System for Worm Observation and Rapid Detection

17 June 2004
NETSEC Meeting

Eric Anderson

anderson@cs.uoregon.edu

University of Oregon

Network Security Research Group

Worms: Who Cares?

- Internet worms have been costly and destructive
  - Traffic causes network collapse.
  - Infected hosts are often unusable.
  - Repair is labor-intensive.
  - Code Red v.2 and Slammer estimated to cost $2bn and $1bn, respectively.
- . . . by accident.
- A truly malicious worm could do much worse.

Why Automated Worm Detection

- Worms can spread incredibly quickly.
  - Slammer reached its peak infection rate in 3 minutes.
  - Theoretical worms: 15 seconds.
  - Time window for useful response even smaller.
- Human-mediated analysis and response are not fast enough.

Intrusion Detection Today

- Signature-based:
  - Monitors keep database of known attacks.
  - Works, after a signature's created.
- Anomaly-based:
  - All traffic is compared to statistical profiles of "normal" traffic.
  - False positive rate is too high to allow automatic responses.
- Specification-based:
  - Rules describe acceptable behavior.
  - Deviant traffic is flagged.

Specification-based Worm Detection

- (Fast) worms are not terribly subtle
  - Infected hosts initiate many connections.
    - Possibly to addresses with no host.
    - Or services which aren't running.
  - Infected hosts start acting like the host which infected them.
- We can observe that a worm exists, but that isn't enough information to do anything.

Our Approach

- A network of monitors:
  - Maintain (shared) set of known worm signatures.
  - Detect “worm-like” events heuristically.
  - Create new signatures from events **not** matching existing ones.
    - Maximize matches with worms.
    - Minimixe matches with other.
- Sharing signatures and data.
  - Individually, monitors may not have enough data.

Approach Overview

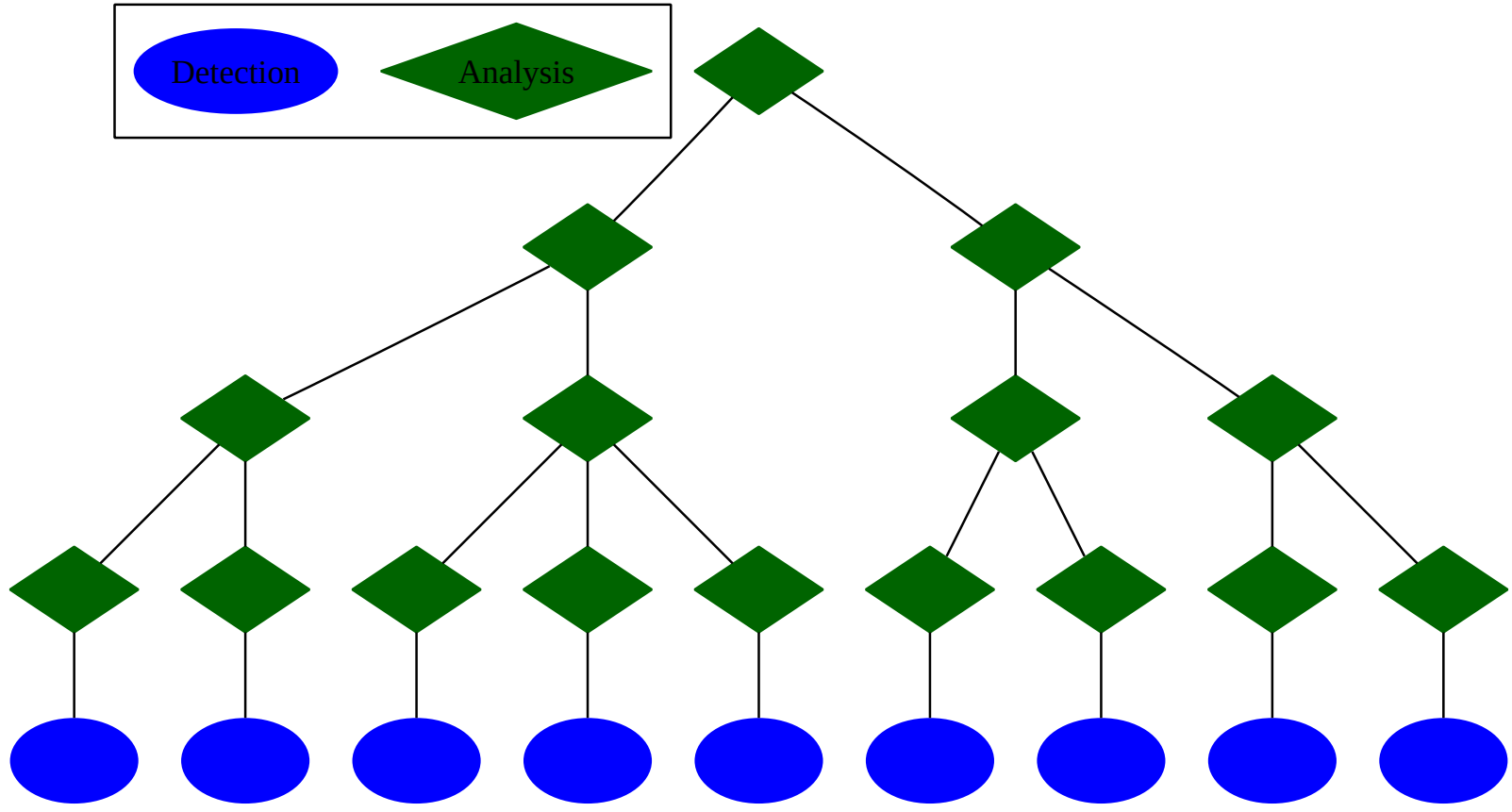
Limitations of Isolated Monitors

- A worm monitor can try to detect new worms
  - Monitor local traffic
  - Create candidate worm signatures
  - Take proper actions
- False positives and false negatives
  - Monitor’s view of network traffic may not be representative
  - Or comprehensive
- Slow detection
  - Won’t know a 0-day worm before seeing actual worm traffic
  - Or seeing "enough" of the worm’s traffic

Worm Monitor Collaboration

- After a monitor finds a local candidate worm signature, it can report that signature to other monitors
  - and info for deriving the signature
- The receiving monitor uses its data to validate & re-optimize that signature
  - Verify against combined data set
  - Fit with worm propagation models
  - Gather accuracy statistics
- Repeated until a global consensus is reached

Tree-structured Monitor Overlay



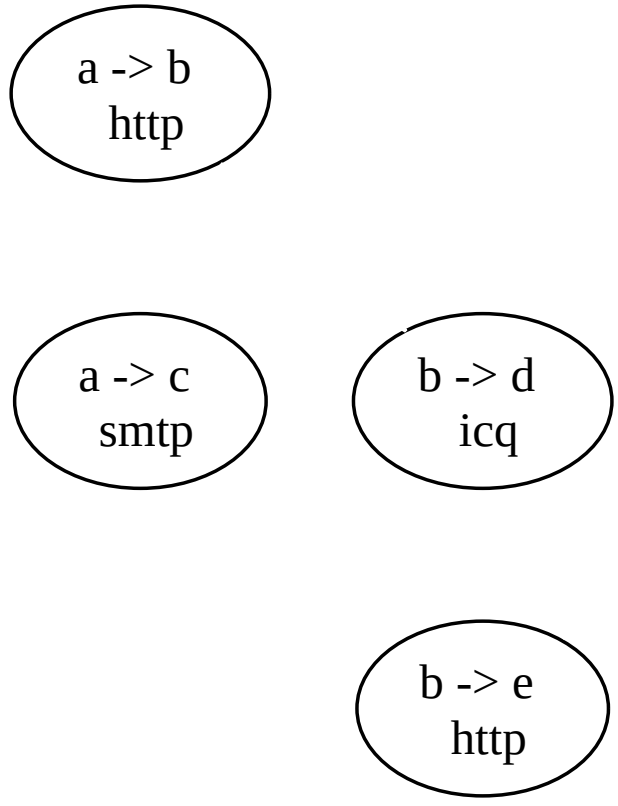
Event History Graphs

Events and Causation

- Represent obseved events and their causal connections as a graph
- Nodes are events (network connections)
- Node labels describe event
  - Source, destination, start time
  - Protocol, ports, duration, size
  - (payload hash, byte distribution, ?)
- Edges are (possible) causation.
- . . . Lamport *happened-before* relation.

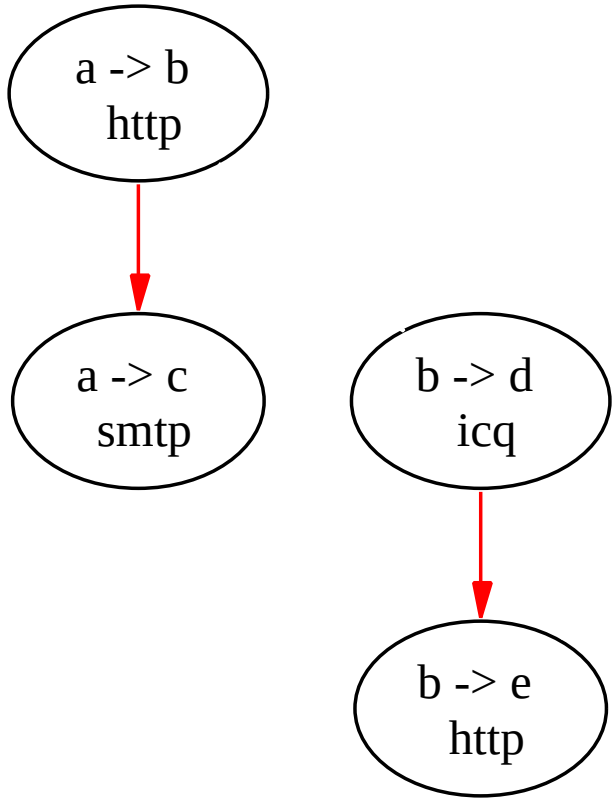
Events Cause Other Events

- Later event in same process
- Or in one communicated with
- Transitive
- Closure $\supseteq$ actual causes



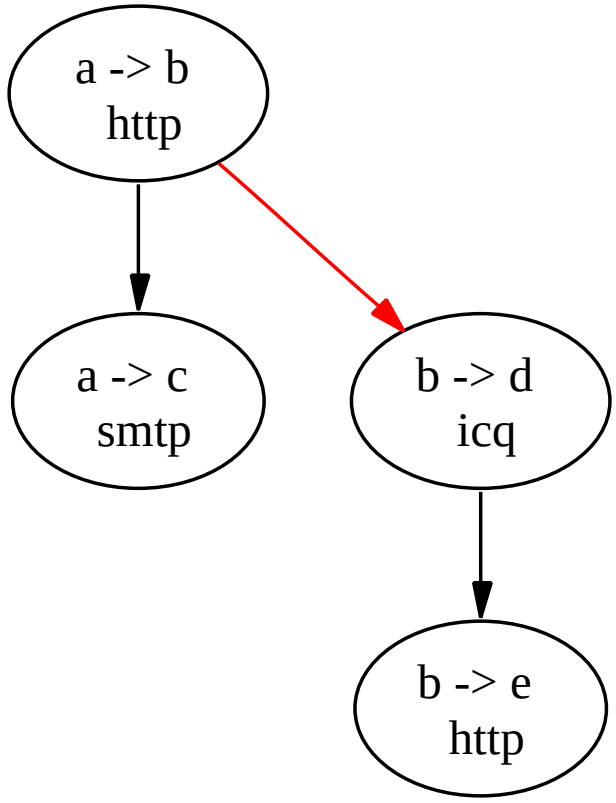
Events Cause Other Events

- Later event in same process
- Or in one communicated with
- Transitive
- Closure $\supseteq$ actual causes



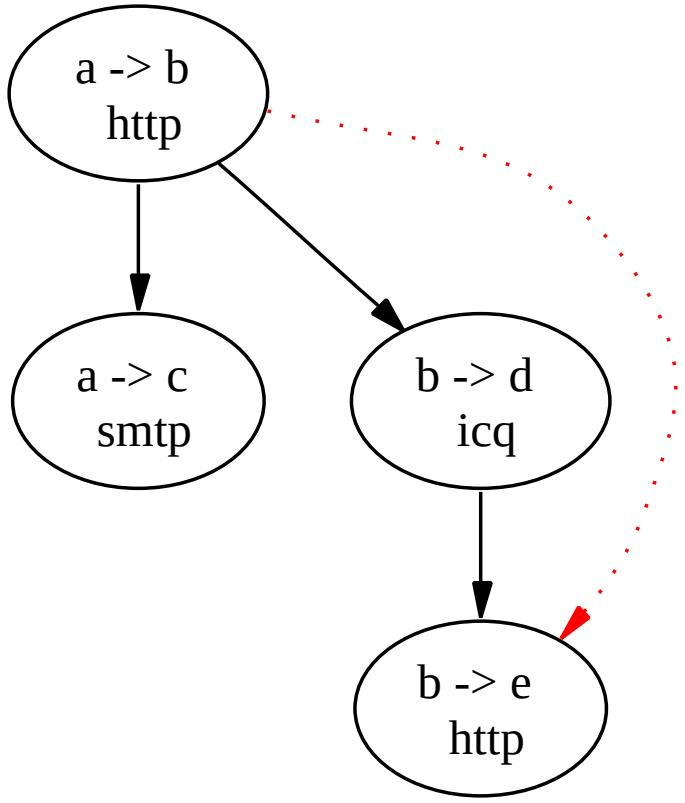
Events Cause Other Events

- Later event in same process
- Or in one communicated with
- Transitive
- Closure $\supseteq$ actual causes



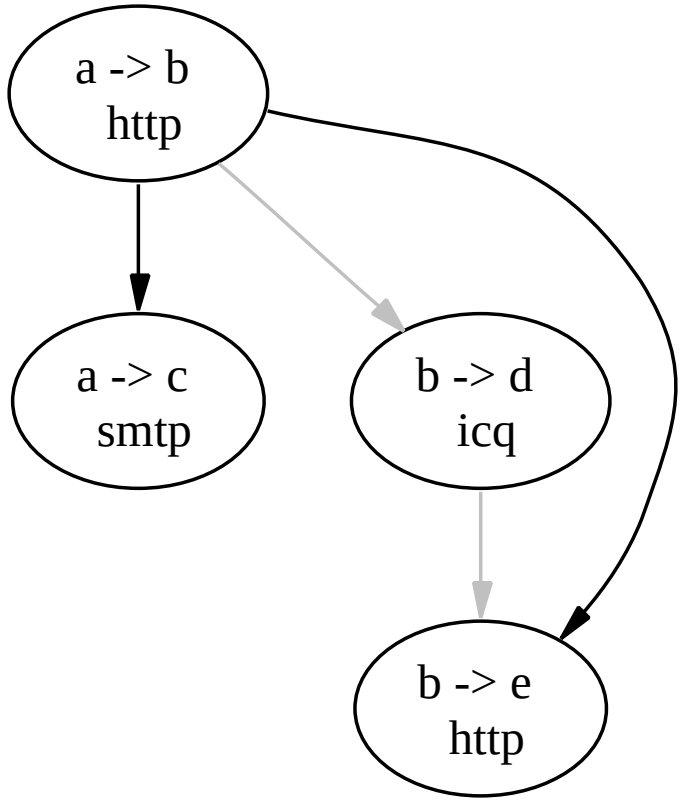
Events Cause Other Events

- Later event in same process
- Or in one communicated with
- Transitive
- Closure $\supseteq$ actual causes



Events Cause Other Events

- Later event in same process
- Or in one communicated with
- Transitive
- Closure $\supseteq$ actual causes



Why Event History Graph?

- Represents pattern of events
  - GrIDS[3] introduced “activity graph:” Hosts are nodes and edges are communication.
  - The same host is multiple nodes in the graph, though.
  - Less clean semantics.
- Worm behavior pattern guaranteed to precede infection event

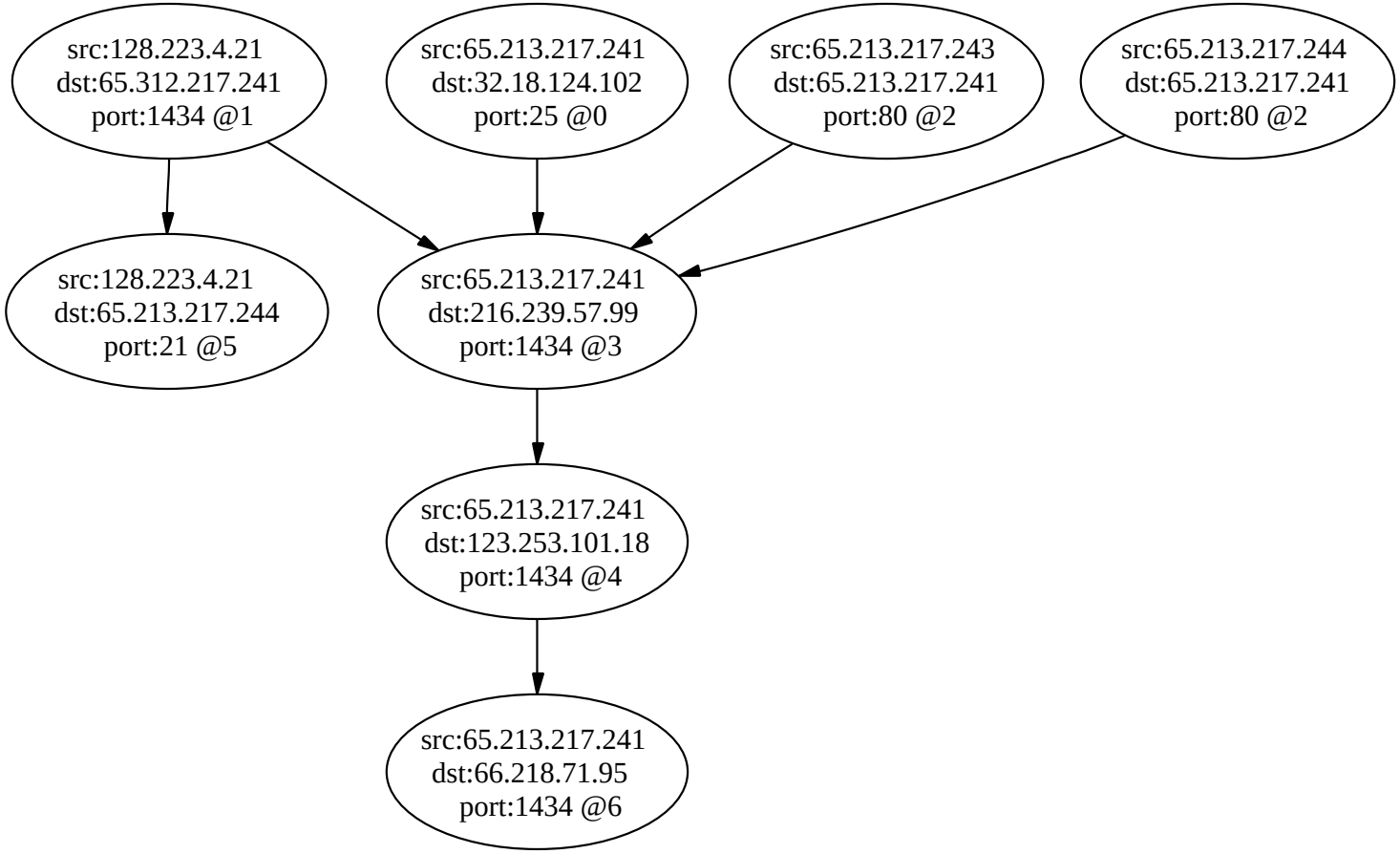
Generating E.H. Graph - Input

Observed Events

Time	Src	Dst	Port
0	65.213.217.241	32.18.124.102	25
1	128.223.4.21	65.312.217.241	1434
2	65.213.217.243	65.213.217.241	80
2	65.213.217.244	65.213.217.241	80
3	65.213.217.241	216.239.57.99	1434
4	65.213.217.241	123.253.101.18	1434
5	128.223.4.21	65.213.217.244	21
6	65.213.217.241	66.218.71.95	1434

Generating E.H. Graph - Output

Observed Events



Subsection: Self-Similarity

Self-Similarity Outline

- **Background**
  - Worm models
  - Detecting scanning
  - . . . and spreading
- Causation and Self-Similarity

Worm Models

- “Scanning” Worm probes random addresses, hoping for a hit. Scans are not so subtle.

Worm Models

- “Scanning” Worm probes random addresses, hoping for a hit. Scans are not so subtle.
- Improved scanning: Prune non-routable or low-probability address space, coordinate scans. Faster, fewer wasted scans.

Worm Models

- “Scanning” Worm probes random addresses, hoping for a hit. Scans are not so subtle.
- Improved scanning: Prune non-routable or low-probability address space, coordinate scans. Faster, fewer wasted scans.
- “Hit list” Pre-determined list of likely vulnerable hosts. No conspicuous scans!

Worm Models

- “Scanning” Worm probes random addresses, hoping for a hit. Scans are not so subtle.
- Improved scanning: Prune non-routable or low-probability address space, coordinate scans. Faster, fewer wasted scans.
- “Hit list” Pre-determined list of likely vulnerable hosts. No conspicuous scans!
- (Well, maybe one)

Worm Models

- “Scanning” Worm probes random addresses, hoping for a hit. Scans are not so subtle.
- Improved scanning: Prune non-routable or low-probability address space, coordinate scans. Faster, fewer wasted scans.
- “Hit list” Pre-determined list of likely vulnerable hosts. No conspicuous scans!
- (Well, maybe one)
- “Topological” detect victim hosts on the fly. More subtle.

Worm Models

- “Scanning” Worm probes random addresses, hoping for a hit. Scans are not so subtle.
- Improved scanning: Prune non-routable or low-probability address space, coordinate scans. Faster, fewer wasted scans.
- “Hit list” Pre-determined list of likely vulnerable hosts. No conspicuous scans!
- (Well, maybe one)
- “Topological” detect victim hosts on the fly. More subtle.
- [2, 5]

Detecting Scanning

Scanning worms do unusual, detectable things:

- Contact unassigned IP addresses
- Contact hosts on closed ports
- Few are chosen, so many are called.

But ...

- Not all worms are scanning
- Not all scans are worms
  - (e.g.) Gnutella peer probing cached hosts.
  - Fake worm as DoS attack.

Detecting Spreading

Birds gotta fly, worms gotta spread

- Track the rate of increase of . . . and react if it's too great.
- But, you need some suspect event to be count in the first place.
- Thus, tracking the rate of increase of scans help to separate worm scans from non-worm scans. [6, 5]

Detecting Spreading

Birds gotta fly, worms gotta spread

- Track the rate of increase of . . . and react if it's too great.
- But, you need some suspect event to be count in the first place.
- Thus, tracking the rate of increase of scans help to separate worm scans from non-worm scans. [6, 5]
- We thought of this before those came out. Too bad we didn't publish sooner!

Causation and Self-Similarity

- One worm infection causes another (and another...)
- Worm processes are “the same” on every infected host
- ... processes are abstractions

Causation and Self-Similarity

- One worm infection causes another (and another...)
- Worm processes are “the same” on every infected host
- ... processes are abstractions
- Events are the manifestation of processes
- Network connections are observable events
- Look for connection events that cause other similar ones.[4]

Similarity

What does it mean that connections (might) cause "similar" ones?

- Worms can vary behavior as much as they like.
- ...but they have to use the vulnerabilities they're given.
- May fix port number(s), TCP options, message sizes.
- How trustworthy is this constraint?

Similarity

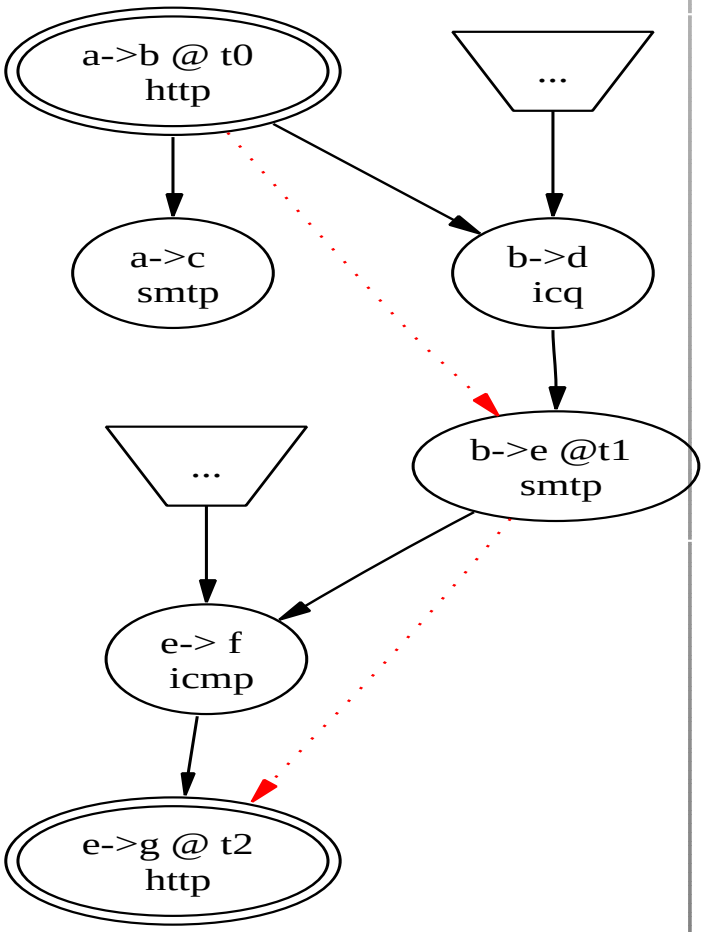
What does it mean that connections (might) cause "similar" ones?

- Worms can vary behavior as much as they like.
- ...but they have to use the vulnerabilities they're given.
- May fix port number(s), TCP options, message sizes.
- How trustworthy is this constraint?
  - No idea!
  - But at least it's not up to the worm author.

Self-Similarity in a Worm

Examining $v =$ `e->g http`

- Infections and noise causally precede connection.
- Two-(or more) vector worm
- Matches $v' =$ `a->b http`
- Infection rate = $((t2 - t0)/2)^{-1}$



A Naïve Similarity Measure σ

To score event v:

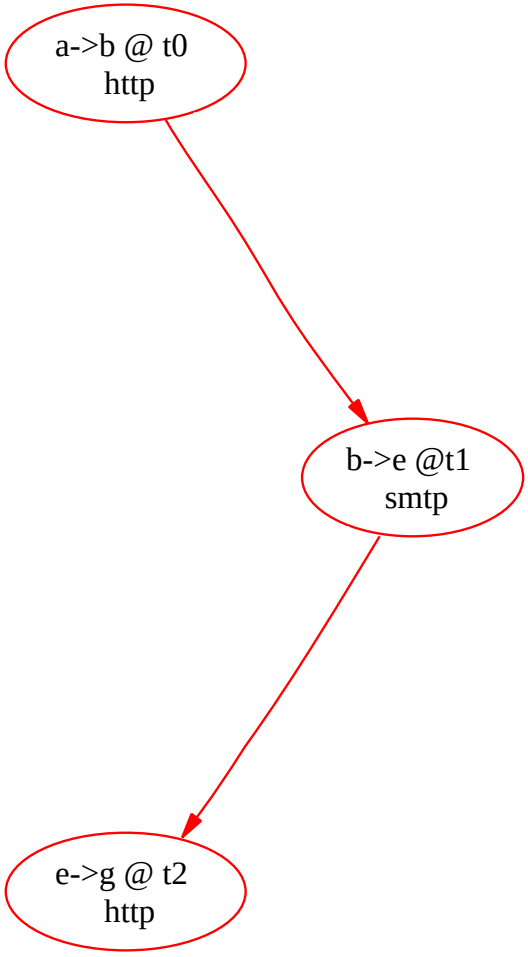
- For each v' in causal history:
  1. Compare each field
  2. Sum similarity scores
- Crude but conservative
- Event score = $max_{v'|v' \rightsquigarrow_w v} \sigma(v, v')$

Field	Type
protocol	nominal
destination port	nominal
duration	cardinal
total bytes sent by sender	cardinal
total bytes sent by receiver	cardinal
payload bytes sent by sender	cardinal
payload bytes sent by receiver	cardinal
sender TCP SYN seen	nominal
⋮	nominal

Signature Generation

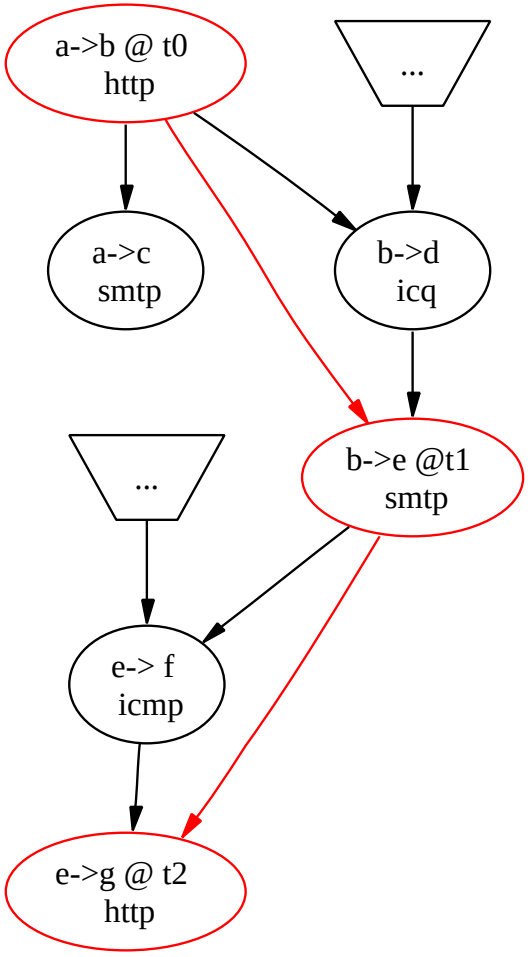
Signatures

- Event (sub-)graph with added constraints.
- A signature matches if there is a subgraph isomorphism which satisfies the constraints (when constraint variables are set to node-specific values.)



Signatures

- Event (sub-)graph with added constraints.
- A signature matches if there is a subgraph isomorphism which satisfies the constraints (when constraint variables are set to node-specific values.)



Signature Generation

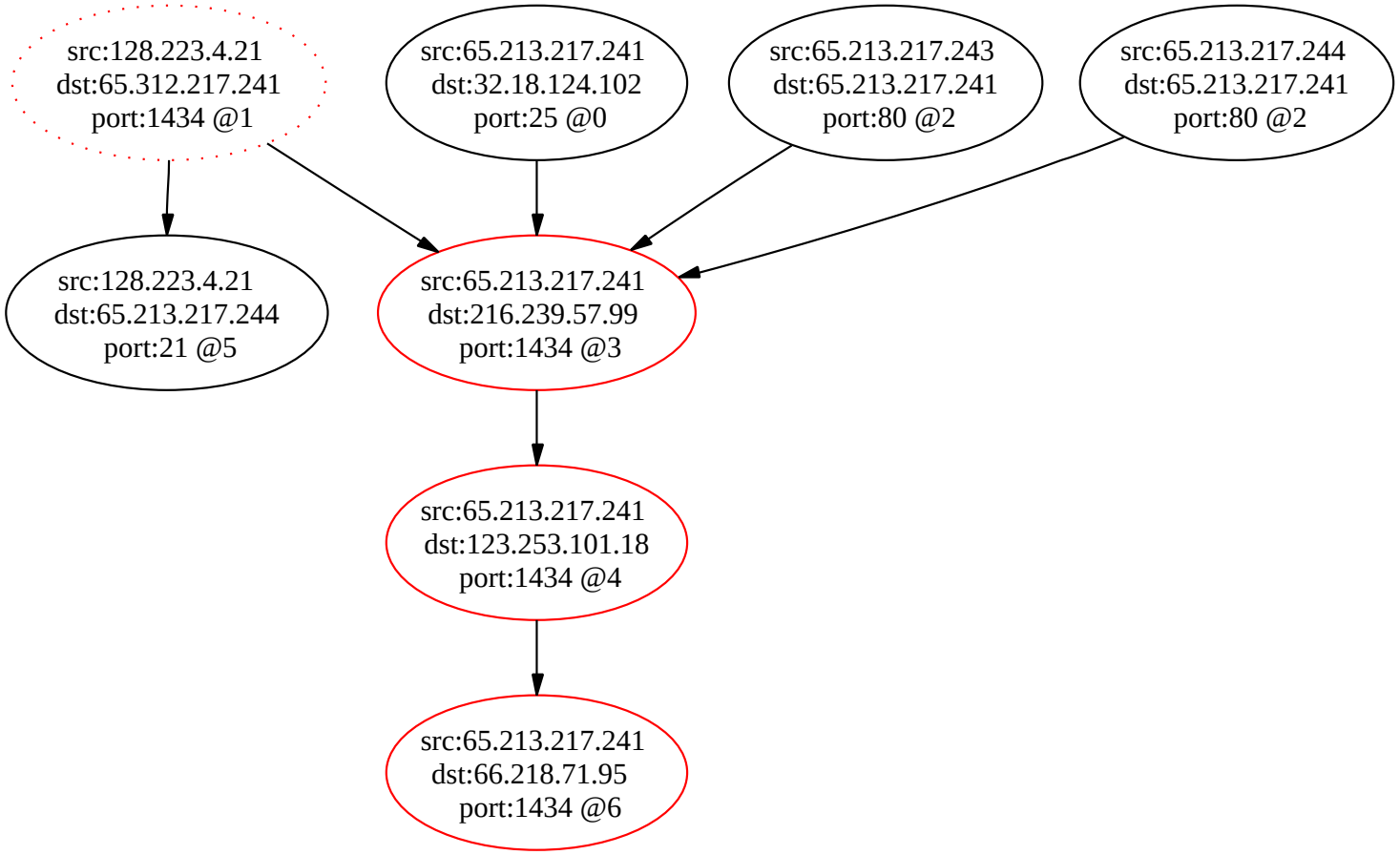
- Happens when (the detection component of) a monitor notices a worm-like event which is not matched by any existing signature.
- Candidate signature is created “dumbly” and then optimized using a best-first search through the space of possible generalization.
- If two candidate signatures overlap excessively, the lower-scoring one is deleted.

Signature Generation

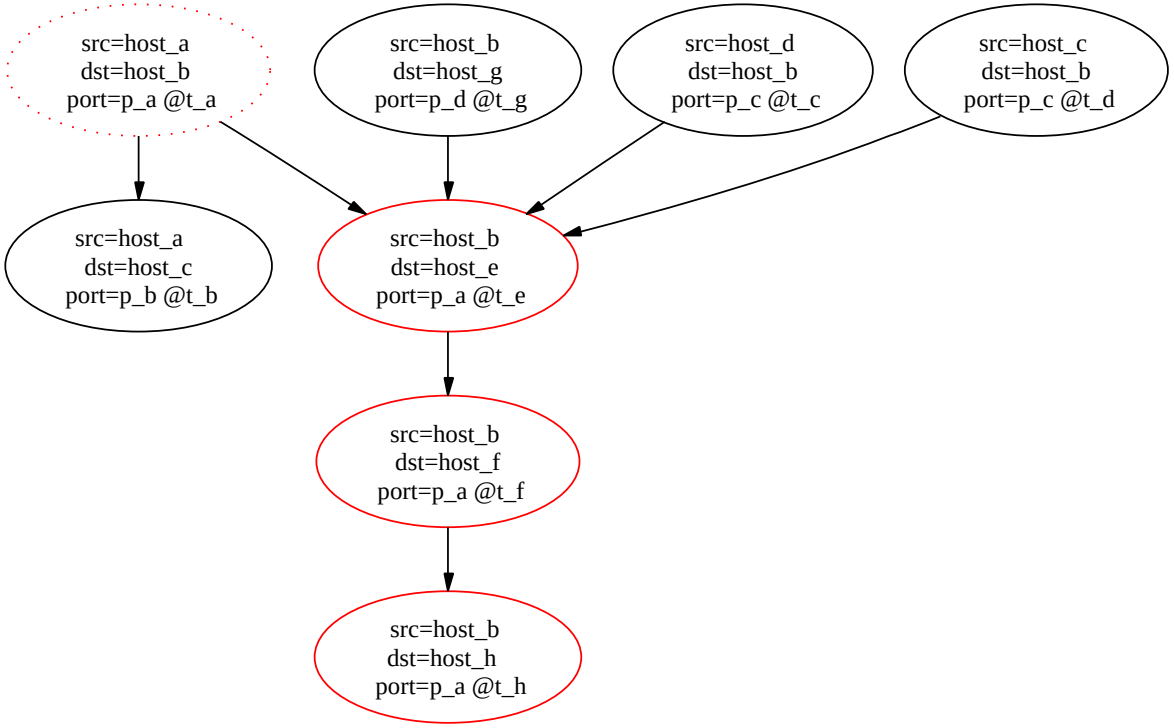
- Snapshot of graph preceding flagged node
  - Literals replaced with (bound) variables
  - Binding dropped for exact time, host addresses
  - Variable bindings, relational expressions are constraints
- “Relaxed” by removing nodes and constraints.
- Goal: Maximize α * (# of newly-matched worm events) β * (# of matched non-worm events) + γ * (# of nodes, constraints).
  - $\beta, \gamma < 0$.
  - $|\gamma| \ll |\alpha| < |beta|$

Example - Possible Worm Traffic

Heuristic has already identified worm events:

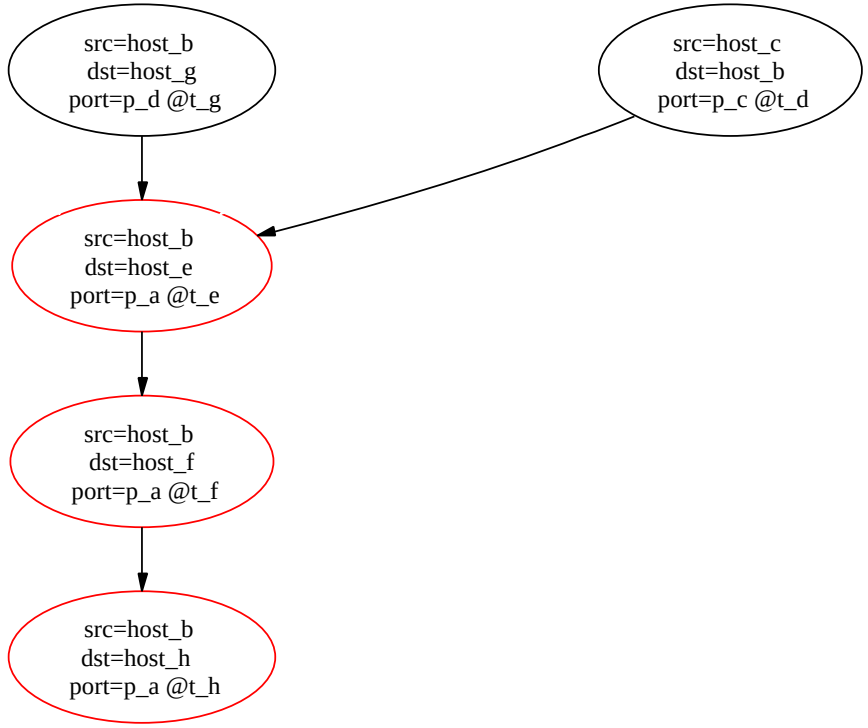


Signature Extraction



$p\_a = 1434,\ p\_b = 21,\ p\_c = 80,\ p\_d = 25,\ host\_a \neq host\_b \neq host\_c \neq \ldots,\ t\_a \leq t\_b \leq t\_c = t\_d \leq \ldots$

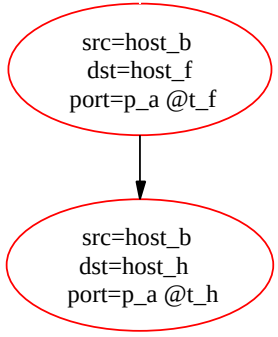
Signature Optimization - Remove nodes & Constraints



$p\_a = 1434, \quad p\_c = 80, \; p\_d = 25,$

$host\_b \neq host\_c \neq \ldots, \; t\_a \leq t\_b \leq t\_c = t\_d \leq \ldots$

Signature Optimization - Remove nodes & Constraints

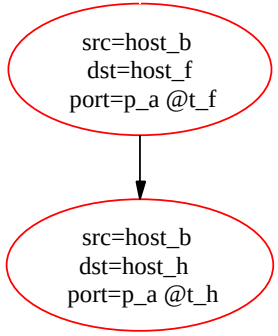


p_a = 1434,

host_b $\neq$

host_f $\neq$ host_h

Signature Optimization - Remove nodes & Constraints



~~p_a = 1434,~~ host_b $\neq$ host_f $\neq$ host_h. Over-generalization: Matches non-worm nodes, lowers score. Backtrack!

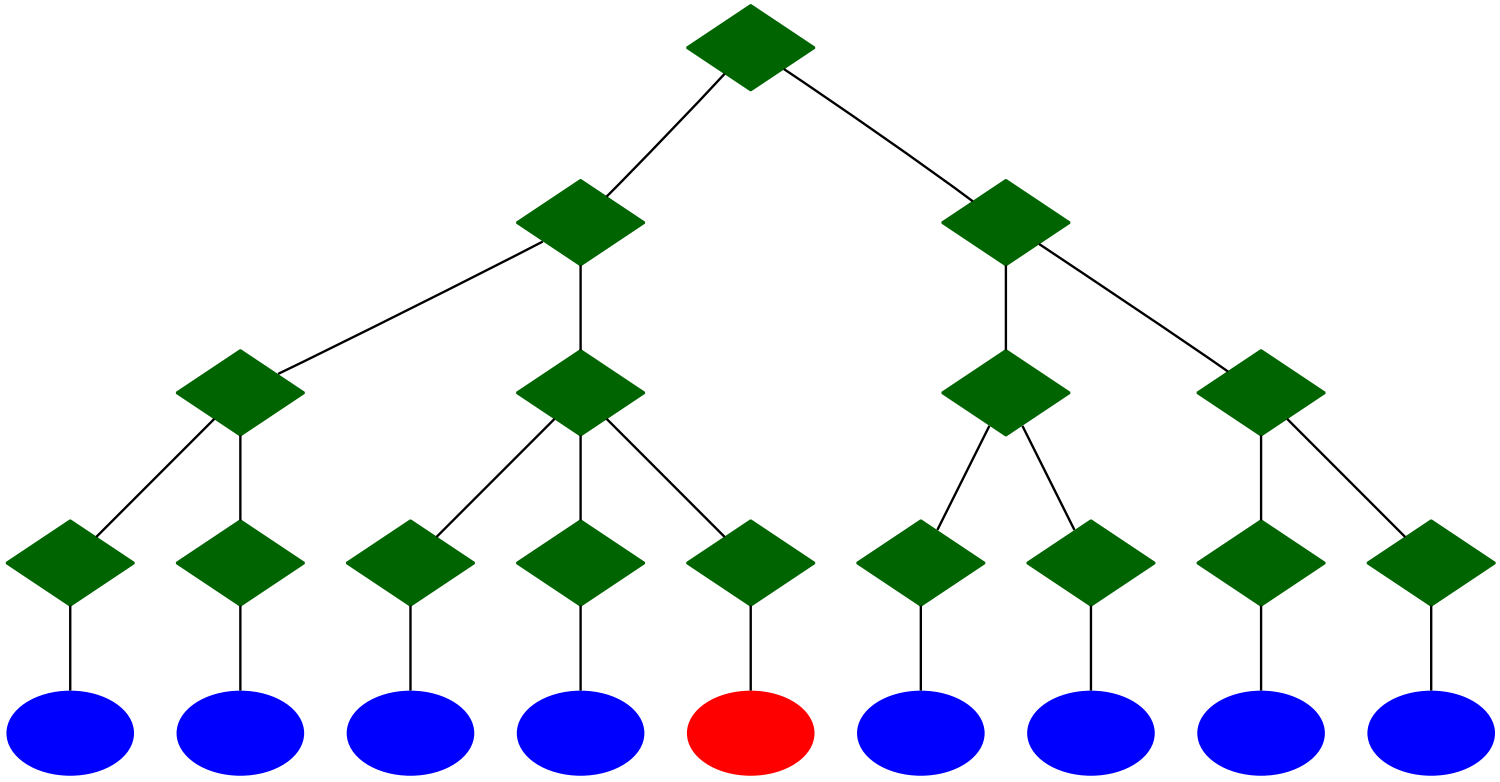
Distributed Signature Optimization

- *False positive problem* - Monitor's local view of network traffic is not necessarily representative
- After a monitor finds a local optimum, the candidate signature is disseminated to a larger set of monitors
- Signature is re-optimized using combined data
- Repeated with increasingly large groups until a global consensus is reached
- Monitors are arranged in a tree-structured overlay

Distributed Signature Optimization Protocol - Synchronous

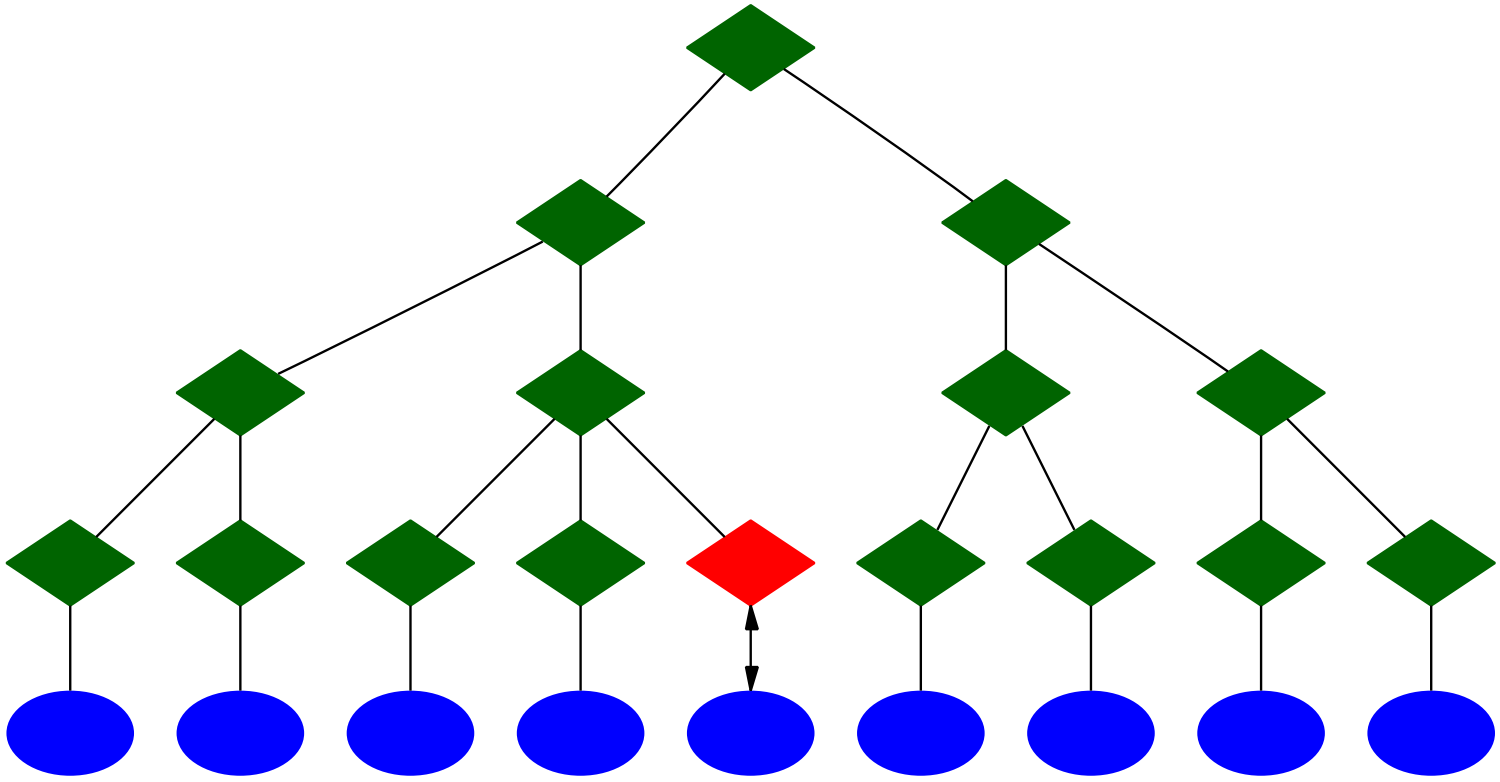
1. Monitor (detector) observes a worm event
2. Monitor (detector) extracts a candidate signature
3. Monitor optimizes signature (repeated):
   (a) Compute k plausible relaxations
   (b) Send relaxation set query to child nodes(if any)
   (c) Computer score for set using local data (if any)
   (d) Wait for score responses from all child nodes.
   (e) Combine scores
   (f) Choose next k relaxations
4. Monitor sends candidate signature search state to parent

Example - 1



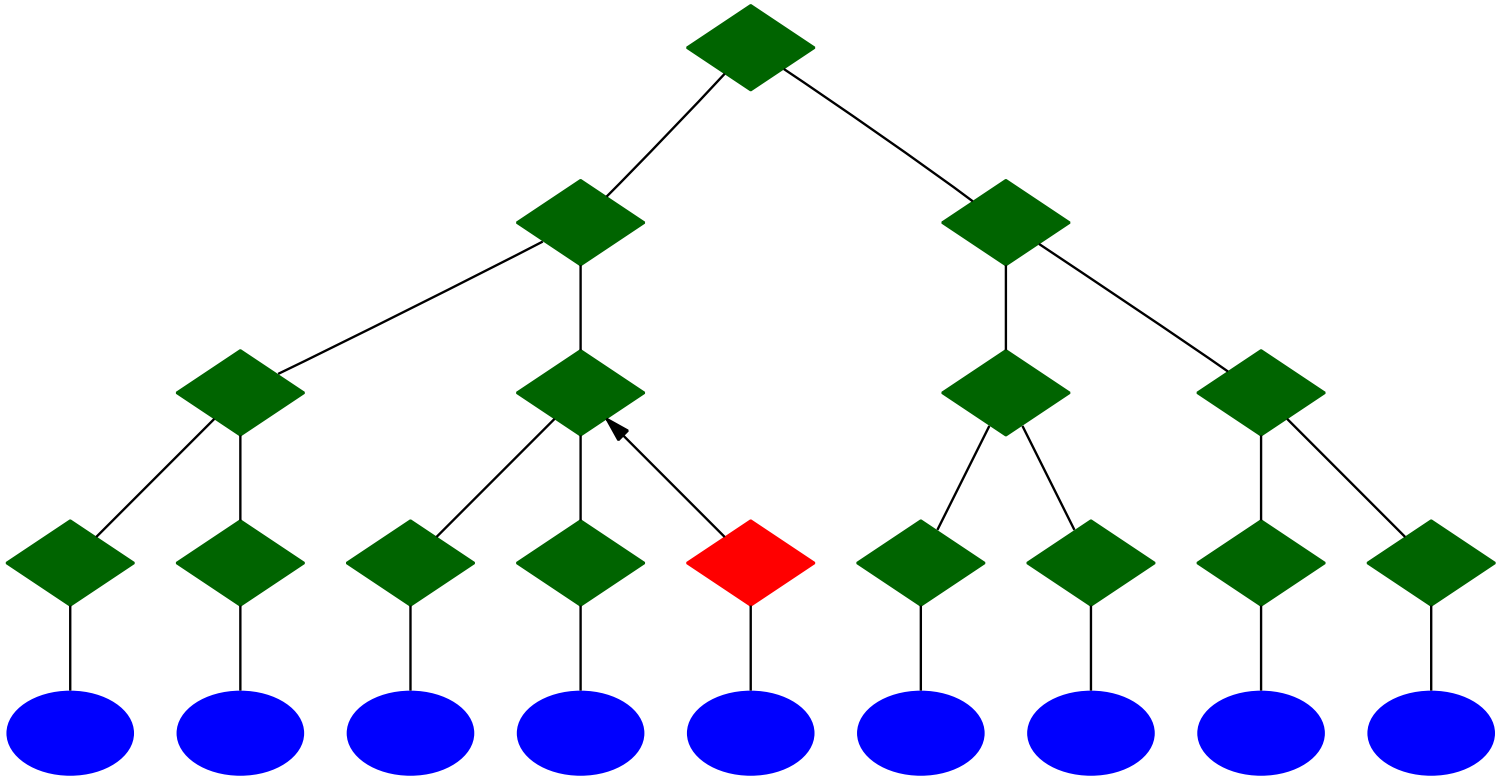
Red detector observes an un-matched worm-like event

Example - 2



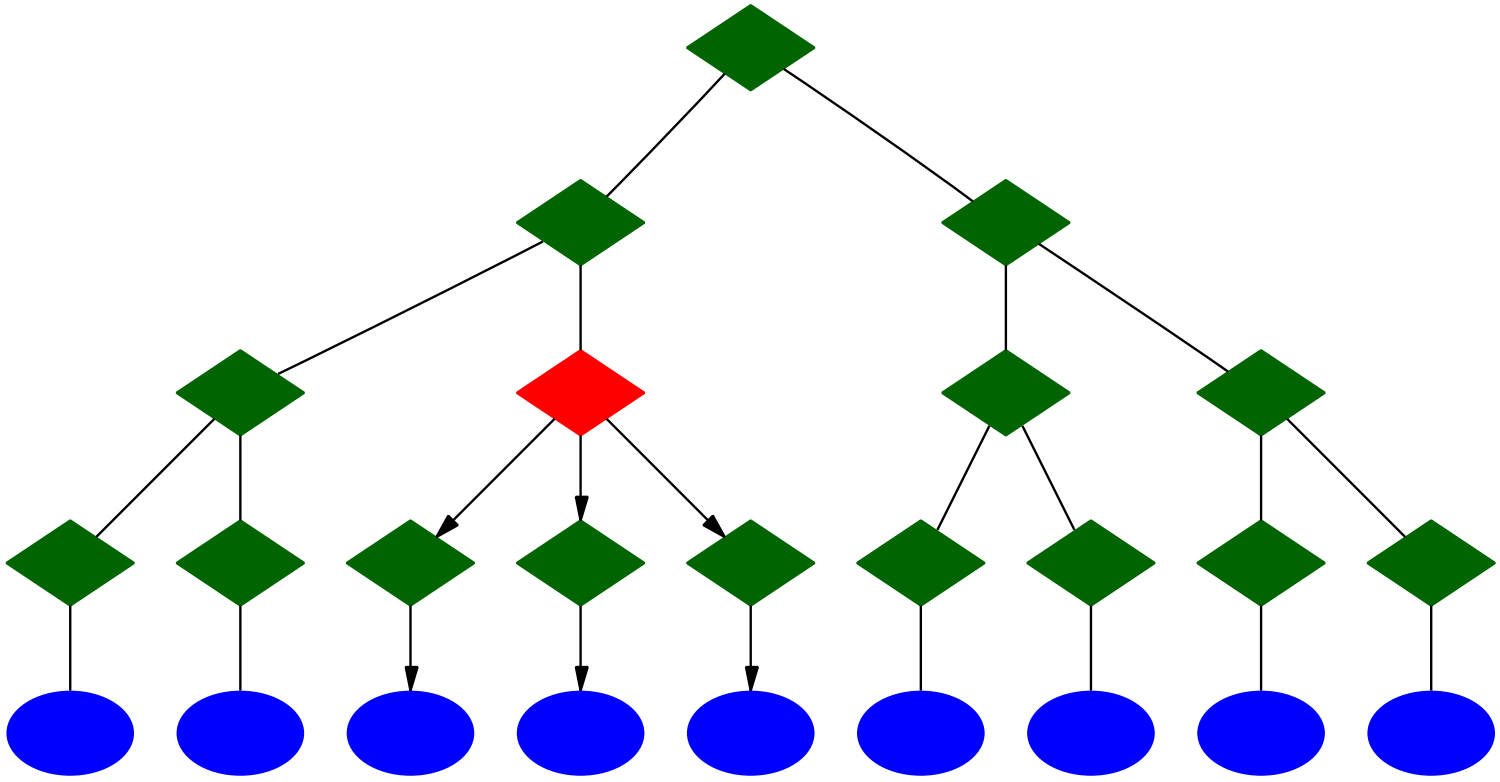
Red analysis node optimizes candidate signature

Example - 3



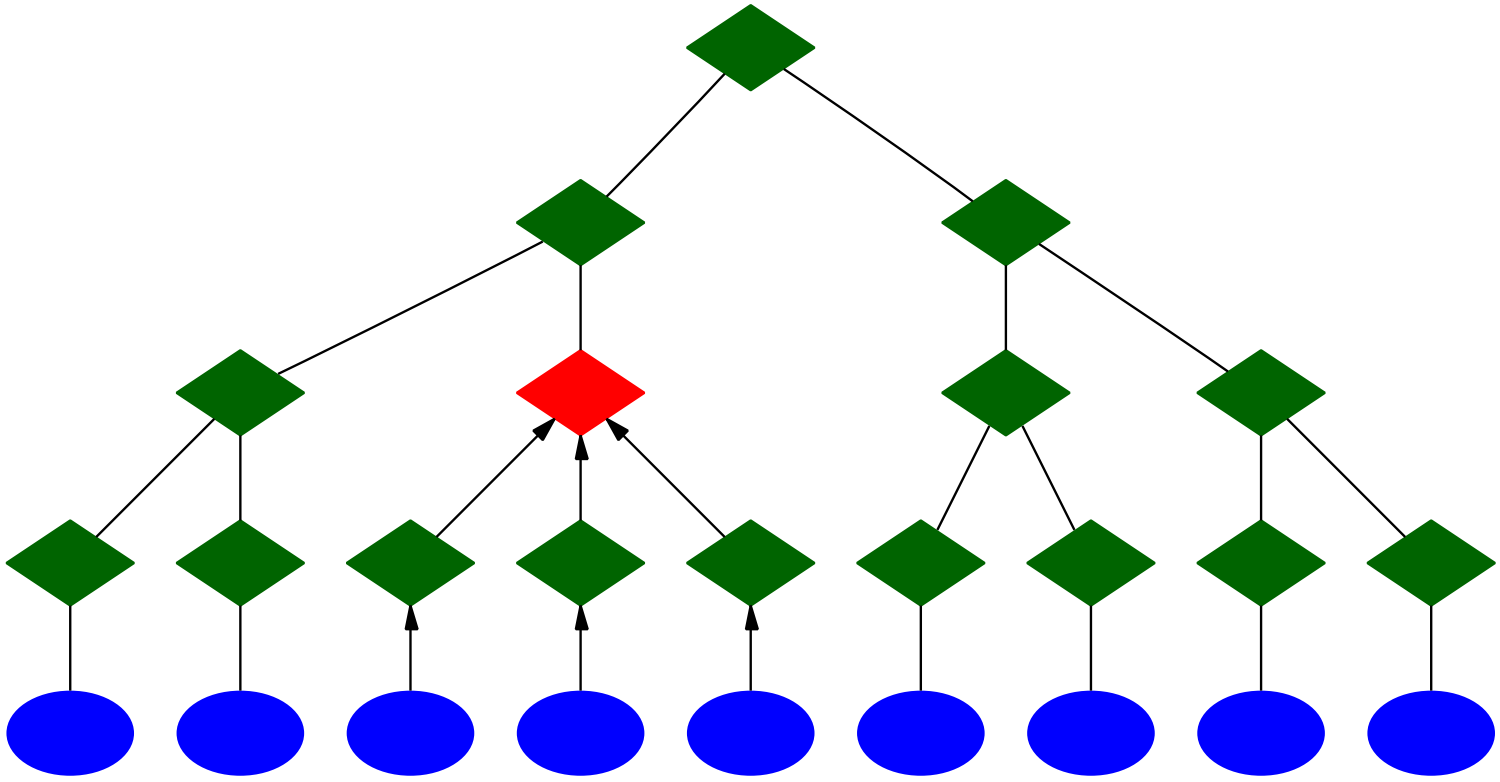
Red analysis node passes state to parent

Example - 4



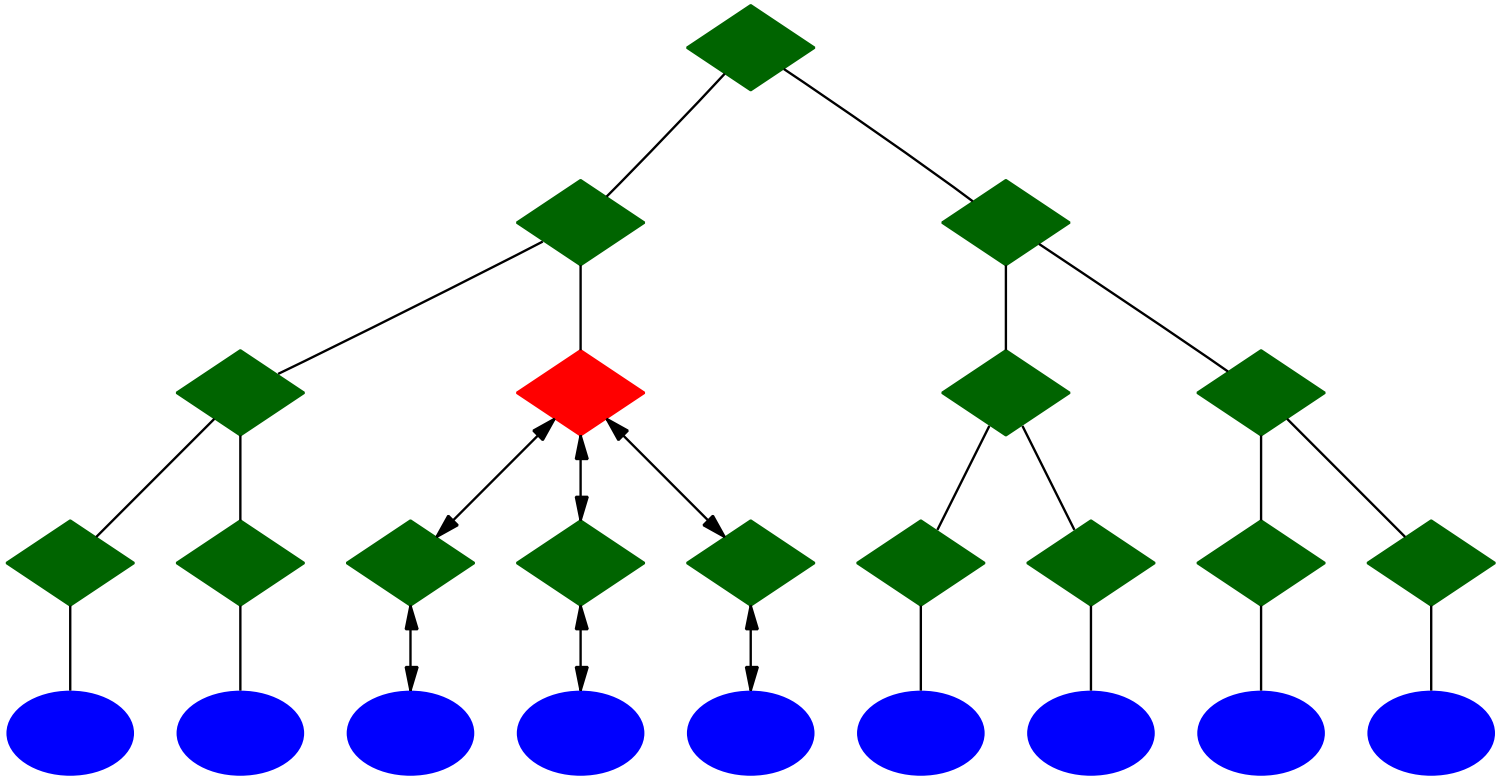
Red analysis node recursively queries children

Example - 5



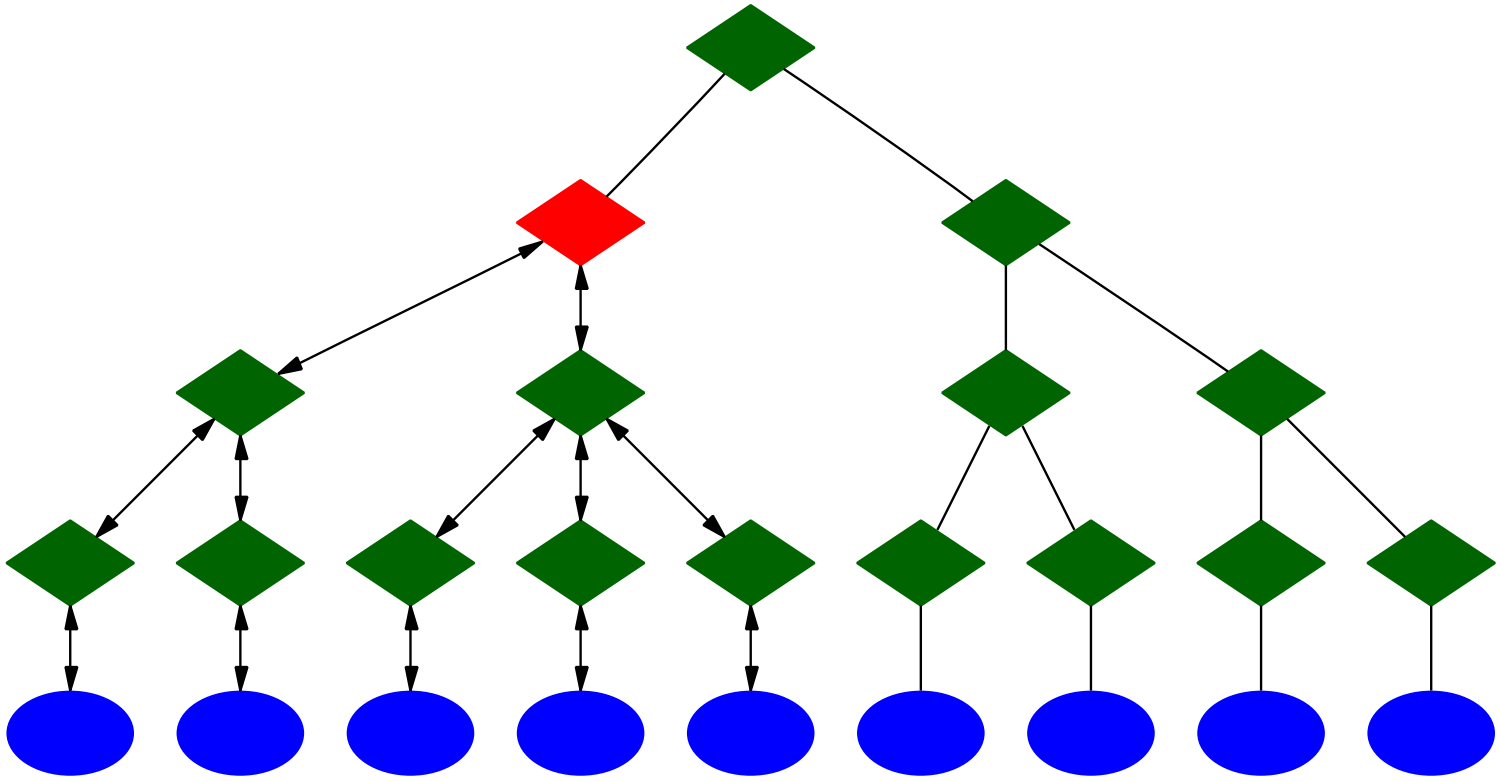
Children reply with real and false positive counts, total data set size

Example - 6



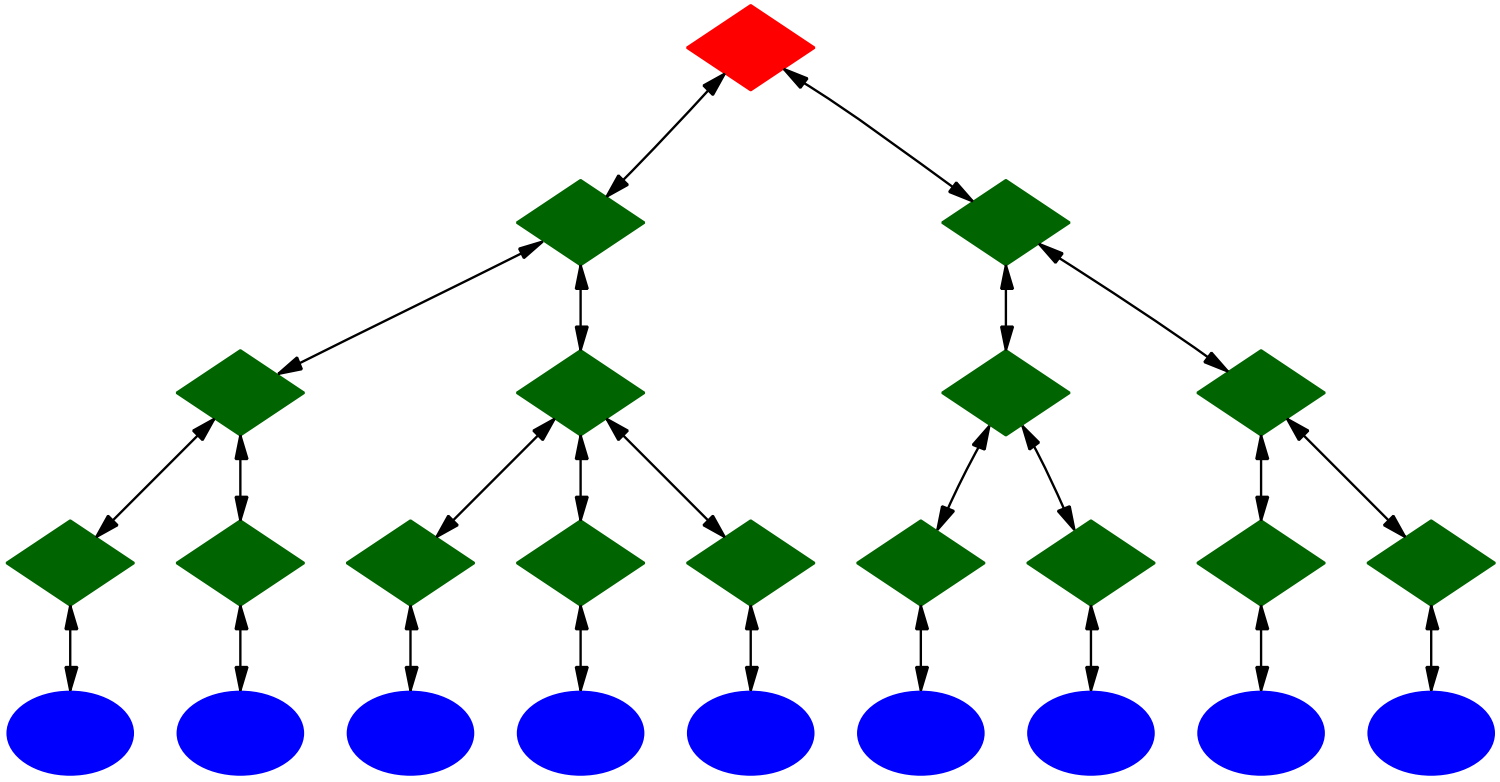
Repeat until local optimum found

Example - 7



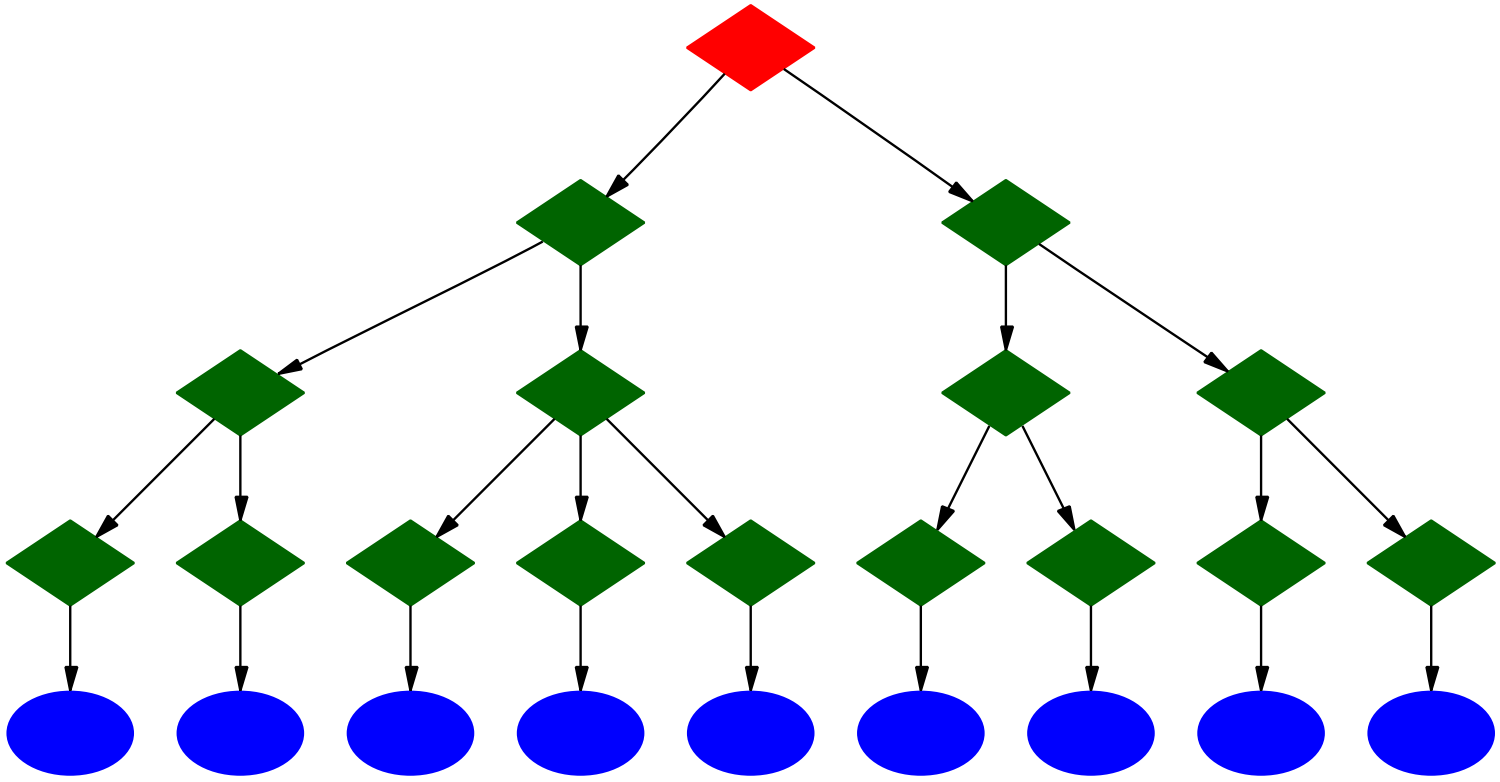
Process repeats at parent

Example - 8



Process repeats at parent (root)

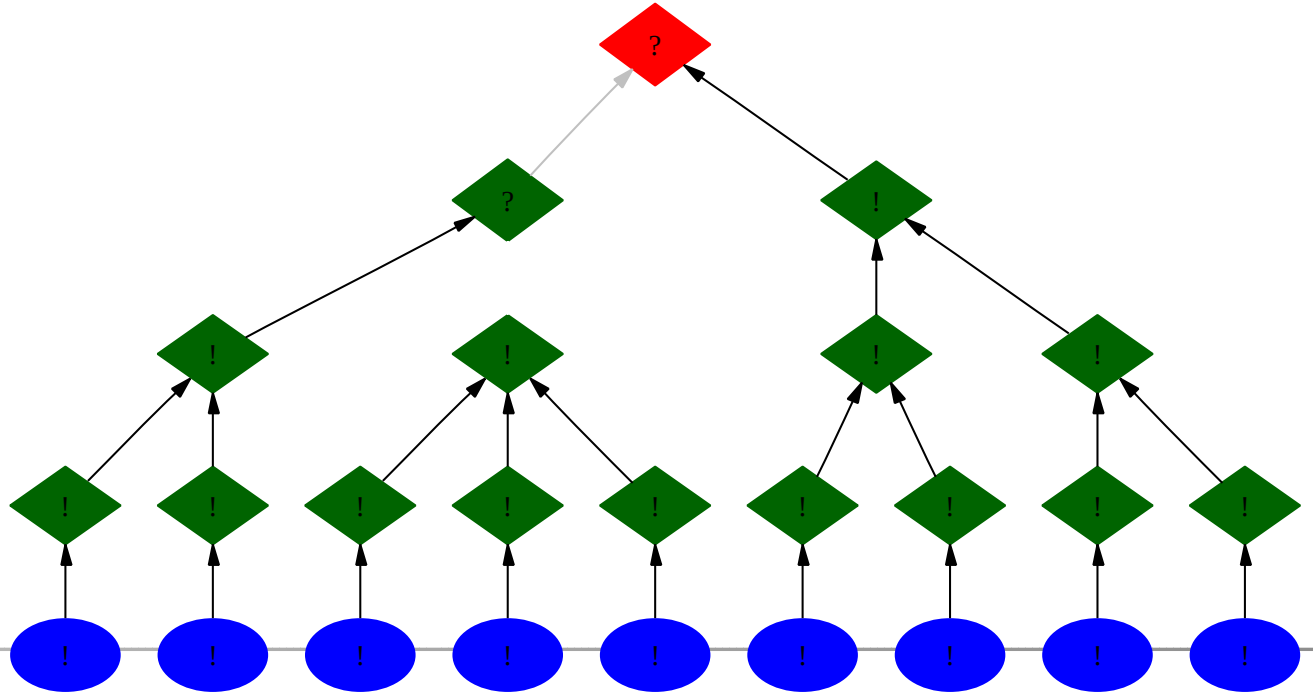
Example - 9



Final signature is disseminated

Problems with Synchronicity

- Slows entire process to rate of slowest node
- Broken links or overloaded nodes are quite plausible



Asynchronous Signature Optimization

- As with synchronous protocol, but:
  - Monitor (analysis node) does not wait for children to respond
  - Tracks most-recently-acted on values
  - Acts whenever local current values change by hysteresis value ϵ.
  - (with a rate limit)
  - “Act” means computing promising relaxations or
  - reporting an answer to a (recursive) query

System Evaluation

Evaulation thus far

- Self-similarity-based Detection
- Worm Monitor Communication

Measurement Evaluation of Self-Similarity

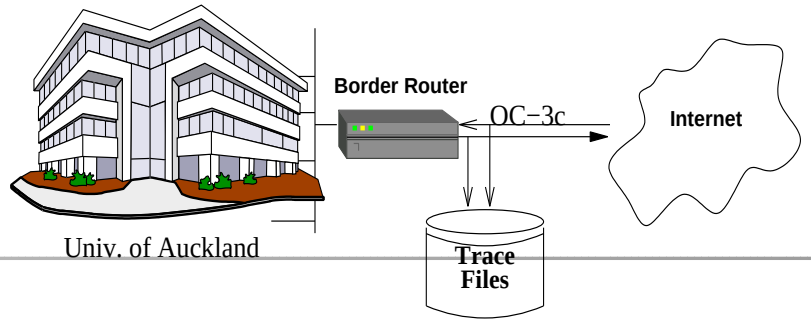
We've argued that worms will show high $max_{v'|v' \rightsquigarrow_w v} \sigma(v, v')$ values. Does that set them apart?

Goals:

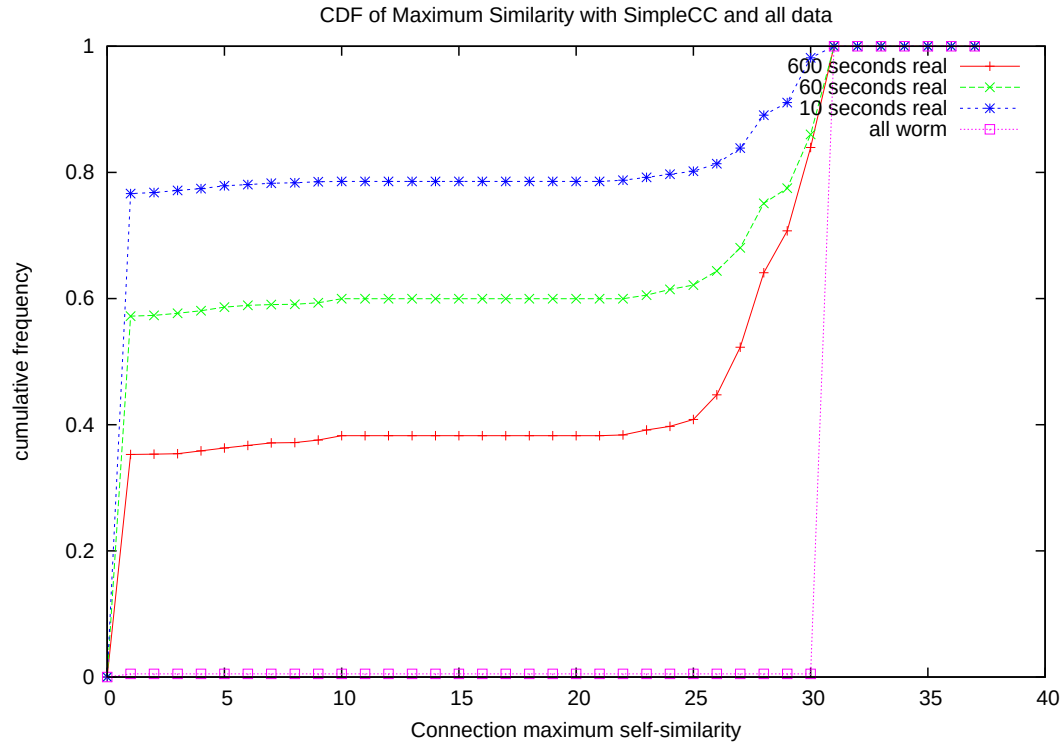
- Determine score distribution for real non-worm traffic for various time windows w.
- For each w, find the relation between score and confidence that traffic is non-worm.
- Offer sensible w and worm score threshold τ.

Approach - In Theory

- Process 2 months of headers from U. of Auckland.
- Aggregate packet events into connection events.
- Exclude known worms from normal traffic.
- Maintain a sliding window of event records.
- Score all recorded events.
- Find score distribution, and confidence levels for each threshold.



Results



- Results from local test traces.
- $\approx$ 95% of connections differ clearly from worms

Conclusions - Self-Similarity

- $\approx$ 95% of connections differ clearly from worms
- Is that good enough? Not by itself, no.
- With secondary reasonableness checks? Maybe.
- How can we improve it?
  - Prioritized comparison function?
  - Heuristics on causation and timing?
  - Hacks to lower false positives break coverage claims! Hmmm...

Communication Structure Evaluation

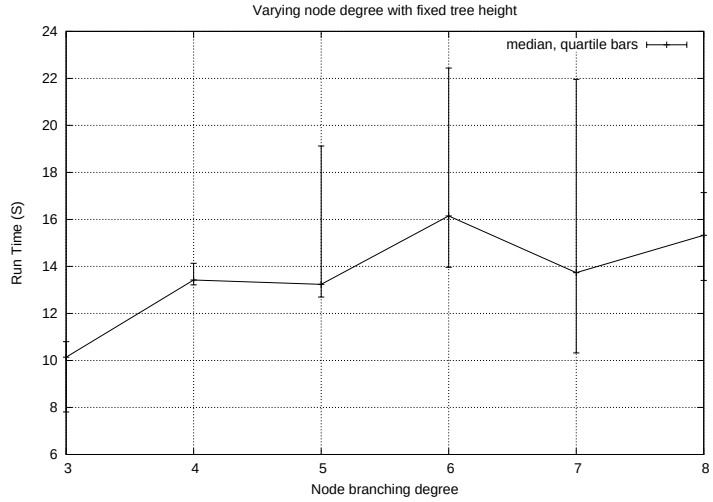
- Developed “connection-level” network simulator
  - ALPYNES - Application-Level PYthon NEtwork Simulator
- Evaluated the network delay of distributed monitoring structure.
- Results for synchronous protocol suggested development of asynchronous one.

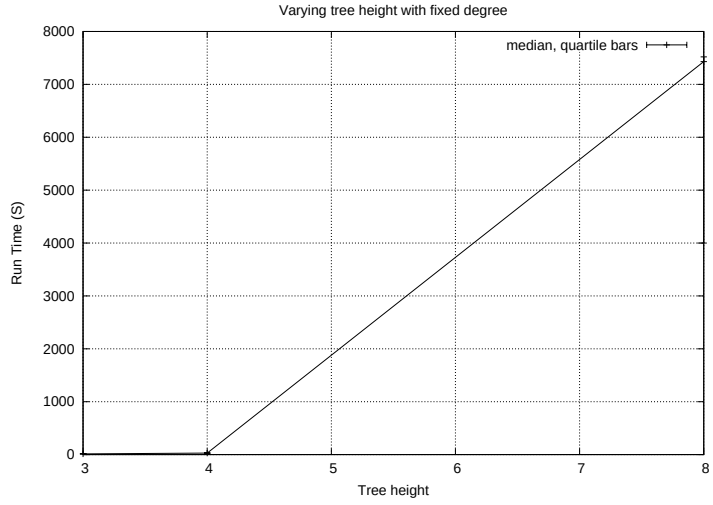
Network Model

- Network consists of e2e “links.”
- Each link represents a path in the Internet
- Path properties are generated randomly using parameters derived from empirical studies. [1, **?**]
- Paths are *domestic* or *international*
  - Domestic mean latency: 10ms, mean b/w: 0.88Mbps
  - International mean latency: 110ms, mean b/w: 0.21Mbps

Tree Structure

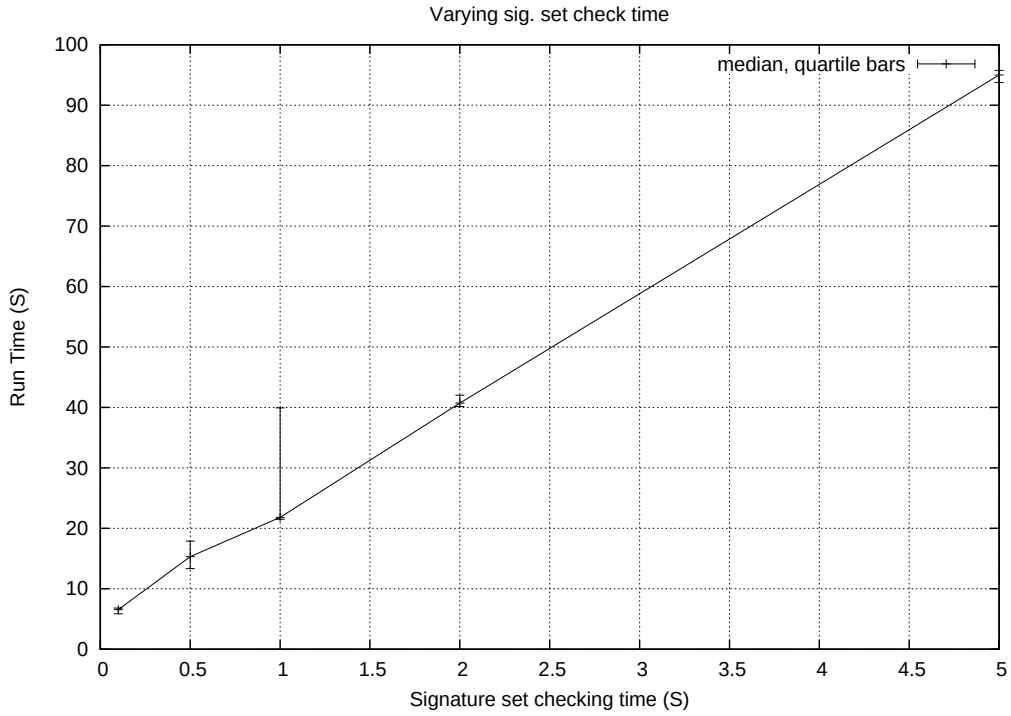
- Reasonable scaling with tree degree
- Not so good with tree height.





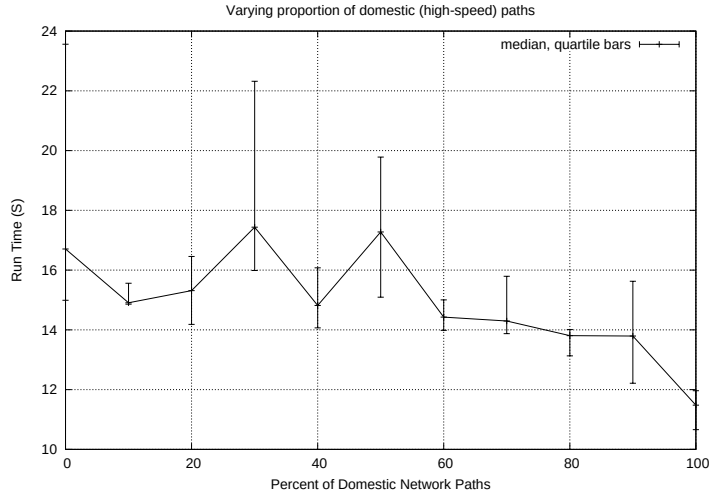
Algorithmic Performance

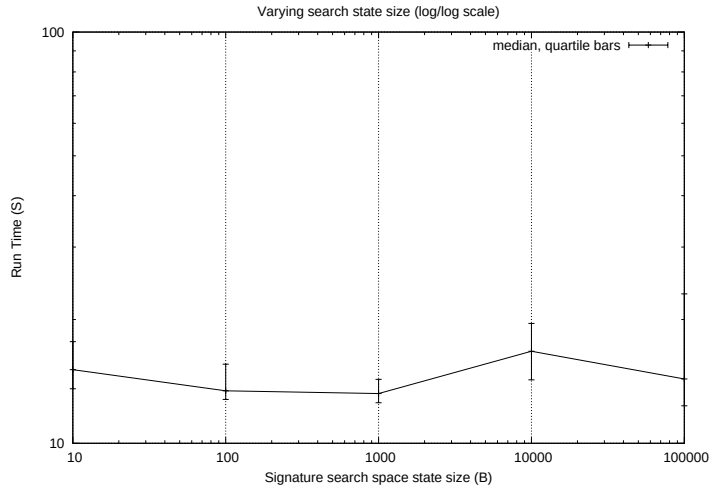
- Signature checking time is a critical bottleneck.



Message Size and Network

- Quick but modest drop-off for international links.
  - Believed to be “clocking” to slowest link.
- Robust to signature optimization state message size.





Conclusions - Tree Structure

- Completion time of well under 15 seconds is possible.
- Tree height and number of optimization rounds are critical factors.
- Message size is not.
- . . . Maximize amount of search information per message.
- Asynchronous communications protocol is probably better.

References

[1] BOLLIGER, J., GROSS, T., AND HENGARTNER, U. Bandwidth modeling for network-aware applications. In INFOCOM (3) (1999), pp. 1300–1309.

[2] STANIFORD, S., PAXSON, V., AND WEAVER, N. How to 0wn the internet in your spare time. In Proceedings of the 11th USENIX Security Symposium (Security '02) (2002).

[3] STANIFORD-CHEN, S., CHEUNG, S., CRAWFORD, R., DILGER, M., FRANK, J., HOAGLAND, J., LEVITT, K., WEE, C., YIP, R., AND ZERKLE, D. GrIDS – A graph-based intrusion detection system for large networks. In Proceedings of the 19th National Information Systems Security Conference (1996).

[4] TOTH, T., AND KRUEGEL, C. Connection-history based anomaly detection. In Proceedings of the 2002 IEEE Workshop on Information Assurance and Security (June 2002), pp. 30–25.

[5] WU, J., VANGALA, S., GAO, L., AND KWIAT, K. An effective architecture and algorithm for detecting worms with various scan techniques. In NDSS (2004).

[6] ZOU, C. C., GAO, L., GONG, W., AND TOWSLEY, D. Monitoring and early warning for internet worms. In 10th ACM

Conference on Computer and Communications Security (October 2003), pp. 190–199.