Model-agnostic and efficient exploration of numerical state space of real-world TCP congestion control implementations

Wei Sun (UNL)

Lisong Xu (UNL)

Sebastian Elbuam (Virginia)

Di Zhao (UNL)

UNIVERSITY OF NEBRASKA–LINCOLN

Example: check whether CUBIC is sometimes too aggressive

create a network with a single link
- speed = 1Kbps to 1Gbps
- loss = 0.00001% to 10%

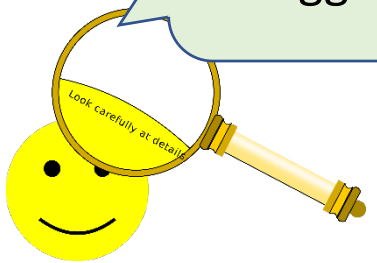
- Impractical to check each of 10^{12} possible combinations
- Propose *automated state space exploration* in order to efficiently solve this type of TCP testing problems

1. write testing script

test CUBIC in the created network, and keep track of CUBIC state variables
- cwnd : congestion window size
- target : expected cwnd after one RTT

2. run script

- check whether target > 2*cwnd
- If so, too aggressive (more aggressive than slow start)



3. check result

Concepts of parameter space P and state space S

create a network with a single link
- speed = 1Kbps to 1Gbps
- loss = 0.00001% to 10%

network environment **parameter space**
P = { (speed, loss) } for this example

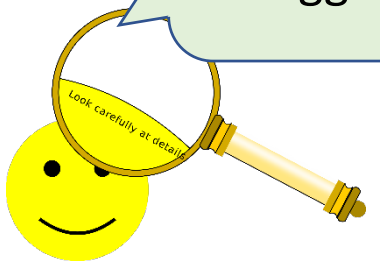
1. write testing script

test CUBIC in the created network, and keep track of CUBIC state variables
- cwnd : congestion window size
- target : expected cwnd after one RTT

congestion control **state space**
S = { (cwnd, target) } for this example

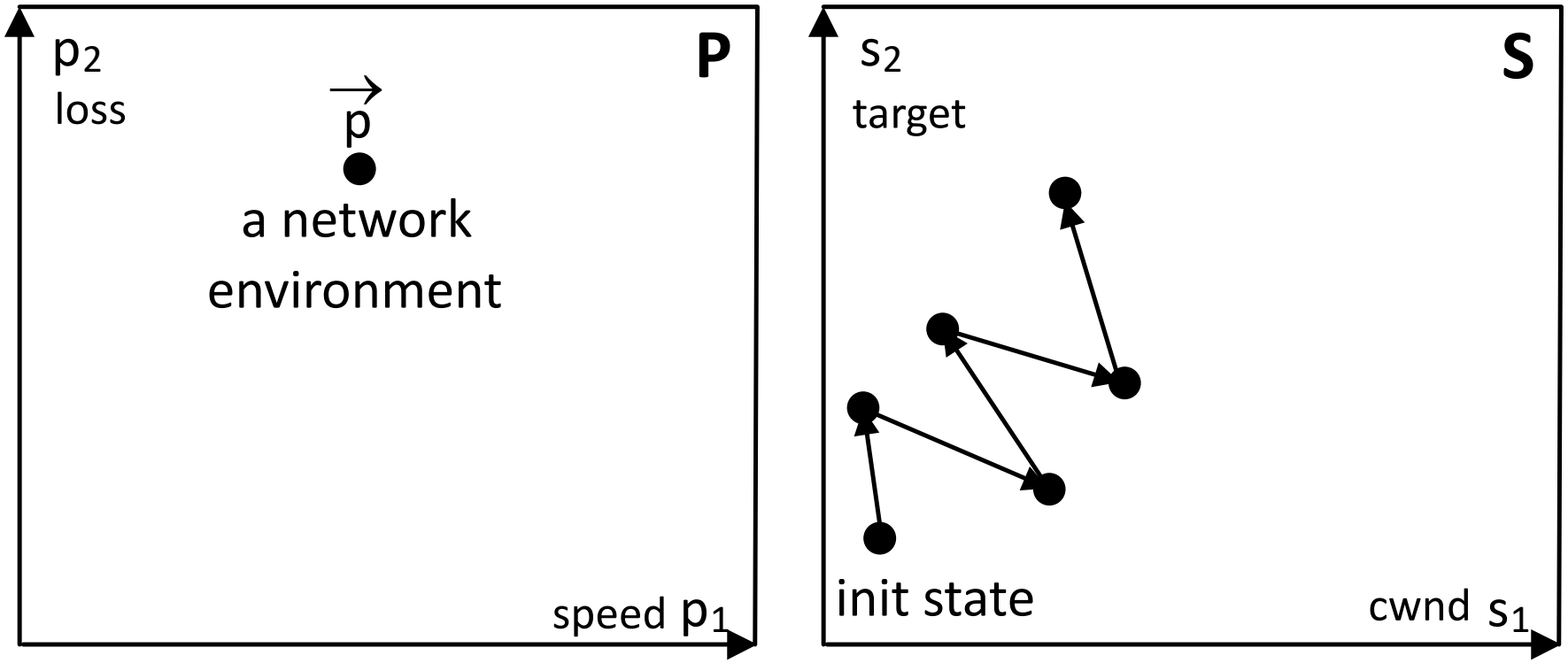
2. run script

- check whether target > 2*cwnd
- If so, too aggressive (more aggressive than slow start)



3. check result

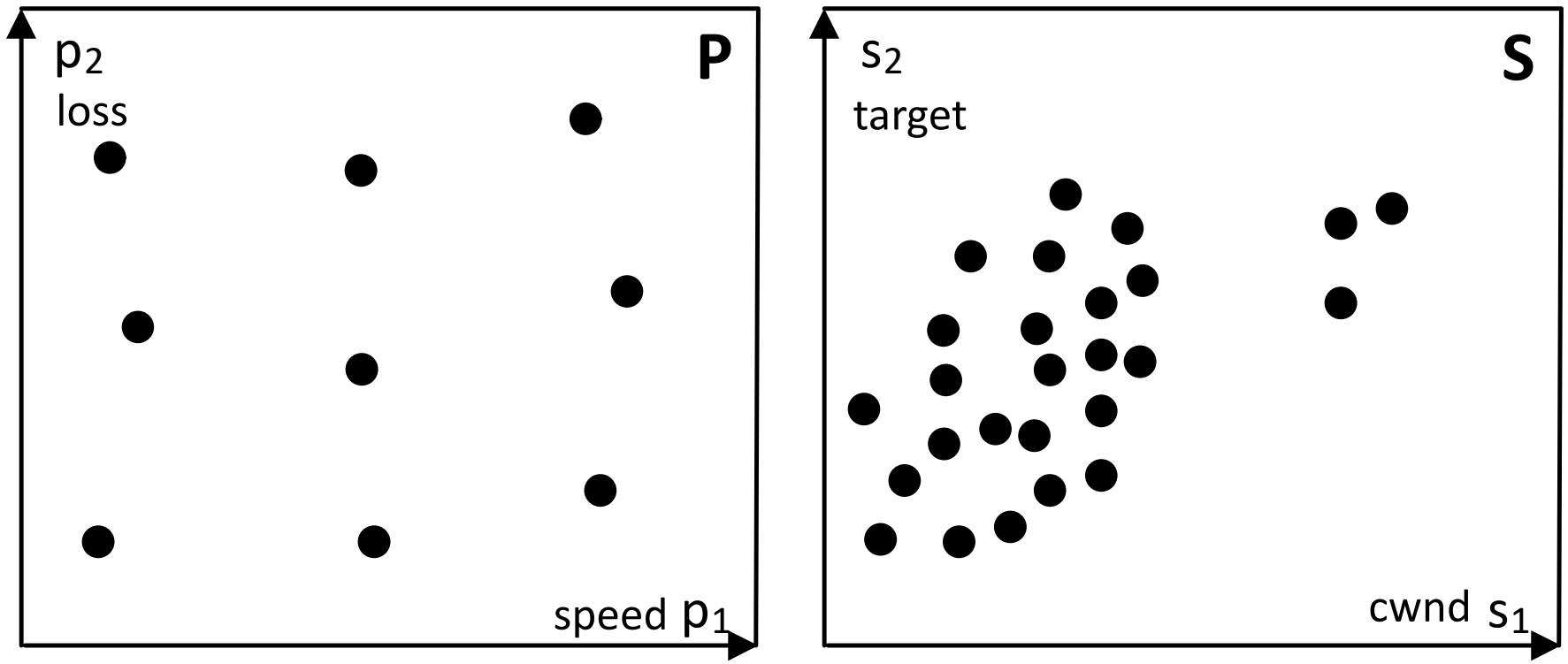
Parameter space P and state space S



Each network environment in P leads CUBIC to visit a sequence of states in S.

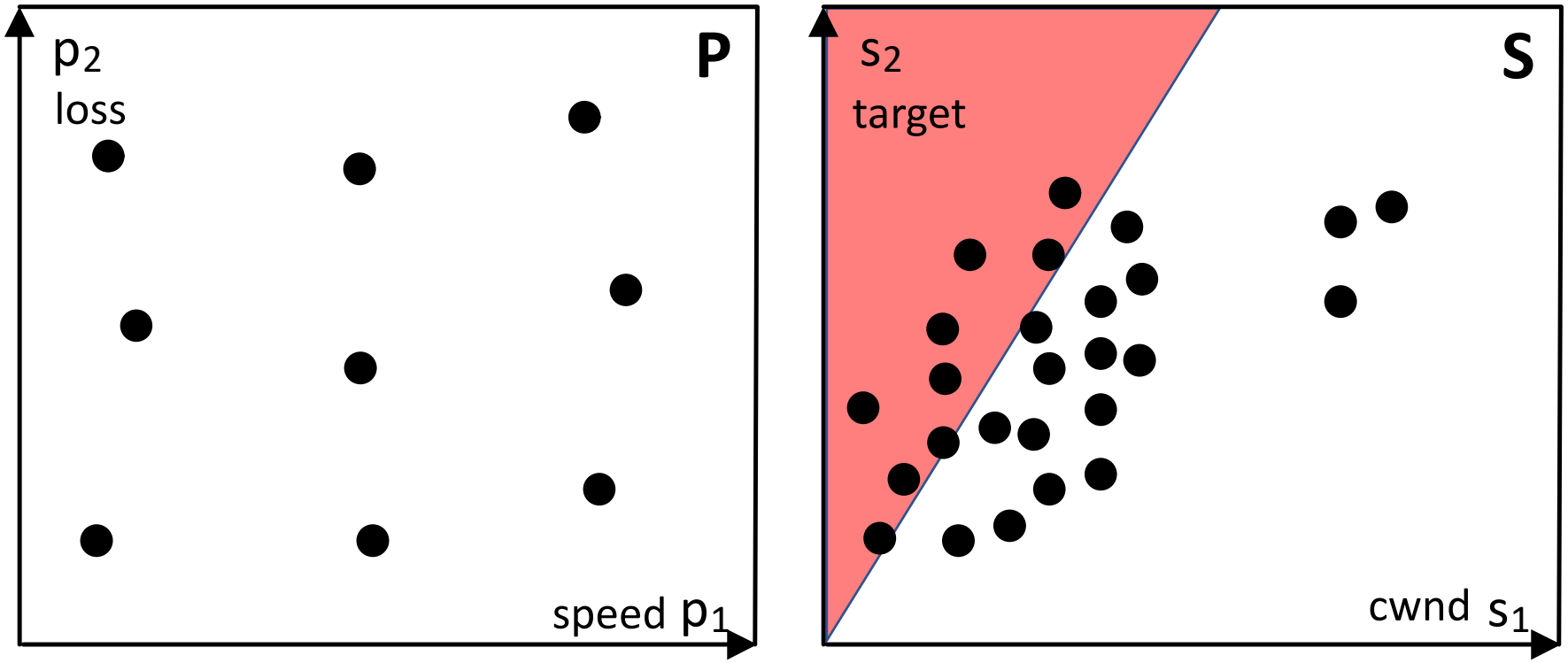
(the sequence of states are for illustration purpose only)

Parameter space P and state space S



As we choose more network environments in P, more states in S will be visited.

Example: check whether CUBIC is sometimes too aggressive



- In order to solve this type of testing problems, we can just find all possible regions of S that can be visited by a congestion control algorithm.
- Red region (target>2*cwnd) is the region where CUBIC is too aggressive. If the red region can be visited by CUBIC, then CUBIC is sometimes too aggressive.

Automated state space exploration

What is automated state space exploration?

- Given
  - a parameter space P for a testing script
  - a state space S for a congestion control algorithm,
- how to automatically choose network environments in P in order to explore as many different regions of S as possible?

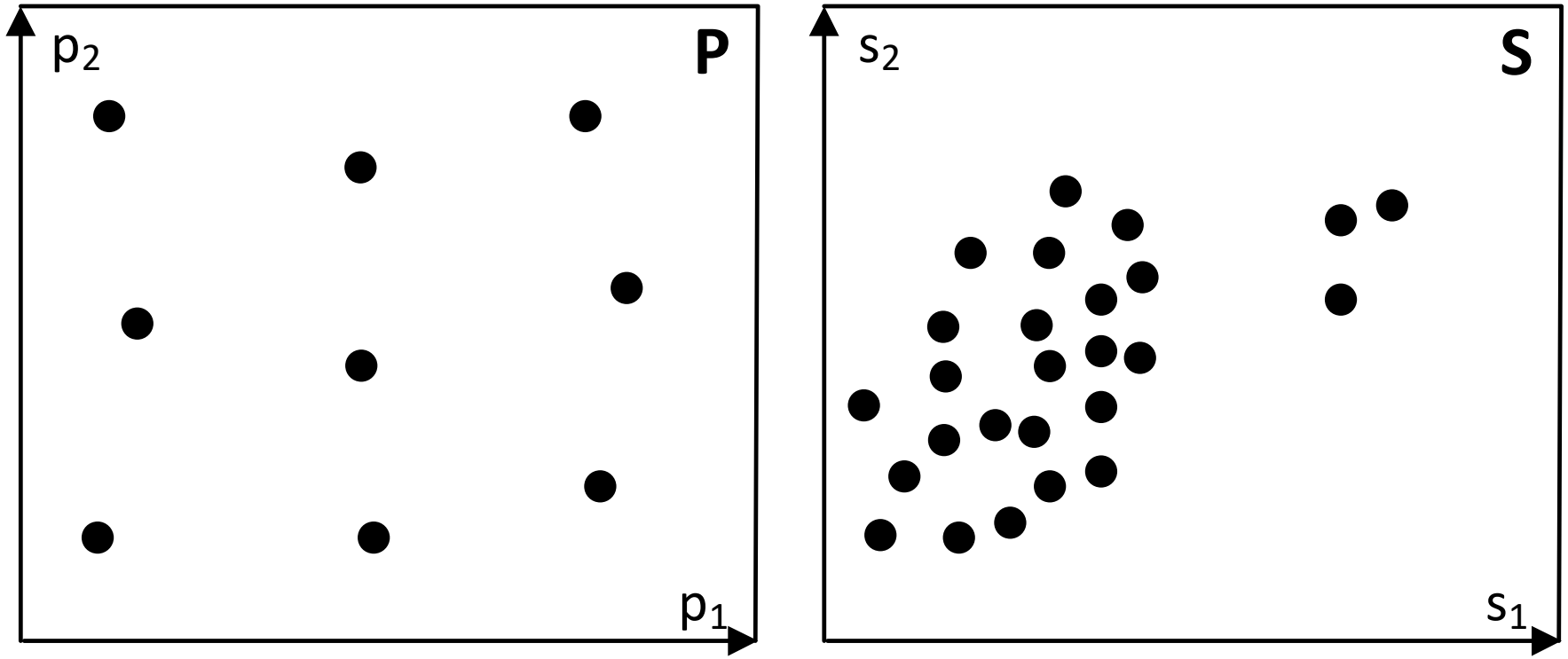
Why it is useful?

- It can be used to test whether a congestion control algorithm has wrong or inappropriate behaviors by just checking whether the corresponding regions of S can be visited.

Our method: Automated Congestion control Testing (ACT)

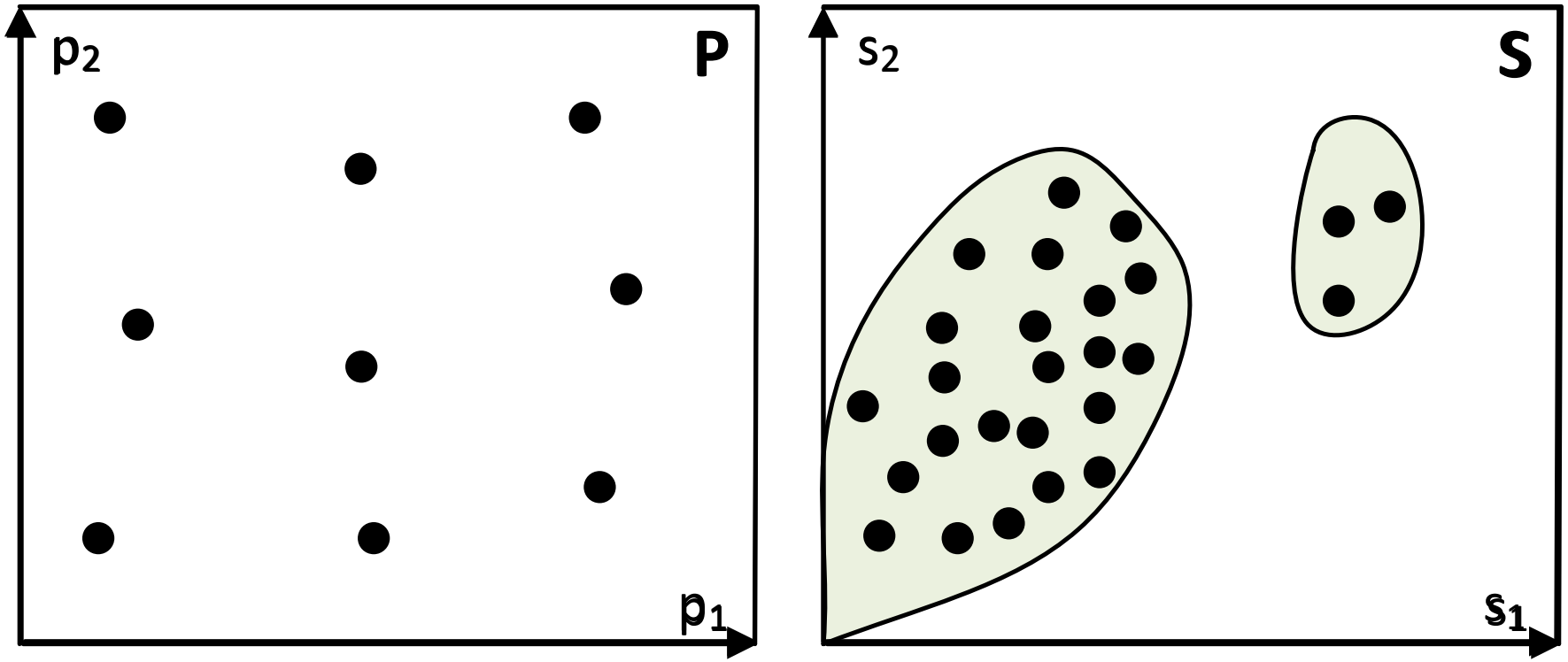
- Goal: Explore as many different regions of S as possible, instead of concentrating on some regions
- Feedback-guided random testing
  - Scalable to large P: ACT randomly selects network environments in P
  - Efficiently explore S: The random selection is guided by the feedback
  - Model-agnostic: Feedback is obtained from previous state coverage information, and does not require abstract models of P and S
- ACT steps
  - ① Random testing
  - ② Parameter estimation
  - ③ Parameter concatenation

ACT ① Random testing, ②Parameter estimation, ③Parameter concatenation



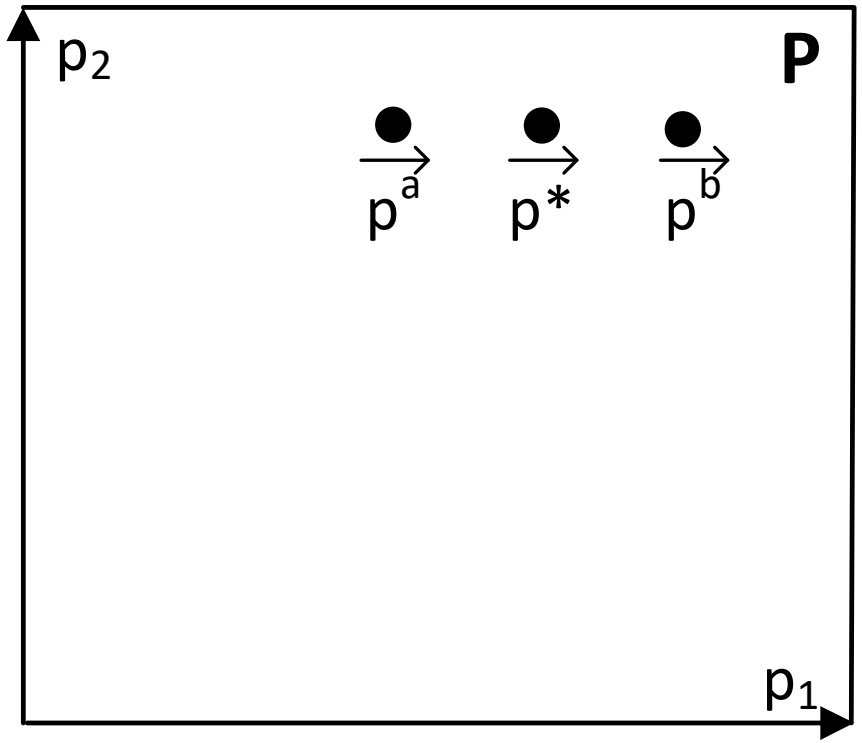
- Uniformly randomly select network environments in P to have an initial coverage of S

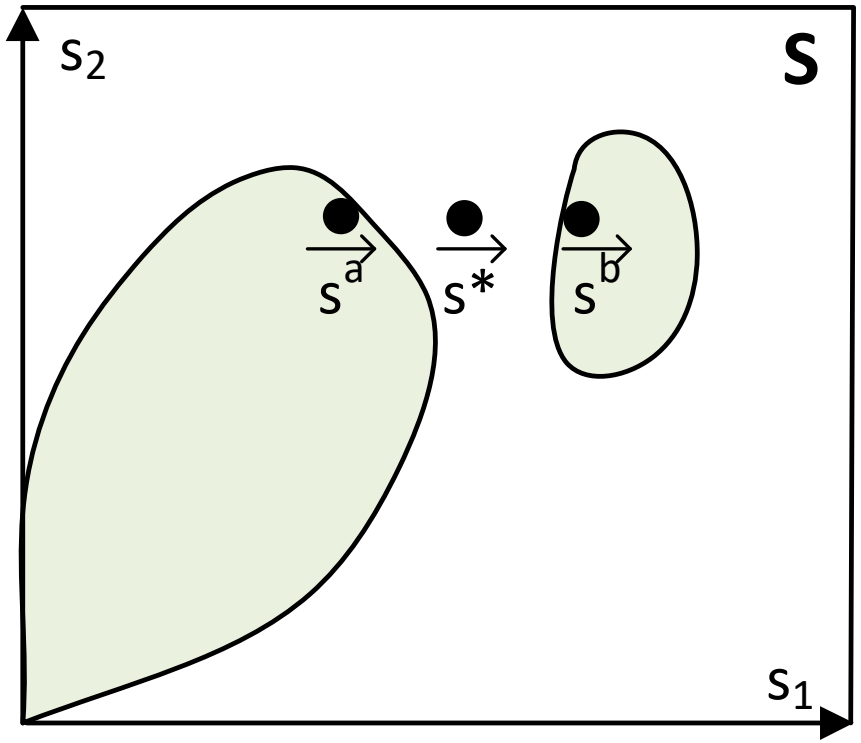
ACT ① Random testing, ②Parameter estimation, ③Parameter concatenation



- Uniform selection in P ≠ uniform coverage of S, due to non-linear mapping
- Use parameter estimation to explore the unvisited gaps and corners.

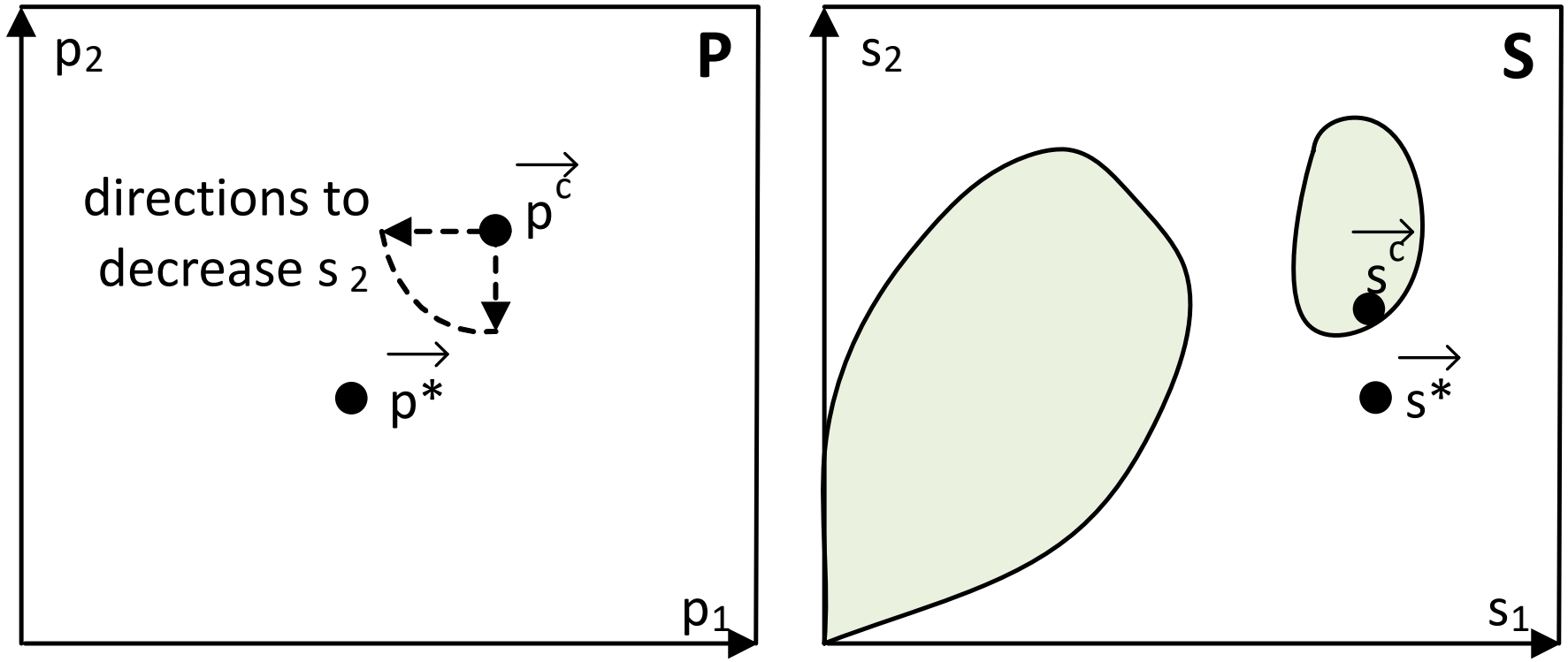
ACT ① Random testing, ②Parameter estimation, ③Parameter concatenation





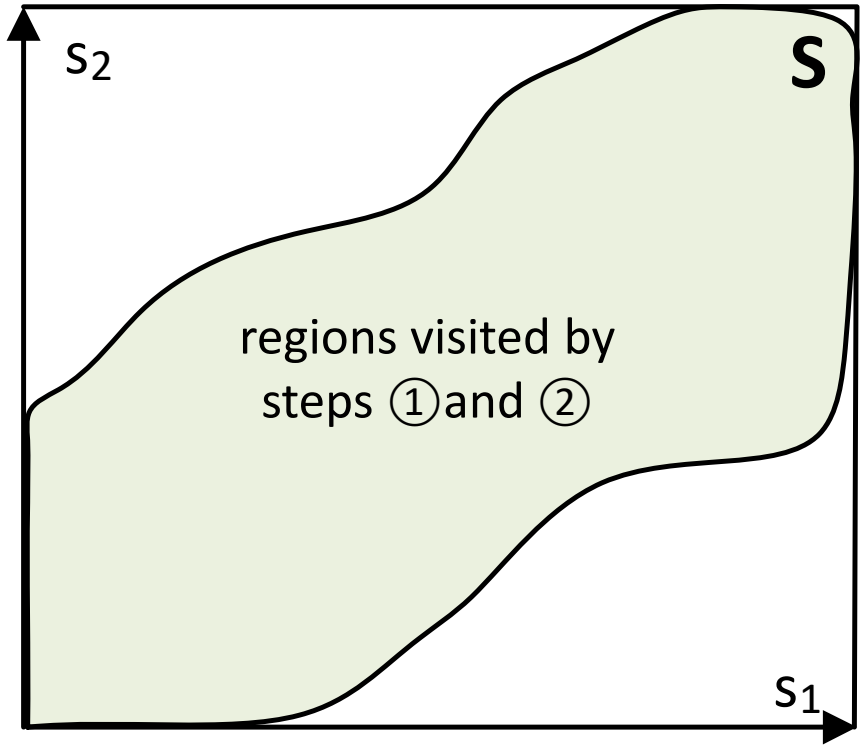
- Use random interpolation to visit the gap between two visited regions

ACT ① Random testing, ②Parameter estimation, ③Parameter concatenation



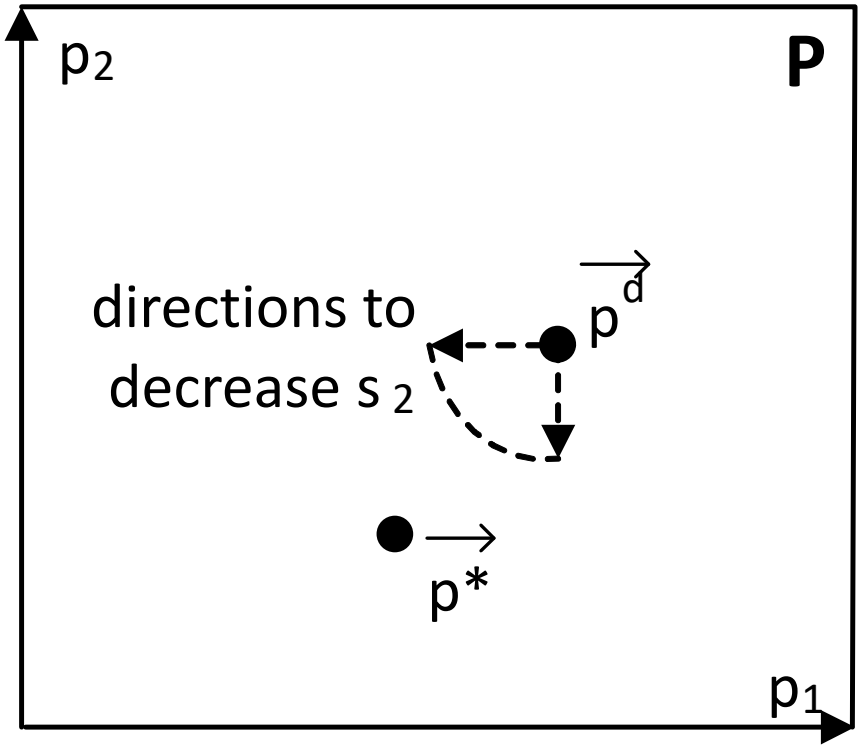
- Use random extrapolation to visit an unvisited corner or side of S
- The directions are estimated using the results of step ①.

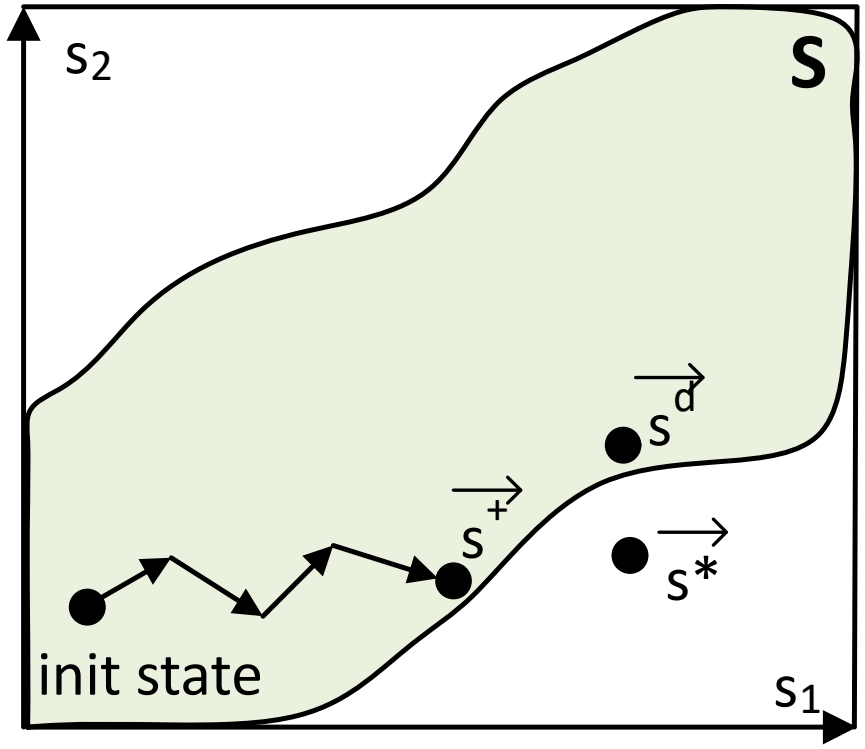
ACT ① Random testing, ②Parameter estimation, ③Parameter concatenation



- Some regions are still not visited, if state variables are correlated (e.g., cwnd and ssthresh)

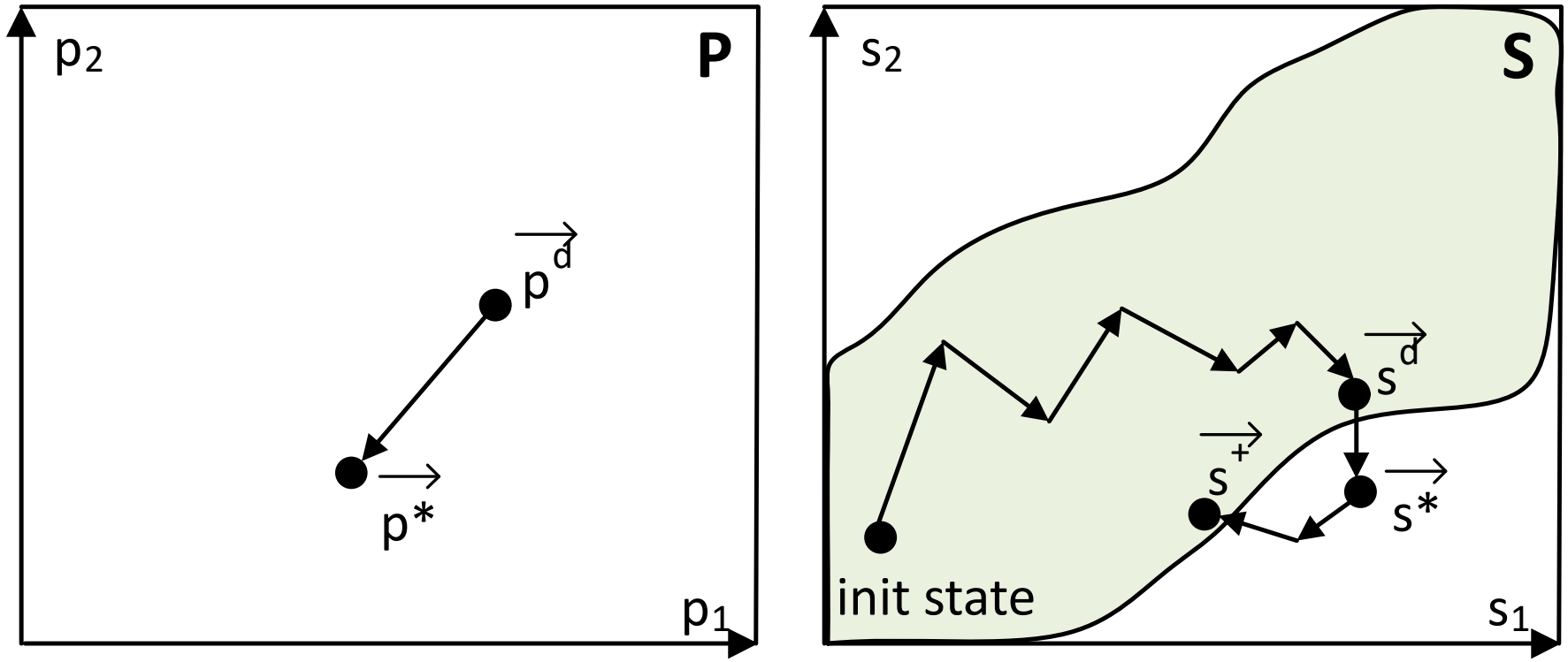
ACT ① Random testing, ②Parameter estimation, ③Parameter concatenation





- If s_1 and s_2 are positively correlated, $\overrightarrow{p^*}$ estimated by extrapolation will lead the algorithm to visit $\overrightarrow{s^+}$ that has smaller s_2 and smaller s_1.

ACT ① Random testing, ②Parameter estimation, ③Parameter concatenation



- Parameter concatenation uses both $\overrightarrow{p^d}$ and $\overrightarrow{p^*}$. That is, we change the network environment in the middle of an experiment.
- The algorithm will follow a different path to state $\overrightarrow{s^+}$, more likely to visit unvisited regions.

Experiment setup

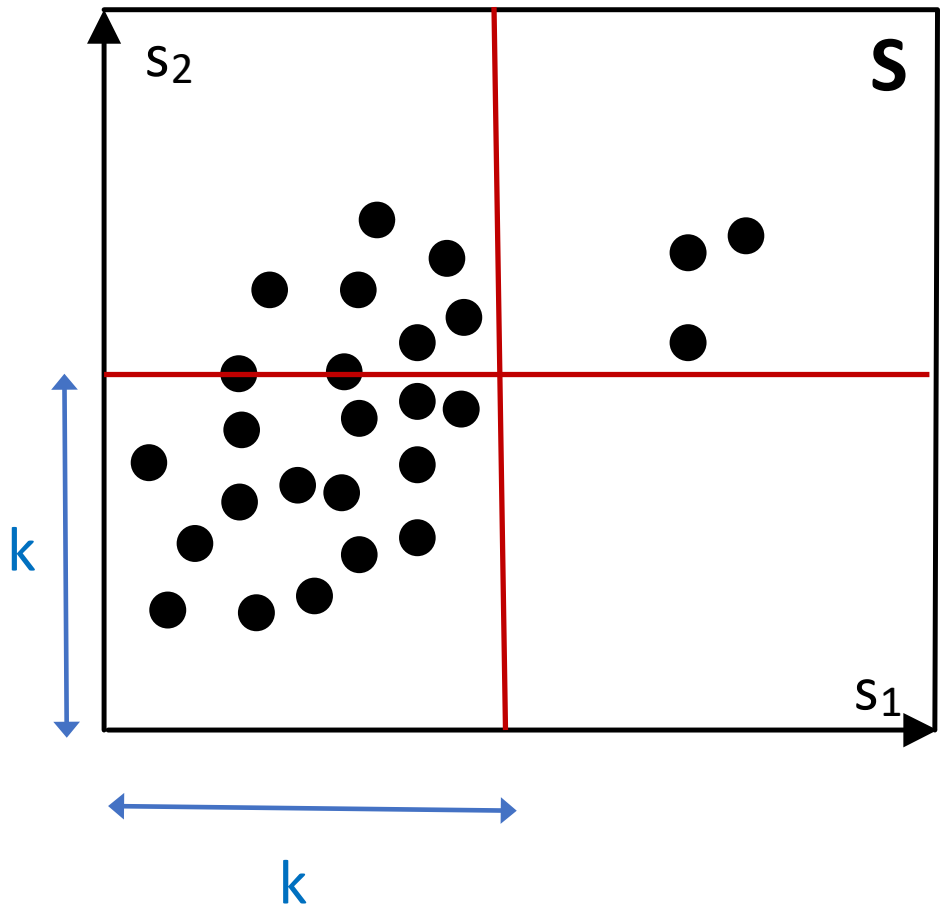
- Real-world TCP congestion control in Linux 3.10:
  - CUBIC, AIMD, HTCP, HSTCP, VENO
- Parameter space P
  - a testing script with a single-link network
  - (loss, speed, propagation delay, random queueing delay, application rate)
- State space S
  - (cwnd, ssthresh, rtt, rttvar, ca_state, …)
  - plus protocol-specific state variables
- Two types of experiments
  - state space coverage
  - bug detection

State space coverage experiments

Measuring state space coverage by four different testing methods

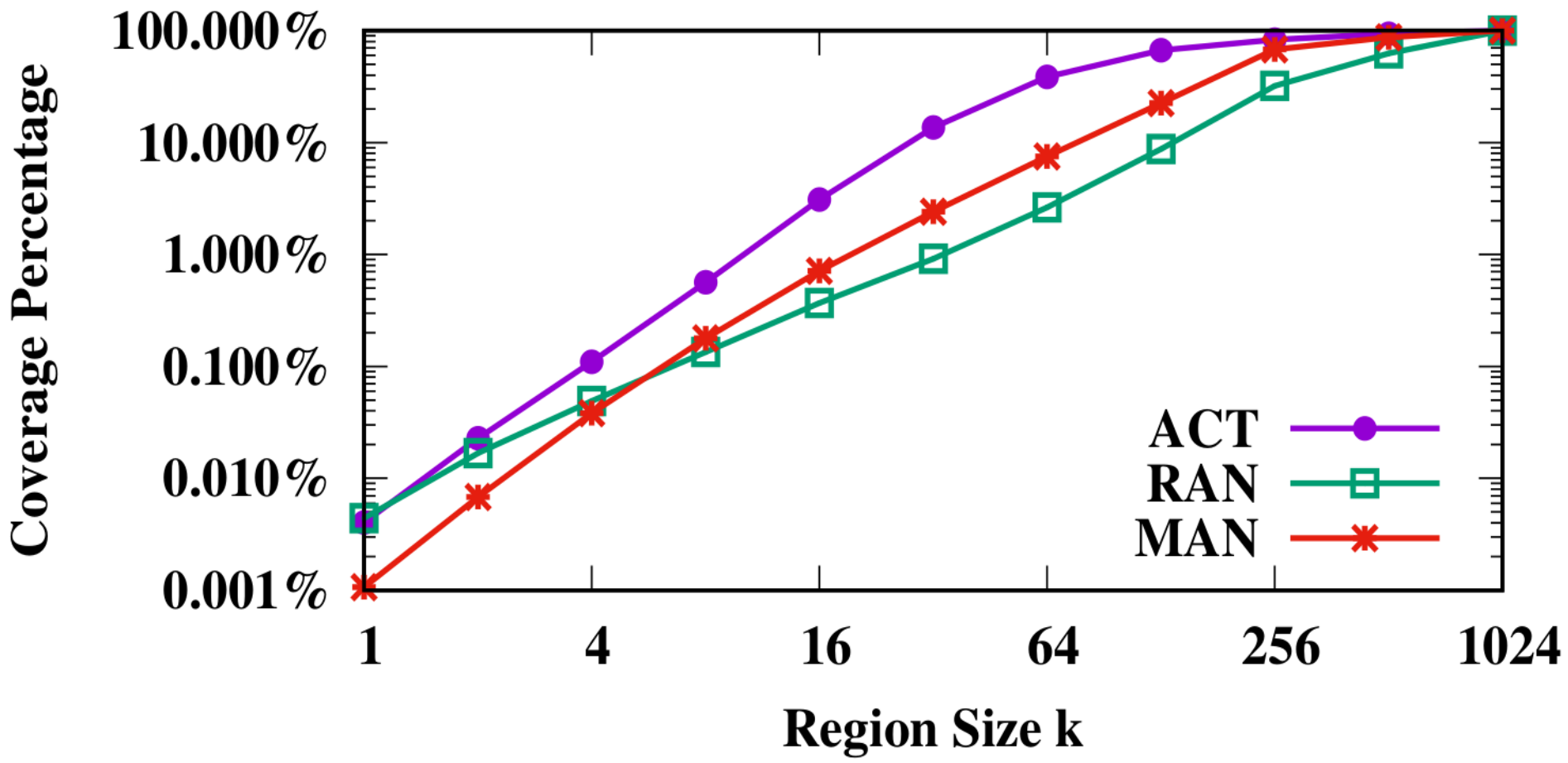
- ACT
- RAN: Undirected random testing
- MAN: Manually choose popular network environments used in previous studies
- SYM: Symbolic execution based testing where packet delays are represented as symbolic variables

Measuring state space coverage



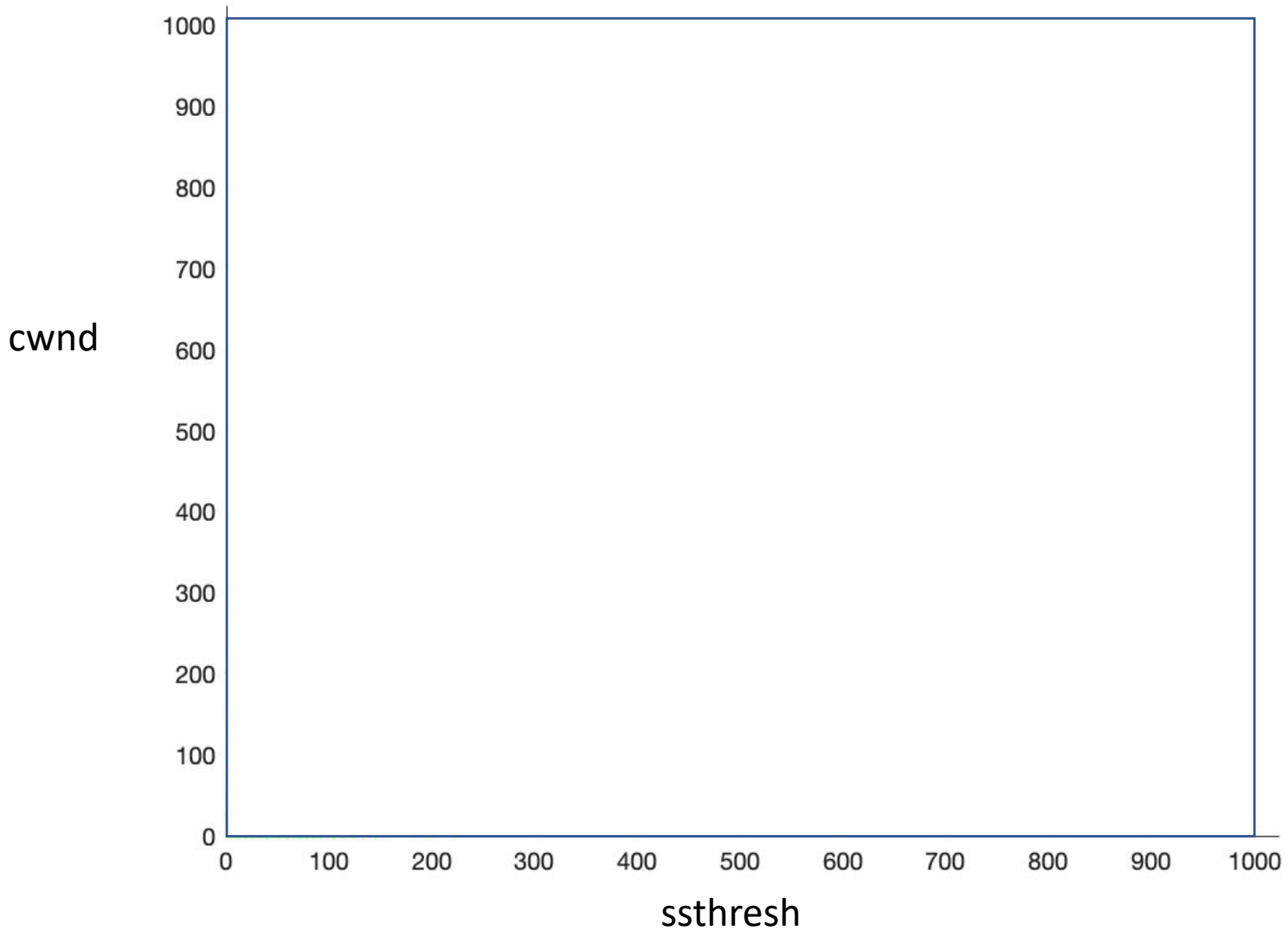
- Divide S into equal-sized regions of size k.
- Measure the percentage of regions covered by each testing method
- Example: 3 out of 4 regions are visited, thus coverage is 75% with region size k.

State space coverage of CUBIC

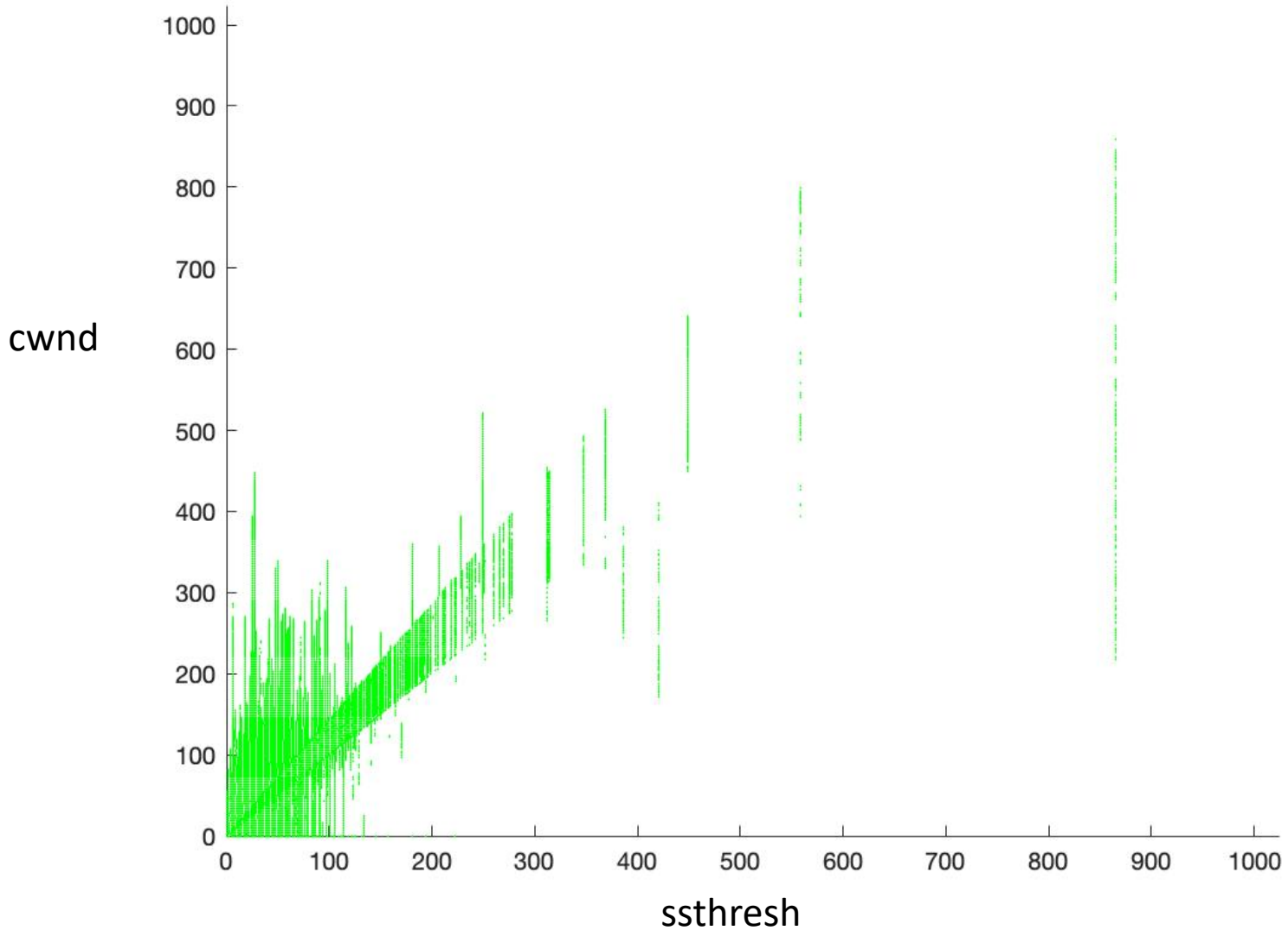


- SYM is not scalable to ten of thousands of packets.

Two-dimensional projection of S: Coverage by ACT

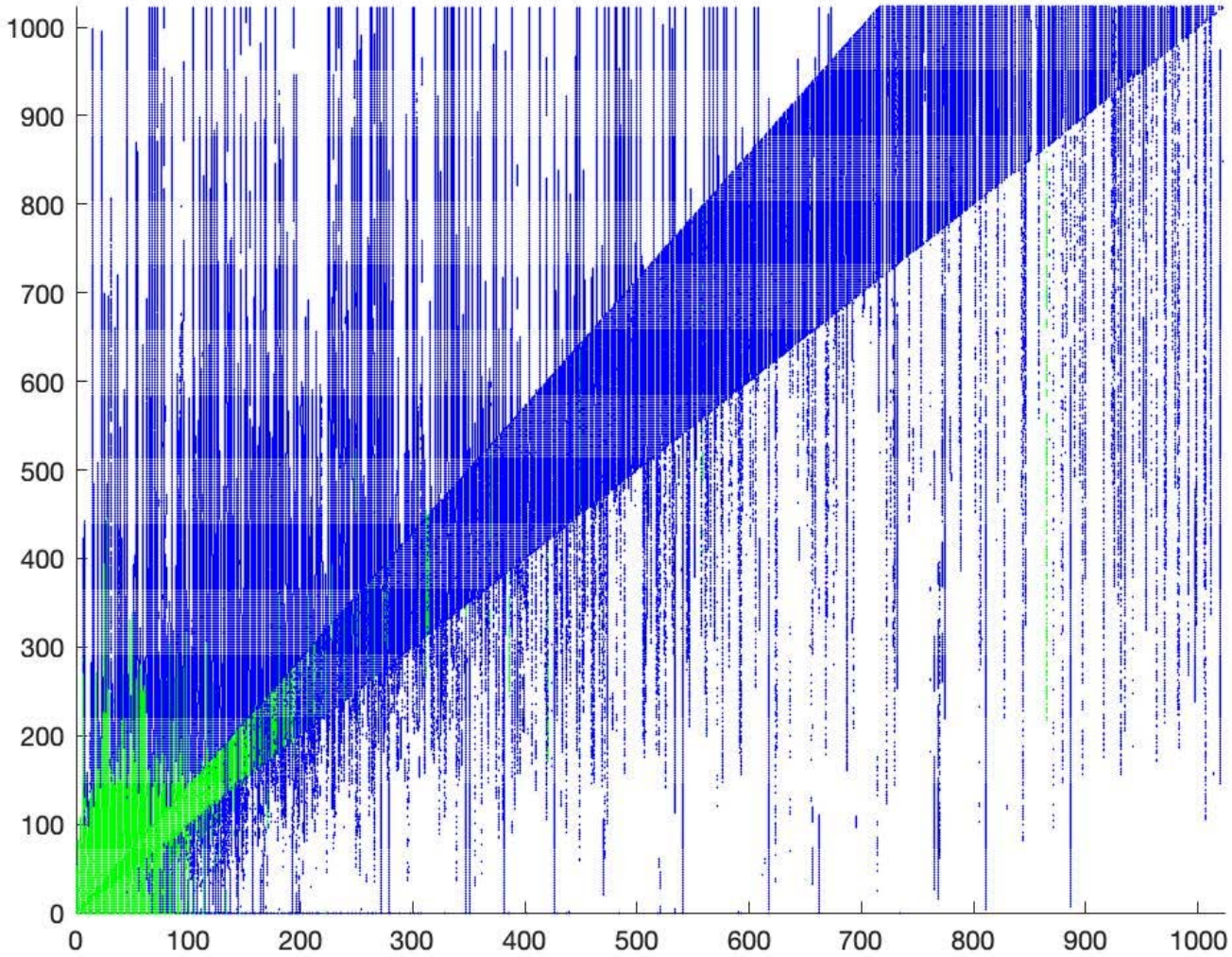


Two-dimensional projection of S: Coverage by ACT



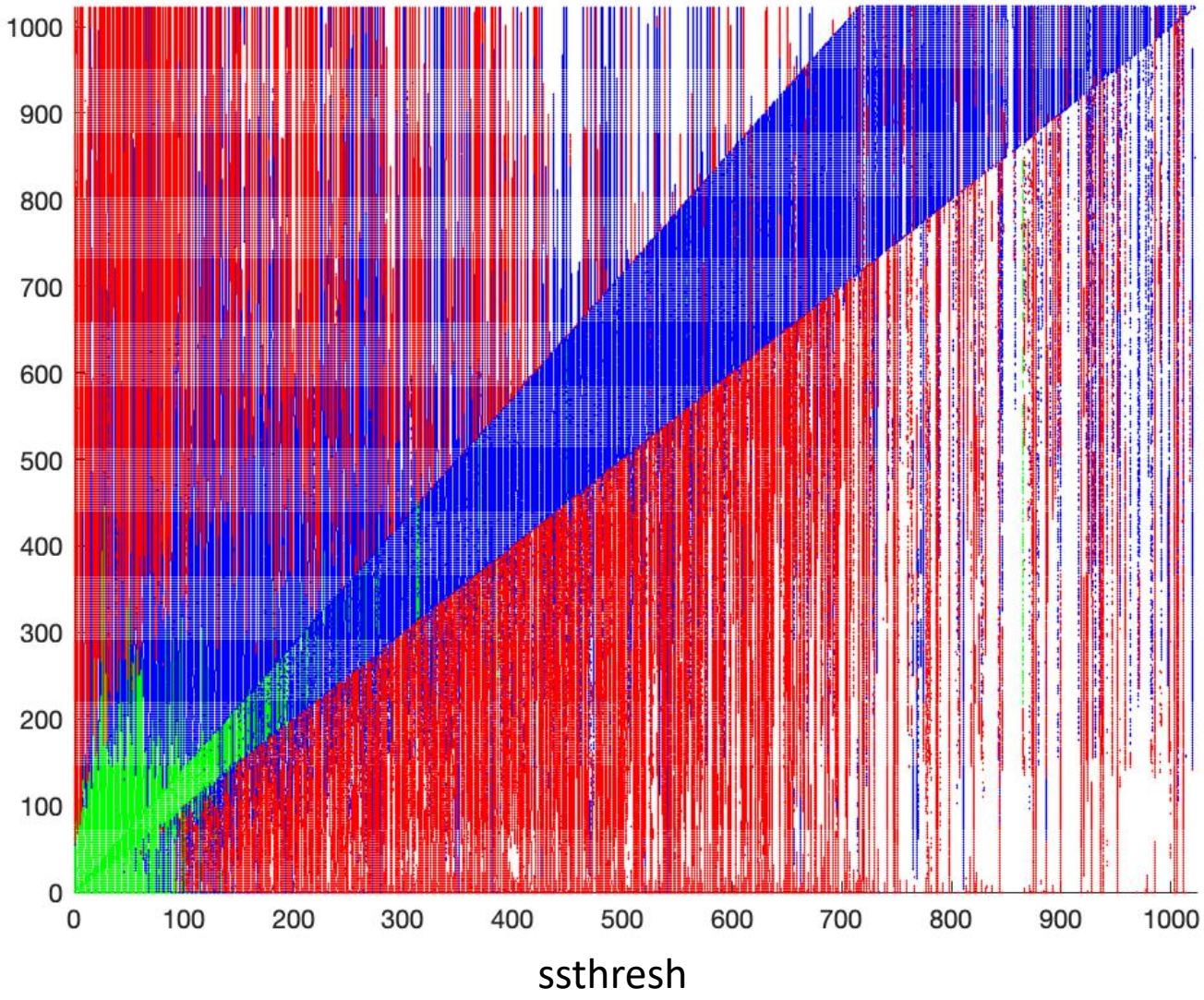
① random testing

Two-dimensional projection of S: Coverage by ACT



① random testing
② parameter estimation

Two-dimensional projection of S: Coverage by ACT

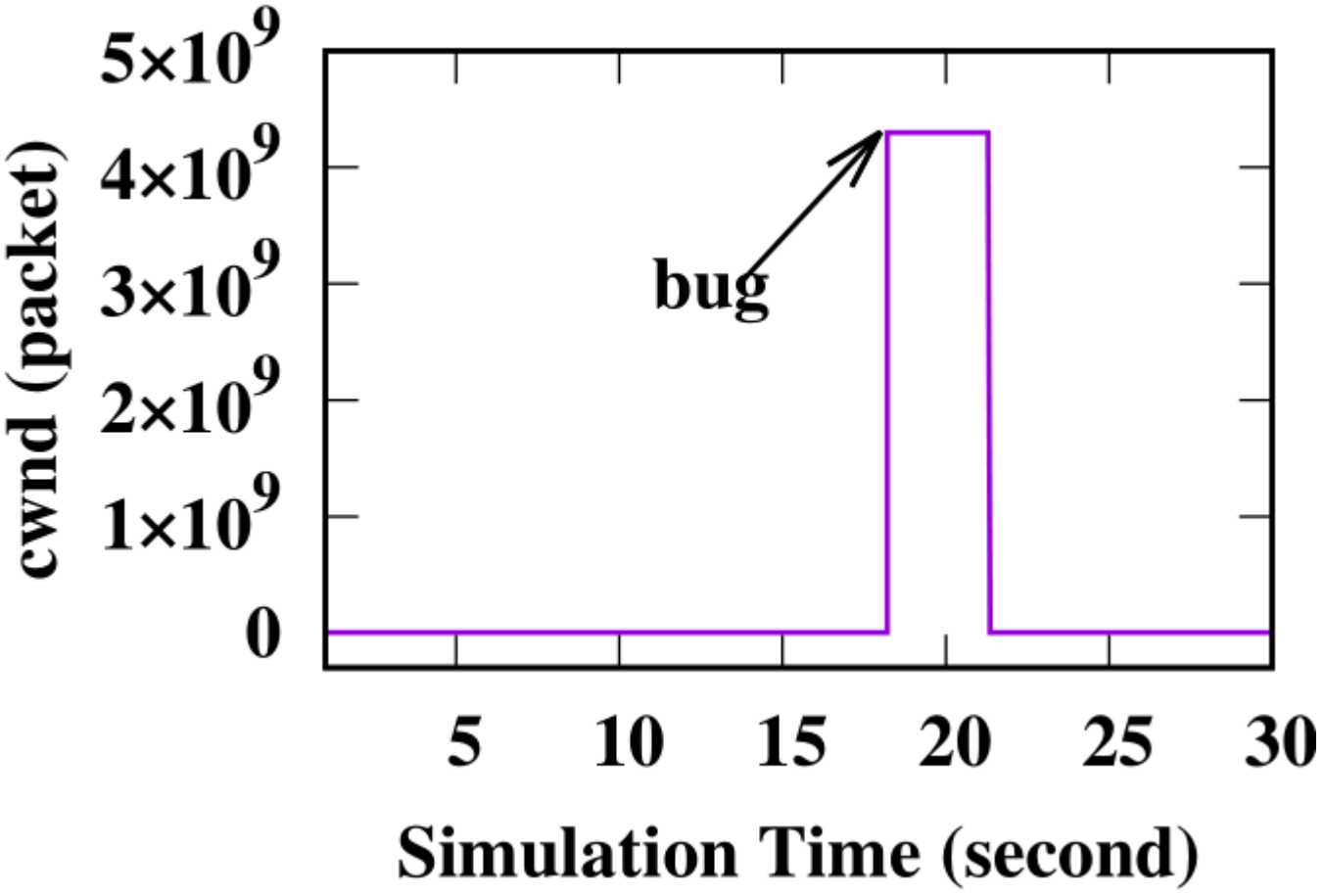


① random testing
② parameter estimation
③ parameter concatenation

Bug detection experiments

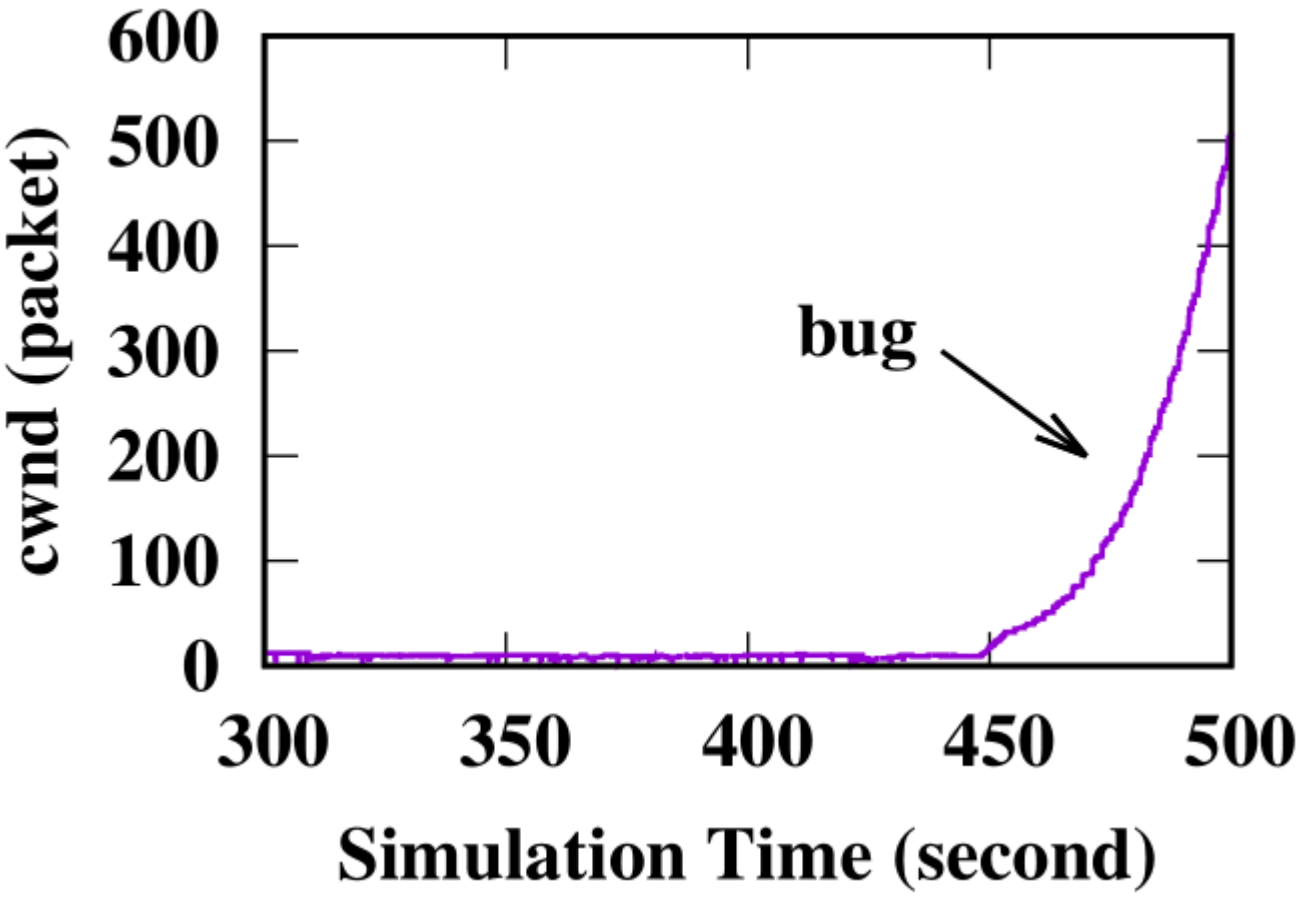
- Check three types of behaviors
  1. Generic behavior
  2. Window increase behavior
  3. Window decrease behavior

Generic behavior: Check whether cwnd > 10^7 packets



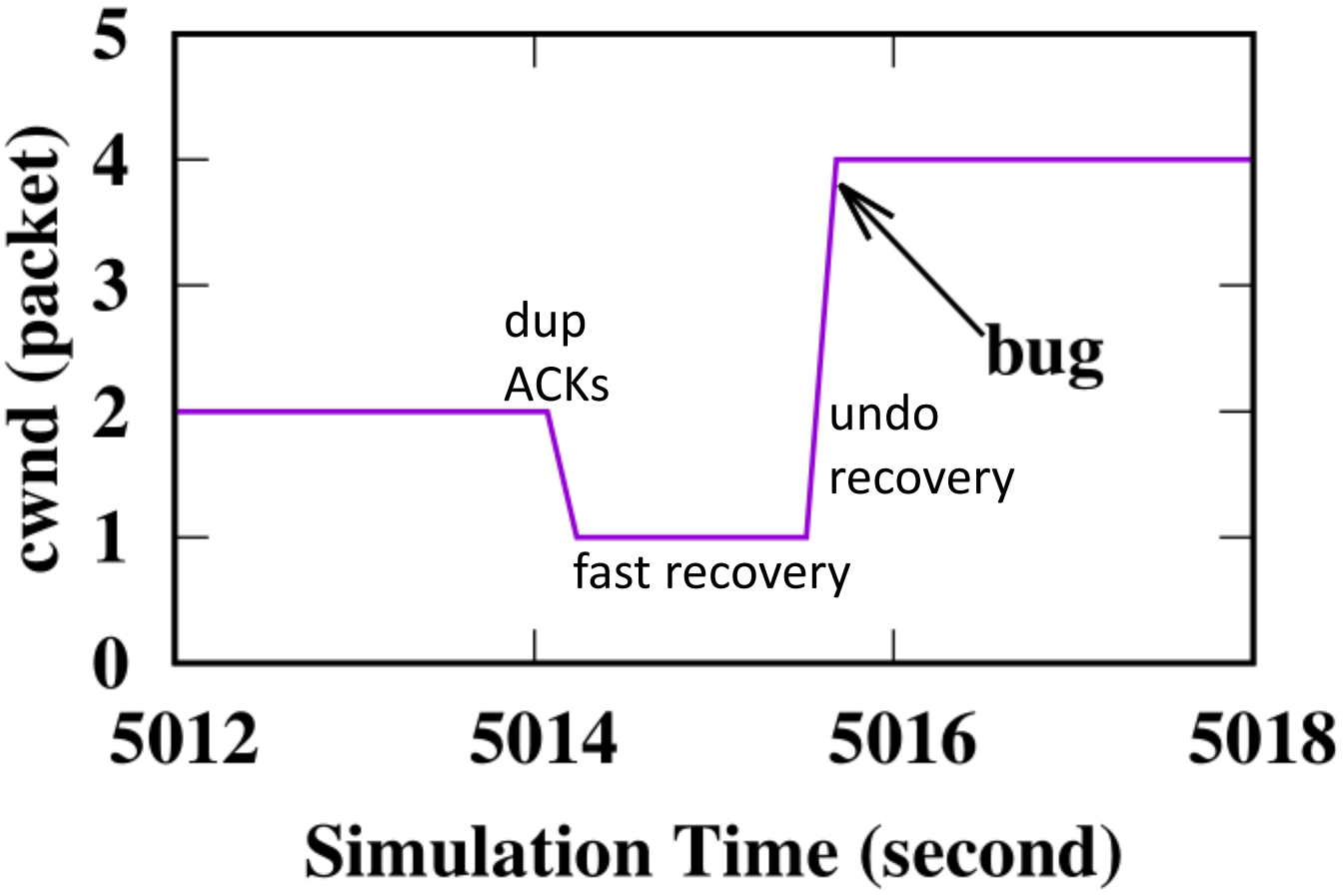
- Bug: Sometimes AIMD, HTCP, HSTCP, VENO set cwnd to 4,294,967,294 packets
- Reported to Linux kernel developers, were told that just fixed

Window increase behavior: Check whether target > 2*cwnd



- One bug: In the application rate limited periods, CUBIC does not increase cwnd, but it still increases target.

Window decrease behavior: Check whether cwnd reduces after fast recovery



- One bug: Sometimes AIMD and HTCP increase cwnd after undoed recovery
- Reported it to Linux kernel developers, and now it has been fixed

Conclusion

- Proposed ACT as a simple, efficient, and effective tool for automated TCP congestion control correctness testing.
- Found several Linux TCP bugs using ACT.