# Big Data in Reliability and Security: Applications

**Saurabh Bagchi**
**School of Electrical and Computer Engineering**
**Department of Computer Science**
**Purdue University**


**Material at: *http://bit.ly/whin-big-data-apps***



# Four Relevant Aspects

1. Principles in order to use Big Data (synonymous with Machine Learning)
2. Big Data helping in Reliable Operation of Systems
3. Big Data helping in Secure Operation of Systems
4. Adversarial ML

# Reliability Principles in order to Use Big Data

- Data cleaning and integration
  - Multiple sources of data being used
  - Structured data sources – in relational databases
  - Unstructured data sources
    - These vary in accuracy
    - More accurate: Sensor data with calibration metadata
    - Less accurate/more noisy: Social network data
  - Hadoop is widely used as an underlying building block for capturing and processing big data – Hadoop Distributed File System (HDFS) and MapReduce
- Streaming data processing is often required for real-time results
  - Data cleaning and integration needs to be done in streaming manner
  - Easier to manage data at rest than data in motion (streaming data)
  - Progressive data cleaning: Initially look for broad patterns; When incorporating into business practice higher importance to clean data

# Big Data Helping in Reliable Operation of Systems

- Dependability solutions driven by 2 emerging trends
  - Execution environments are changing: "All components are developed in-house, are open source and well-understood" → "Systems are made up of third-party software components, some of which are partially opaque to the system owner and interactions amongst the components have many evolving patterns"
  - Dependability solutions are found fragile: "Rule-based, always on, not adaptive to dynamic workloads" → "Data analytics based monitoring for correctness, which adapts to heterogeneous architectures and new workloads"
- But 3 characteristics are important for ML applied to reliability
  1. Interpretability
  2. Fast execution
  3. Resilience to noise in training data
  4. (Optionally) Semi-supervised learning

# Analyzing Job Failures on Compute Clusters

- Jobs submitted to Purdue's central compute cluster
- Analyze 3M jobs over 28 months (Mar 2015-Jun 2017); 617 unique users
- 3 kinds of data:
  - Job accounting logs (from TORQUE)
  - Resource utilization stat on each node (from TACC_Stats)
  - Node failure data (from Kickstand, Sensu, Nagios)

| | | |
|---|---|---|
| **Compute** | Node | 580 |
| | Memory | 64 GB/node |
| | Processor | Xeon E52670 + 2x Xeon Phi/node |
| | Resilience | ECC-protected CPU & Xeon Phi memory |
| **Local I/O** | I/O bandwidth | 100MB/s |
| | Capacity | 500 GB |
| | Type | SATA |
| **Network I/O** | I/O bandwidth | 23GB/s |
| | Capacity | 1.4 PB |
| | Resilience | Disks: RAID 6, Index Disks: RAID 1+ 0, OSS : Active-Active HA pair, MDS: Active-Passive HA pair |
| **Network** | Peak node inj. | 40Gb/s |
| | Topology | Fat Tree |
| | Resilience | Infiniband Forward Error Correction (FEC) |

**Open data set at: *https://purdue.edu/fresco***

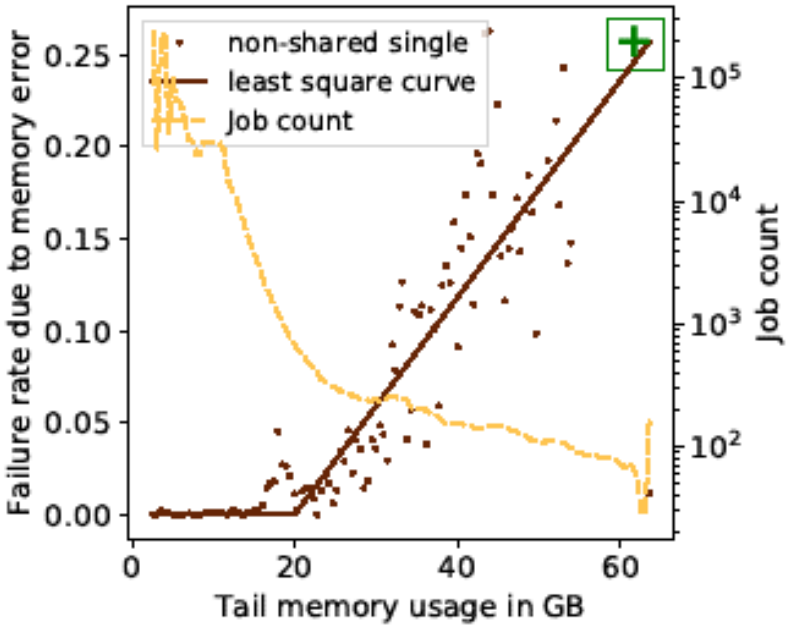


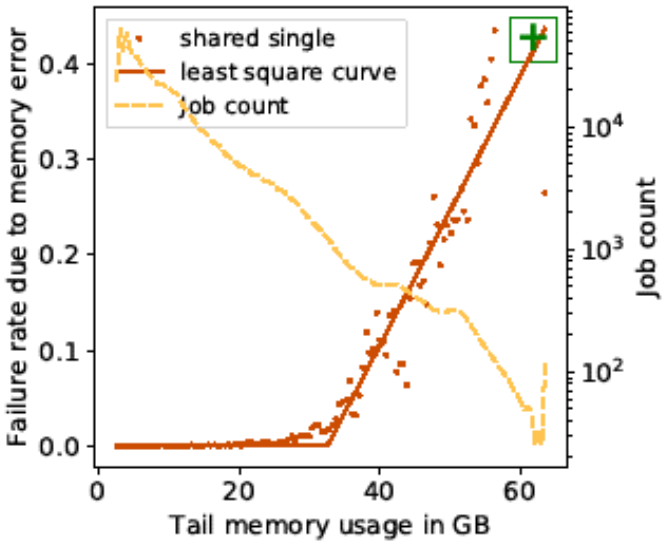
PURDUE UNIVERSITY

# Effect of Resource Usages on Job Failures

- 5 primary kinds of resources, both local and remote, namely, memory, local and remote IO, network and job node-seconds
- Resource utilization considered in the tail period
- Data preprocessing:
  - Create equal-sized bins for the resource usages
  - Only consider bins that have ≥ 100 jobs
  - Aggregate read and write rates
  - Use log scale where there is wide range of values
- Perform hypothesis testing with null hypothesis of the form: "Job failure rate is *not* correlated with resource usage of resource X"
  - So if null hypothesis is rejected, we conclude there is effect of that resource usage on job failures
- We model failure rate plot using best-fit statistical distribution
  - Sometime $R^2$ is not significant but $H_0$ rejected

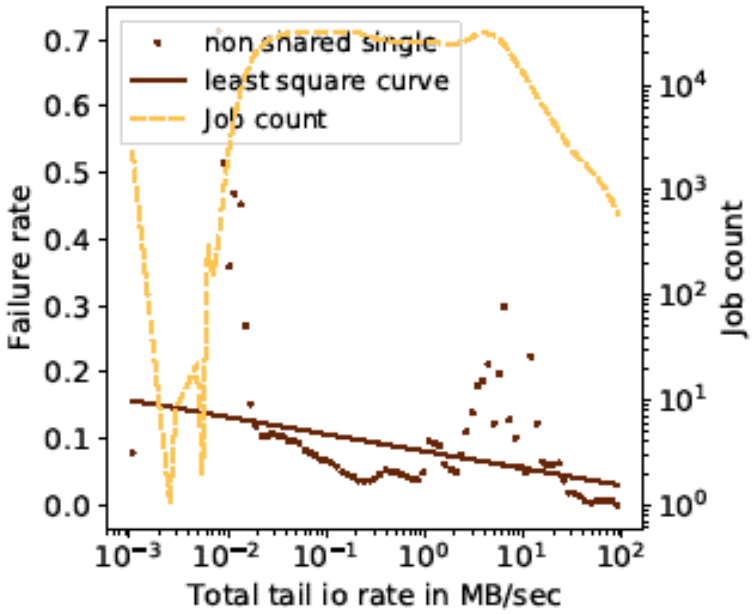
# Effect of Memory Usage

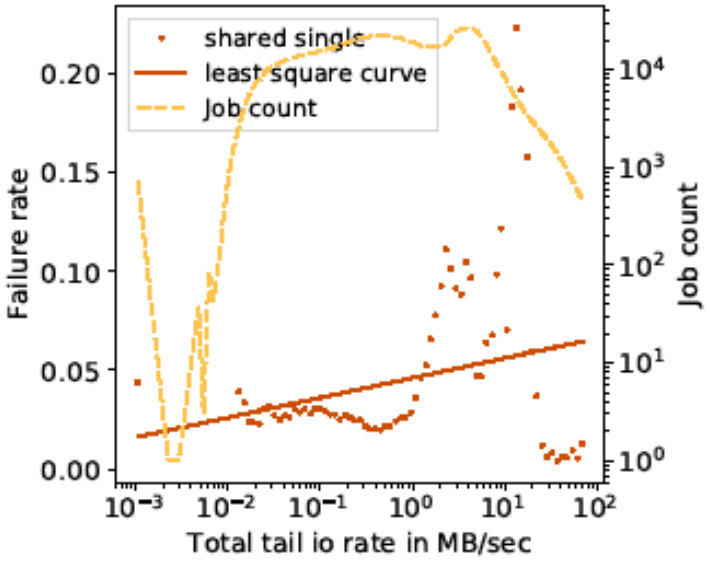



- Memory related errors are common among failed jobs
- Hypothesis testing shows positive correlation
- Failure rate increasing even when the available free memory is more than half of the total node memory capacity!

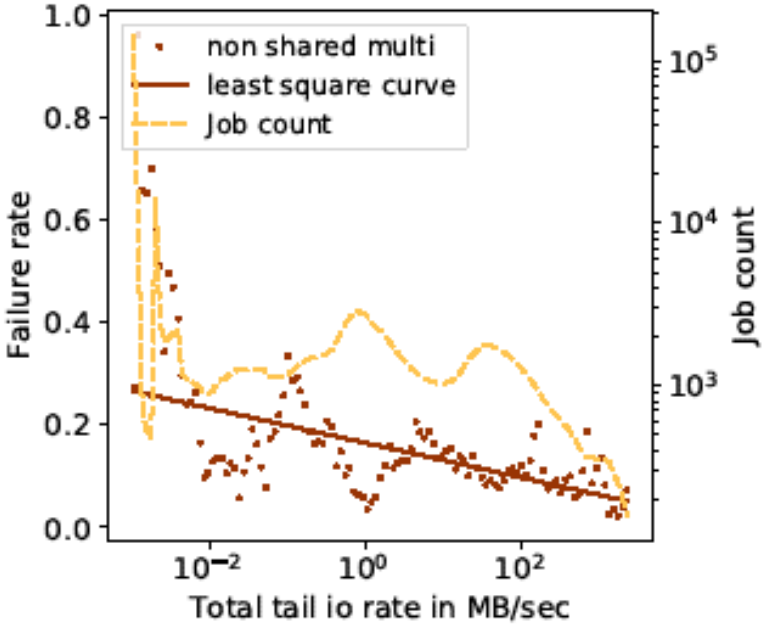
# Local IO

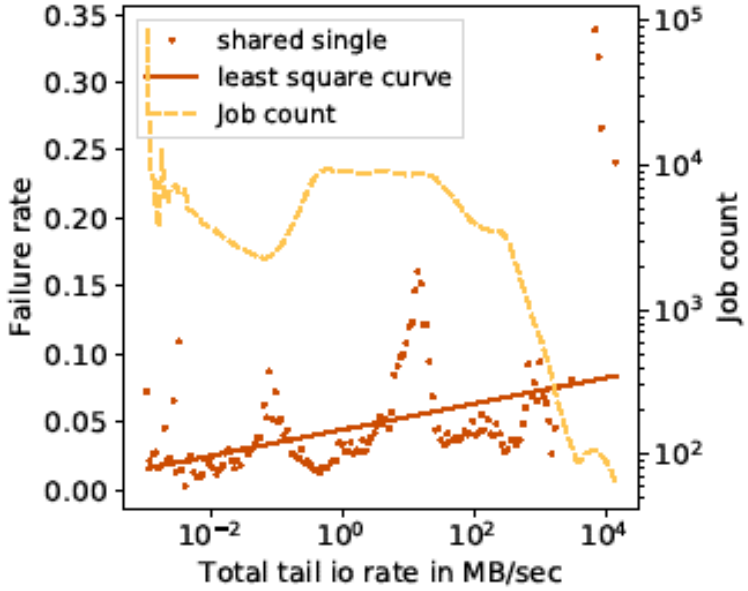



- For non-shared, negative correlation of failures with local IO!
  - Jobs are failing due to systems issues such as disk failure or faulty drivers that restrict I/O usages
  - Peak in the failure rate around 6 MB/s is much lower than the specified operating I/O limit of available local discs (100 MB/s)

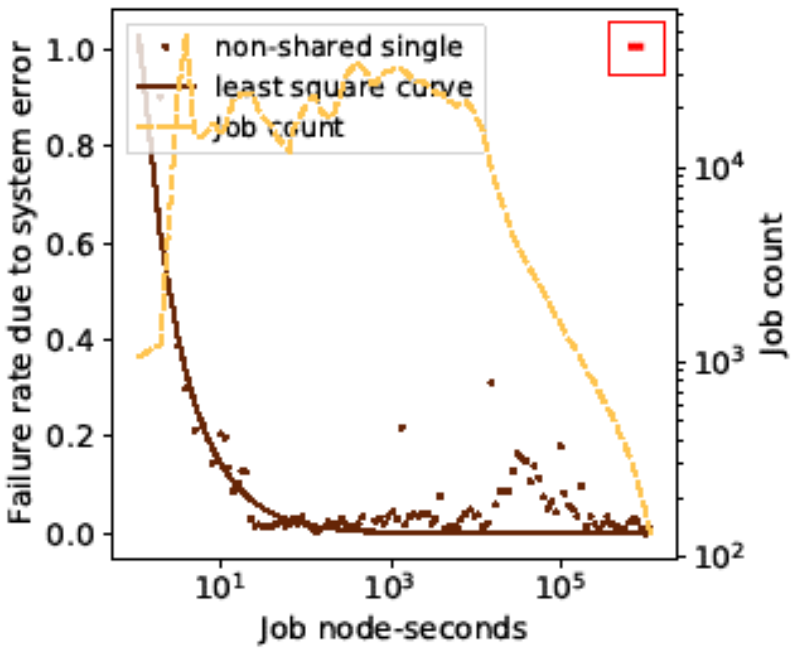
# Network File System

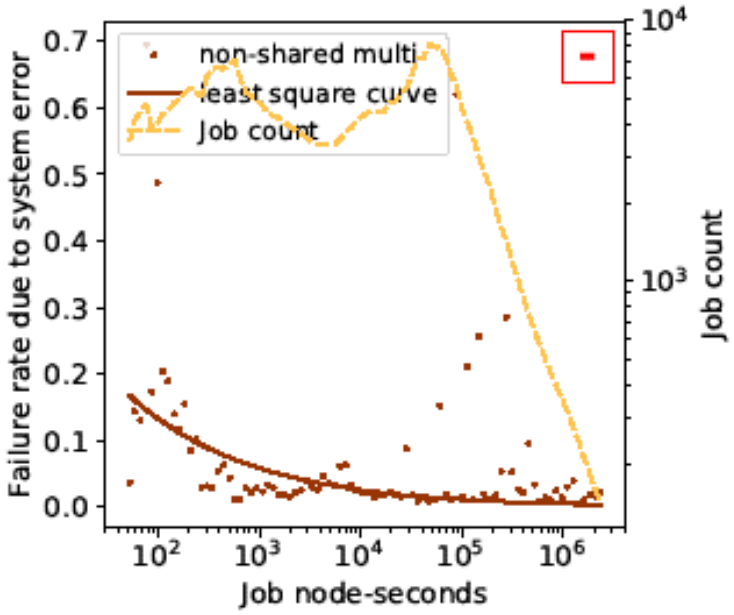



- Metric: total remote I/O per node
- Negative correlation of failure rate in the non-shared environment
- Positive correlation of failure rate in shared environment

# Job Execution Time




- Prior studies have found that there is a positive correlation of failure rate with the total execution time of an application
- We find a negative slope for all three categories of jobs—non-shared single, non-shared multi, and shared single
- Counter-intuitive?

Novice users jobs fail quickly while jobs that have run for a long time run to success

# Big Data Helping in Secure Operation of Systems

- Consider a standard security control: Intrusion Detection System (IDS)
  - Misuse-based (synonymously, signature-based)
  - Anomaly-based
  - Hybrid
- ML approach usually consists of two phases: training and
- testing. Often, the following steps are performed:
  1. Identify class attributes (features) and classes from training data
  2. Do dimensionality reduction
  3. Learn the model using training data
  4. Use the trained model to classify the unknown data (here, host traces or network traces)
  - Often, also a validation phase, to fit parameters
- Prediction for security domain can be two-class (benign or malicious), or occasionally 3 class (also suspect class)

# Features Used in Security Classification

| IP Header (IPv4) | |
|---|---|
| Internet Header Length | The number of 32-bit words in the header |
| Total Length | The entire packet size, including header and data, in bytes |
| Time To Live | This field limits a datagram's lifetime, in hops (or time) |
| Protocol | The protocol used in the data portion of the IP datagram |
| Source address | This field is the IPv4 address of the sender of the datagram |
| Destination address | This field is the IPv4 address of the receiver of the datagram |
| **TCP Packet** | |
| Source port | Identifies the sending port |
| Destination port | Identifies the receiving port |
| Sequence number | Initial or accumulated sequence number |
| Acknowledgement number | The next sequence number that the receiver is expecting |
| Data offset | Specifies the size of the TCP header in 32-bit words |
| Flags (control bits) | NS, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN |
| **UDP Packet** | |
| Source port | Identifies the sending port |
| Destination port | Identifies the receiving port |
| Length | The length in bytes of the UDP header and UDP data |

Network packet header fields

| NetFlow Data – Simple Network Management Protocol (SNMP) | |
|---|---|
| Ingress interface (SNMP ifIndex) | Router information |
| Source IP address | |
| Destination IP address | |
| IP protocol | IP protocol number |
| Source port | UDP or TCP ports; 0 for other protocols |
| Destination port | UDP or TCP ports; 0 for other protocols |
| IP Type of Service | Priority level of the flow |
| **NetFlow Data – Flow Statistics** | |
| IP protocol | IP protocol number |
| Destination IP address | |
| Source IP address | |
| Destination port | |
| Source port | |
| Bytes per packet | The flow analyzer captures this statistic |
| Packets per flow | Number of packets in the flow |
| TCP flags | NS, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN |

Router NetFlow fields

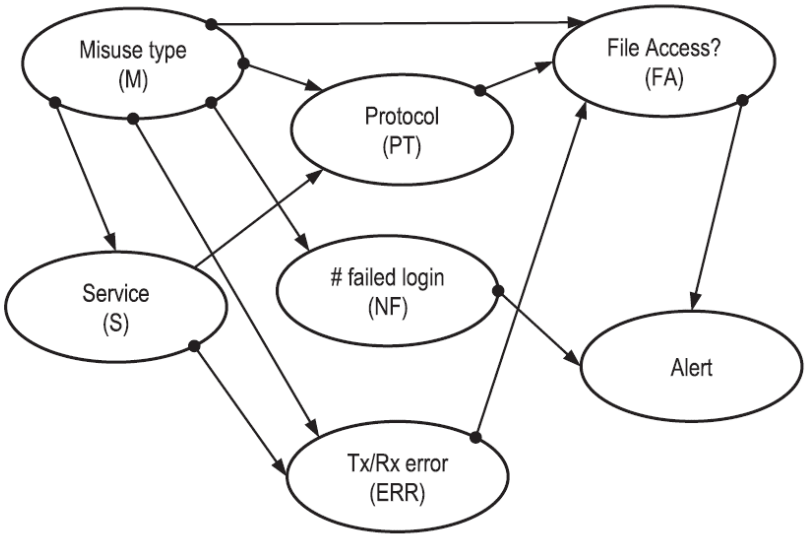
# Early Days

- Association Rules and Fuzzy Association Rules
  - Introduced in 1993 to discover interesting co-occurrences in supermarket data
  - Finds frequent sets of items and from the frequent items sets such as {X, Y}, generates association rules of the form: X → Y and/or Y → X.
- Security context:
  - If (Packet *i* header has A and Packet *i* body has B) → (Packet *i*+1 header has C): Example of anomaly-based rule
  - High-level design decision: What level of abstraction to use for the rules?

R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in Proc. ACM Sigmod, 1993, pp. 207-216.

# Bayesian Network

- Probabilistic graphical model that represents the variables and the relationships between them




Example Bayesian Network for Signature Detection

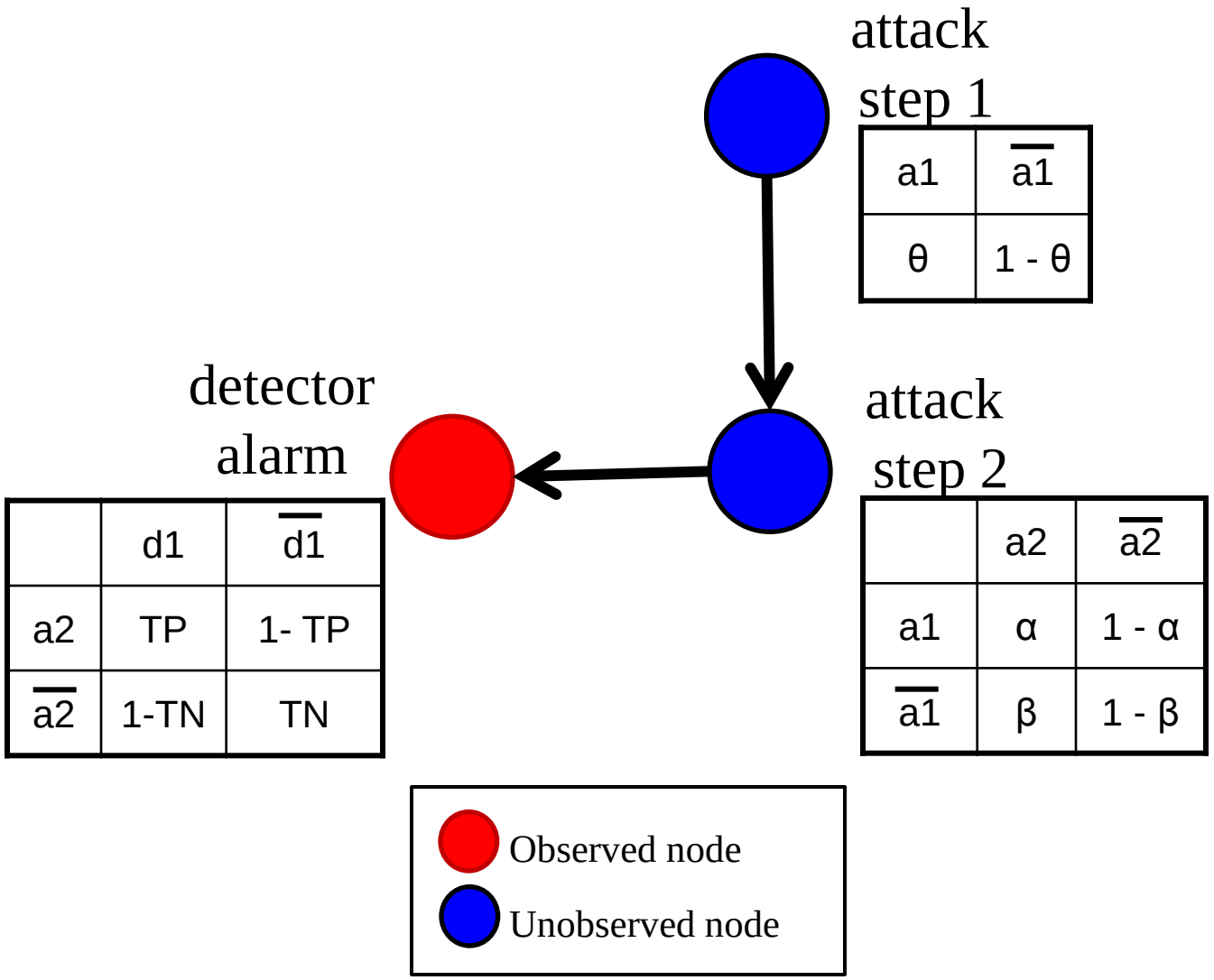
Example of anomaly detection using BN on TCP/IP packets [Kruegel, ACSAC-03]

- DARPA 1999 data set is used to excite the OS kernel by TCP/IP packets
- A set of attributes based on these system calls, such as system call argument string length and distribution and character distribution, are compared using a Pearson test
- These features are used in a Bayesian network to calculate the probability of a normal state or an anomaly state

# Use of BN for Sensor Placement

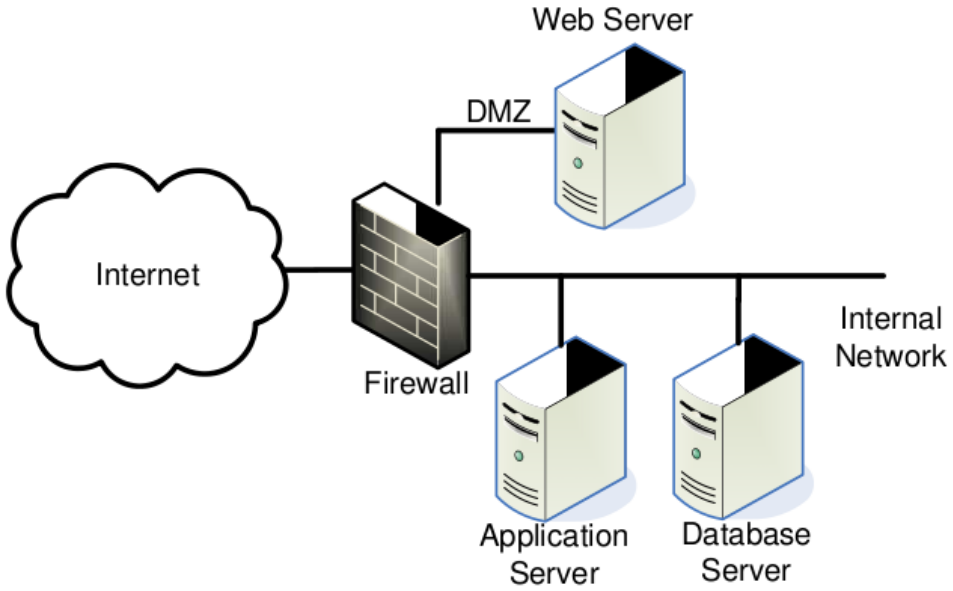
1. Use **attack graphs** as input for our method
   - Definition: Graphical representation of the different steps used by attacker to compromise different elements
   - Nodes represent attack stages, edges represent attack paths
2. Convert attack graphs to **Bayesian networks (BNets)**
   - Definition: BNets provide convenient framework for modeling relationship between attack steps and detector alerts
   - Include probability values and connect detectors to attack graph

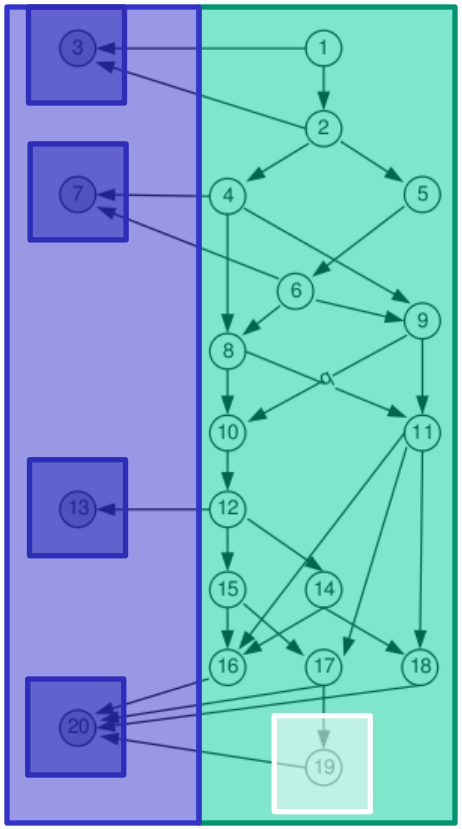
# From attack graph to a Bayesian Network



# Application to be Protected

- E-Commerce system and corresponding Bayesian network



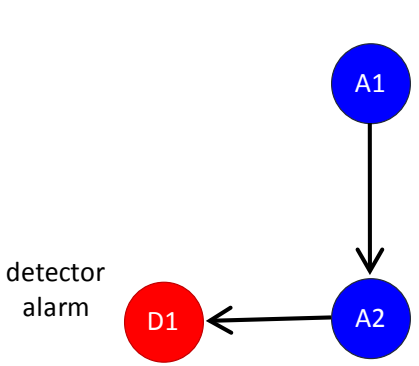




attack goal

Compute $P(attack\_goal|d_i)$

# Problem 7 - **Bayesian Network for Security**

Consider that we have an attack graph fragment like below. Which of the conditional probability tables is most likely for the Intrusion Detection Sensor (IDS) node *D1*? Also think why. We are considering a commercial strength IDS here.




| | $A2$ | $\overline{A2}$ |
|---|---|---|
| **A1** | 0.8 | 0.2 |
| $\overline{A1}$ | 0.1 | 0.9 |

1.

| | **D1** | $\overline{D1}$ |
|---|---|---|
| **A2** | 0.9 | 0.1 |
| $\overline{A2}$ | 0.2 | 0.8 |

2.

| | **D1** | $\overline{D1}$ |
|---|---|---|
| **A2** | 0.1 | 0.9 |
| $\overline{A2}$ | 0.2 | 0.8 |

3.

| | **D1** | $\overline{D1}$ |
|---|---|---|
| **A2** | 0.1 | 0.9 |
| $\overline{A2}$ | 0.8 | 0.2 |

4.

| | D1 | $\overline{D1}$ |
|---|---|---|
| A2 | 0.1 | 0.9 |
| $\overline{A2}$ | 0.3 | 0.8 |

# Proposed Method

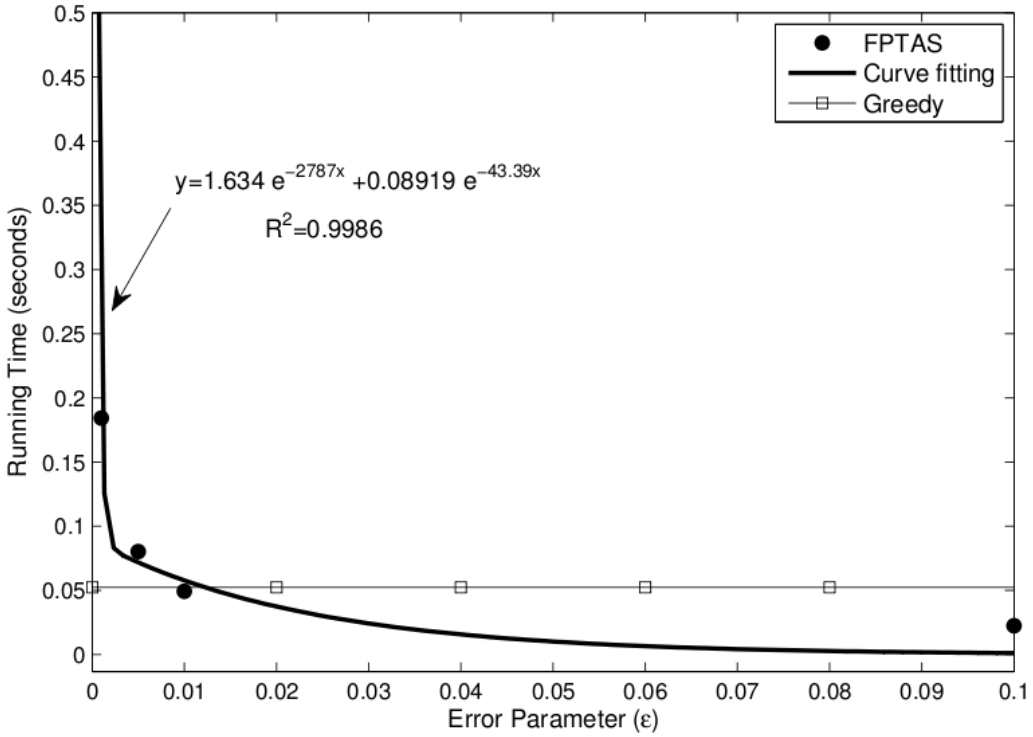
3. Run **FPTAS algorithm** on Bayesian network
   - Mapped our problem to 0-1 Knapsack problem (NP-Hard)
   - FPTAS trade off the running time with how close the solution is to the optimal → *(1-ε)·OPT*

$$B[k,w] = \begin{cases} B[k-1,w] & if\, w_k > w \\ max B[k-1,w], B[k-1,w-w_k] + b'_k & else \end{cases}$$

   - Cost/Benefit analysis
     - Cost in terms of TP, FP, FN and cost to respond/not respond
     - Benefit defined in terms of precision and recall (F-measure)
   - In simulations, FPTAS provided equal or better results than Greedy approach ($\varepsilon \geq 0.01$), showing similar running times

# Experimental Results

- Execution time comparison between Greedy and FPTAS



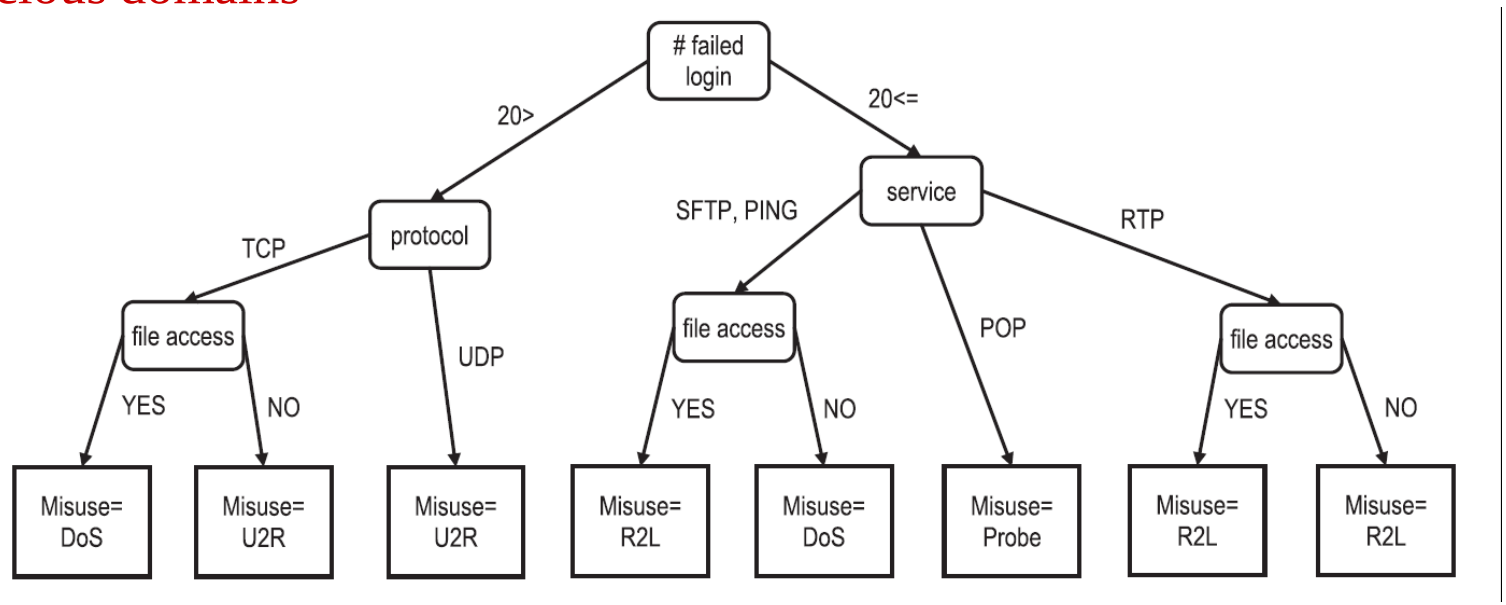

- Values of $\varepsilon$ equal or larger than 0.01 allow FPTAS to run faster than Greedy

# Conclusions

- By using attack graphs and Bayesian inference, we can quantify overall intrusion detection performance
  - Looked at different choices and placements of detectors
  - Quantified the information gain due to detector as function of distance to attack step
- In our experiments, FPTAS consistently outperformed Greedy
  - Greedy could be used in scenarios with time constraints

# Decision Trees

- Tree-like structure that has leaves, which represent classifications and branches, which represent the conjunctions of features that lead to those classifications
  - Algorithms such as ID3, C4.5 build DT by maximizing information gain at each variable split
- Advantage: Intuitive operation
- Example: [Bilge, NDSS-11] Uses DNS query activity to create a DT to detect malicious domains



# Support Vector Machine

- A separating hyperplane in the feature space between two classes in such a way that the distance between the hyperplane and the closest data points of each class is maximized
- Handles well the case of lots of features – fast classifier
- Example: [Wagner+Jerome+Thomas-Networking '11]
  - Anomaly based detection using binary SVM classifier
  - Used NetFlow data collected from realworld and simulated attack data using the Flame tool and ISP sources NetBIOS scans, DoS attacks, POP spams, and Secure Shell (SSH) scans
  - A new window kernel was introduced to help find an anomaly based on time position of the NetFlow data

# Considerations for ML in Security

- Need low time complexity for running time
  - HMM, SVM +
  - ANN, BN –
- Need to be streaming capable
  - Natural for BN
  - Harder for sequence mining
- IDS should be able to integrate network-level *and* host-level data
- Ensemble methods have been successful
  - Overcome limitations of individual techniques for specific kinds of attacks
- Interpretability of the classification model is important
  - Enables security admins to understand anomalous features
  - Leads to patching systems more quickly
- ML algorithms must be able to deal with large volumes of data, online and partial labeling of data

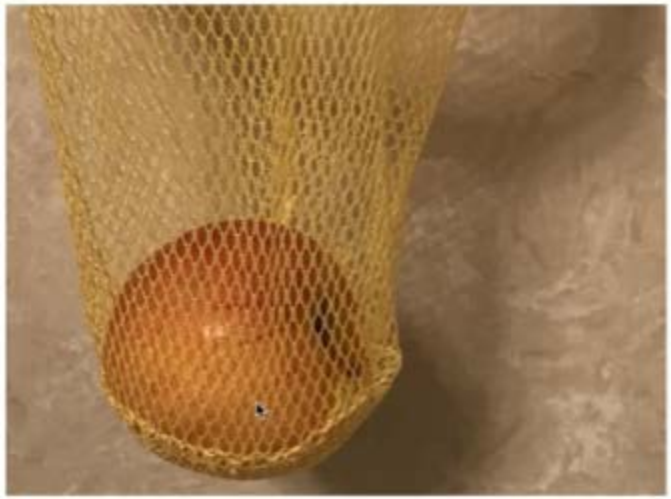
# Adversarial ML

- The adversarial example (AE) $x^*$ is the one that minimally perturbs $x$ to mislead $x$'s class label. Let's say model is $F(x)$
- The basis of most existing defense mechanisms against AE is what is often called *gradient masking*
  - Attempts to hide a useful gradient in the vicinity of the input data points
  - Hide $[\delta F(x)]_\varepsilon / \delta x$
- **Black-box attacks**: Fool a target model by AE made on a substitute model.
  - Adversaries do not know the internal parameters of $F(x)$ and the definition of cost function $J(x; y)$ in the target model
  - But using the same training data set, they can train their own model $F'(x)$ with a cost function $J'(x; y)$ so that they can construct the gradients of the target model with high similarity.
- **White-box attacks**: Attacks that attempt to mislead the target model using the adversarial examples crafted on the target model itself
  - Adversaries are assumed to know $F(x)$ and $J(x; y)$ of the target model so that they can compute the gradients of the target model
- Most existing defenses perform poorly against black-box attacks, although they are quite effective against white-box attacks

# Adversarial ML

## Also Adversarial Examples


(Eykholt et al, 2017)



(Goodfellow 2018)