Challenges Associated with Implementing Sub-Second / (Near) Real-Time Alerting Capabilities for Active Cyber Defense

Dr. Abdul Rahman

AFCEA Symposium
George Mason University
C^{4}I Center
May 20, 2014

Enlighten IT Consulting Inc.
7467 Ridge Road
Suite 140
Hanover, MD 21076

Agenda

Evaluation

Optimizations

Future Ideas

What is the Alerting

Architecture

Ingest

Optimizations

Streams Concepts

Alerting Overview

Based on our experience, alerting is a way to query a stream of data based on any of the following criteria:

Set of cyber data
Ex:
IPs and IP ranges (including CIDR)

Geo: A geolocation bounding box

Data types:
Ex:
XML, json, csv, binary

Boolean expression such as (a && (b || c))

Challenges in CND

IDS, IPS, Firewalls, Routers, Switches, and other devices produce traps, logs, and alerts that are critical to CND analysts

How do we rapidly process, filter, analyze, alert, tip, share, and store this information?

Key Challenges:

- Data volumes
- Processing
- Filtering
- Optimization
- Speed

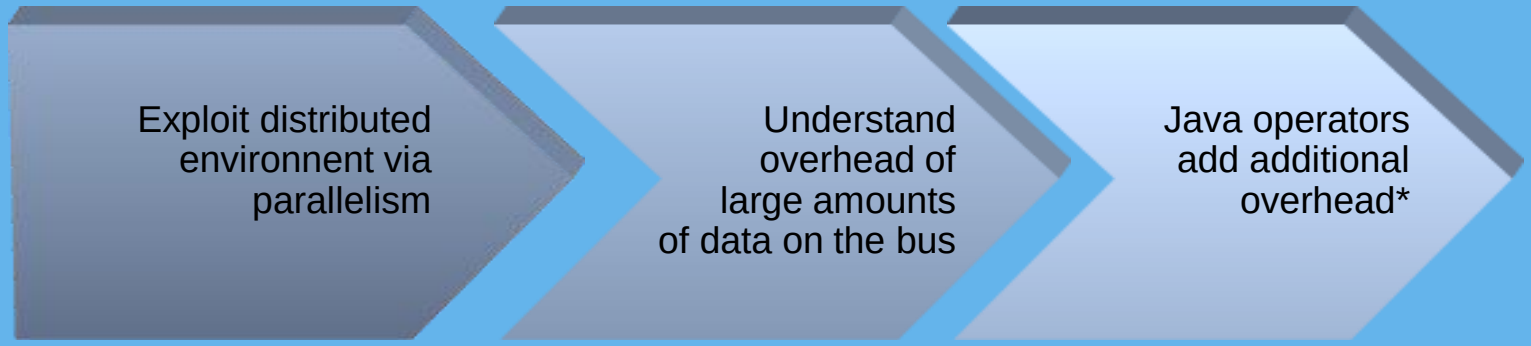
Alerting Engine

The alerting engine is the portion of the system that runs and checks the user's queries against the event data in the stream.

If the data in an event matches then the engine will fire off an alert with the raw event data and some rule information.

This data then get persisted and pushed up to users monitoring the rule.

Important Streams Concepts

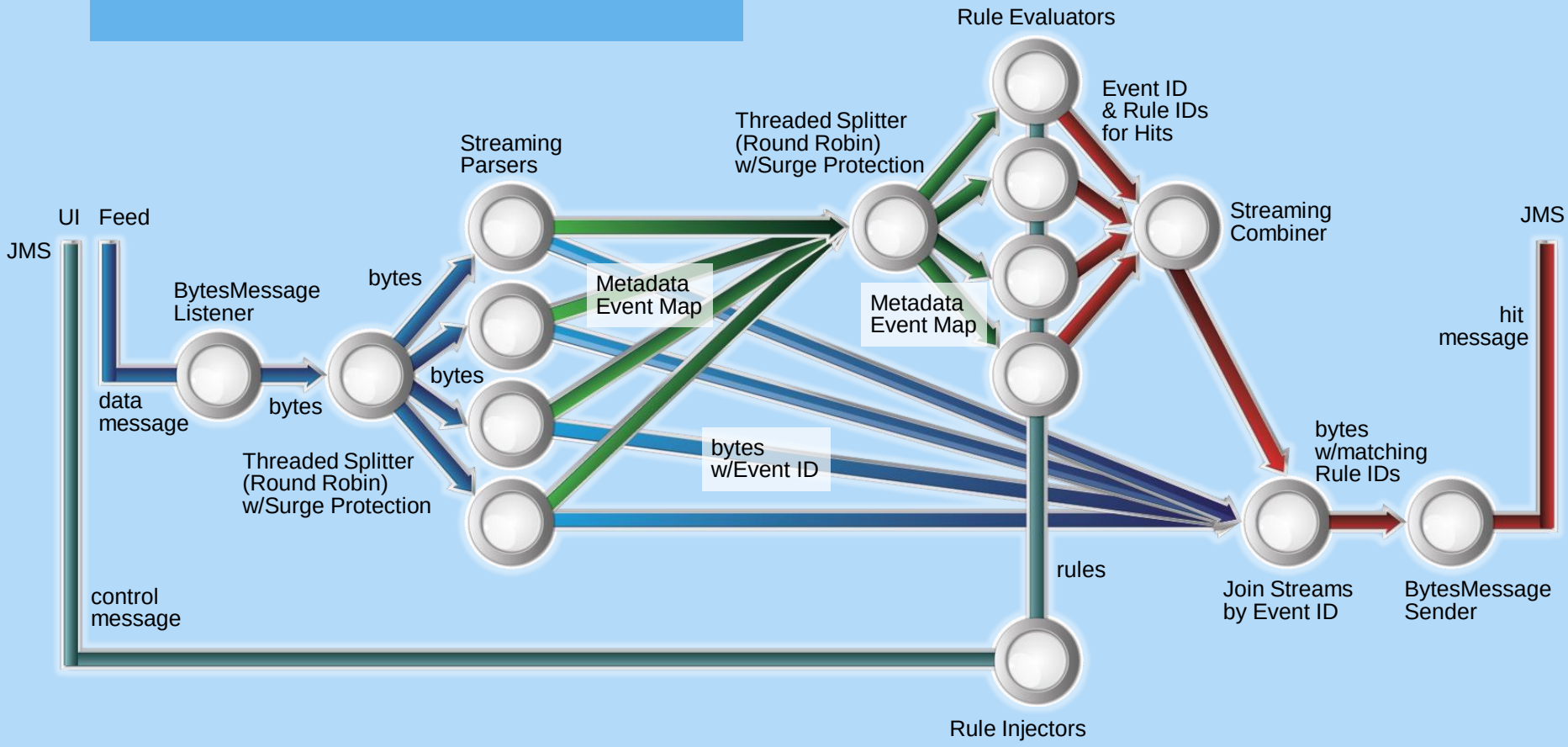


- Pipeline-parallelism
- Data-parallelism
- Use threading with caution

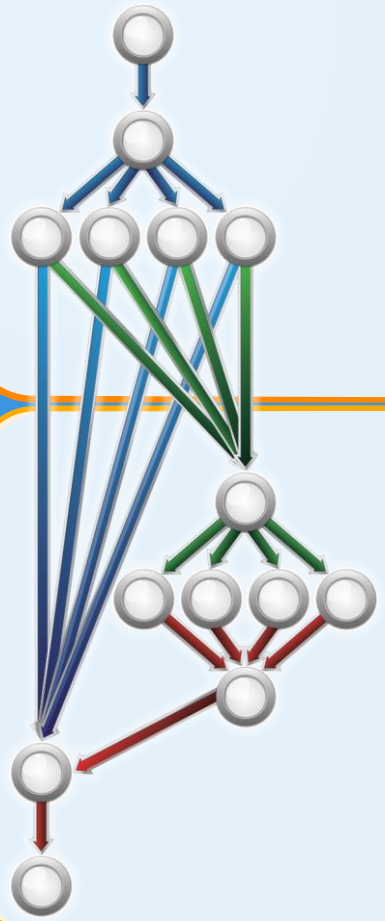
- Serialization costs
- Network limitations

- Offers integration with other streaming functions but has processing power ‘tax’

Alerting Engine Architecture



Alerting Engine Ingest



1. The entry point is a JMS listener. This operator simply reads a BytesMessage and pushes the raw byte contents out as quick as possible.

2. The next step in the flow is simply a ThreadedSplit operator. It distributes the raw byte data to four parsers. Also incorporates some surge protection via a drop policy.

3. Finally, the parsers take the raw data and examine it for each event contained in the byte data. It then outputs every single event as a parsed map and the event in raw data form.

Note: The raw data is simply a serialized (catalog) object.

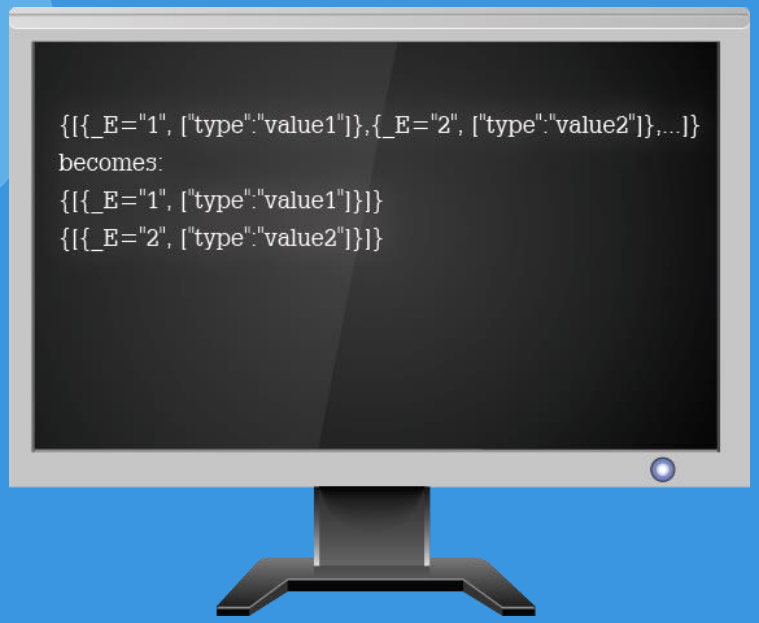
Ingest Optimizations

The biggest optimization to ingest was related to how events were parsed.

1. Use of Jackson's streaming JSON api to parse events.

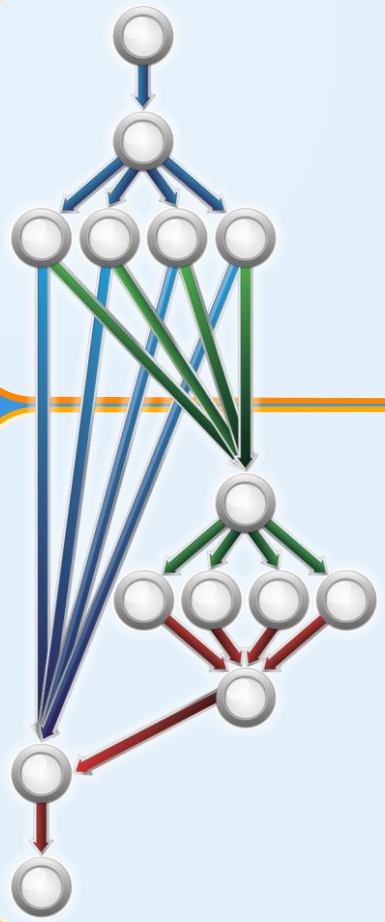
2. Extract raw event data from by stream as JSON is being parsed. This prevents having to reserialize each single event in order to generate the raw data.

For Example



To extract each event in the following data simple keep track of where the data begins within the JSON as it is being parsed. When the parser is completed with the event object, the parser copies the entire json object from the byte stream into a new byte stream and decorates it with metadata

Alerting Engine
Rule Evaluation



1. First is another threaded split sending the parsed data to 8 rule evaluators. Like in ingest, it incorporates some surge protection via a drop policy.
2. The rule evaluators will evaluate each event received against all the rules. The evaluator then emits a single event if any of the rules evaluated to true. The alert event contains a single event id and all the rules that evaluated to true.
3. The alert streams are then combined into a single stream.
4. Each alert is then joined with the raw data stream to get the matching raw event data.
5. Finally the alerts with the raw event data are sent out over JMS as hits..

Alerting Addressing Challenges using Alerting Engine Optimizations

1

Drastically shortened

the path in which event data must be sent through the pipeline. Opted instead for data-parallelism.

2

Reduced number

of threaded queue as this was having a impact on throughput.

3

Rule evaluators

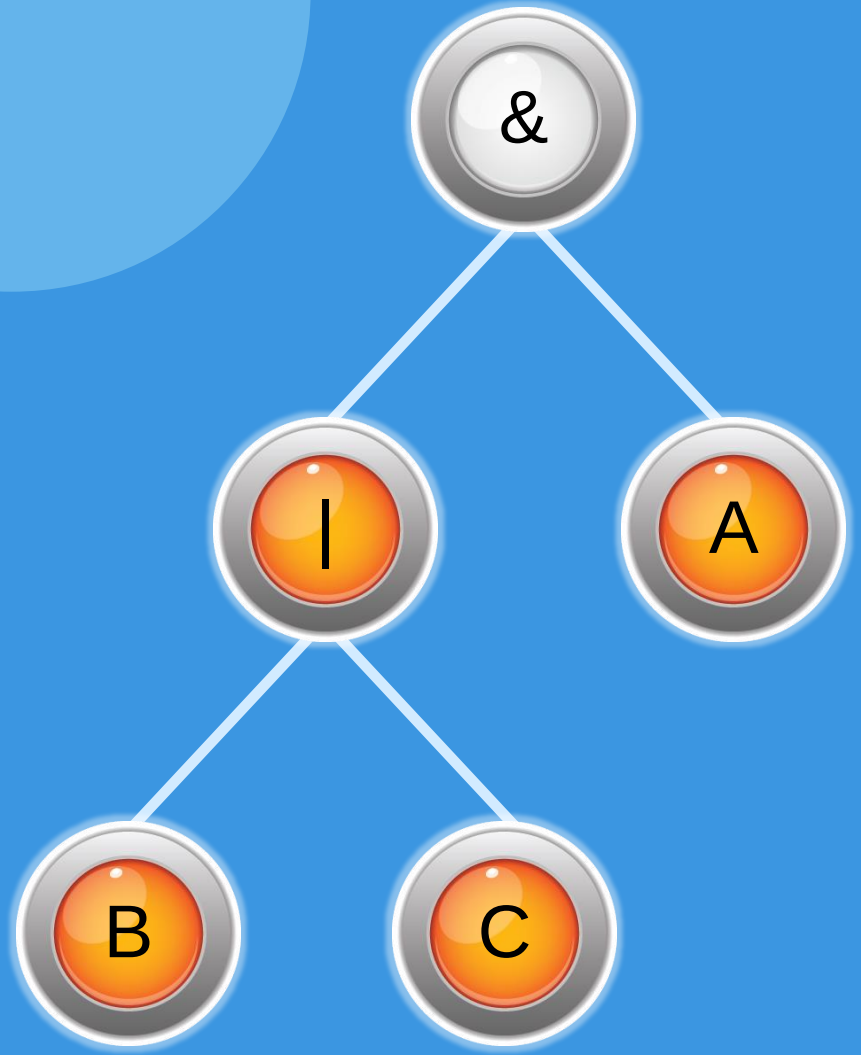
are written in C++ to minimize Java overhead and more importantly to utilize the Boost which provide the most efficient libraries for things such as interval sets..

4

Finally and most importantly

we used boolean expression trees to represent the rules to allow for shortcutting the evaluation of the rule. This also allows for optimizations on each rule to further reduce the amount of work required for rule evaluation.

Boolean Expression Optimizations



- Constant Tree Reduction

  Ex:

  a & false = false
  a | false = a

- Same Relational Operator

  Ex:

  a & (b & c) = a & b & c
  a | (b | c) = a | b | c

- Single Child on Relational Operator

  Ex:

  a & (b) = a & b

- Duplicate Criteria Removal

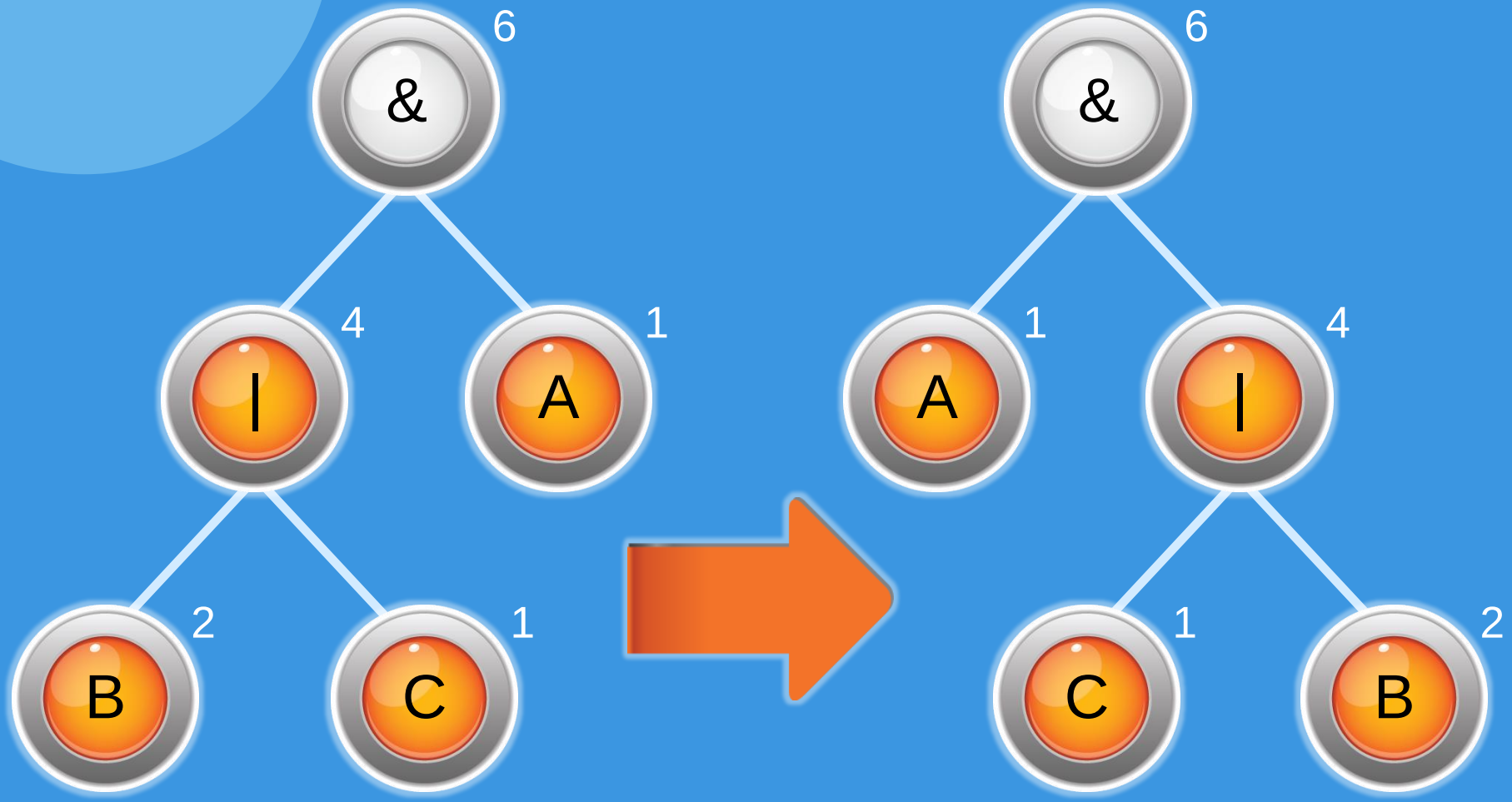
  Ex:

  a & a = a
  a | a = a

Boolean Expression Optimizations

- Weighted path sorting

  In the following example each node is evaluated from left to right down the tree.



Possible Criteria Types

2 Relational Criteria

And
- Returns false on first false, else true

Or
- Returns true on first true, else false

Comparison Criteria
- Evaluates the value using the following operators = , != , > , <

Regex Criteria
- Evaluates the value to a regular expression
- Always evaluated as string no matter what type field it is.

Interval Set Criteria
- Compares a value of a field on an event to a set of intervals
- Utilizes Boost's interval library for fast evaluation of large interval sets

Constant Tree Reduction
- Always returns true or false
- Mostly used as a placeholder for optimizations

4 Field Criteria

Optimization Results

- The largest bottleneck in the system is now the throughput of the JMS topic providing the event data
- The amount of dropped events has dropped to less than 0.0001%*

Possible Future Improvements

Command and control

- Execution
- Monitoring
- Management
- Configuration

Optimize path for delivery of events

- Utilize direct connection protocols instead of JMS

Runtime rule optimizations

- Use information about how individual criteria are being evaluated to enable earlier shortcutting during future evaluations.
- Throttle rules with high evaluation cost, even though they rarely hit.

Additional Boolean algebra optimizations

- Ex:
  a & (a | c) = a
  a | (a & b) = a

QUESTIONS

ENLIGHTEN
Empowering Solutions

Enlighten IT Consulting, Inc.
7467 Ridge Road, Suite 140
Hanover, MD 21076

Phone: 410-850-7305 x171
Fax: 410-255-5522
info@eitccorp.com