Bayesian Detection of Router Configuration Anomalies

Khalid El-Arini and Kevin Killourhy

August 26, 2005

Carnegie Mellon

Motivation

- On January 23, 2001, Microsoft’s websites went down for nearly 23 hours.
- Why?

Motivation

- On January 23, 2001, Microsoft’s websites went down for nearly 23 hours.
- Why?
  - "We screwed up. [Tuesday] night at around 6:30 p.m. Pacific time we made a configuration change to the routers on the DNS network," spokesman Adam Sohn said Wednesday evening.

Introduction

- Problem and Approach
  - Router misconfigurations can be costly, and existing tools can only detect certain types.
  - Under a Bayesian framework, router misconfigurations will appear as statistical anomalies.
- Methodology
  - Adapted three machine learning techniques for configuration file anomaly detection
- Results and Analysis
- Discussion
- Conclusion

Prior Work

- Feldmann and Rexford (2001) build a pattern matching tool to find known misconfigurations.
- Feamster and Balakrishnan (2005) build a tool to compare BGP configurations to a specification.
- Caldwell et al. (2003) define the problem, and suggest a rule learner approach.

Prior Work

- Feldmann and Rexford (2001) build a pattern matching tool to find known misconfigurations.
- Feamster and Balakrishnan (2005) build a tool to compare BGP configurations to a specification.
- Caldwell et al. (2003) define the problem, and suggest a rule learner approach.
- We aim to detect misconfigurations **without** prior knowledge of their form.

Methodology

- Obtain router configuration files
- Parse files
- Train and test three anomaly detection algorithms:
  - Naïve Bayes
  - Joint Bayes
  - Structured Bayes
- Evaluate performance

Router Data

- We obtained 24 (sanitized) configuration files from CMU computing services
- Cisco IOS format
- Modified extensively over the years, and thus diverged from common source
- Misconfigurations expected

Parsing

- IOS files are highly unstructured
- List of commands, many with multiple attributes

```
logging facility local5
logging 128.2.4.8
access-list 2 deny   10.0.0.0 0.255.255.255
access-list 2 deny   127.0.0.0 0.255.255.255
access-list 2 deny   172.16.0.0 0.15.255.255
access-list 2 deny   192.168.0.0 0.0.255.255
access-list 2 deny   169.254.0.0 0.0.255.255
access-list 2 permit any
access-list 2 deny   any
```

- Extract command name and list of arguments

Naïve Bayes

- We make some simplifying assumptions:
  - Each line of configuration file is independent of every other line
  - For a given command, each attribute is independent of every other attribute
- We want to estimate the probability of seeing a specific instance of a command (i.e. a single line in the configuration file)

line = [cmd, ($attr_1$=a_1, $attr_2$=a_2, …, $attr_N$=a_N)]

P(line | cmd)

= P($attr_1$=a_1, $attr_2$=a_2, …, $attr_N$=a_N | cmd)

= P($attr_1$=a_1 | cmd) P($attr_2$=a_2 | cmd) … P($attr_N$=a_N | cmd)

Naïve Bayes

- How do we compute these probabilities?
  - Estimate from router data
  - For each command:

$$P(attr_i = a_i \mid cmd) = \frac{\text{\# of instances of } a_i}{\text{\# instances of cmd}}$$

- What is an anomaly?
  - Probability significantly below its expected value
  - $P(line \mid cmd) < a \cdot E[P(line \mid cmd)]$
    - Where a is an empirically determined multiplier

Joint Bayes

- Assumptions:
  - Each line of configuration file is independent of every other line
  - No longer assume that attributes are independent of each other
- Now,

$$P(\text{line} \mid \text{cmd}) = P(\text{attr}_1 = a_1, \text{attr}_2 = a_2, \ldots, \text{attr}_N = a_N \mid \text{cmd})$$

Joint Bayes

- How do we compute these probabilities?
  - For each command:

$$P(\text{line} \mid \text{cmd}) = \frac{\#\text{ of instances of }(a_1\ a_2,\ldots,a_N)}{\#\text{ instances of cmd}}$$

- What is an anomaly?
  - Consider two situations (where cmd1 and cmd2 are commands that take a single argument):
    - "cmd1 x_1" appears once, "cmd1 x_2" appears 23 times
    - "cmd2 y_i" appears once for $1 = i = 24$

Joint Bayes

- cmd1 x_1 and cmd2 y_1 both have the same probability of occurring (1/24)
  - cmd1 x_1 seems anomalous, but cmd2 y_1 does not
  - How do we differentiate between these scenarios?
- Entropy is a measure of how unpredictable a distribution is
  - In this case, cmd1 has low entropy while cmd2 has high entropy
  - A threshold weighted by entropy will differentiate between these cases
  - line is anomalous if $P(\text{line} \mid \text{cmd}) < a \cdot [H(\text{cmd})]^{-1}$

Structured Bayes

- Assumptions:
  - Each line of configuration file is independent of every other line
  - Groups of attributes are mutually dependent, while others are independent
- We manually selected attributes which appear to be mutually dependent (e.g. ip address and subnet), and joined them as one attribute
- We then proceeded as in the Naïve Bayes case to compute probabilities, but used the entropy-based threshold from Joint Bayes

Evaluate Performance

- From literature, we identified three critical types of misconfigurations:
  - Lone commands
  - Suppressed commands
  - Dangling commands
- We built tools to automatically find instances in CMU data
- We determine how many other commands someone has to look through in order to find each misconfiguration as an anomaly

Evaluate Performance

- Lone commands
  1. ip ospf authentication null (pod-b-cyh)
  2. exec-timeout 0 0 (rtrbone)
  3. version 12.2 (rtrbone)
- Suppressed commands
  1. access-list 2 permit any
     access-list 2 deny any (campus)
  2. access-list 2 permit any
     access-list 2 deny any (rtrbone)
- Dangling commands
  1. ip access-group 198 (pod-c-cyh)
  2. ip access-group 133 (core255)

Evaluate Performance

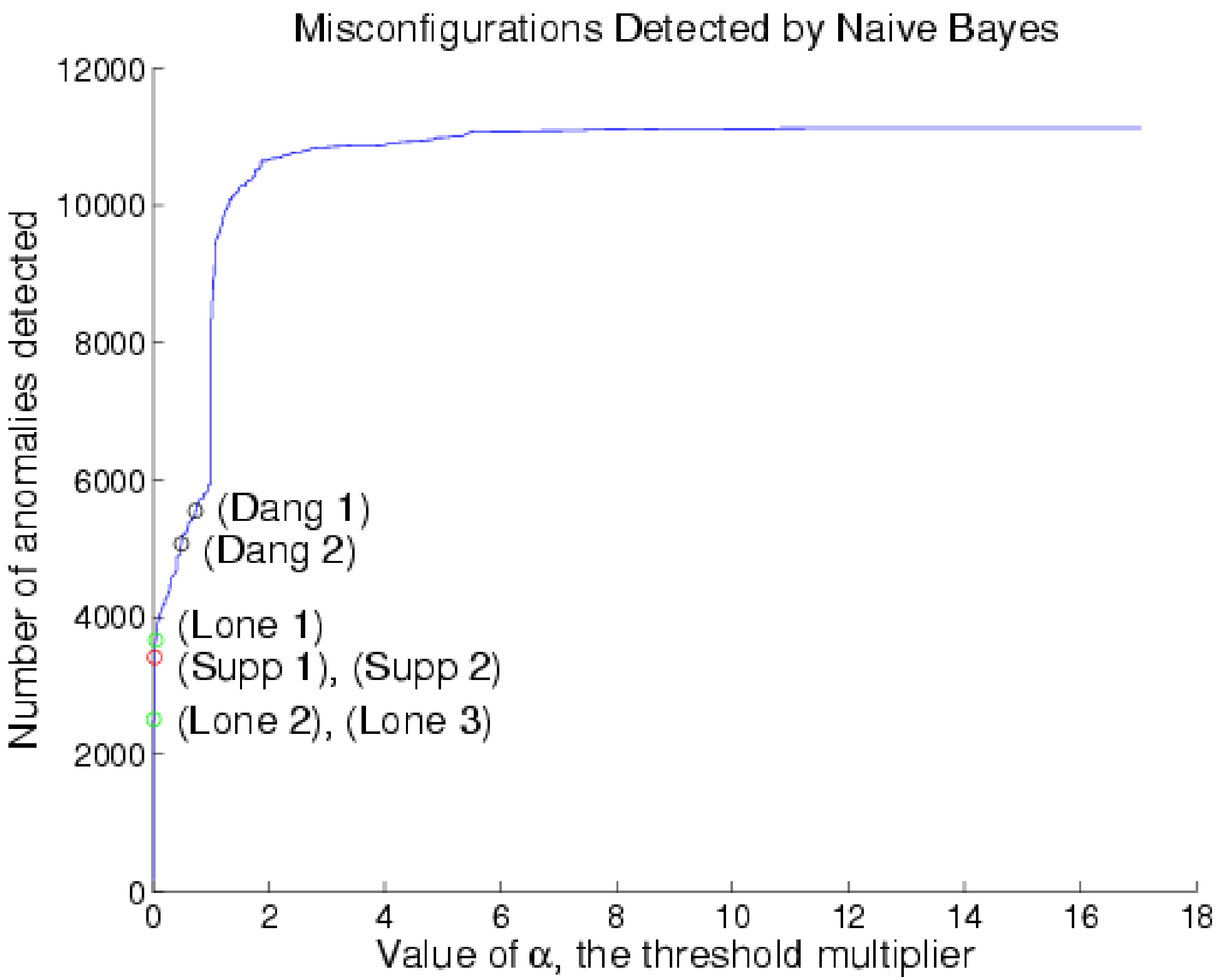
- Each detector was trained and tested on all 24 CMU router files
  - Training involves modeling probability distribution of each command
  - Testing involves classifying individual commands as anomalies using these probabilities
- We compute the minimum value for a necessary to classify each command as anomalous
- For each misconfiguration and each detector, we determine how many commands have to be classified as anomalous in order to detect it (those with a lower minimum a value)

Results

# commands detected with anomaly	Naïve Bayes	Joint Bayes	Structured Bayes
Lone 1	3661	2539	4498
Lone 2	2511	0	2608
Lone 3	2511	0	2608
Supp 1	3414	2539	1955
Supp 2	3414	2539	1955
Dang 1	5543	5591	5845
Dang 2	5065	4734	5700

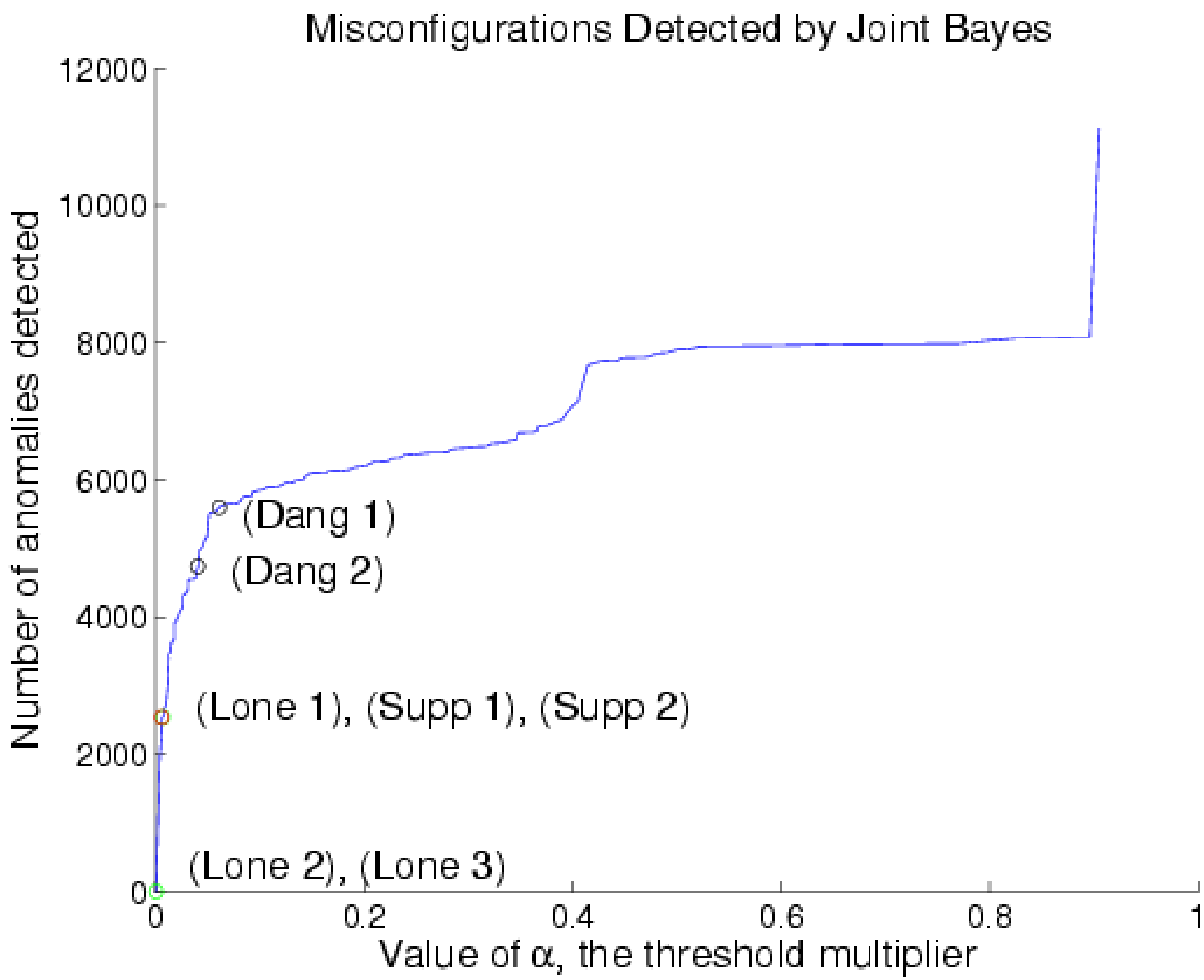
Total: 11,125 commands

Results



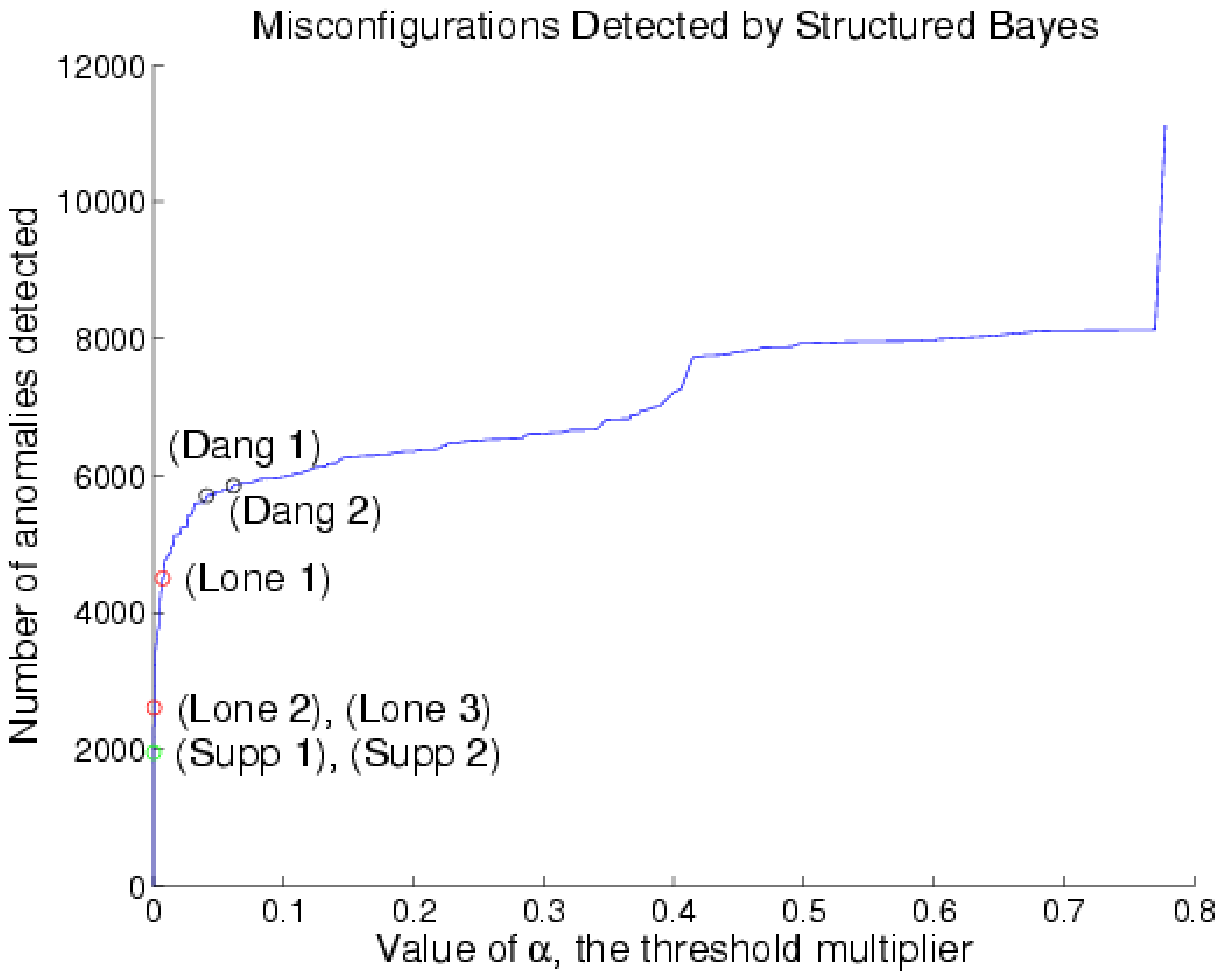
Misconfigurations Detected by Naive Bayes
12000
10000
8000
6000
4000
2000
0
Number of anomalies detected
0
2
4
6
8
10
12
14
16
18
Value of α, the threshold multiplier
(Dang 1)
(Dang 2)
(Lone 1)
(Supp 1), (Supp 2)
(Lone 2), (Lone 3)

Results



Misconfigurations Detected by Joint Bayes
Number of anomalies detected
12000
10000
8000
6000
4000
2000
0
(Dang 1)
(Dang 2)
(Lone 1), (Supp 1), (Supp 2)
(Lone 2), (Lone 3)
0
0.2
0.4
0.6
0.8
1
Value of α, the threshold multiplier

Results


Misconfigurations Detected by Structured Bayes
Number of anomalies detected
Value of α, the threshold multiplier
0
2000
4000
6000
8000
10000
12000
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
(Dang 1)
(Dang 2)
(Lone 1)
(Lone 2), (Lone 3)
(Supp 1), (Supp 2)

Analysis

- Joint Bayes is able to detect lone commands better than other two methods
- Structured Bayes has the interesting quality that it finds suppressed command anomalies earlier than the other detectors
- Dangling commands are hardest to find

Discussion

- Joint Bayes is the only method able to detect misconfigurations without a flood of other commands also being detected (specifically the type of anomaly Caldwell et al. mention in their paper)
- Relaxing the independence assumption among commands is likely to produce better results
  - With local context, we can do a better job detecting suppressed command anomalies
  - With global context, we can better detect dangling references

Conclusion

- With some success we were able to detect misconfigurations as statistical anomalies