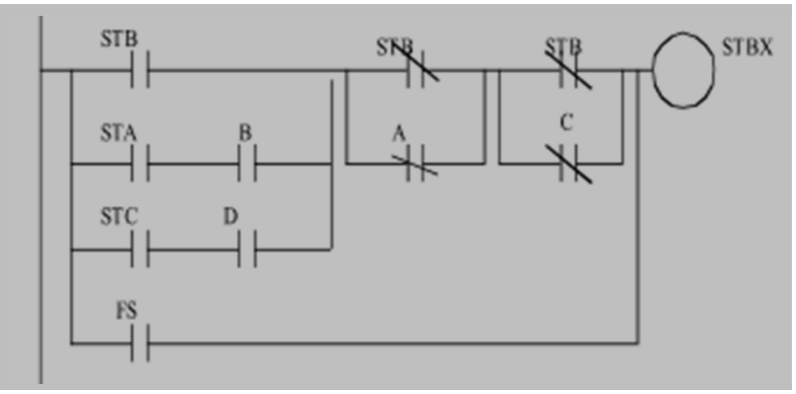
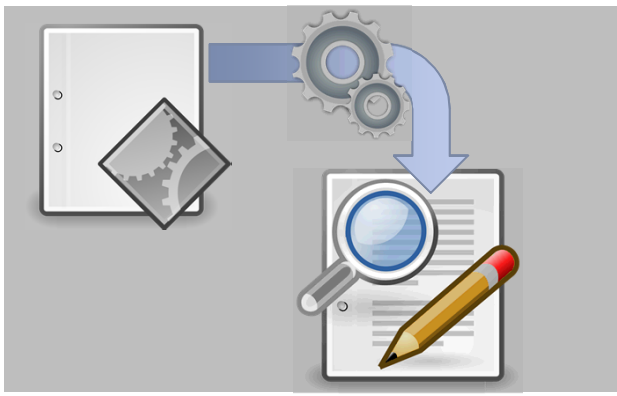
Exceptional service in the national interest

SAND2015-10536C

Sandia National Laboratories





Evolving Decision Trees to Detect Anomalies in Recurrent ICS Networks

Jasenko Hosic, Jereme Lamps, and Derek Hart

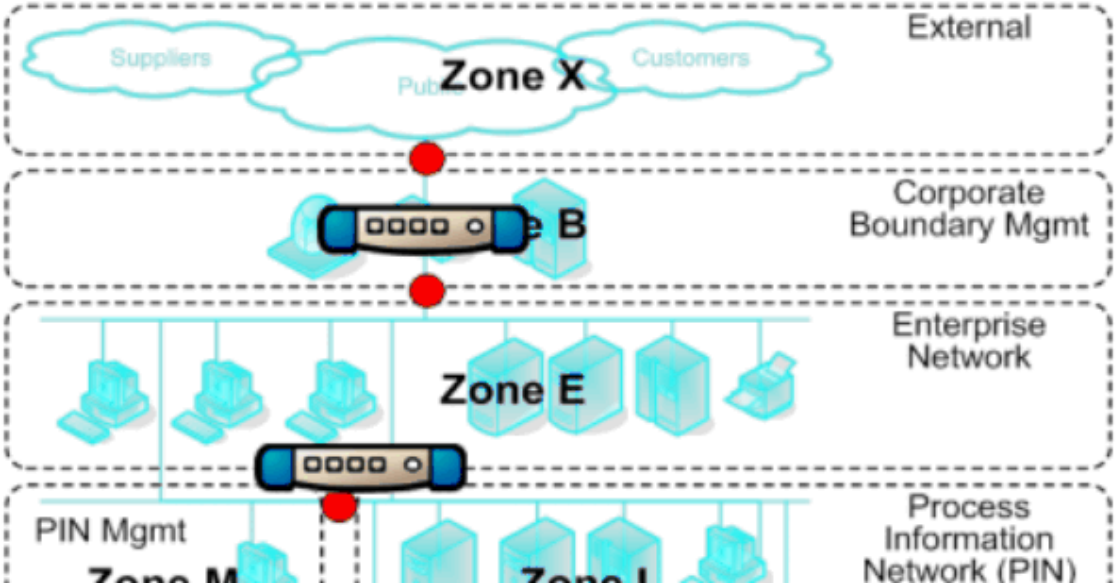
U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

Cyber attacks on ICS are *on the rise…*

Sandia National Laboratories

NEXT GENERATION CYBER ATTACKS TARGET OIL AND GAS SCADA

By Eric Byres, P. Eng., ISA Fellow, Tofino Security Product Group, Belden Inc. | **February 2012, Vol. 239 No. 2**



Anyone working with SCADA or industrial control systems (ICS) in the oil and gas industry is aware of the pressure to increase productivity and reduce costs through network integration. For example, sharing real-time data from field operations with management is standard practice for most companies. Similarly, the demand for remote support has made many pipeline control systems accessible via Internet-based technologies.

The Problem

Sandia National Laboratories

- ICS is a vulnerable and valuable target for attackers
- Security was largely an afterthought in the pre-stuxnet world
- ICS devices are old and have minimal processing capability

Related Work

Sandia National Laboratories

- Many papers focus on using machine learning to detect new attacks
- Machine learning experts have shown that these techniques are not useful for determining meaningful outliers, only recognizing patterns
- Recurrent nature of ICS networks lends itself well to this idea
- Multiple papers have been published about the potential of machine learning applicability in this domain
- Only a couple proof of concepts (using SOM with Bro and SVM with live data), but still produce an unreasonable number of false positives

Objectives

1. Detect various ICS network anomalies or failures with few or no false positives
2. Analyze the accuracy from different points of presence on the network
3. Stress the robustness of the solutions with numerous red herring events
4. Create a semi-real-time solution
5. Develop a general solution with minimal human intervention requirements for new event detection

Method Overview

- Generate “sensor trees” that take in buffered network data (pcaps, netflow, etc.) to determine if an event occurred
- **Trees are generated using a genetic programming algorithm. Machine learning algorithms can be trained to recognize patterns, which lends itself well to the recurrent nature of ICS networks**
- Each tree is responsible for one event (ie, a specific router going down, any router going down, etc)
- Multiple trees can be placed in multiple locations, depending on the training data available, the amount of events that need to be detected, the required level of accuracy, and the desired presence on the network
- Leaf nodes of the trees are extracted numeric metrics from the network data, normalized to values between 0 and 1

Method Overview – Cont.

- Internal nodes are fuzzy logic operators that relate the features (or metrics) together
- Buffered data allows for semi-real-time analysis. Multiple buffer sizes were tested, but ultimately ~20 minute buffers produced optimal results
- Data was collected using Minimega to make a "training set" as input to the fitness function – ultimately a supervised reinforcement learning approach
- Features vary in complexity
- High level features such as packet type collocation
- Simple metrics such as average packets sent per minute

Network Diagrams

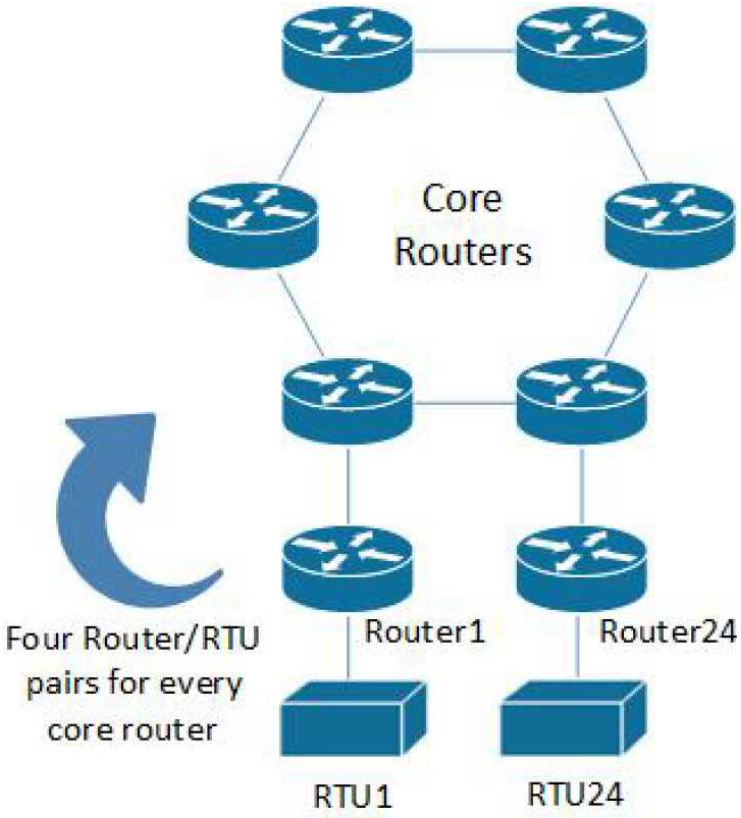


Fig. 1. A subset of our 24-bus network

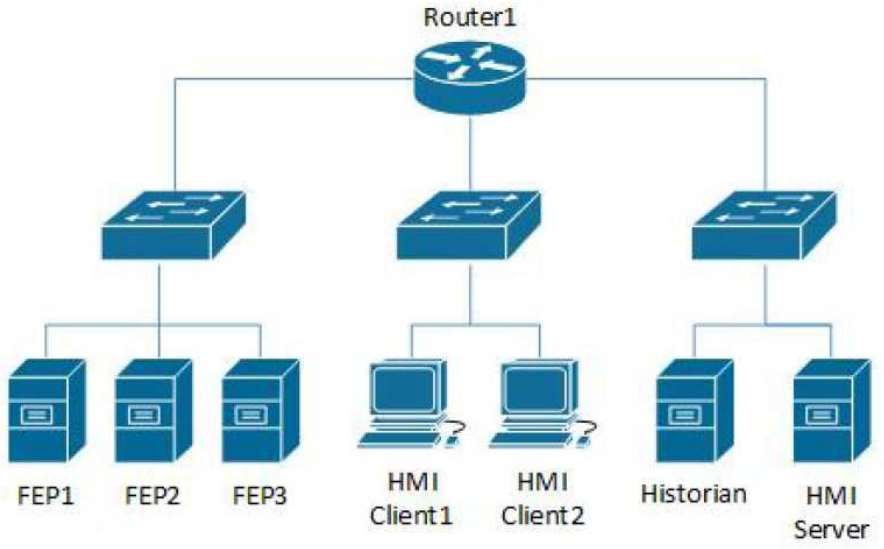


Fig. 2. An example of devices connected to a router in the 24-bus network

Data Collection

- Collected data from various ‘tap’ locations through out the network, gathering pcaps and netflow data
- Attempting to detect 3 events: any router down, specific router down, specific FEP down
- Performed 20 different data collection chunks, where each lasts for ~20 minutes, and from 3 different ‘tap’ locations, resulting in 1200 minutes (20x20x3) of data

Data Collection – Cont.

TABLE I
EXPERIMENT DESCRIPTIONS

Exp. No.	Fail Event	Tap Location
1	Specific router	Next to target router
2	Specific router	Several hops, next to RTU
3	Specific router	Several hops, next to FEP
4	Any router	Next to router
5	Any router	Next to RTU
6	Any router	Next to FEP
7	Specific FEP	Several hops, next to router
8	Specific FEP	Several hops, next to RTU
9	Specific FEP	Several hops, next to diff. FEP

Feature Extraction

- The feature extraction functions and genetic programming (GP) framework was developed in Python 2.7 (> 1000 LoC).
- Imported various python modules (dpkt, pcap, and pyevolve) to aid in rapid development
- Developed over 10 specific features to extract from the data
- Some example features were:
  - Packet type collocation
  - Average flow time
  - Communication pattern breaks
  - Average packet length
  - Average TTL

Individual Feature Performance

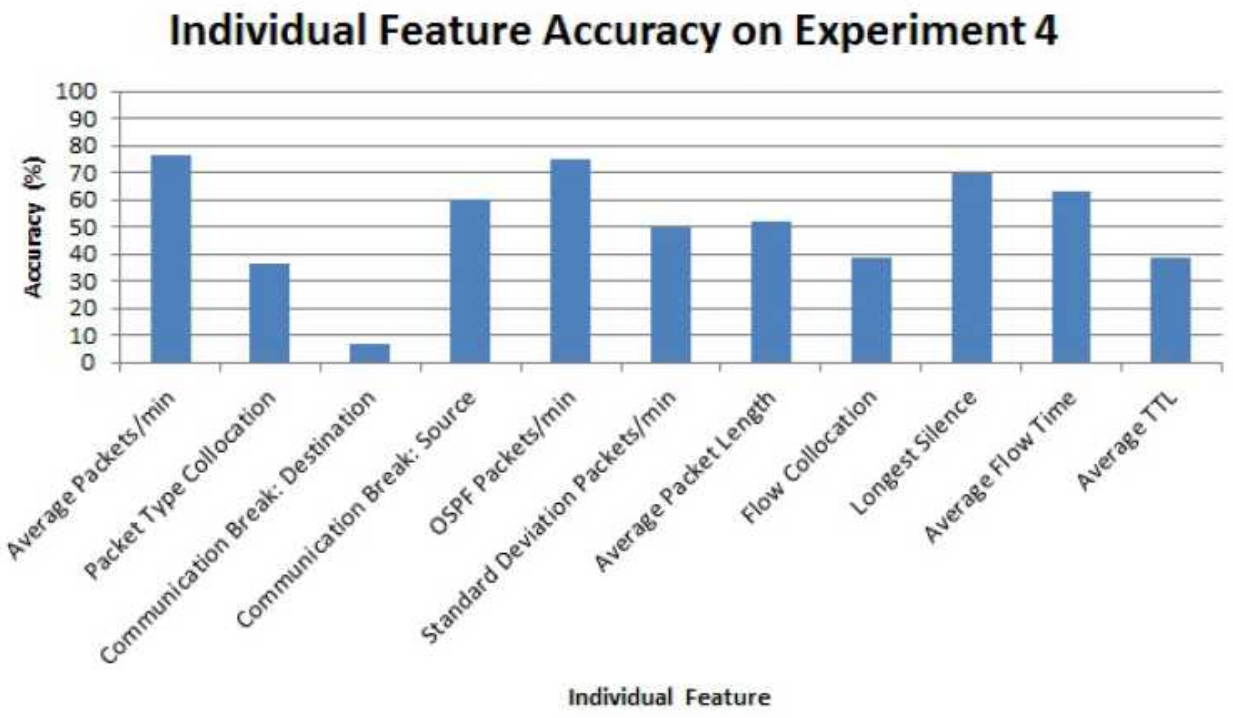


Fig. 3. Individual feature accuracy on Experiment 4

Genetic Programming Parameters

TABLE III
GP PARAMETERS

Parameter	Default	Exp. 5	Exp. 6
μ	100	200	200
λ	20	40	40
max depth	4	4	5
selection	k-tourn.	k-tourn.	k-tourn.
survival	k-tourn.	k-tourn.	k-tourn.
k	7	5	5
crossover	single-point	single-point	single-point
mutation	sub-tree	sub-tree	sub-tree
mutation rate	.15	.8	.85
termination	5000 eval.	5000 eval.	5000 eval.
Acceptance Thresh.	.9	.75	.75

*Parsimony pressure introduced to prefer smaller solutions with equal fitness

Genetic Programming Parameters

- Parameters were chosen to be “generic defaults”
- Algorithm would likely perform better with proper parameter tuning (ie, meta-algorithm or hand-tuning)
- Algorithm performs really well without tuning
- Intended to be used by people that are not machine learning experts, so no emphasis was placed on this

Fitness Function

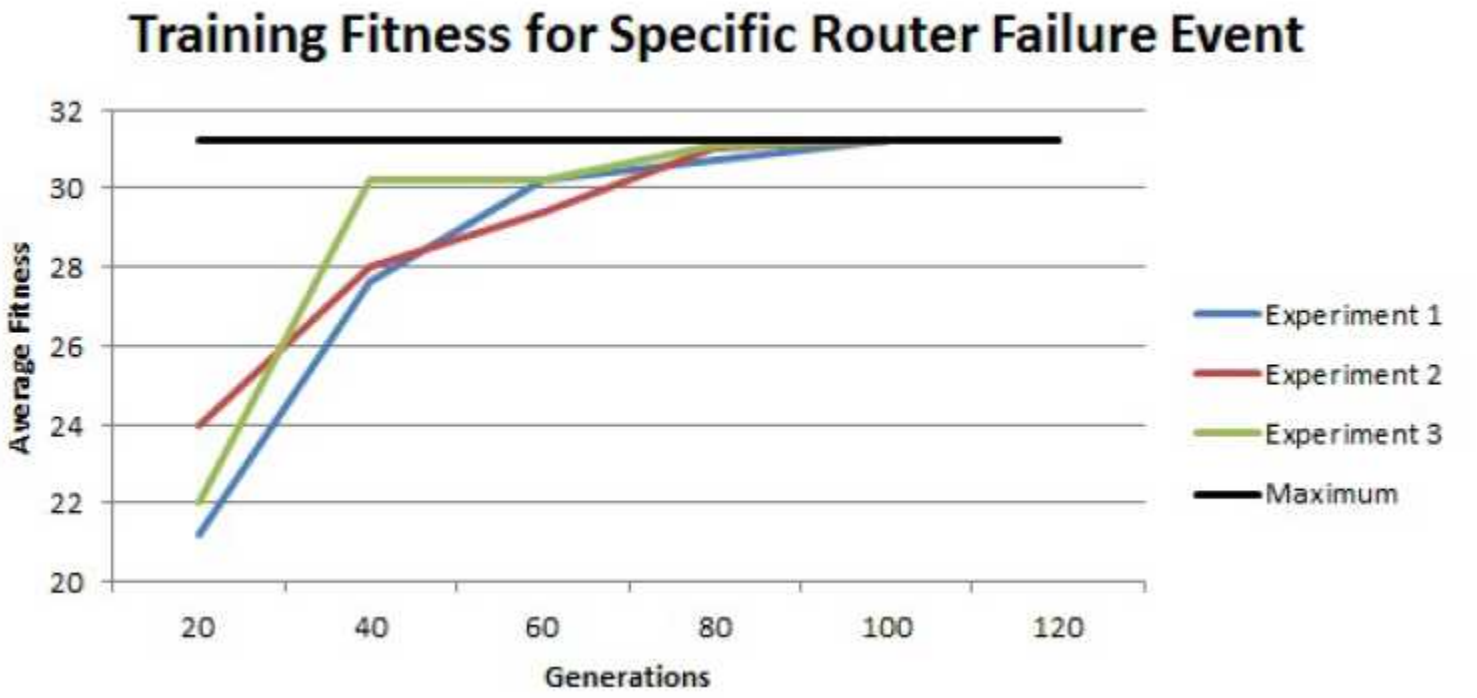
Algorithm 1 Fitness function pseudocode

```
function get_fitness(tree)
    total ← 0
    for all data ∈ data_sets do
        val ← eval_tree(tree)
        if val ≥ accept_thresh && event then
            total ← total + num_not_event
        end if
        if val ≤ 1 − accept_thresh && not_event then
            total ← total + num_event
        end if
    end for
return total
end function
```

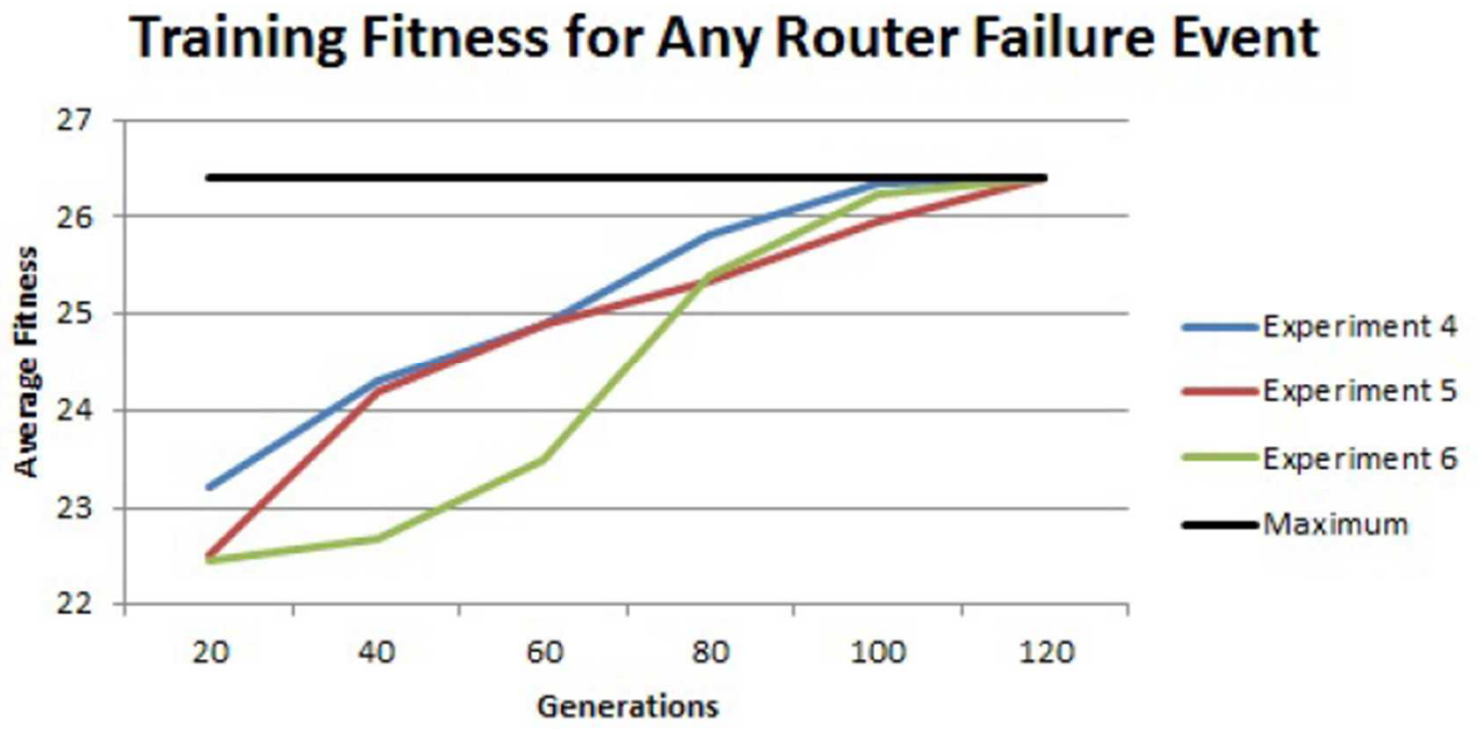
Experiments

- 80% of the data was used for training, 20% used for testing
- Each of the experiments from Table I were run 30 times with 5-point cross-validation such that each dataset from Table II was in the testing set at least once

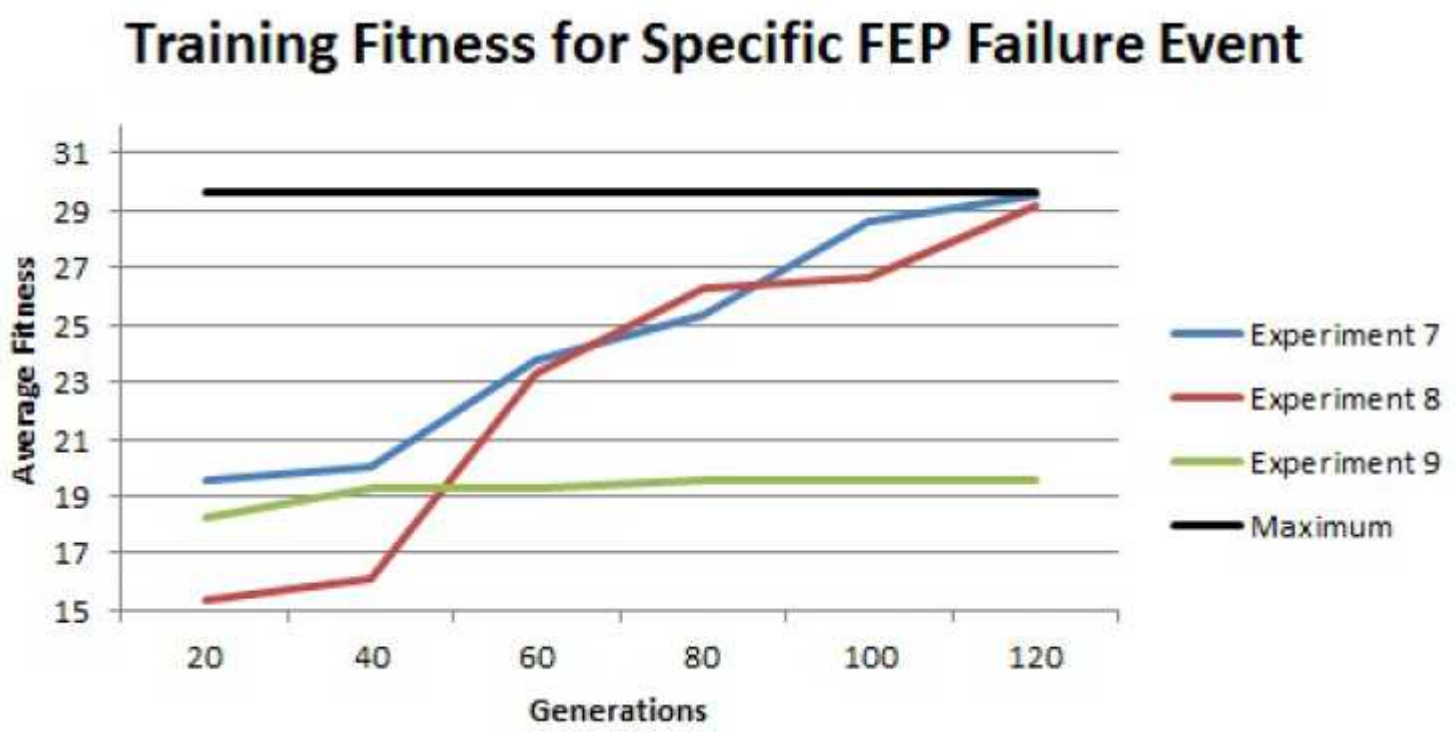
Average Fitness vs. Generations



Average Fitness vs. Generations



Average Fitness vs. Generations



Accuracy

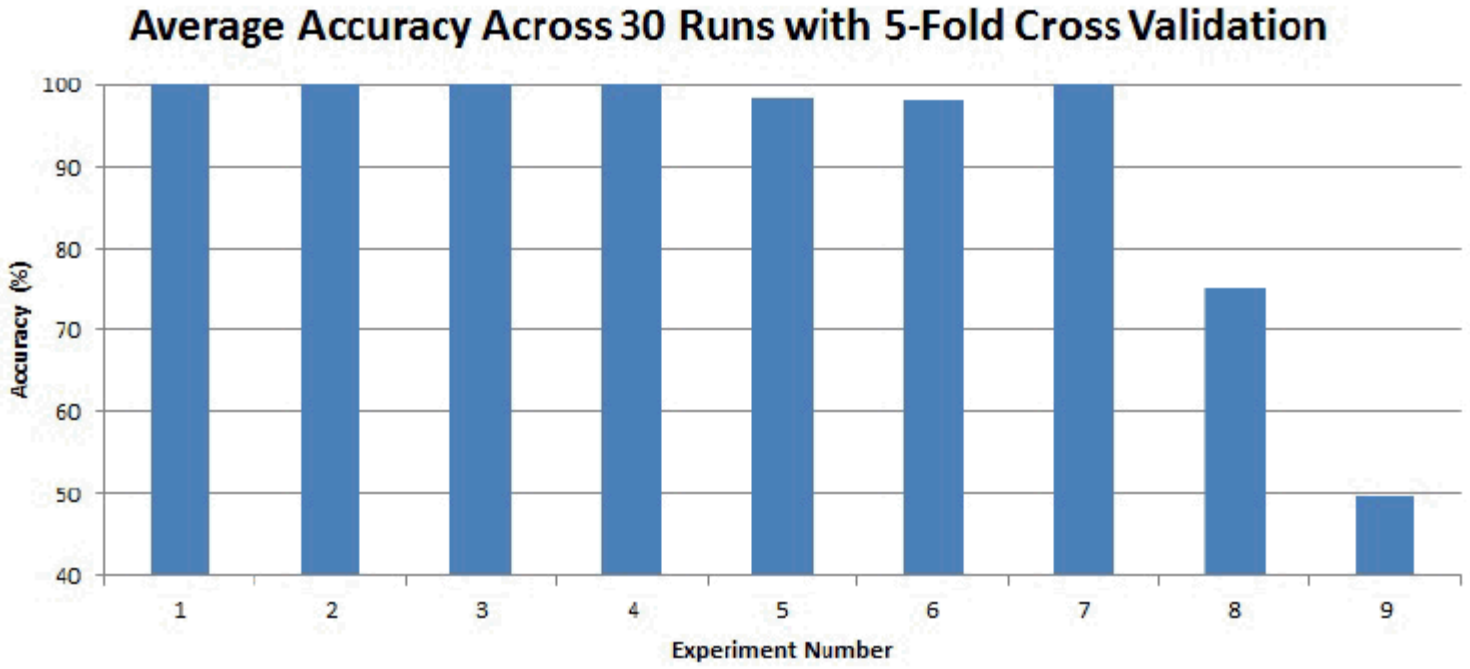
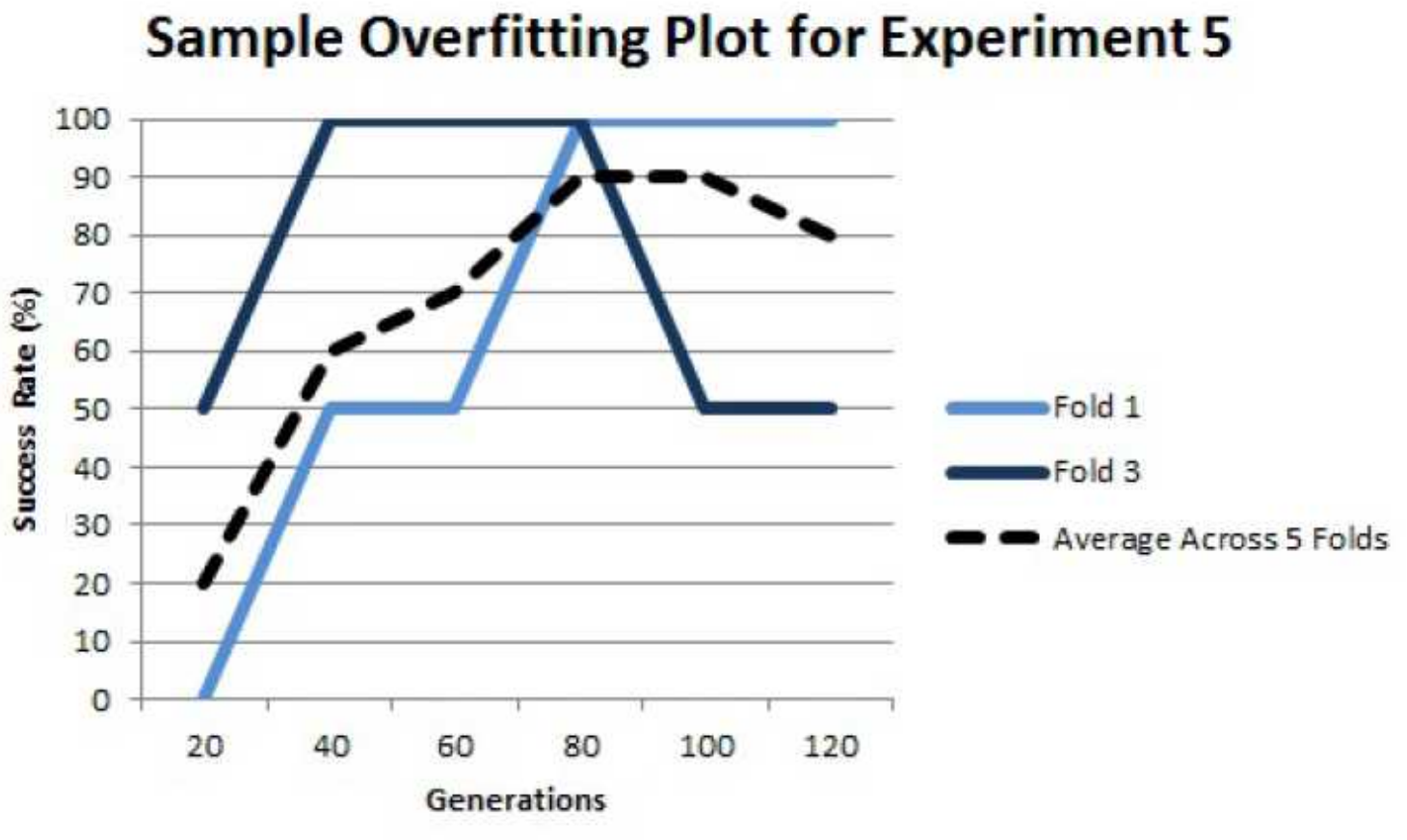


Fig. 7. Accuracy per experiment, averaged over all runs and cross validations

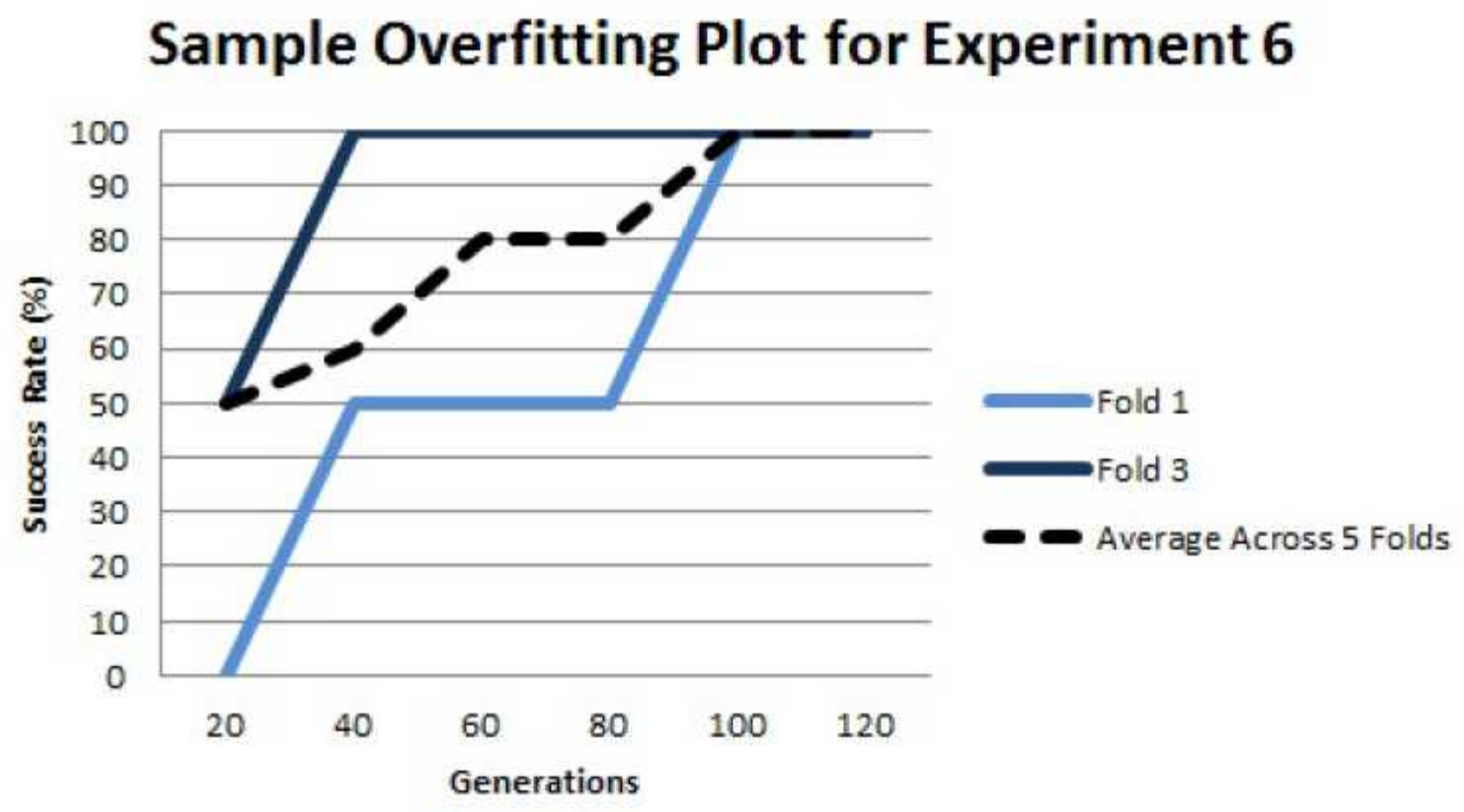
Overfitting

- When parameters are not optimized, there is a chance of overfitting
- The GP algorithm could force the trees to over-train to the training set and perform worse on the eventual testing set
- Analysis shows that this occurs on several folds of a few of our imperfect experiment results
- This the trade-off with avoiding intensive parameter tuning

Overfitting – Cont.



Overfitting – Cont.



Example Solutions

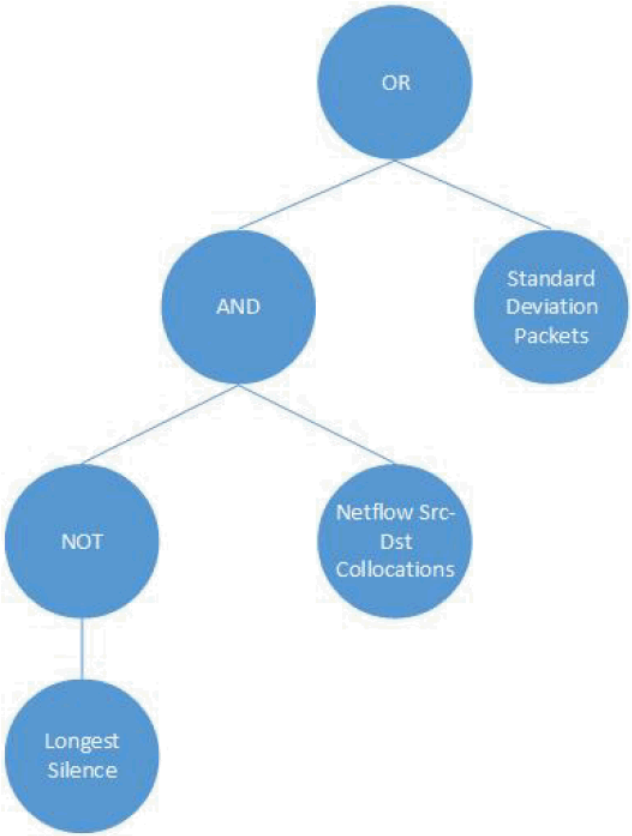


Fig. 10. Sample decision tree for experiment 2

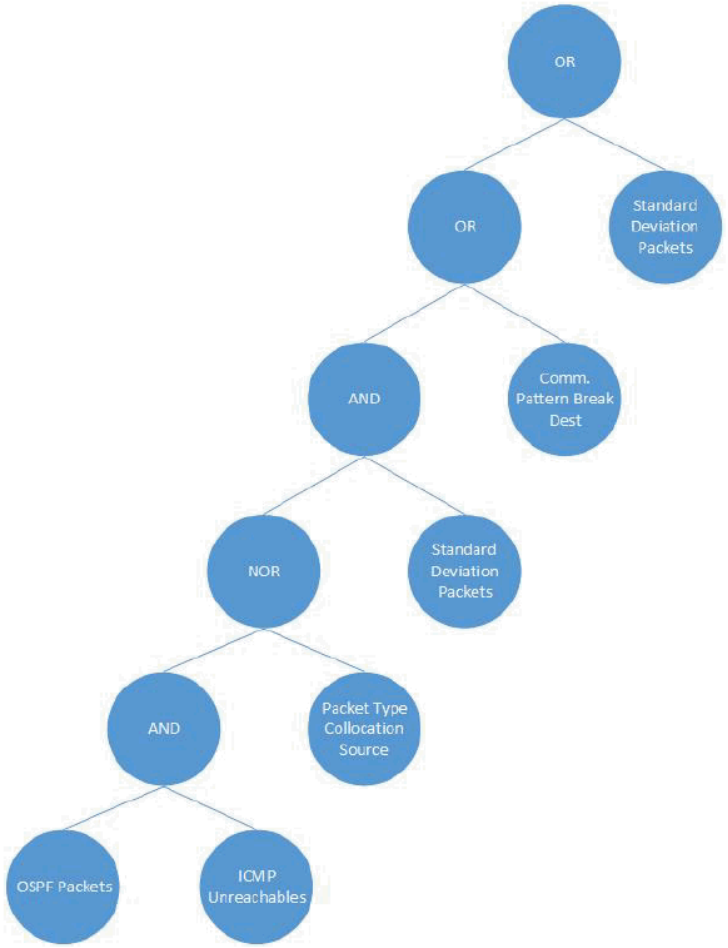


Fig. 11. Sample decision tree for experiment 2

*Different folds used for each tree. Fig. 11 training set was harder to optimize against

Analysis

- Complexity of resultant trees varies depending on the training set provided and point of presence on the network
- Solutions make logical sense when evaluated
- Some redundant logic exists (ie, NOT-NOT-NOT-feature) but parsimony pressure filters out most of those issues
- This method works best when the trees are trained on data collected nearest to the event in question
- For truly optimal results, multiple trees would be generated for one event and aggregated; due to low processing cost, they can be correlated to make a decision

Impact

- General solution to the problem
- All major processing is done a-priori, and the resultant solutions are models that do not need to "learn" more
- Can be deployed in semi-real-time to sit quietly on nearly any node or several different nodes at once
- Small network presence can be achieved – needs to just have access to the necessary data for the feature extraction algorithms
- Could be ported to accept data from multiple other network nodes, aggregate it, and evaluate the data; does not need to be directly in the network loop
- When an event is detected, the resultant action can be anything the operator desires (ie, send an alert)

Impact – Cont.

- No parameter tuning is required, so operators not versed in machine learning can run the algorithm with great results
- Stressed the method by causing many false events atypical of real environment, such as multiple devices failing and coming back up in order to “trick” it, but still got solid results . Would perform even better in a normal ICS environment with less noise
- Resulting trees can process data quickly and require little computing power
- Different types of data can be introduced, such as system logs, images, etc. if the proper feature extraction algorithms accompany the data
- Can easily be abstracted to an ensemble approach, such as using multiple evolved trees to create a Learning Classifier System (LCS), or using a classification algorithm as a single feature in the trees

Impact – Cont.

- Any other event can theoretically be detected using the same method if appropriate training data is given
- This method is only as strong as the feature extraction algorithms
- For detecting other events, new feature extractions will likely need to be implemented that fit the situation
- Other possible events that could be detected:
- PLC process logic update
- PLC firmware update
- Abnormal values sent by PLC
- Data exfiltration
- PLC communication with new nodes or at odd intervals

Future Work

1. Reinforce the results by using a HITL or pure-hardware approach as opposed to a simulated environment with Minimega

2. Use the approach to detect the more nuanced events listed in the previous slide

3. Run multiple trees in a live system to detect a specific set of failures, then attack the system

4. Minimize the amount of data given to the algorithm to further stress its robustness

5. Gather specific statistics about processing power requirements

Questions?

Sandia National Laboratories