Network Verification Using Atomic Predicates

Difficulty in Managing Large Networks

- Complexity of network protocols
  - unexpected protocol interactions
  - links may be physical or virtual (e.g., point to point, Ethernet, VLAN)
  - access control list (ACL) - complex syntax, ACLs designed and configured by different people over a long period of time
  - packet transformations (e.g., NATs, MPLS and IP tunnels)
- Operator error was the largest single cause of failures - with configuration errors being the largest category of operator errors

Data Plane Verification

- How do we know packet networks are working correctly?

- A uniform model for verifying packet networks
  - Seminal framework by Xie et al. (IEEE Infocom 2005)
  - A graph where each node is a packet filter or a packet transformer

Prior Work

Two approaches:

- Reformulate the network verification problem within the context of a verification tool previously designed for another domain *(less effort but inefficient)*
  - Symbolic model checking [2009]
  - SAT/SMT solvers [2011]
  - Datalog [2015]
  - Symbolic execution [2016]
- Custom design new data structures and algorithms to directly compute *reachability trees* *(much more effort but much more efficient)*
  - Header Space Analysis/Hassel in C [2012-2013]
  - Atomic Predicates Verifier [2013]

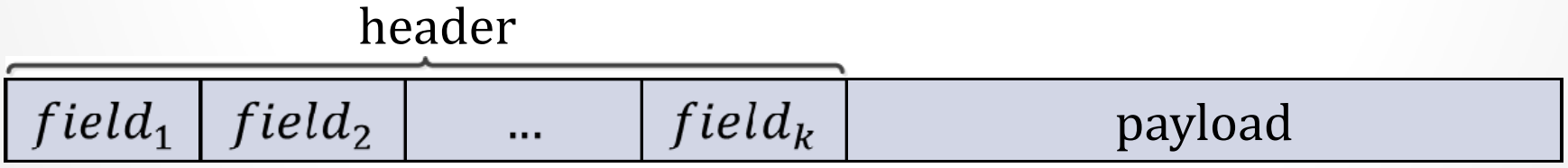
Network Reachability Properties

- Properties
  - loop-freedom (no forwarding loop for any packet)
  - reachability via waypoints (e.g. firewalls)
  - nonexistence of black holes in routers
  - network slice isolation (i.e., virtual networks)
  - . . .

- Compute packet sets that can travel from port *x* to port *y*
  - forward reachability trees rooted at a source port
  - reverse reachability trees rooted at a destination port

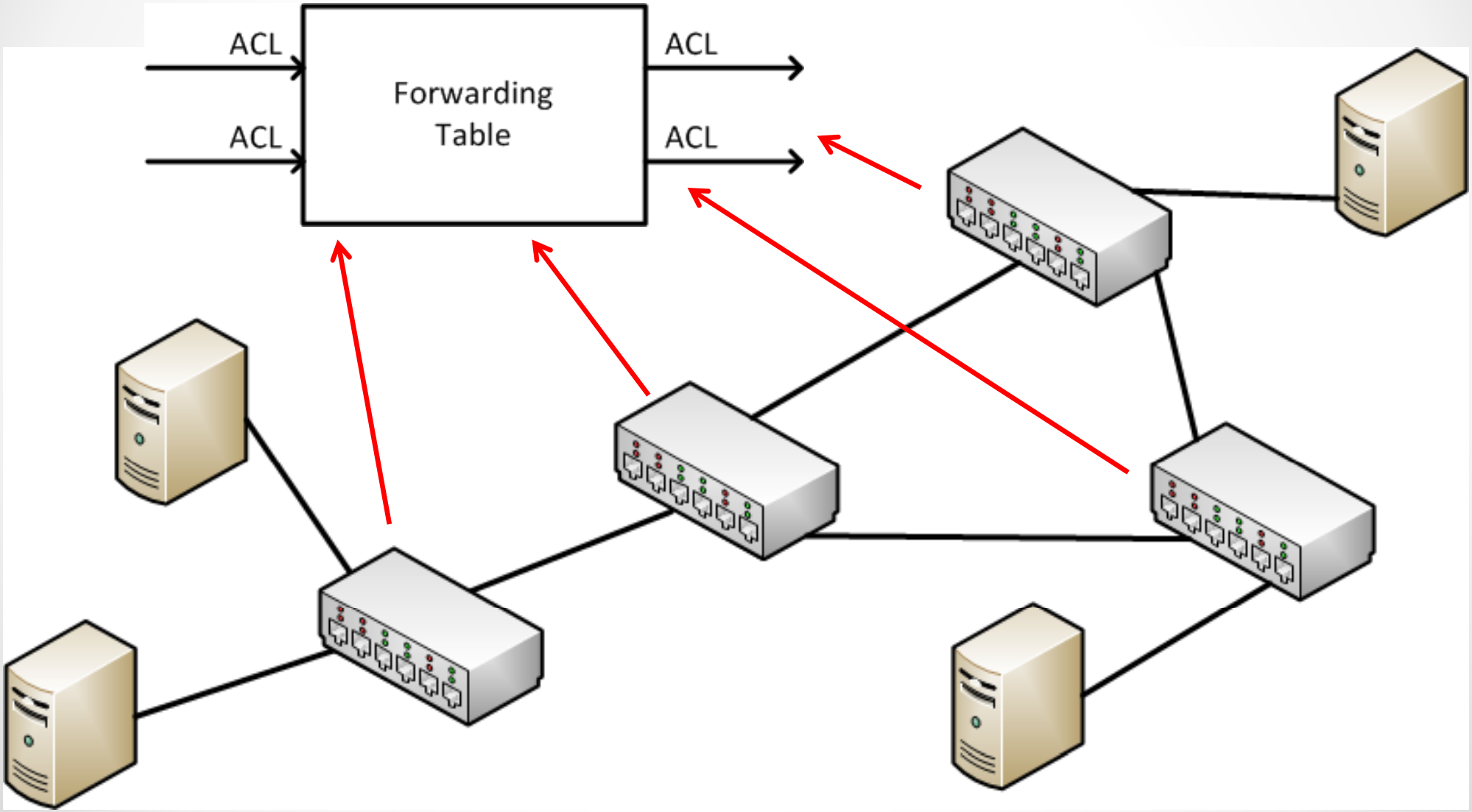
Packet

- Each packet has a header and a payload
- A packet header is partitioned into multiple fields
- Packets with identical values in their header fields are considered to be the same by packet filters



Packet Network

(assume no transformer for now)



Packet filters

- Routers/switches
  - forwarding table determines packet sets to output ports
- Access control list (ACL)
  - guard input and output ports of boxes
  - determines set of packets that can pass through
  - a firewall is an ACL with a large number of rules
- The set of packets that can travel through a sequence of packet filters can be computed by **intersection** of the packet sets that represent the filters
  - reachability set along multiple paths is the **union** of reachability sets along individual paths

Intersection and Union of Packet Sets are Computation-intensive

- Multidimensional sets
  - with many allowed intervals in each dimension and arbitrary overlaps
  - **worst case computation time of set intersection/union is $O(2^h)$**
- Efficiency of these operations determines the efficiency of reachability analysis
- The time and space performance of a network verification tool depends on
  - data structure for representing packet sets, and
  - algorithm for computing reachability sets

Box Model in AP Verifier

- Each ACL is converted to a predicate specifying the packet set allowed by the ACL
- For each output port, a predicate is computed from the forwarding table
  - specifying the packet set forwarded to the output port

Predicates represent packet sets

- Each variable in a predicate represents one packet header bit
- Predicate *P* specifies the set of packets for which *P* evaluates to *true*
- In AP Verifier, predicates are implemented as binary decision diagrams (BDDs) which are rooted, directed acyclic graphs
  - intersection and union of packet sets are replaced by conjunction and disjunction of predicates
  - BDD operations are performed using highly efficient graph algorithms [R. Bryant , 1986]

BDD Representation

- Uniqueness
- Representation size for each rule

  Theorem 1. If the length of a packet header is h bits, and an ACL rule specifies each header field by an interval, a prefix or a suffix, then the number of nodes in the BDD graph representing an ACL rule is less or equal to $\mathbf{2+2h}$.

- Logical operations
  - conjunction (disjunction) requires time proportional to the product of operand sizes in the worst case; complement is easy

h is the number of header bits relevant for verification

Datasets

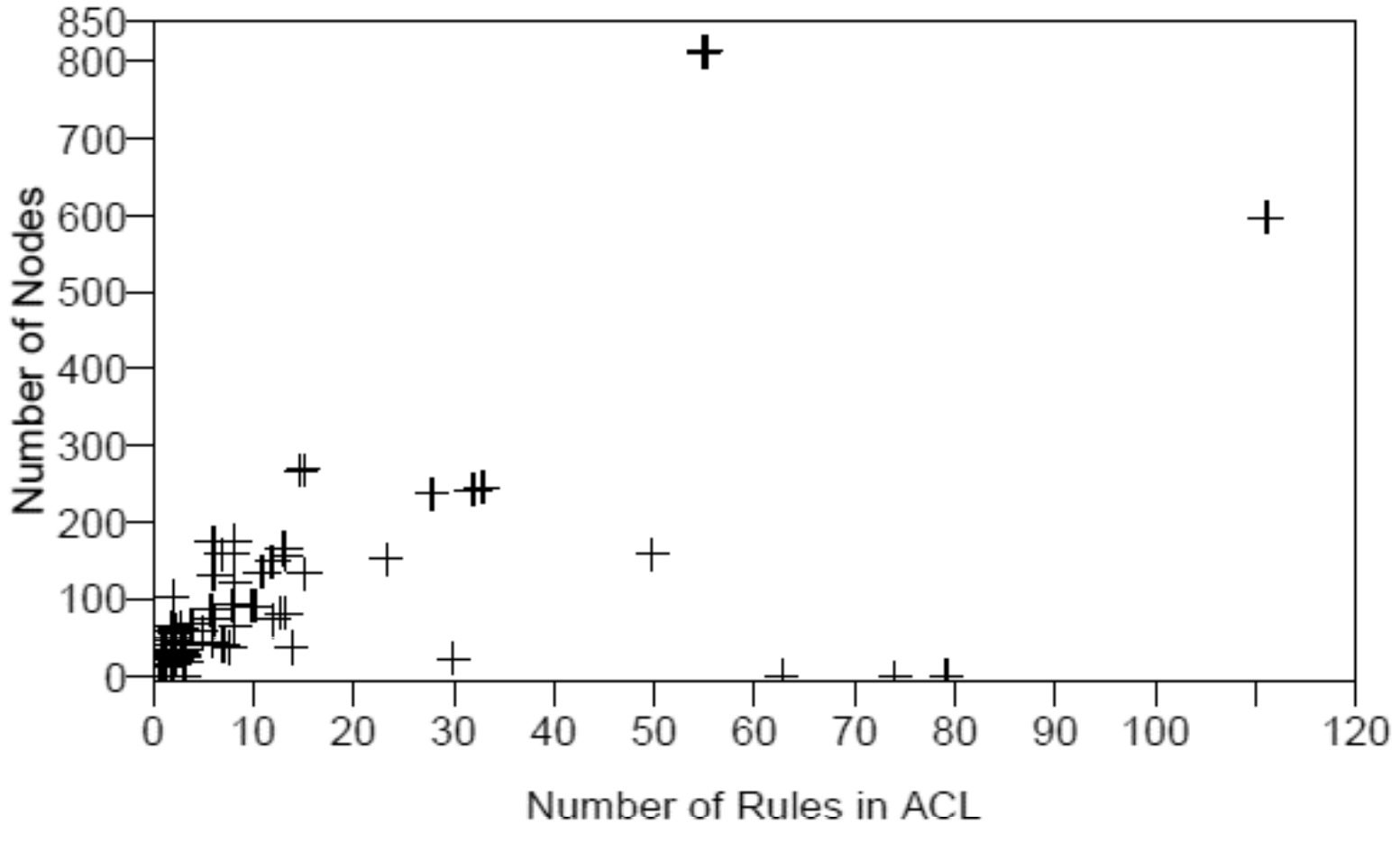
	Stanford	Internet2	Purdue
No. of boxes	16	9	1,646
No. of ports used	58	56	2,736

	Stanford		Internet2	Purdue
No. of rules	Forwarding	ACL	Forwarding	ACL
	757,170	1,584	126,017	3,605

Statistics of three real networks.

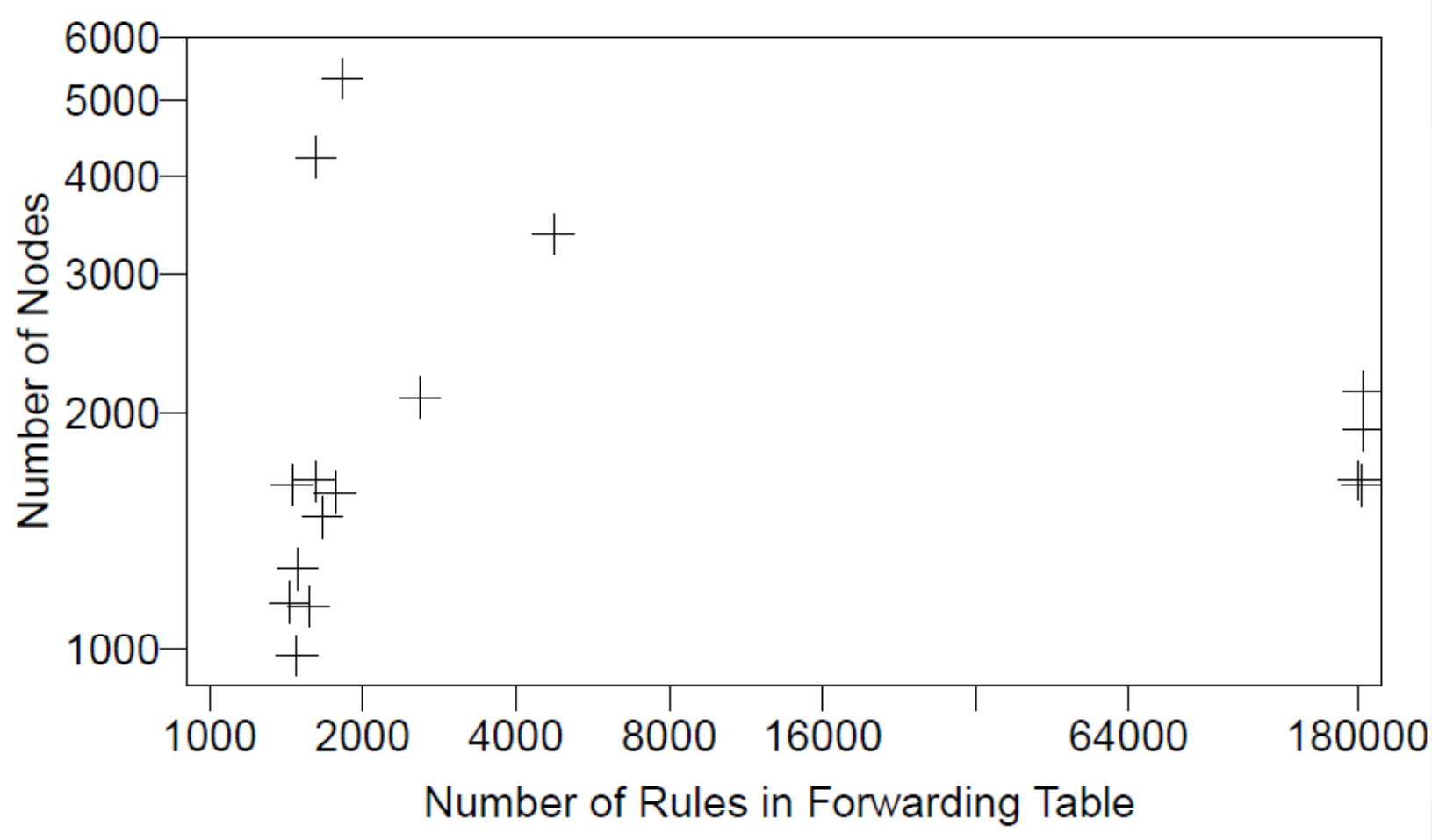
- All boxes in Stanford and Internet2 dataset are routers
- Boxes in Purdue dataset consist of routers and switches

Representation Size - ACL



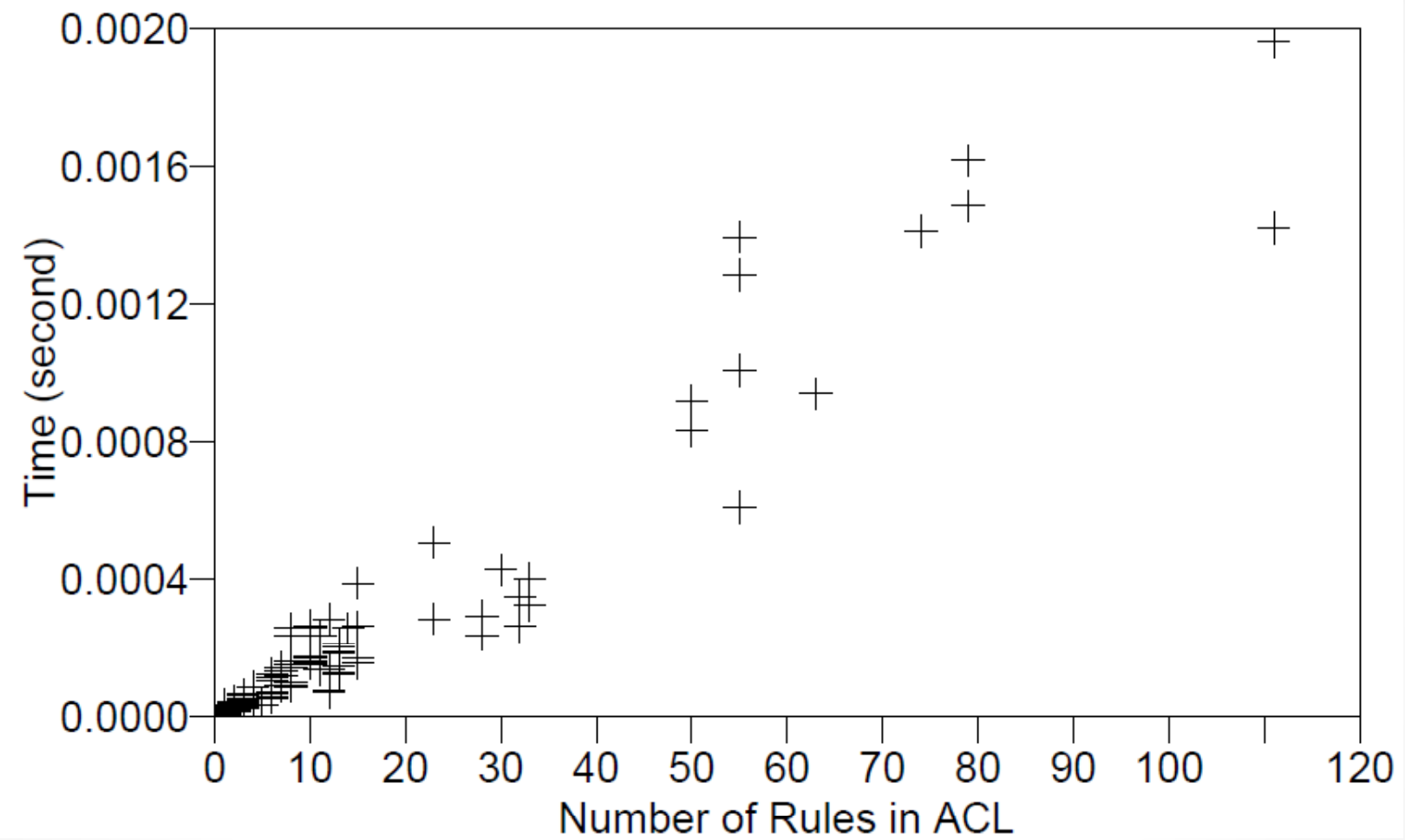
Stanford network.

Representation Size – Table



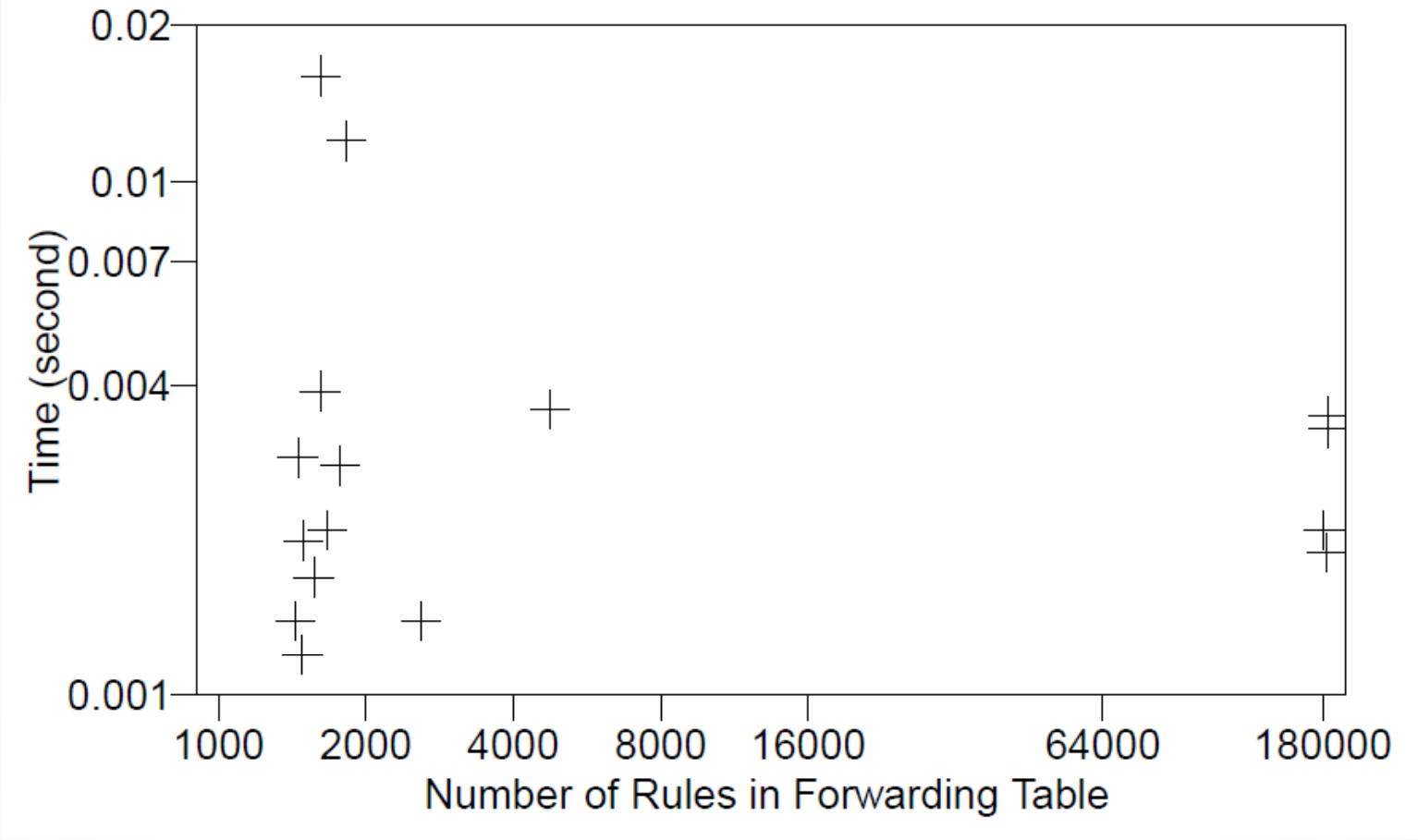
Stanford network.

Computation Times



Time to compute predicate for an ACL in Stanford network.

Computation Times



Time to compute all predicates of a forwarding table in Stanford network.

Observations

- Increasing the number of rules in an ACL or a forwarding table does not always mean more BDD nodes
- Computing BDDs for ACLs and for forwarding tables is fast
  - in milliseconds for each ACL or table

Atomic Predicates - Definition

Given a set $\mathcal{P}$ of predicates, its set $\{p_1, \ldots, p_k\}$ of atomic predicates satisfies five properties

1. $p_i \neq false, \forall i \in \{1, \ldots, k\}$
2. $\bigvee p_i = true$
3. $p_i \wedge p_j = false$, for $i \neq j$
4. Each predicate $P \in \mathcal{P}, P \neq false$, is equal to the disjunction of a subset of atomic predicates:

$$P = \bigvee_{i \in S(P)} p_i, \text{ where } S(P) \subseteq \{1, \ldots, k\}$$

5. k is the minimum number such that the set $\{p_1, \ldots, p_k\}$ satisfies the above four properties

Meaning of Atomic Predicates

- Given a set $\mathcal{P}$ of predicates, there are numerous sets of predicates that satisfy the first four properties
  - interested in the set with the **smallest** number of predicates*
- An equivalence class C is a packet set
  - pkt_1 and pkt_2 both $\in C$ if and only if each predicate in $\mathcal{P}$ evaluates to the same value for both packets
- The meaning of atomic predicates

Theorem 2. For a given set $\mathcal{P}$ of predicates, the atomic predicates for $\mathcal{P}$ specify the equivalence classes in the set of all packets with respect to $\mathcal{P}$.

*Note: The equivalence classes specified by atomic predicates are the coarsest equivalence classes.

Computing Atomic Predicates

- Compute the set of atomic predicates for predicate P:

$$\mathcal{A}(\{P\}) = \begin{cases} \{true\} & \text{if } P = false \text{ or } true \\ \{P, \neg P\} & \text{otherwise} \end{cases}$$

- $\mathcal{P}_1, \mathcal{P}_2$ two sets of predicates. $\mathcal{P}_1$'s set of atomic predicates is $\{b_1, \cdots, b_l\}$ and $\mathcal{P}_2$'s set of atomic predicates is $\{d_1, \ldots, d_m\}$. Compute a set of predicates, $\{a_1, \ldots, a_k\}$:

$$\{a_i = b_{i_1} \wedge d_{i_2} | a_i \neq false, i_1 \in \{1, \ldots, l\}, i_2 \in \{1, \ldots, m\}\}$$

Theorem 3. The set of atomic predicates for $\mathcal{P}_1 \cup \mathcal{P}_2$ is $\{a_1, \ldots, a_k\}$ where, for $i \in \{1, \ldots, k\}$, a_i is computed by the above formula.

In the worst case, the above set $\{a_i\}$ can have ℓm predicates; in practice most of them are false

Atomic Predicates in Real Networks

- Datasets of three real networks from Stanford University, Purdue University, and Internet2
- Compute separate sets of atomic predicates for ACLs and forwarding tables

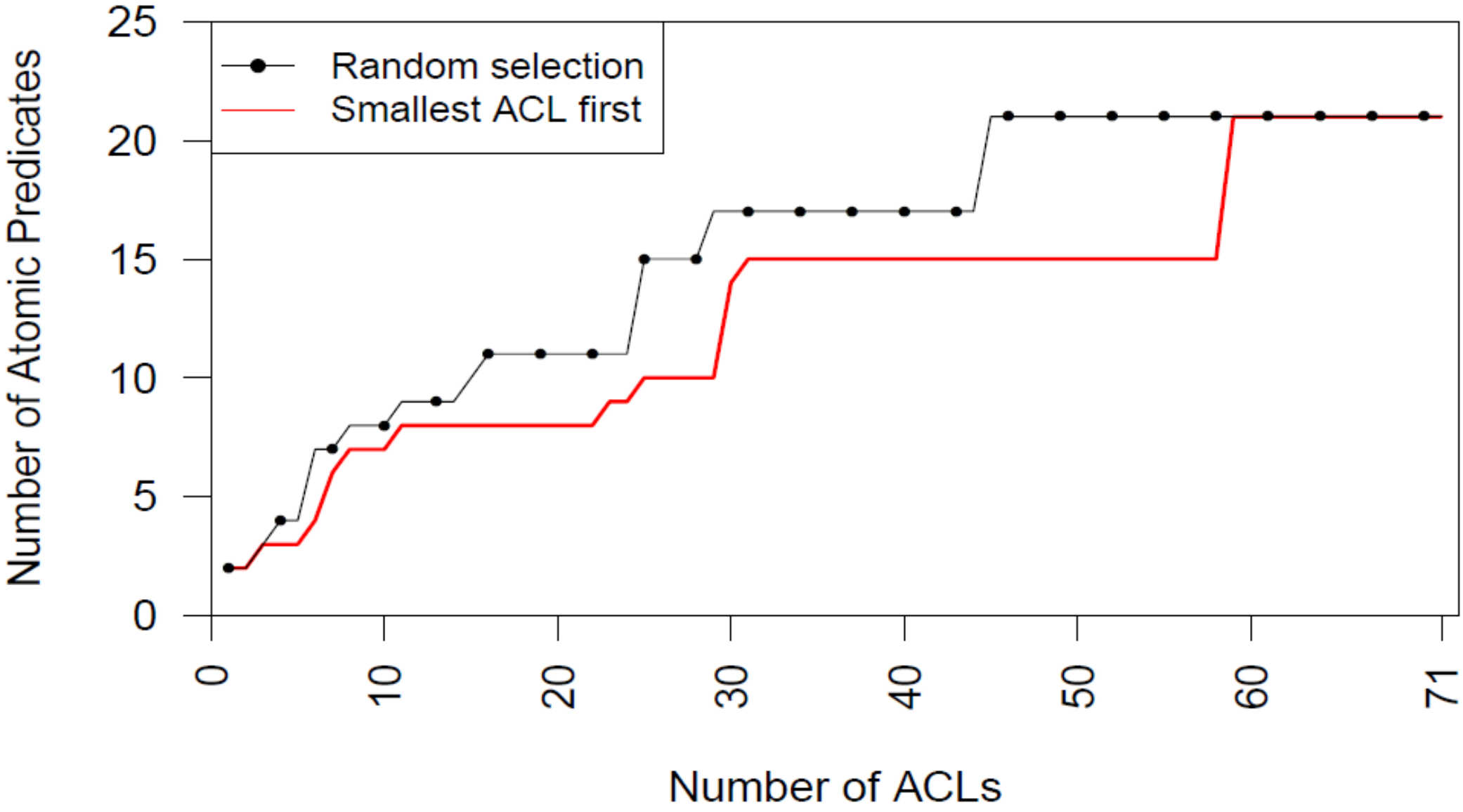
	Stanford		Internet2	Purdue
No. of rules	Forwarding	ACL	Forwarding	ACL
	757,170	1,584	126,017	3,605
No. of atomic predicates	494	21	216	3,917

Time to Compute Atomic Predicates

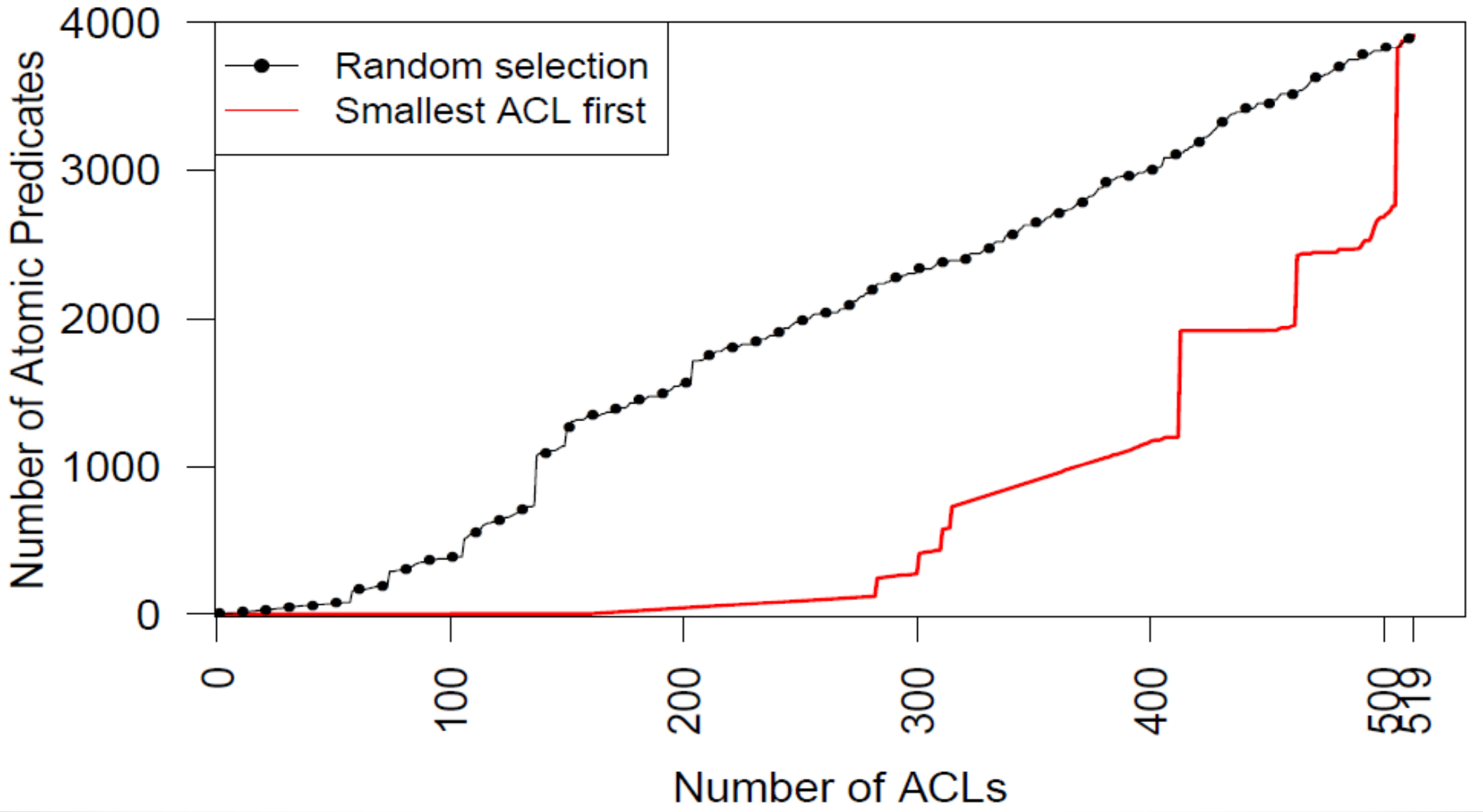
atomic predicates for ACLs		
	random selection (ms)	*smallest ACL first* (ms)
Stanford	1.56	0.84
Purdue	886.21	450.31

atomic predicates for forwarding		
	random selection (ms)	*selection by box* (ms)
Stanford	210.26	201.40
Internet2	154.91	148.28

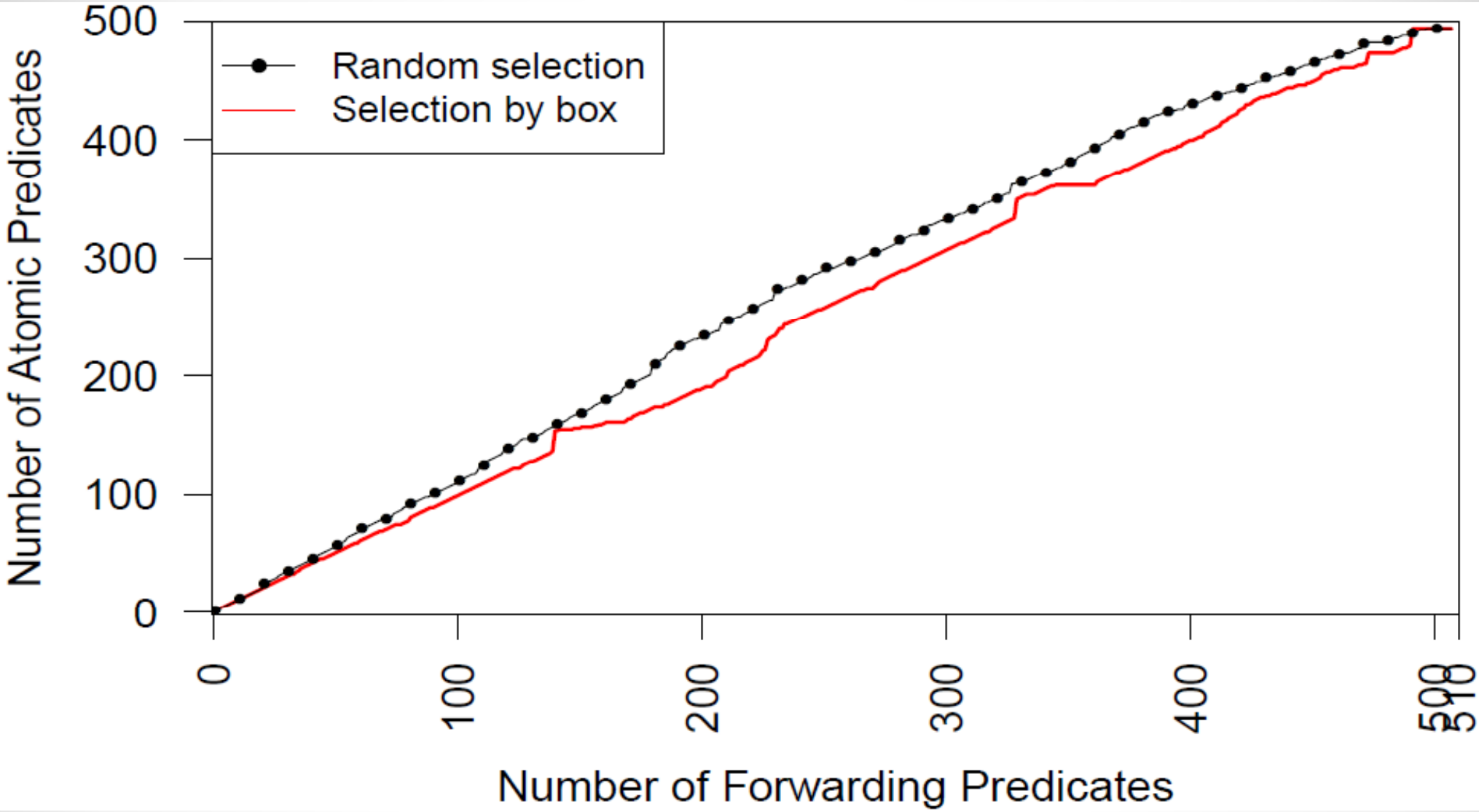
- Order of predicates affects computation time
- Computed in a fraction of second



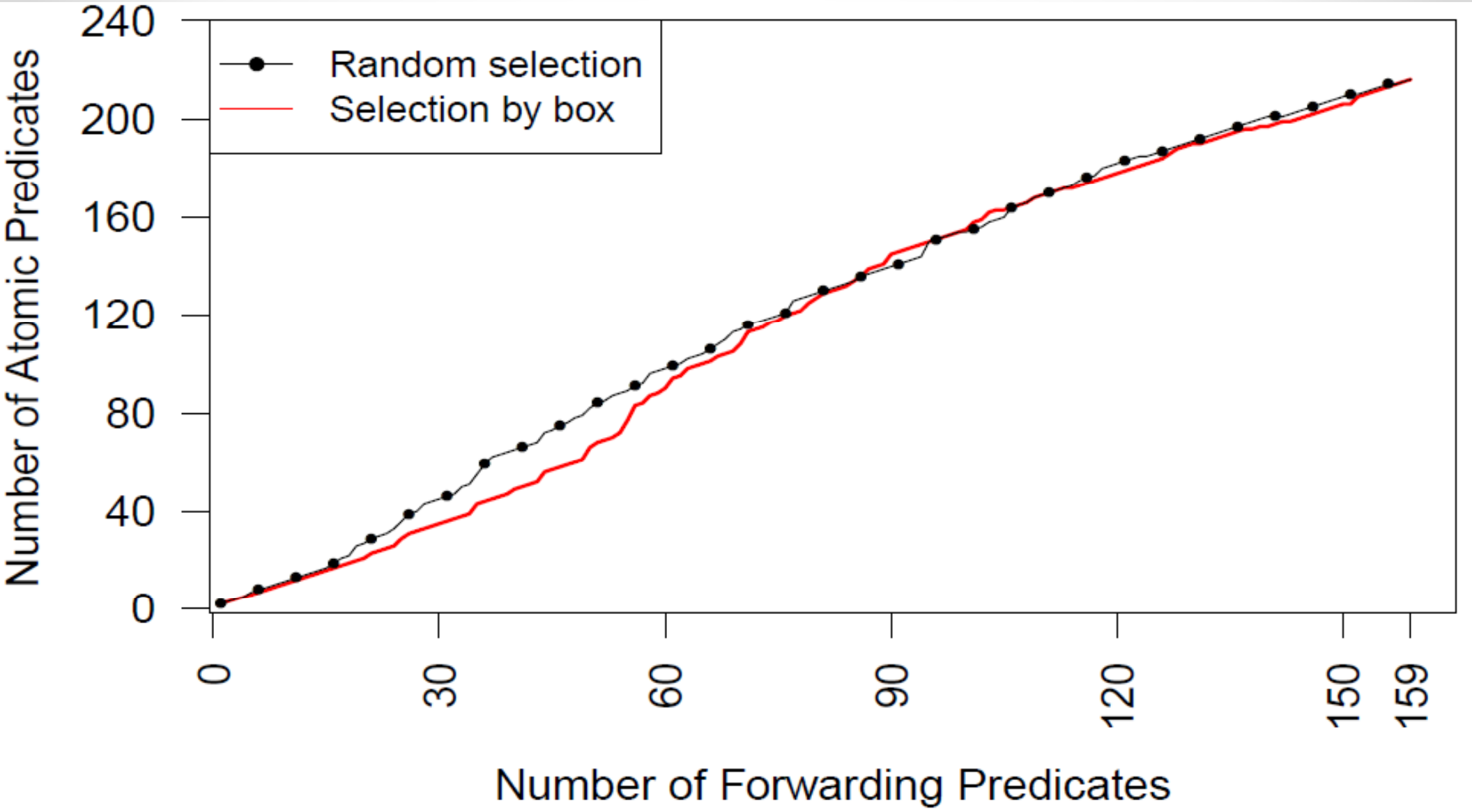
Number of atomic predicates for ACLs in Stanford network.



Number of atomic predicates for ACLs in Purdue network.



Number of atomic predicates for forwarding in Stanford network.



Number of atomic predicates for forwarding in Internet2.

Packet Set Specification

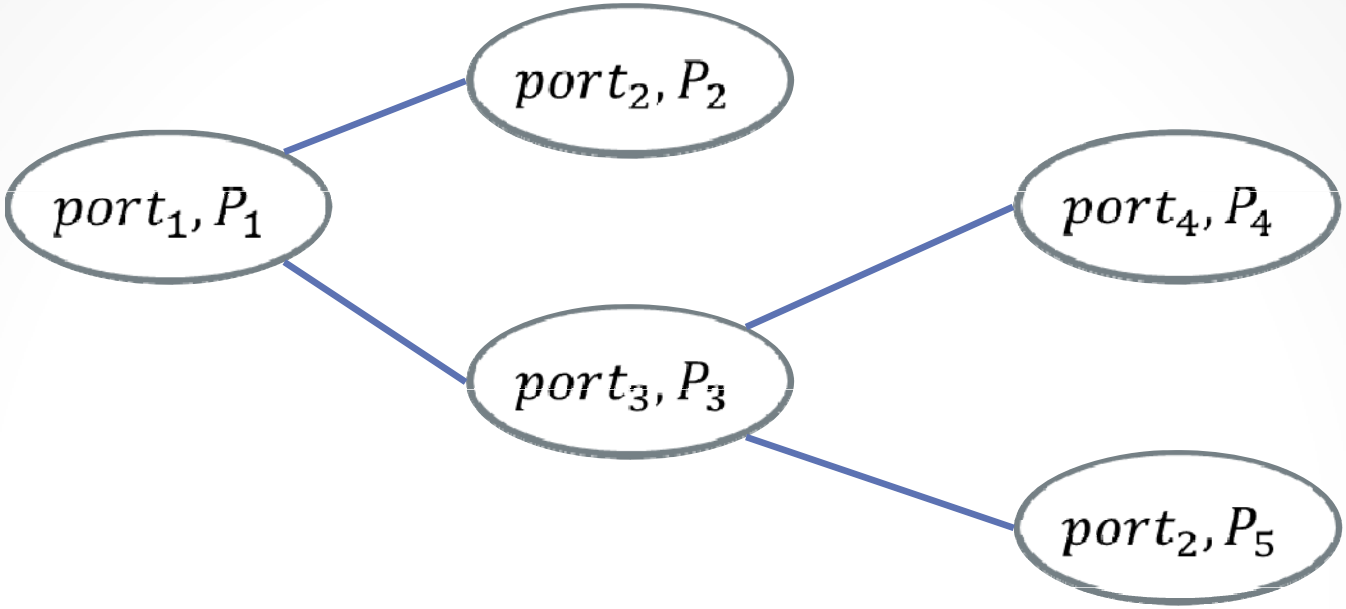
- The set of packets P that can pass through an output port is specified by the conjunction of its predicates for forwarding and ACLs
  - represented by two sets of identifiers of atomic predicates

- P is specified by

$$P = \left(\bigvee_{i \in S_F} f_i\right) \wedge \left(\bigvee_{j \in S_A} a_j\right)$$

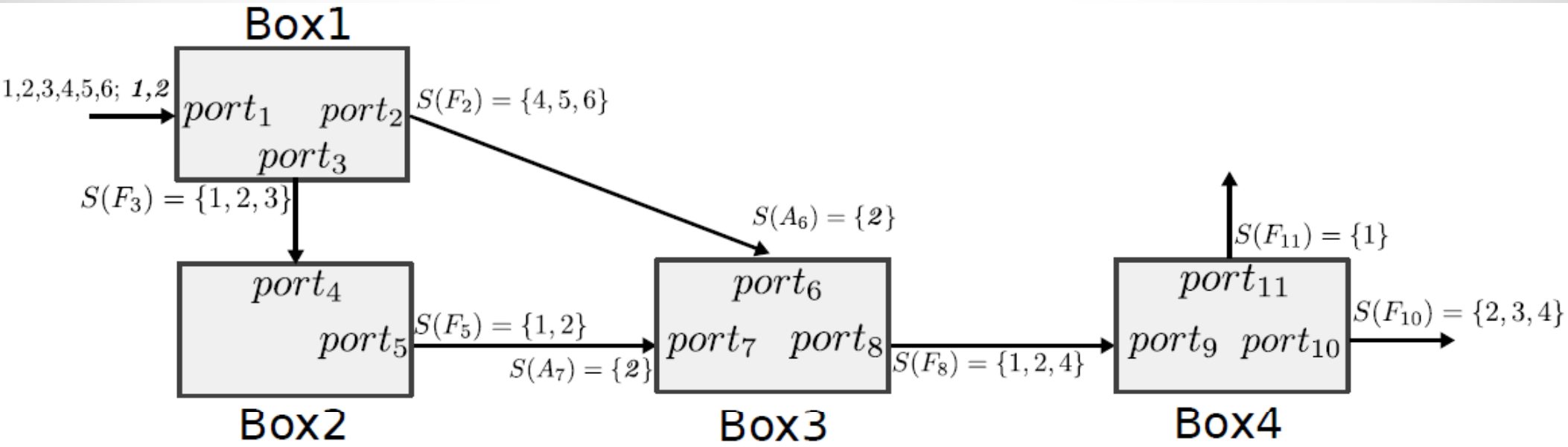
where S_F is the set of integer identifiers of atomic predicates for forwarding, S_A is the set of integer identifiers of atomic predicates for ACLs.

Reachability Tree

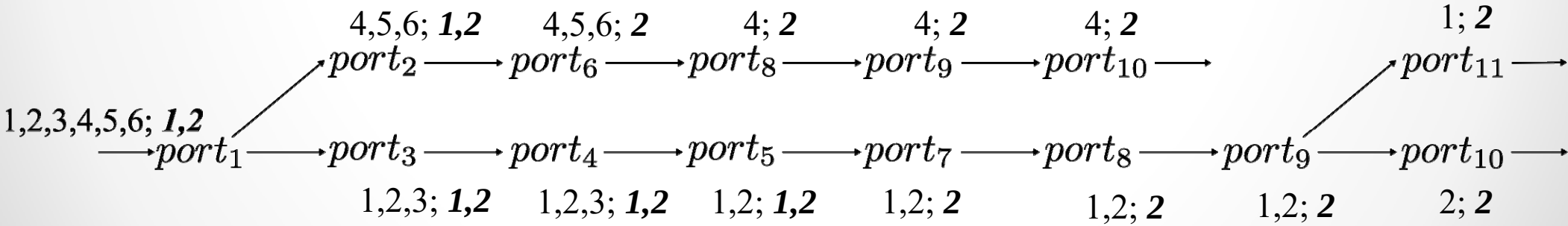


- Reachability tree consists of every path along which a nonempty set of packets can travel from source port to another port in the network
- Each node stores a port number and the set of packets that can reach the port from the source
  - The packet set of is represented by identifiers of atomic predicates
  - The same port may appear in multiple paths of the tree

A Reachability Tree Example



A small network example.



The reachability tree of $port_1$.

Storage Cost of Reachability Trees

	Size (MB)
Hassel in C	323.06
AP Verifier	8.70

Stanford network (58 ports).

	Size (MB)
Hassel in C	187.60
AP Verifier	6.72

Internet2 (56 ports).

- Storing reachability trees for all ports
  - Hassel in C required 37 times more memory for the Stanford network and 28 times more memory for Internet2
- Storing intermediate data
  - maximum memory was over 400 MB for Hassel in C and was less than 1 MB for AP Verifier

Loop Detection by Computing the Reachability Tree for One Port

	Average (ms)	Median (ms)	Maximum (ms)
Hassel in C	218.22	53.45	1881.41
AP Verifier	0.95	1.03	1.38
AP Verifier (BDD)	4.23	4.21	10.67

Reachability tree computation from one port (loop detection) in Stanford network.

AP Verifier is 230 times faster

	Average (ms)	Median (ms)	Maximum (ms)
Hassel in C	754.19	609.14	5873.44
AP Verifier	0.27	0.29	0.45
AP Verifier (BDD)	2.96	3.05	8.34

Reachability tree computation from one port (loop detection) in Internet2.

AP Verifier is 2793 times faster

- Twelve infinite loop paths in the Stanford network
- Two infinite loop paths in Internet2

Black Hole Detection

- A black hole in the forwarding table is a set of packets that are dropped due to no forwarding entry
- No black hole in forwarding tables of the Stanford network. Black holes in every forwarding table of Internet2
  - forwarding tables of Stanford network have default routes

	Average (ms)	Median (ms)	Maximum (ms)
AP Verifier	0.011	0.0064	0.040

Black hole detection for each forwarding table in Stanford network.

	Average (ms)	Median (ms)	Maximum (ms)
AP Verifier	0.014	0.014	0.027

Black hole detection for each forwarding table in Internet2.

Slice Isolation

- Different network slices for different customers (applications)
  - slices do not overlap
- A slice can be defined by a set of ports together with a set of packets allowed in the slice
- $Slice_i$ have set T_i of ports, a set of packets represented by S_{F_i} and S_{A_i}, for i =1, 2

  If $T_1 \cap T_2 = \emptyset$ or $S_{F_1} \cap S_{F_2} = \emptyset$ or $S_{A_1} \cap S_{A_2} = \emptyset$ then

  return "two slices are isolated"

  else $Slice_1$ overlaps $Slice_2$ at $T_1 \cap T_2$

Required Waypoints

- From a source port *s* to a set of destination ports
  - traverses the reachability tree from *s* to check that every path in the tree passes through an input port of the waypoint before reaching any destination port in the specified set
- From a set of source ports to a set of destination ports
- All packets from port *s* pass through any member of a set of waypoints or several waypoints in a specified sequence

Benefits of Atomic Predicates

- Atomic predicates for a given set of predicates
  - They specify the (*coarsest) equivalence classes of packets*
  - Observation: An atomic predicate represents a very large number of equivalent packets in numerous "fragments" of the packet space
- Each predicate stored and represented as a set of integers
  - space efficient
- **Conjunction (disjunction)** of two predicates computed as **intersection (union)** of two sets of integers
  - time efficient
- Automated tool based upon a formal method

Networks with Packet Transformers

(a very short introduction)

Let U denote the set of all elements.[4] Without qualification, an element x is always in set U, and a set of elements is always a subset of U. A predicate specifies a set of elements in U. Predicate $true$ specifies U. Predicate $false$ specifies the empty set.

The indicator function for a set D of elements and an element x is defined as follows:

$$I_D(x) = \begin{cases} 1 & x \in D, \\ 0 & x \notin D. \end{cases}$$

Transformers

Let $\mathcal{T}$ denote a set of transformers. A transformer $T \in \mathcal{T}$ maps an element from its *domain* to a set of elements in its *range*. Both the domain and the range of T are subsets of U.

For a transformer T, and an element x in the domain of T, $T(x)$ denotes the set of elements after transformation. For a set D of elements, we define

$$T(D) = \bigcup_{x \in D} T(x) \tag{8}$$

Assumption. For each transformer T, its inverse T^{-1} is a function from the range of T to the domain of T.

For an element $x \in U$, $T^{-1}(x)$ is *undefined* if x is not in the range of T.

Packet Equivalence

Definition 3 (*Equivalence w.r.t. $\mathcal{P}$ and $\mathcal{T}$*). *Given a set $\mathcal{P}$ of predicates and a set $\mathcal{T}$ of transformers. Let $\{C_1, C_2, \ldots, C_n\}$ denote equivalence classes specified by the atomic predicates for $\mathcal{P}$.*

Two elements x_1, x_2 in set U are equivalent w.r.t. $\mathcal{P}$ and $\mathcal{T}$ if and only if the following two conditions hold:

1) $I_{C_i}(x_1) = I_{C_i}(x_2)$ *for each* $i \in \{1, \ldots, n\}$.
2) *Either both* $T_{\alpha_k}^{-1} \cdots T_{\alpha_1}^{-1}(x_1)$ *and* $T_{\alpha_k}^{-1} \cdots T_{\alpha_1}^{-1}(x_2)$ *are undefined, or*

$$I_{C_i}(T_{\alpha_k}^{-1} \cdots T_{\alpha_1}^{-1}(x_1)) = I_{C_i}(T_{\alpha_k}^{-1} \cdots T_{\alpha_1}^{-1}(x_2)) \quad (9)$$

for each $i \in \{1, \ldots, n\}$, any positive integer k and any possible sequence $T_{\alpha_k} \ldots T_{\alpha_1}$ of transformers, $T_{\alpha_j} \in \mathcal{T}, j \in \{1, \ldots, k\}$.

Algorithm

Notation. Given any set, $\mathcal{Q}$, of predicates, we use $\mathcal{A}(\mathcal{Q})$ to denote the set of atomic predicates for $\mathcal{Q}$, which can be computed using one of the algorithms in [26], [28].

Algorithm 1 for computing atomic predicates after adding transformers

Input: a set $\mathcal{P}$ of predicates, a set $\mathcal{T}$ of transformers
Output: a set $\mathcal{B} = \{b_1, b_2, \ldots, b_l\}$ of predicates

1: $\mathcal{P}' \leftarrow \mathcal{P}, \mathcal{B} \leftarrow \mathcal{A}(\mathcal{P}')$
2: Compute the following set:

$$\mathcal{R} = \{T(b_i) | \text{ for each } T \in \mathcal{T}, \text{ and for each } b_i \in \mathcal{B} \text{ that is transformed by } T\}$$

3: **if** $\mathcal{B} = \mathcal{A}(\mathcal{P}' \cup \mathcal{R})$ **then**
4: **return** $\mathcal{B}$.
5: **else**
6: $\mathcal{P}' \leftarrow \mathcal{P}' \cup \mathcal{R}$, $\mathcal{B} \leftarrow \mathcal{A}(\mathcal{P}')$
7: **goto** line 2
8: **end if**

Performance for two large networks

	provider cone 1	provider cone 2	ratio
customer net	AS 37684	AS 52941	
# routers	51	92	1.80
# duplex ports	1,048	1,920	1.83
ave. # of neighbors per router	20.5	20.9	1.02
# rules	6,958,862	11,691,232	1.68
# tier-1 ISPs	15	15	

TABLE II: Statistics of the two provider cone datasets.

	provider cone 1	provider cone 2	ratio
with 51 NATs	0.02800	0.07800	2.79
with 40 IP-in-IP	0.03420	0.06221	1.82
with 40 MPLS	0.04021	0.06542	1.63

TABLE V: Ave. time to compute reachability tree from one port (seconds).

	provider cone 1	provider cone 2	ratio
predicates, atomic predicates and 40 MPLS	355.57	739.42	2.08
one reachability tree (ave.)	2.07	4.45	2.15

TABLE VI: Memory space usage (Mbytes).

Sources:

1. Hongkun Yang and Simon S. Lam, "Real-time Verification of Network Properties Using Atomic Predicates," *Proceedings of IEEE ICNP 2013*, Göttingen, Germany, October 2013; extended version in

   IEEE/ACM Transactions on Networking, April 2016, Vol. 24, No. 2, pages 887-900.

2. Hongkun Yang and Simon S. Lam, "Scalable Verification of Networks with Packet Transformers using Atomic Predicates," The University of Texas at Austin, Department of Computer Science. Report# TR-16-12 (regular tech report). August 16, 2016.

References:

1. P. Kazemian, G. Varghese, and N. McKeown, "Header Space Analysis: Static Checking for Networks." In *Proc. of USENIX NSDI, San Jose,* California, 2012.
2. Header Space Library and NetPlumber. In *https://bitbucket.org/peymank/hassel-public/ .*

Summary

- Definition of packet equivalence for packet networks with filters and transformers
- Definition of atomic predicates which specify the (coarsest) equivalence classes of packets
- Algorithm to compute atomic predicates
- Algorithm to compute reachability tree from a port to all other ports in a network
- By representing a very large set of equivalent packets by a single integer, the use of atomic predicates reduces the computation time and space by orders of magnitude
- Verification tools (AP Verifier and APT) designed to recover quickly from network changes including link/box status change, addition/removal or a NAT or tunnel, and rule updates

The end