# VAST

A Unified Platform for Interactive Network Forensics

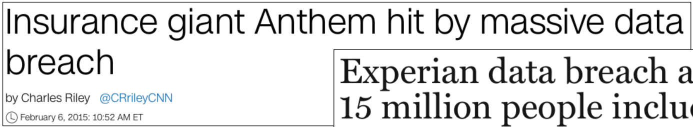
Matthias Vallentin[1,2] Vern Paxson[1,2] Robin Sommer[2,3]

[1] *UC Berkeley*
[2] *International Computer Science Institute (ICSI)*
[3] *Lawrence Berkeley National Laboratory (LBNL)*

March 17, 2016

USENIX NSDI

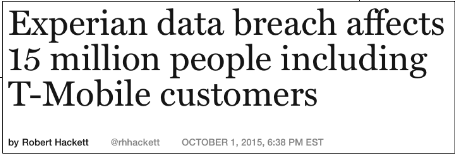
# Omnipresent Data Breaches



Insurance giant Anthem hit by massive data breach

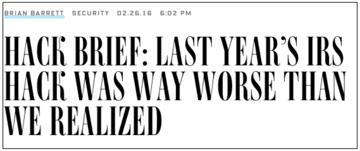
by Charles Riley @CRrileyCNN

February 6, 2015: 10:52 AM ET



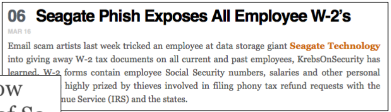
Experian data breach affects 15 million people including T-Mobile customers

by Robert Hackett @rhhackett OCTOBER 1, 2015, 6:38 PM EST



BRIAN BARRETT SECURITY 02.26.16 6:02 PM

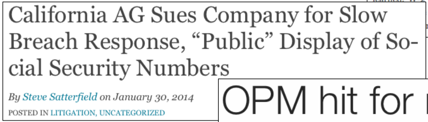
HACK BRIEF: LAST YEAR'S IRS HACK WAS WAY WORSE THAN WE REALIZED



06 **Seagate Phish Exposes All Employee W-2's**

MAR 16

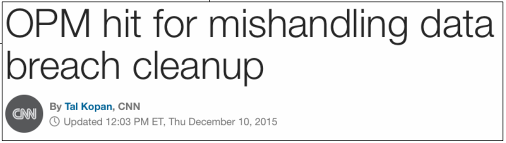
Email scam artists last week tricked an employee at data storage giant **Seagate Technology** into giving away W-2 tax documents on all current and past employees, KrebsOnSecurity has learned. W-2 forms contain employee Social Security numbers, salaries and other personal highly prized by thieves involved in filing phony tax refund requests with the nue Service (IRS) and the states.



California AG Sues Company for Slow Breach Response, "Public" Display of Social Security Numbers

*By Steve Satterfield on January 30, 2014*

POSTED IN LITIGATION, UNCATEGORIZED

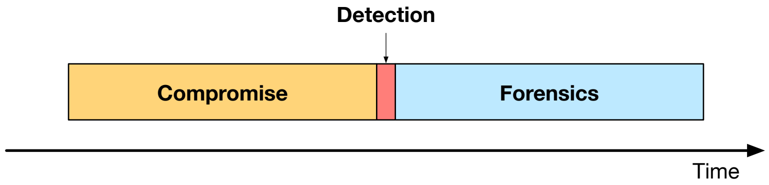


OPM hit for mishandling data breach cleanup

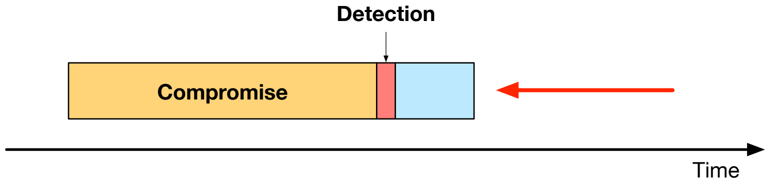
By Tal Kopan, CNN

Updated 12:03 PM ET, Thu December 10, 2015

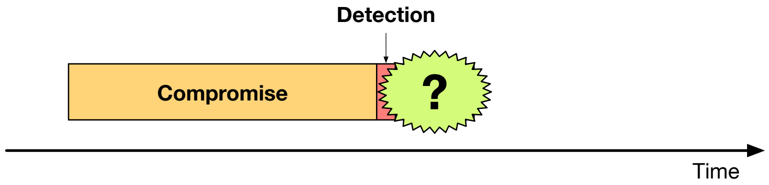
## Breach Timeline



## Breach Timeline



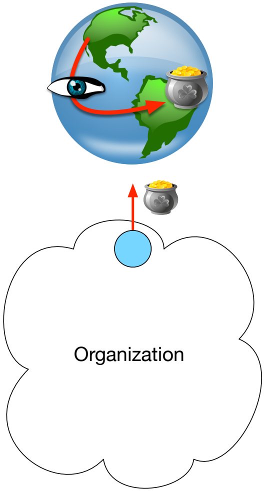
# Breach Timeline



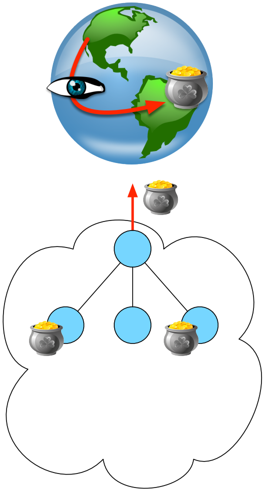
## Network Forensics — Characteristics

## Network Forensics — Characteristics

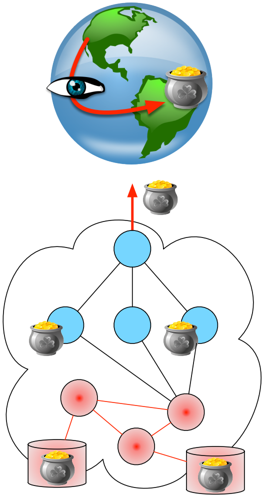
## Network Forensics — Characteristics



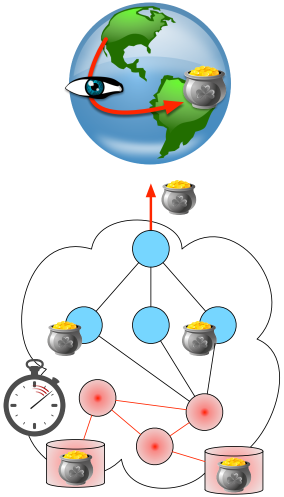
## Network Forensics — Characteristics



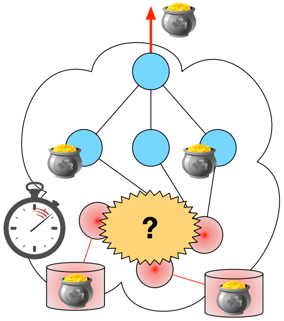
## Network Forensics — Characteristics



# Network Forensics — Characteristics



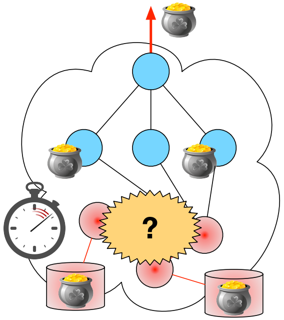
## Network Forensics — Characteristics



## Network Forensics — Characteristics

**Interactive data exploration**

- Iterative query refinement
- High-dimensional search



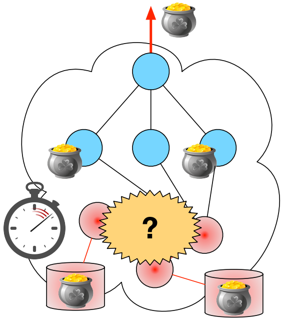
## Network Forensics — Characteristics

**Interactive data exploration**

- Iterative query refinement
- High-dimensional search

**Disparate data access**

- Temporal
- Spatial



## Network Forensics — Characteristics

**Interactive data exploration**

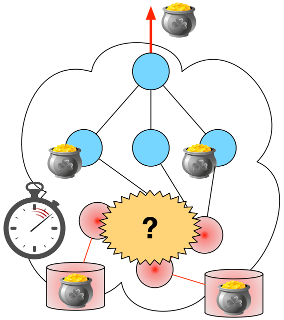
- Iterative query refinement
- High-dimensional search

**Disparate data access**

- Temporal
- Spatial

**Massive data volumes**

- 50–100K events/sec
- 10s TBs/day



## Log Example — Bro Connection Log

```
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path conn
#open 2016-01-06-15-28-58
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto service duration orig_bytes resp_bytes conn_..
#types time string addr port addr port enum string interval count count string bool bool count string
1258531.. Cz7SRx3.. 192.168.1.102 68 192.168.1.1 67 udp dhcp 0.163820 301 300 SF - - 0 Dd 1 329 1 328 (empty)
1258531.. CTeURV1.. 192.168.1.103 137 192.168.1.255 137 udp dns 3.780125 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258531.. CUAVTq1.. 192.168.1.102 137 192.168.1.255 137 udp dns 3.748647 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258531.. CYoxAZ2.. 192.168.1.103 138 192.168.1.255 138 udp - 46.725380 560 0 S0 - - 0 D 3 644 0 0 (empty)
1258531.. CvabDq2.. 192.168.1.102 138 192.168.1.255 138 udp - 2.248589 348 0 S0 - - 0 D 2 404 0 0 (empty)
1258531.. CViJEOm.. 192.168.1.104 137 192.168.1.255 137 udp dns 3.748893 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258531.. CSC2Hd4.. 192.168.1.104 138 192.168.1.255 138 udp - 59.052898 549 0 S0 - - 0 D 3 633 0 0 (empty)
1258531.. Cd3RNm1.. 192.168.1.103 68 192.168.1.1 67 udp dhcp 0.044779 303 300 SF - - 0 Dd 1 331 1 328 (empty)
1258531.. CEwuIl2.. 192.168.1.102 138 192.168.1.255 138 udp - - - - S0 - - 0 D 1 229 0 0 (empty)
1258532.. CXxLc94.. 192.168.1.104 68 192.168.1.1 67 udp dhcp 0.002103 311 300 SF - - 0 Dd 1 339 1 328 (empty)
1258532.. CIFDQJV.. 192.168.1.102 1170 192.168.1.1 53 udp dns 0.068511 36 215 SF - - 0 Dd 1 64 1 243 (empty)
1258532.. CXFISh5.. 192.168.1.104 1174 192.168.1.1 53 udp dns 0.170962 36 215 SF - - 0 Dd 1 64 1 243 (empty)
1258532.. CQJw4C3.. 192.168.1.1 5353 224.0.0.251 5353 udp dns 0.100381 273 0 S0 - - 0 D 2 329 0 0 (empty)
1258532.. ClfEd43.. fe80::219:e3ff:fee7:5d23 5353 ff02::fb 5353 udp dns 0.100371 273 0 S0 - - 0 D 2 369 0 0
1258532.. C67zf02.. 192.168.1.103 137 192.168.1.255 137 udp dns 3.873818 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258532.. CG1FKF1.. 192.168.1.102 137 192.168.1.255 137 udp dns 3.748891 350 0 S0 - - 0 D 7 546 0 0 (empty)
1258532.. CNFkeF2.. 192.168.1.103 138 192.168.1.255 138 udp - 2.257840 348 0 S0 - - 0 D 2 404 0 0 (empty)
1258532.. Cq4eis4.. 192.168.1.102 1173 192.168.1.1 53 udp dns 0.000267 33 497 SF - - 0 Dd 1 61 1 525 (empty)
1258532.. CHpqv31.. 192.168.1.102 138 192.168.1.255 138 udp - 2.248843 348 0 S0 - - 0 D 2 404 0 0 (empty)
1258532.. CFoJjT3.. 192.168.1.1 5353 224.0.0.251 5353 udp dns 0.099824 273 0 S0 - - 0 D 2 329 0 0 (empty)
1258532.. Cc3Ayyz.. fe80::219:e3ff:fee7:5d23 5353 ff02::fb 5353 udp dns 0.099813 273 0 S0 - - 0 D 2 369 0 0
```

## Existing Solutions

### MapReduce (Hadoop)

✓ Scalability

✗ Batch-oriented: no iterative, exploratory analysis

## Existing Solutions

### MapReduce (Hadoop)

- ✓ Scalability
- ✗ Batch-oriented: no iterative, exploratory analysis

### In-Memory Cluster Computing (Spark)

- ✓ Efficient & complex analysis
- ✗ Thrashing when working set does not fit in aggregate memory

## Contribution

**VAST**

**V**isibility **A**cross **S**pace and **T**ime

## Contribution

**VAST**

**V**isibility **A**cross **S**pace and **T**ime

**Architecture**

- **Performance**: concurrent & modular design
- **Scaling**: intra-machine & inter-machine
- **Typing**: strong & rich

## Contribution

### VAST

**V**isibility **A**cross **S**pace and **T**ime

### Architecture

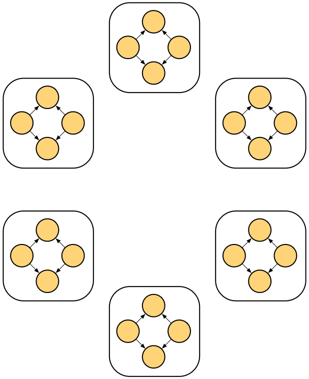
- **Performance**: concurrent & modular design
- **Scaling**: intra-machine & inter-machine
- **Typing**: strong & rich

### Implementation

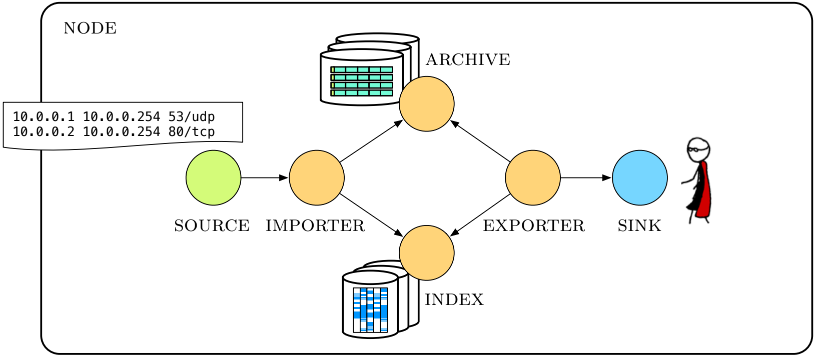
- **Composition**: high-level bitmap indexing framework
- **Adaptation**: fine-grained component flow-control
- **Asynchrony**: finite state machines for query execution

## Outline

## VAST Architecture — Single Machine

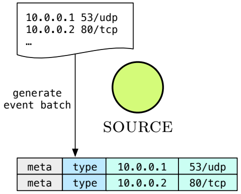


# VAST Architecture — Single Machine

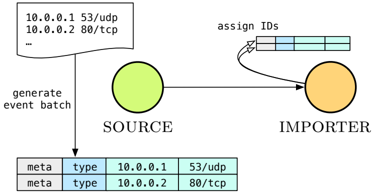


```
10.0.0.1 10.0.0.254 53/udp
10.0.0.2 10.0.0.254 80/tcp
```

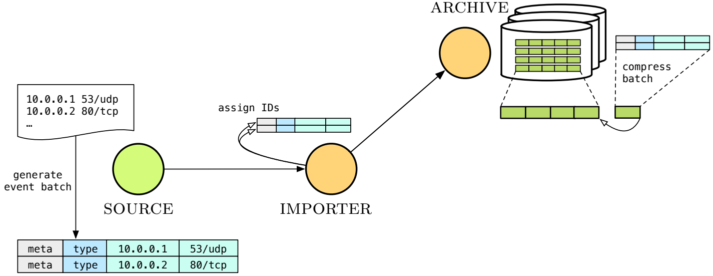
## VAST Architecture — Ingestion


10.0.0.1 53/udp
10.0.0.2 80/tcp
…
generate event batch
SOURCE
meta | type | 10.0.0.1 | 53/udp
meta | type | 10.0.0.2 | 80/tcp

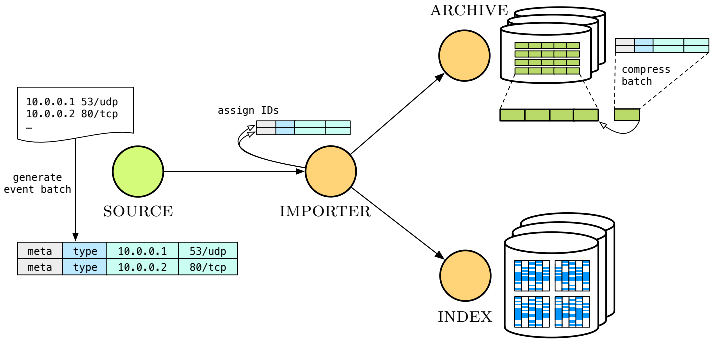
## VAST Architecture — Ingestion



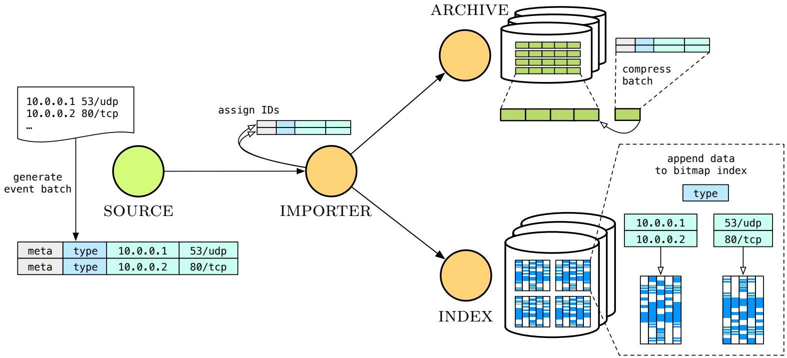
# VAST Architecture — Ingestion



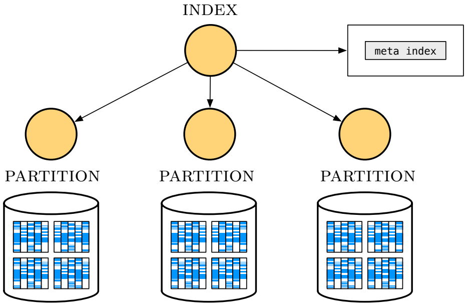
## VAST Architecture — Ingestion



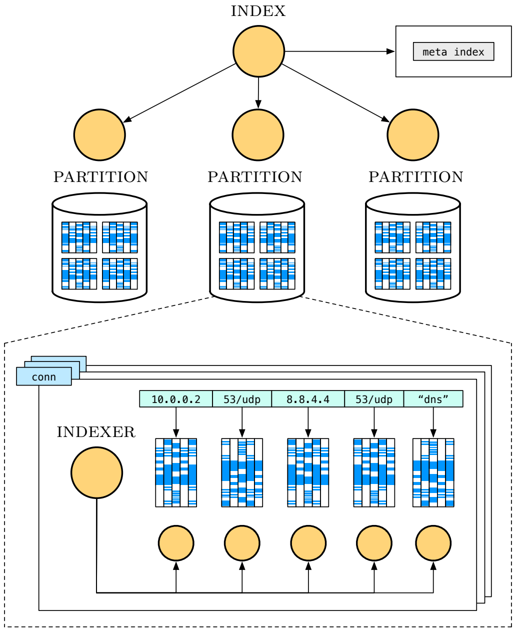
# VAST Architecture — Ingestion



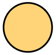
## VAST Architecture — Index



# VAST Architecture — Index

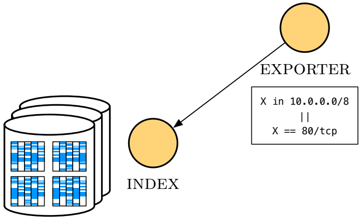


## VAST Architecture — Querying



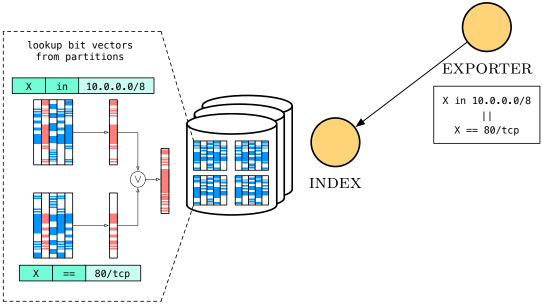
```
X in 10.0.0.0/8
      ||
X == 80/tcp
```

## VAST Architecture — Querying

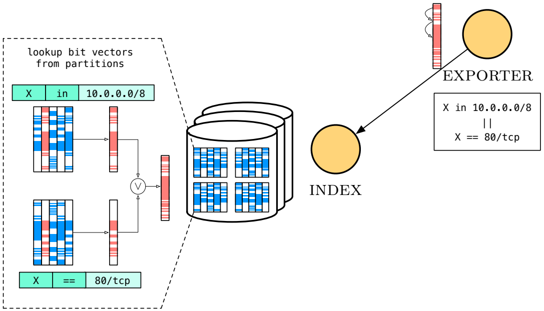


```
X in 10.0.0.0/8
      ||
 X == 80/tcp
```

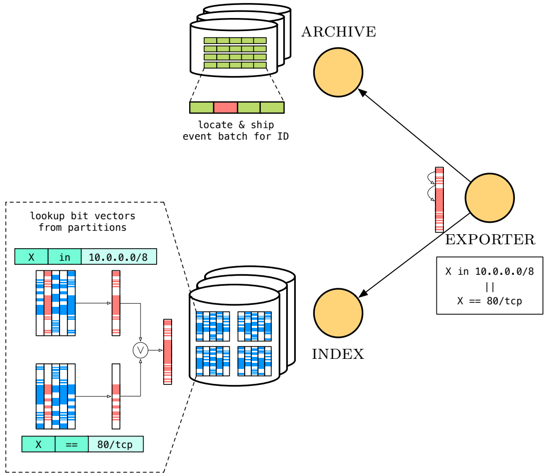
# VAST Architecture — Querying



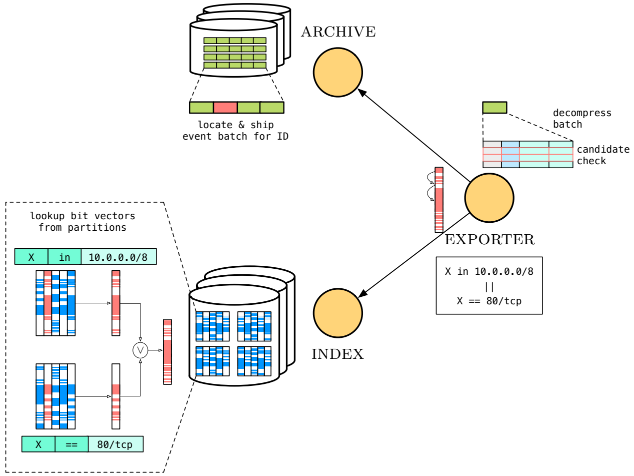
# VAST Architecture — Querying



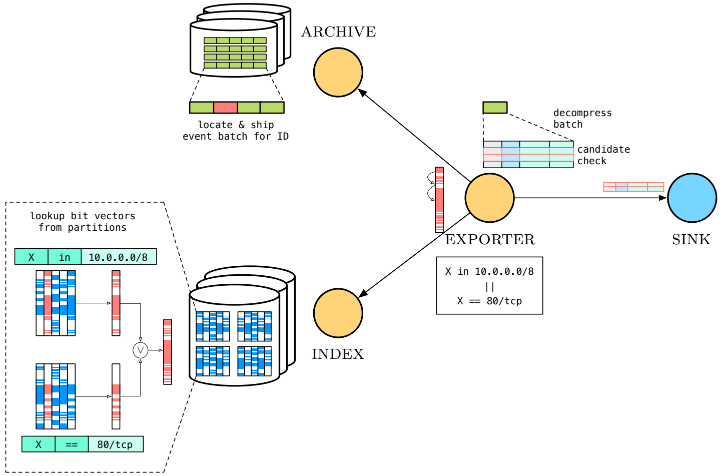
# VAST Architecture — Querying



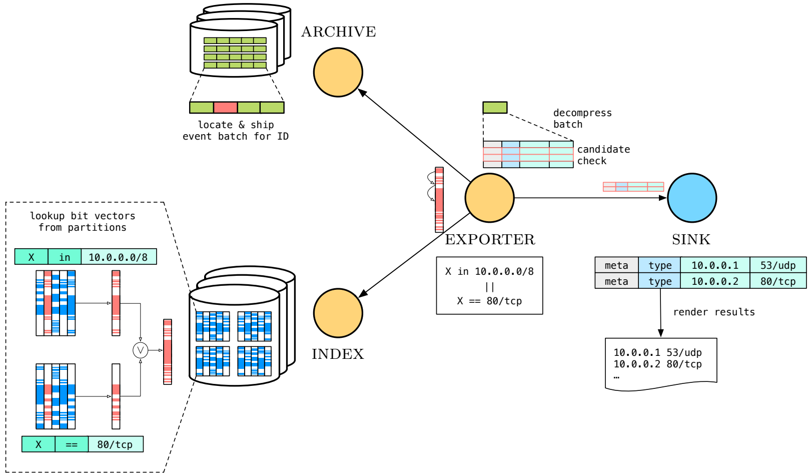
# VAST Architecture — Querying



## VAST Architecture — Querying



## VAST Architecture — Querying

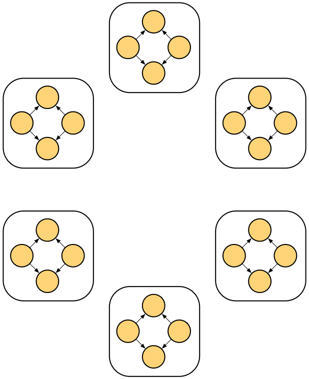


ARCHIVE
locate & ship event batch for ID
decompress batch
candidate check
EXPORTER
SINK
INDEX
lookup bit vectors from partitions
X in 10.0.0.0/8
X == 80/tcp
X in 10.0.0.0/8 || X == 80/tcp
render results

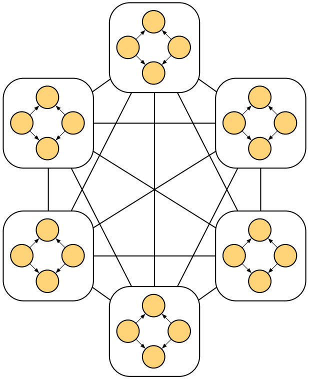
| | | | |
|---|---|---|---|
| meta | type | 10.0.0.1 | 53/udp |
| meta | type | 10.0.0.2 | 80/tcp |

```
10.0.0.1 53/udp
10.0.0.2 80/tcp
…
```

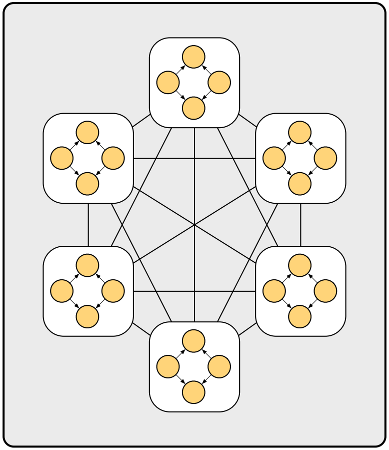
# VAST Architecture — Distributed



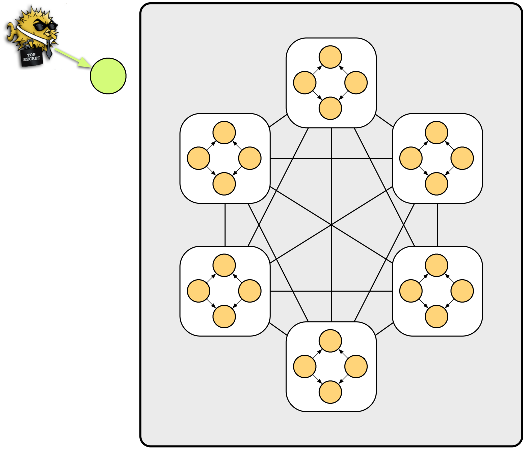
## VAST Architecture — Distributed



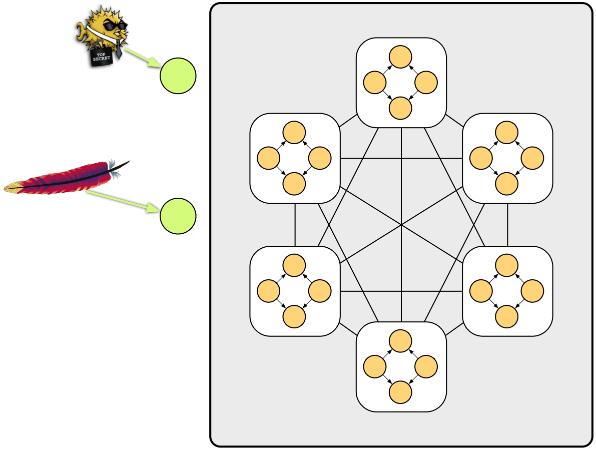
# VAST Architecture — Distributed



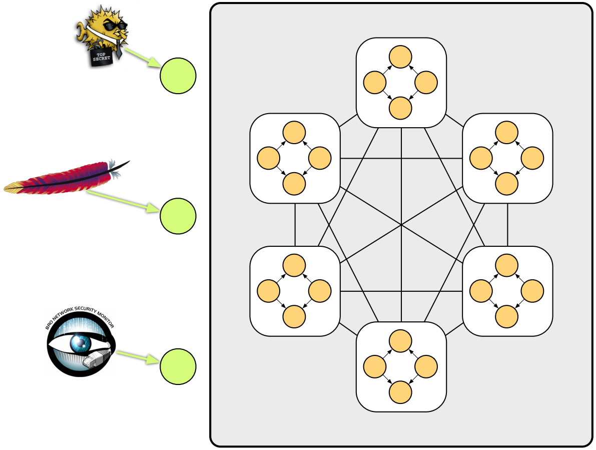
## VAST Architecture — Distributed



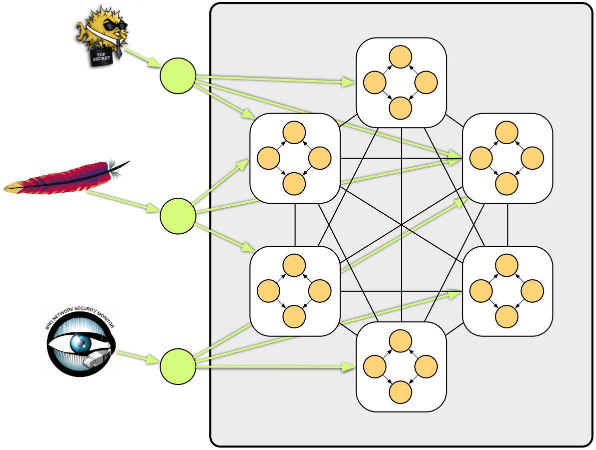
# VAST Architecture — Distributed



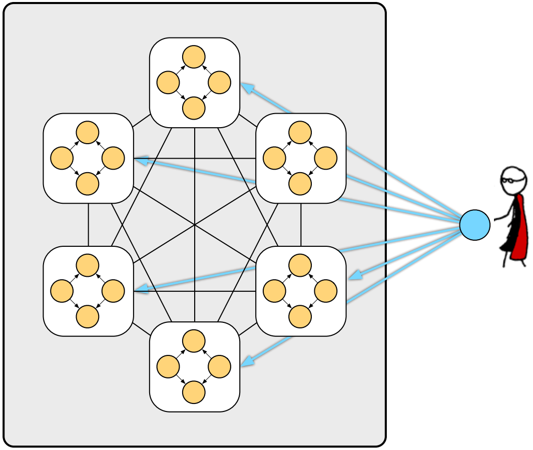
# VAST Architecture — Distributed



## VAST Architecture — Distributed

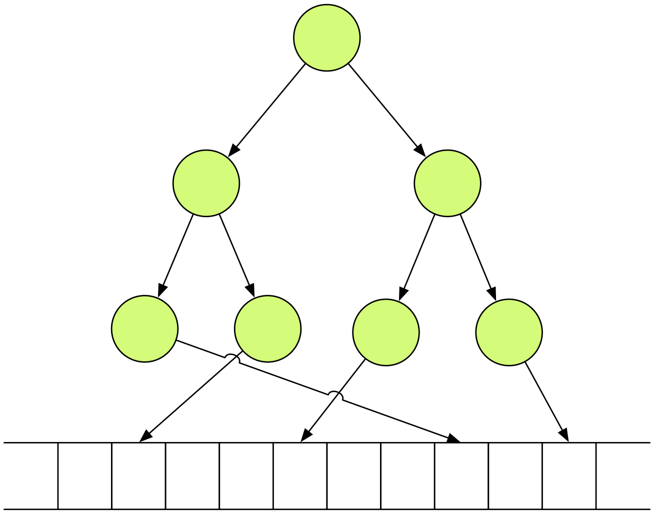


## VAST Architecture — Distributed

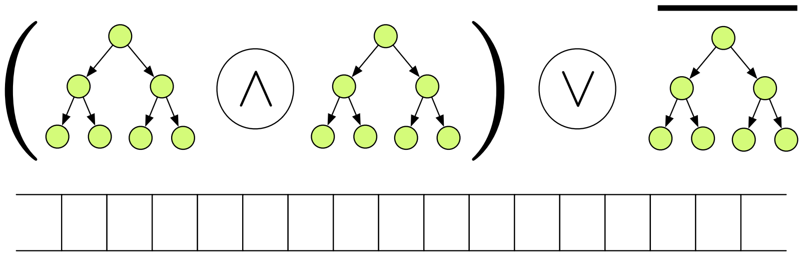


## Outline

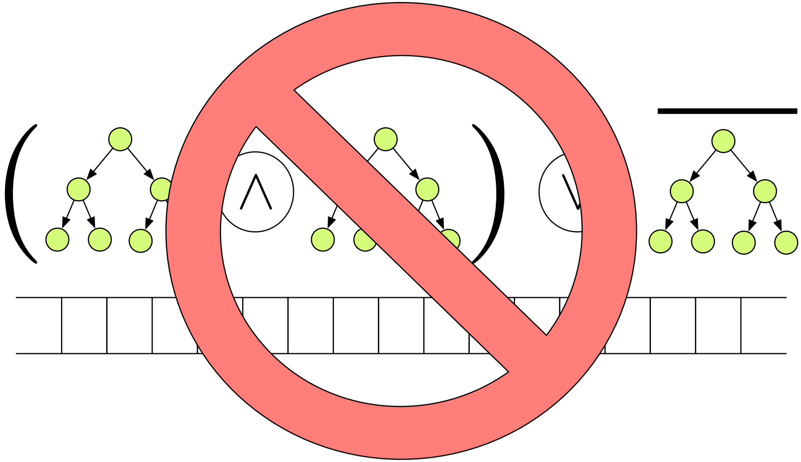
# Indexing Basics — Tree Indexes



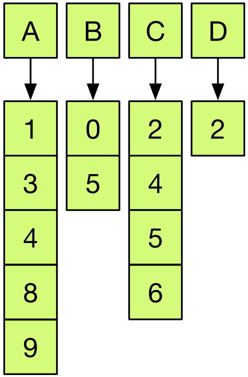
## Indexing Basics — Composition

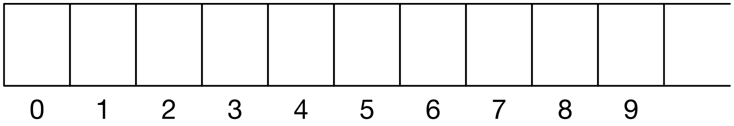


## Indexing Basics — Composition

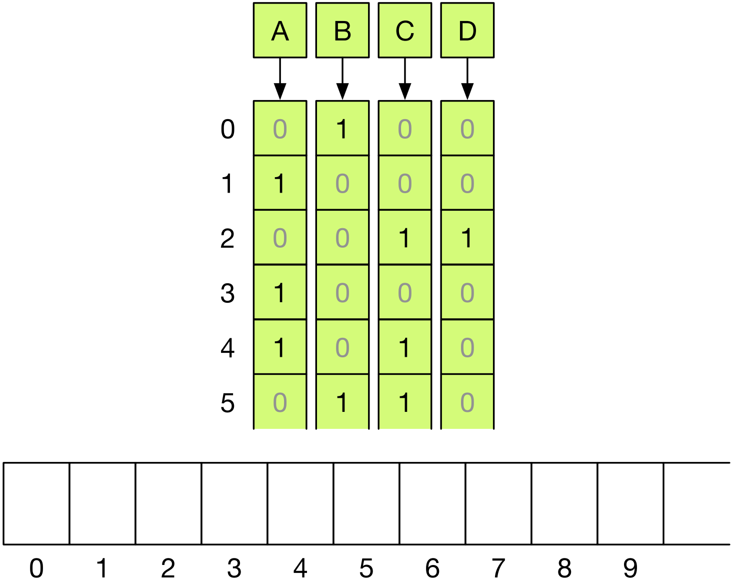


## Indexing Basics — Inverted Index

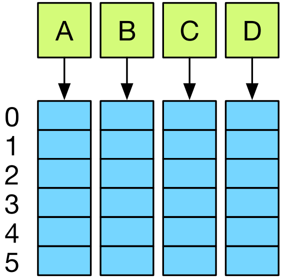


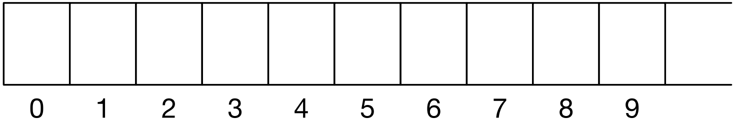


## Indexing Basics — Bitmap Index

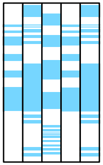


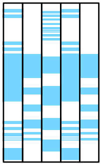
## Indexing Basics — Bitmap Index





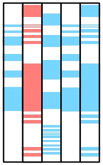
## Indexing Basics — Bitmap Composition



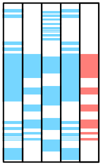


# Indexing Basics — Bitmap Composition

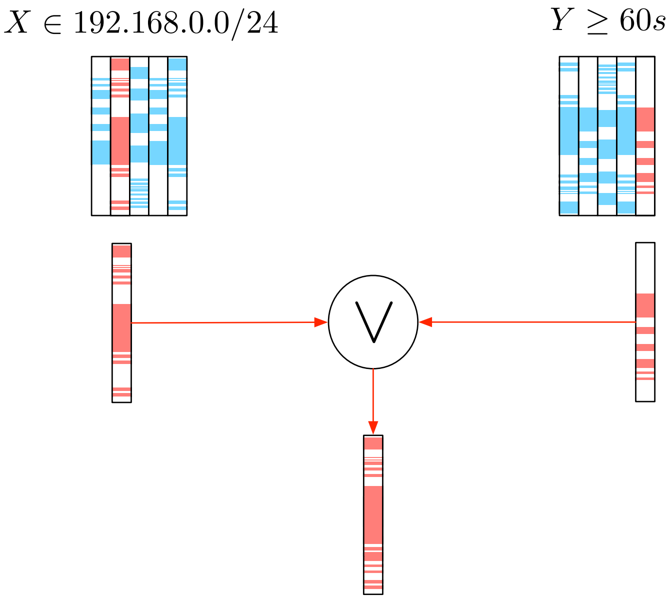
$X \in 192.168.0.0/24$



$Y \geq 60s$



# Indexing Basics — Bitmap Composition



## Indexing Challenges

**High-cardinality values**

- Represent *millions* of distinct values compactly
- Provide low-latency lookups

**High-level operations**

- Support type-specific operations
- Relational operators: $\{<, \leq, =, \neq, \geq, >, \in, \notin\}$

## Query Language

### Boolean Expressions

- Conjunctions `&&`
- Disjunctions `||`
- Negations `!`
- Predicates
  - `LHS op RHS`
  - `(expr)`

### Examples

- `A && B || !(C && D)`
- `orig_h in 10.0.0.1 && &time < now - 2h`
- `&type == "conn" || "foo" in :string`
- `duration > 60s && service == "tcp"`

### Extractors

- `&tag`
- `x.y.z`
- `:type`

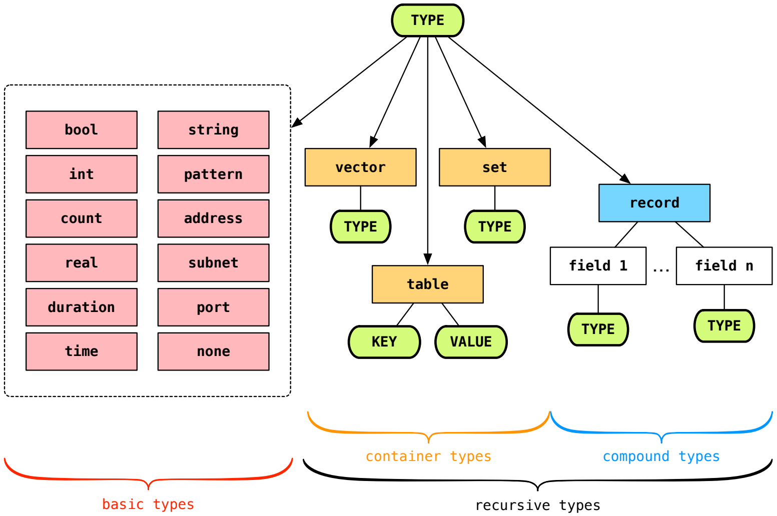
### Relational Operators

- `<`, `<=`, `==`, `>=`, `>`
- `in`, `ni`, `[+`, `+]`
- `!in`, `!ni`, `[-`, `-]`
- ∼, !∼

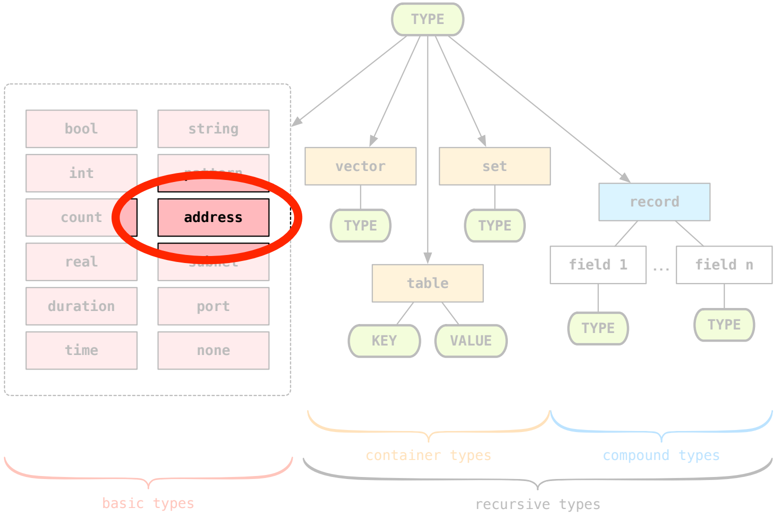
### Values

- `T`, `F`
- `+42`, `1337`, `3.14`
- `"foo"`
- `10.0.0.0/8`
- `80/tcp`, `53/?`
- `{1, 2, 3}`

## Data Model


TYPE
bool
string
int
pattern
count
address
real
subnet
duration
port
time
none
vector
TYPE
set
TYPE
table
KEY
VALUE
record
field 1
...
field n
TYPE
TYPE
container types
compound types
basic types
recursive types

## Data Model


TYPE
bool
string
int
pattern
count
address
real
subnet
duration
port
time
none
vector
set
TYPE
TYPE
table
KEY
VALUE
record
field 1
...
field n
TYPE
TYPE
container types
compound types
basic types
recursive types

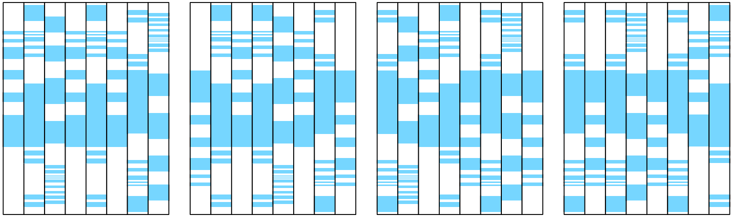
## Bitmap Index for IP Addresses

192.168.0.42

## Bitmap Index for IP Addresses

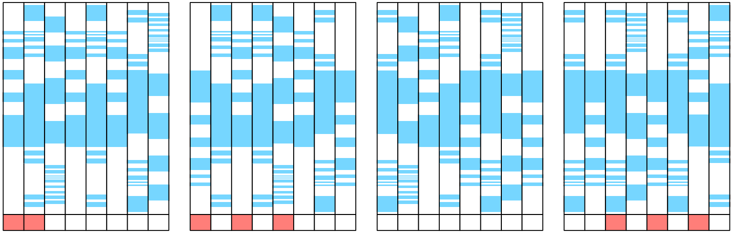
11000000.10101000.00000000.00101010

## Bitmap Index for IP Addresses



11000000.10101000.00000000.00101010

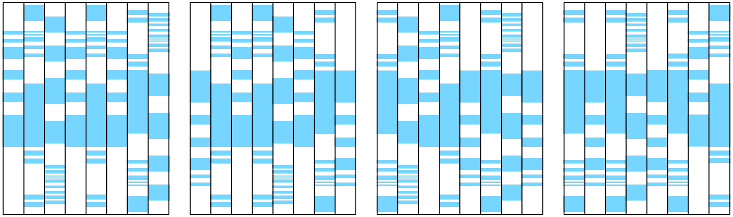
## Bitmap Index for IP Addresses



11000000.10101000.00000000.00101010

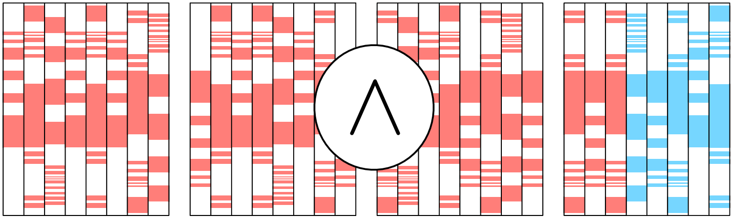
## Bitmap Index for IP Addresses

$$X \in 192.168.0.0/27$$



## Bitmap Index for IP Addresses

$$X \in 192.168.0.0/27$$



## Outline

## Data Set

### Single-Machine

Data:

- 10 M packets from a 24-hour trace (5 fields/event)
- 3.4 M derived Bro connection logs (20 fields/event)

Machine:

- 2 × 8-core Intel Xeon CPUs
- 128 GB RAM
- 4 × 3 TB SAS 7.2 K disks
- 64-bit FreeBSD

# Data Set

## Single-Machine

Data:

- 10 M packets from a 24-hour trace (5 fields/event)
- 3.4 M derived Bro connection logs (20 fields/event)

Machine:

- 2 $\times$ 8-core Intel Xeon CPUs
- 128 GB RAM
- 4 $\times$ 3 TB SAS 7.2 K disks
- 64-bit FreeBSD

## Cluster

Data:

- 1.24 B Bro connection logs (152 GB)
- Split into $N$ slices for $N$ nodes
- $N \in [1, 24]$

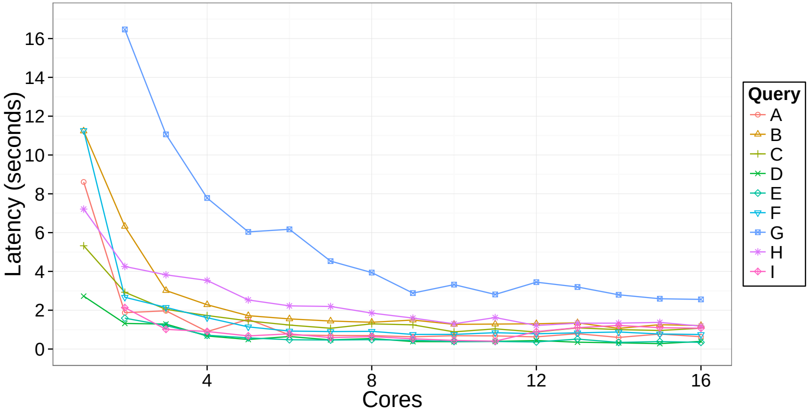
Nodes:

- 2 $\times$ 8-core Intel Xeon CPUs
- 12 GB of RAM
- 2 $\times$ 500 MB SATA disks
- 64-bit FreeBSD

## Queries

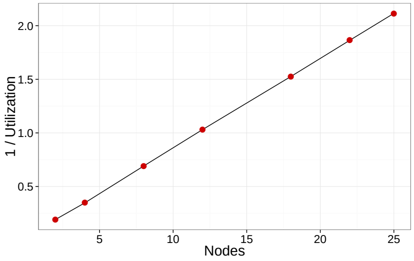
| Label | Results | Query |
|---|---|---|
| A | 374 | `resp_h == 2001:7fe::53` |
| B | 942 | `(duration > 1000s \|\| resp_bytes > 40000) && service == "dns"` |
| C | 13 | `orig_h in 192.150.186.0/23 && orig_bytes > 10000 && service == "http"` |
| D | 3 | `duration > 1h && service == "ssh"` |
| E | 969,092 | `conn_state != "SF"` |
| F | 4812 | `:addr in 192.150.186.0/23 && :port == 3389/?` |
| G | 1,077 | `:addr in 192.150.186.0/23 && :port == 3389/?` |
| H | 34 | `&time > 2015-02-04+10:00:00 && &time < 2015-02-04+11:00:00 && ((src == 77.255.19.163 && dst == 192.150.187.43 && sport == 49613/? && dport == 443/?) \|\| (src == 192.150.187.43 && dst == 77.255.19.163 && sport == 443/? && dport == 49613/?))` |
| I | 187,015 | `&time > 2015-02-04+10:00:00 && &time < 2015-02-04+11:00:00 && :addr == 192.150.187.43` |

## Performance – Index Latency

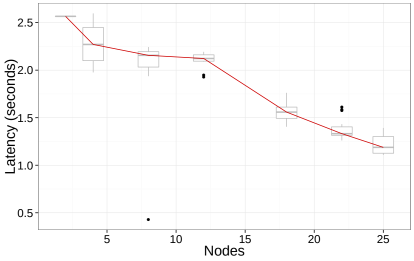


# Performance — Scaling

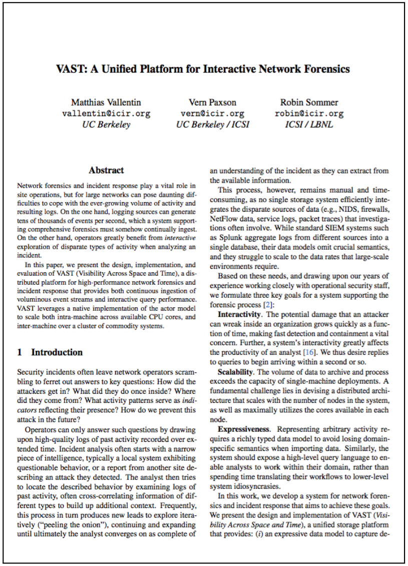
Import



Export



## Details in the paper



# VAST: A Unified Platform for Interactive Network Forensics

Matthias Vallentin
vallentin@icir.org
*UC Berkeley*

Vern Paxson
vern@icir.org
*UC Berkeley / ICSI*

Robin Sommer
robin@icir.org
*ICSI / LBNL*

**Abstract**

Network forensics and incident response play a vital role in site operations, but for large networks can pose daunting difficulties to cope with the ever-growing volume of activity and resulting logs. On the one hand, logging sources can generate tens of thousands of events per second, which a system supporting comprehensive forensics must somehow continually ingest. On the other hand, operators greatly benefit from *interactive* exploration of disparate types of activity when analyzing an incident.

In this paper, we present the design, implementation, and evaluation of VAST (Visibility Across Space and Time), a distributed platform for high-performance network forensics and incident response that provides both continuous ingestion of voluminous event streams and interactive query performance. VAST leverages a native implementation of the actor model to scale both intra-machine across available CPU cores, and inter-machine over a cluster of commodity systems.

## 1 Introduction

Security incidents often leave network operators scrambling to ferret out answers to key questions: How did the attackers get in? What did they do once inside? Where did they come from? What activity patterns serve as *indicators* reflecting their presence? How do we prevent this attack in the future?

Operators can only answer such questions by drawing upon high-quality logs of past activity recorded over extended time. Incident analysis often starts with a narrow piece of intelligence, typically a local system exhibiting questionable behavior, or a report from another site describing an attack they detected. The analyst then tries to locate the described behavior by examining logs of past activity, often cross-correlating information of different types to build up additional context. Frequently, this process in turn produces new leads to explore iteratively ("peeling the onion"), continuing and expanding until ultimately the analyst converges on as complete of an understanding of the incident as they can extract from the available information.

This process, however, remains manual and time-consuming, as no single storage system efficiently integrates the disparate sources of data (e.g., NIDS, firewalls, NetFlow data, service logs, packet traces) that investigations often involve. While standard SIEM systems such as Splunk aggregate logs from different sources into a single database, their data models omit crucial semantics, and they struggle to scale to the data rates that large-scale environments require.

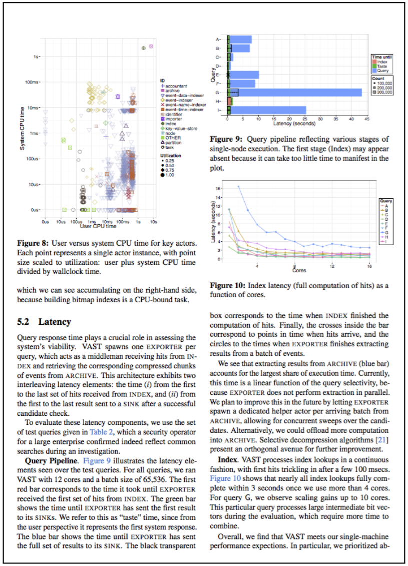
Based on these needs, and drawing upon our years of experience working closely with operational security staff, we formulate three key goals for a system supporting the forensic process [2]:

**Interactivity**. The potential damage that an attacker can wreak inside an organization grows quickly as a function of time, making fast detection and containment a vital concern. Further, a system's interactivity greatly affects the productivity of an analyst [16]. We thus desire replies to queries to begin arriving within a second or so.

**Scalability**. The volume of data to archive and process exceeds the capacity of single-machine deployments. A fundamental challenge lies in devising a distributed architecture that scales with the number of nodes in the system, as well as maximally utilizes the cores available in each node.

**Expressiveness**. Representing arbitrary activity requires a richly typed data model to avoid losing domain-specific semantics when importing data. Similarly, the system should expose a high-level query language to enable analysts to work within their domain, rather than spending time translating their workflows to lower-level system idiosyncrasies.

In this work, we develop a system for network forensics and incident response that aims to achieve these goals. We present the design and implementation of VAST (*Visibility Across Space and Time*), a unified storage platform that provides: (*i*) an expressive data model to capture de-



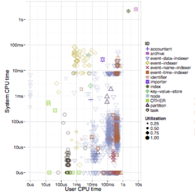


**Figure 8:** User versus system CPU time for key actors. Each point represents a single actor instance, with point size scaled to utilization: user plus system CPU time divided by wallclock time.

which we can see accumulating on the right-hand side, because building bitmap indexes is a CPU-bound task.

### 5.2 Latency

Query response time plays a crucial role in assessing the system's viability. VAST spawns one EXPORTER per query, which acts as a middleman receiving hits from INDEX and retrieving the corresponding compressed chunks of events from ARCHIVE. This architecture exhibits two interleaving latency elements: the time (*i*) from the first to the last set of hits received from INDEX, and (*ii*) from the first to the last result sent to a SINK after a successful candidate check.

To evaluate these latency components, we use the set of test queries given in Table 2, which a security operator for a large enterprise confirmed indeed reflect common searches during an investigation.

**Query Pipeline**. Figure 9 illustrates the latency elements seen over the test queries. For all queries, we ran VAST with 12 cores and a batch size of 65,536. The first red bar corresponds to the time it took until EXPORTER received the first set of hits from INDEX. The green bar shows the time until EXPORTER has sent the first result to its SINKs. We refer to this as "taste" time, since from the user perspective it represents the first system response. The blue bar shows the time until EXPORTER has sent the full set of results to its SINK. The black transparent box corresponds to the time when INDEX finished the computation of hits. Finally, the crosses inside the bar correspond to points in time when hits arrive, and the circles to the times when EXPORTER finishes extracting results from a batch of events.

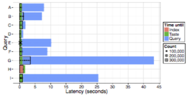


**Figure 9:** Query pipeline reflecting various stages of single-node execution. The first stage (Index) may appear absent because it can take too little time to manifest in the plot.

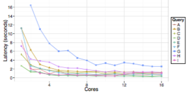


**Figure 10:** Index latency (full computation of hits) as a function of cores.

We see that extracting results from ARCHIVE (blue bar) accounts for the largest share of execution time. Currently, this time is a linear function of the query selectivity, because EXPORTER does not perform extraction in parallel. We plan to improve this in the future by letting EXPORTER spawn a dedicated helper actor per arriving batch from ARCHIVE, allowing for concurrent sweeps over the candidates. Alternatively, we could offload more computation into ARCHIVE. Selective decompression algorithms [21] present an orthogonal avenue for further improvement.

**Index**. VAST processes index lookups in a continuous fashion, with first hits trickling in after a few 100 msecs. Figure 10 shows that nearly all index lookups fully complete within 3 seconds once we use more than 4 cores. For query G, we observe scaling gains up to 10 cores. This particular query processes large intermediate bit vectors during the evaluation, which require more time to combine.

Overall, we find that VAST meets our single-machine performance expections. In particular, we prioritized ab-

## Conclusion

### Network Forensics Challenges

- Explorative high-dimensional search
- Disparate data access
- Massive data volumes

### VAST: Visibility Across Space and Time

- Platform for network forensics
- Interactive & iterative search
- *Inter*-machine and *intra*-machine scaling
- Open-source, permissive license (BSD)

## Questions?

http://vast.io