested priority. This priority is added to the alarm and can be used by the operator to sort the alarm-list.

B. Test Configurations

The prototype was tested with different settings both for learning rate, momentum and neural network structure. The tests are outlined in Table I. Since the testing is very slow, approximately 3 minutes per epoch, we aborted the learning when the error-rate stabilized. Each test was performed with data *not* used during the training and is randomly chosen from the data-set. About 10% of the data-set was used for training.

Test	Epochs	L	N	O	LR	M	Tr E	Te E
1	1200	3	200	6	0,01	0,3	3,1%	18,1%
2	680	3	100	6	0,03	0,3	2,4%	17,7%
3a	100	4	50	6	0,03	0,2	6,1%	16,0%
3b	1000	4	50	6	0,03	0,2	4,7%	16,9%
4	300	3	70	1	0,05	0,2	31,8%	12,8%
5	1000	2	100	6	0,05	0,2	3,1%	30,0%

TABLE I

TEST RESULTS: L = LAYERS, N = NEURONS, O = OUTPUT, LR = LEARNING RATE, M = MOMENTUM, Tr E = TRAINING ERROR, Te E = TEST ERROR

Layers, Neurons and Output in Table I describe the architecture of the neural network. The number of neurons used in each hidden layer is given by the Neurons field of the table. The 6 neuron Output (Test 1, 2, 3 and 5) was a mapping where each neuron represented one priority and the one that got the highest result was the proposed priority. The 1 neuron Output (Test 4) was a scaled priority where 0,1 represented priority 1 and 0,25 represented 2 and so on. Training Error is the average of the mean square error of the last epoch in the training. Testing Error is the average error of each prioritized alarm. A priority error of one, e.g. assigning priority 3 instead of 4, equaled 20% in the 6 neuron output and 15% in the 1 neuron output.

The tests show us that the number of layers are more important than the number of neurons in each layer. The extremely low error on Test 3 shows that 4 layers is a better alternative than 3. Test 5 shows that with only 2 layers the application does not learn to prioritize. Adding more neurons does not make the neural network prioritize better; notice how the error drops when decreasing the number of neurons between Test 1 and Test 2. Although the Testing Error drops considerably when using the 1 neuron Output the high Training Error makes us reluctant to use it.

In Test 3 we can see that we don't necessarily get a better result from longer training. More testing is needed to find a suitable method of when to stop training. We have no direct correlation between Training Error and Testing Error. The high Training Error on Test 4 is accompanied by a low Testing Error and in Test 5 we have the opposite relation.

Figure 7 shows how mean square error descends during training for test 1. All tests, except test 4, produced similar graphs.

C. Results

We have a good match between the suggested priority and the manually set priority. As can be seen in Figure 8, 49% of the alarms had exactly the same priority as the ones manually assigned by network administrators. 22% of the alarms were prioritized with one step wrong.

IV. RELATED WORK

Most research efforts related to alarm handling focus on correlation [6], [8], [11], [12], [13]. The aim is "the determination of the cause" [16]. Wietgreffe [20] uses neural networks

Fig. 7. Test 1: X-axis = epoch and Y-axis = total error rate

Fig. 8. Neural network priorities mapped to manually assigned priorities.

to perform the correlation. Common for these efforts is that they look at the stream of alarms and tries to find the root cause. All correlated alarms are then grouped into one root-cause alarm.

We are not trying to find the root cause of the alarms, neither to group them. In contrast, our problem is in some sense a simpler one, but overlooked; *to prioritize the individual alarms*.

Se Woo et. al. [4] shows that neural networks are able to perform well in diagnostic environments. The authors emphasizes the “ability to learn from actual plant and/or simulator results, without requiring explicit rules and extensive human efforts” and the “large numbers of the training data that cover a wide range of representative alarm patterns are required to assure adequate network performance”. The work of Se Woo et. al. is focused on finding the initial cause based on the equipment emitting the alarm.

Chan [3] has the same motivation as we for using neural networks in alarm handling

This ability to learn and build unique structures for a particular problem, without requiring explicit rules and extensive human effort, makes neural networks especially useful in pattern recognition applications where rules are to complex to define and maintain.

One important statement made by Chan is that neural networks is not supposed to be better than operators in prioritizing tasks but faster.

Klemettinen et. al. [9] discusses how to use old data to find expert rules to feed into expert systems. They discuss how “changes in equipment, software, and network load means that the characteristics of the alarm data change.” In their solution “Knowledge about the topological relationships of senders of alarms is crucial.” They discuss the problem that “The number of correlation patterns can be very large, and acquiring them from technical experts is a tedious task.” and how

Both networks and network elements evolve quickly over time, so a correlation system is never complete. It also takes time for the experts to learn new correlations and to modify existing ones.

V. CONCLUSION AND FUTURE WORK

The mobile operator under study stated that a suggested priority with one level wrong would be relevant. We have shown that neural networks can assign relevant priorities to 71% of the alarms as shown by Figure 8. This is a much better guidance for operators rather than the currently available alarm severity which gives a relevant priority in 47% of the cases as shown by Figure 4. If we compare the mapping for *correct* priority we see that severities are only correct in 17% of the alarms whereas the neural network is correct in 49%. The figure also illustrates that severities performs worse in suggesting bad priorities, (2 or more step wrong).

The important characteristics of our solution is that it adjusts automatically to network administrators by learning from the trouble-ticket database. This is an efficient use of human expert knowledge compared to traditional approaches using rule-based systems which has the underlying problem of converting human knowledge to rules. It also avoids the complex problem of having a complete set of rules and topology information in an ever changing environment.

We have run several different configurations of neural networks and found an optimal network design. The best configuration is Test 3 where the output neurons match the output priorities. Configuration 3 also is less computing intensive than Test 1 & 2 so it is no question that this is the recommended network architecture. This solution has several benefits for the service providers:

- Priorities are available immediately as the alarms arrive.
- Captures network administrators’ knowledge without disturbing their business-critical work.
- Adapts to changing behavior and changing network topologies.
- Integrates easily.
- Completely automated.

The work presented in this paper was run using a historic database of alarms and trouble tickets. The next step is to deploy the test configuration in a running system to study how it adapts continuously. The test database only included alarms with associated tickets. A valid improvement is to perform the analysis and evaluation using alarms without associated tickets in the learning process as well. This could help the neural network to filter out non-relevant alarms.

ACKNOWLEDGEMENTS

The authors would like to thank Viktor Leijon, Gary Webster, and David Partain for valid input and reviews, as well as Christer Dahl for help with the graphics.

REFERENCES

- [1] Mats Andersson and Robert Wedin. Python scripting for network management. Bachelor's thesis, Lulea University of Technology LTU Skelleftea, 2006.
- [2] R. Callan. *Essence of Neural Networks*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1998.
- [3] Edward H. P. Chan. Using neural network to interpret multiple alarms. *IEEE Computer Applications in Power*, 3(2):33–37, 1990.
- [4] Se Woo Cheon, Soon Heung Chang, Hak Yeong Chung, and Zeung Nam Bien. Application of neural networks to multiple alarm processing and diagnosis in nuclear power plants. *IEEE Transactions on Nuclear Science*, 40(1):11–20, 1993.
- [5] C.L. Forgy. Rete: a fast algorithm for the many pattern/many object pattern match problem. *IEEE Computer Society Reprint Collection*, pages 324–341, 1991.
- [6] P. Frihlich, W. Nejd, L. Laube, K. Jobmann, and H. Wietgreffe. Model-Based Alarm Correlation in Cellular Phone Networks.
- [7] X733: Information technology - open systems interconnection - systems management: Alarm reporting function, 1992.
- [8] G. Jakobson and M. Weissman. Alarm correlation. *Network, IEEE*, 7(6):52–59, 1993.
- [9] Mika Klemettinen, Heikki Mannila, and Hannu Toivonen. Rule discovery in telecommunication alarm data. *Journal of Network and Systems Management*, 7(4):395–423, 1999.
- [10] libf2n2 Feedforward Neural Networks. Accessed 10th of August 2007. <http://libf2n2.sourceforge.net/>.
- [11] G. Liu, AK Mok, and EJ Yang. Composite events for network event correlation. *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on*, pages 247–260, 1999.
- [12] D.M. Meira. A Model For Alarm Correlation in Telecommunications Networks. *Computer Science, Institute of Exact Sciences (ICEx) of the Federal University of Minas Gerais, Belo Horizonte, Brazil*, page 149.
- [13] G. Penido, JM Nogueira, and C. Machado. An automatic fault diagnosis and correction system for telecommunications management. *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on*, pages 777–791, 1999.
- [14] ARS Remedy. Accessed 10th of August 2007. <http://www.bmc.com/remedy/>.
- [15] DE Rumelhart, GE Hinton, and RJ Williams. *Learning internal representations by error propagation*. MIT Press Cambridge, MA, USA, 1986.
- [16] R. Sterritt. Towards autonomic computing: effective event management. *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pages 40–47, 2002.
- [17] TeMIP Fault Management. Accessed 10th of August 2007. <http://h20229.www2.hp.com/products/tmpfm/index.html>.
- [18] Stefan Wallin and Viktor Leijon. Rethinking network management solutions. *IT Professional*, 8(6):19–23, 2006.
- [19] Stefan Wallin and Viktor Leijon. Multi-Purpose Models for QoS Monitoring. *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops-Volume 01*, pages 900–905, 2007.
- [20] H. Wietgreffe, K.D. Tuchs, K. Jobmann, G. Carls, P. Frohlich, W. Nejd, and S. Steinfeld. Using neural networks for alarm correlation in cellular phone networks. *International Workshop on Applications of Neural Networks to Telecommunications (IWANNNT)*, May, 1997.
- [21] J. Wilkonzon and D Lucas. Better alarm handling- a practical application of human factors. *Measurement and control(London. 1968)*, 35(2):52–55, 2002.