Evaluation of Optimization Methods for Network Bottleneck Diagnosis

Alina Beygelzimer
Jeff Kephart
Irina Rish

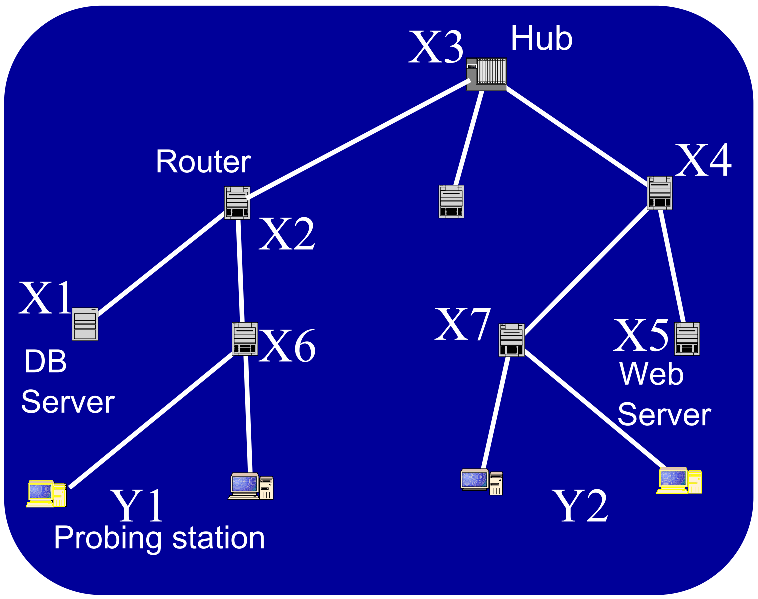
Real-time Network Performance Diagnosis

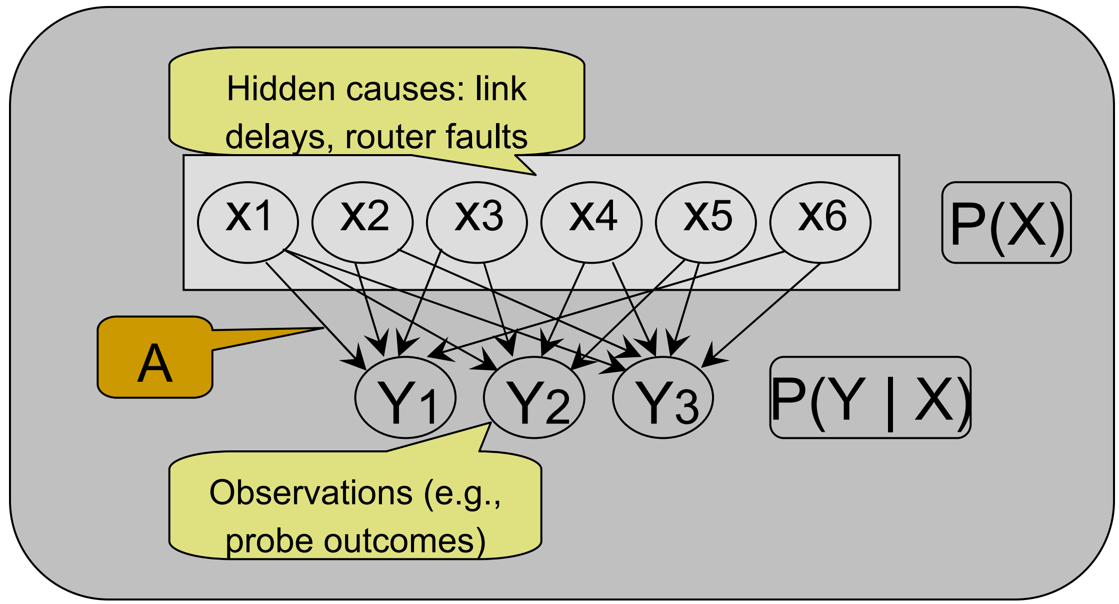
- Nodes in computer networks often experience performance problems

- Problem localization can be difficult
  - Typically, we don't have information about individual link delays
  - But we can monitor end-to-end transmission times for selected probes
    - Sometimes, we can pick the probes

- Approach: combine measurements from multiple probes to localize bottlenecks
  - We compare several algorithms
    - In a small real network in our lab
    - In a simulated medium-scale network (Gnutella topology)

Inference about Hidden Causes in Networks: "Network Tomography"



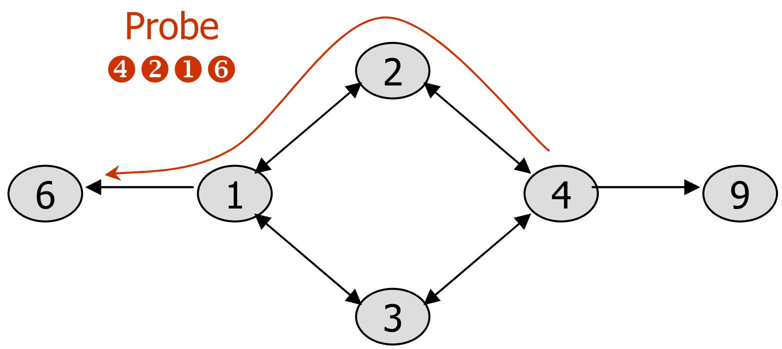


Suppose we have

- Table A of known dependencies (routing table)
- Set X of actual node delays that we can't measure directly
- Set Y of end-to-end probe latencies that we measure

Then we want to infer X from Y, and over time identify when X is anomalously large

Choosing end-to-end probes



- In some situations, we may have a choice of probes

- There is a simple algorithm that chooses a set of probes of minimum size that suffices for recognizing single faults

Links

All possible probes of length >1

Sufficient for unique fault diagnosis

	1 6	1 2	1 3	2 4	3 4	4 9
4216	1	1	0	1	0	0
4316	1	0	1	0	1	0
1249	0	1	0	1	0	1
312	0	1	1	0	0	0
124	0	1	0	1	0	0
349	0	0	0	0	1	1
421	0	1	0	1	0	0
431	0	0	1	0	1	0
216	1	1	0	0	0	0
249	0	0	0	1	0	1
213	0	1	1	0	0	0
316	1	0	1	0	0	0
134	0	0	1	0	1	0
1349	0	0	1	0	1	1

Four probes suffice for detection and single fault diagnosis: all columns (“signatures” of problems) are unique, all links are covered

Isolating Bottlenecks: Problem Formulation

N links (problems)

vector X of link delays

X	X_1	X_2	X_3	X_4	X_5	X_6
	❶❻	❶❷	❶❸	❷❹	❸❹	❹❾
❹❷❶❻	1	1	0	1	0	0
❹❸❶❻	1	0	1	0	1	0
❶❷❹❾	0	1	0	1	0	1
❸❶❷	0	1	1	0	0	0

Routing matrix A

vector Y of end-to-end observations

Y
Y_1
Y_2
Y_3
Y_4

M probes

$$Y = AX + \text{noise}$$

M□□N (under-constrained system), so in general we cannot uniquely solve for X!

- We need to introduce additional constraints, e.g.
  - Assume that X is sparse (small number of bottlenecks)
  - Don’t attempt to determine X_i precisely (just whether it’s significant)

Various approaches

- Regression-based: find X that minimizes $||Y - AX||_2$
  - May add **positivity** constraint: $X_i >= 0$
  - May add **regularization**
    - $||X||_0 \leq k$, or
    - minimize $||X||_0$ subject to $||Y\text{-}AX||_2 \leq \delta$, or
    - **convexify**: minimize $||X||_1$ subject to $||Y\text{-}AX||_2 \leq \delta$
      - leads to a quadratic program
  - Can require combinatorial optimization

- Greedy approaches
  - Add one bottleneck X_i at a time
  - May or may not revise X_i as we add more non-zero X components

Greedy approaches: add one “bottleneck” at a time

1) Initialization:
   Set the residual to Y and the approximation to 0.
2) Pick a component (column of A) that best matches the observations, given the ones already chosen:
   Variant L1: Pick the component maximizing the dot product to the residual (over all coefficients).
   Variant L2: Pick the component minimizing the squared distance to the residual (over all coefficients).
3) Update the approximation and the residue. Stop if the residue is small enough; otherwise repeat step 2.

Two variants:

1) Pure greedy: the coefficients chosen at previous rounds don’t change.
2) Orthogonal greedy: the coefficients are re-optimized after each round.

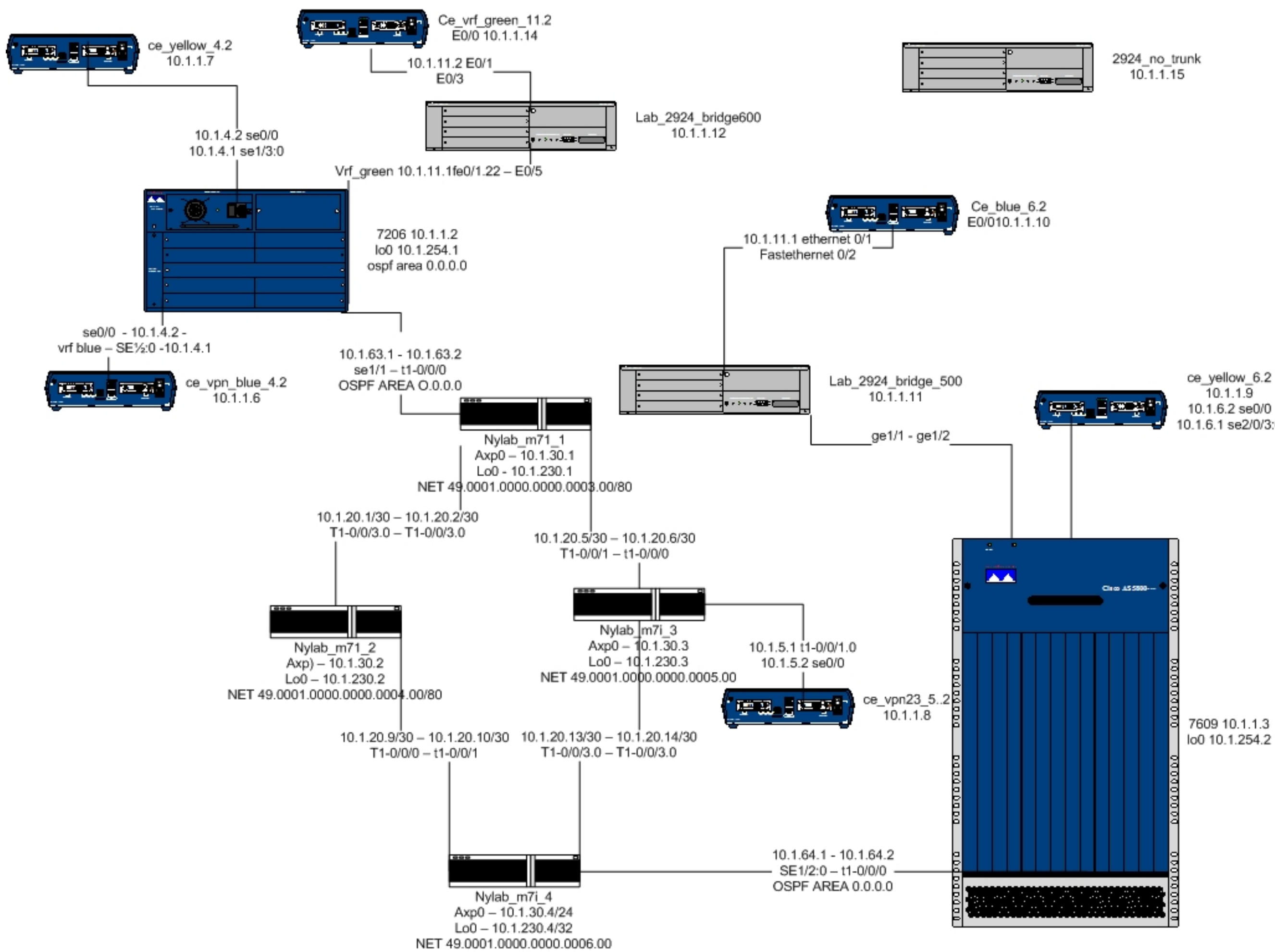
Another variant: impose a positivity constraint on the coefficients.

Our Experimental Network

We have a small network with Cisco and Juniper routers

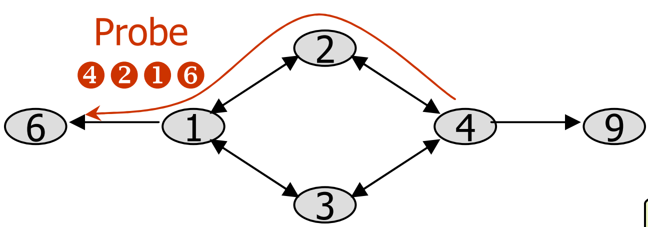
We induce performance problems at specified nodes by excessive pings

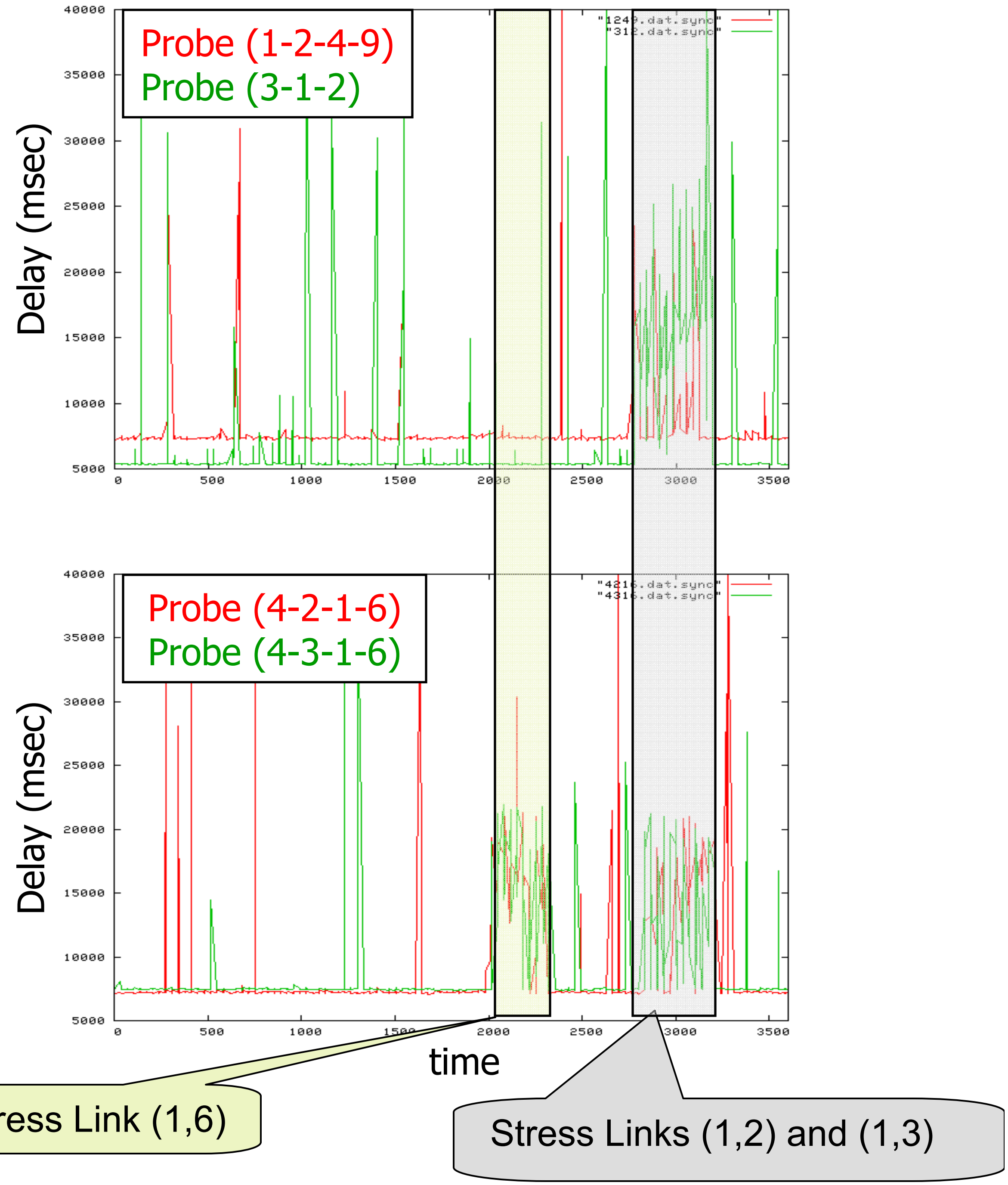
We use monitoring probes to measure delay, jitter and packet loss
- SAA/rtr on Cisco
- RPM on Juniper



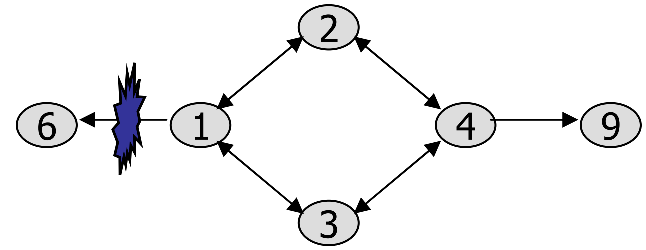
Experiments

- We induce performance problems at specific links by pinging them with 30K 1400-byte ECHO_REQUEST packets.
- For each of our four probes, we measure the end-to-end transmission delay.
- Which algorithms best determine the bottlenecked links from the probe delays?





Incident 1: link ❶❻ is slow



In microseconds

	❶❻	❶❷	❶❸	❷❹	❸❹	❹❾	mean normal delay	mean stressed delay	difference dY_i
❹❷❶❻	1	1	0	1	0	0	**7,762**	**15,912**	**8,166**
❹❸❶❻	1	0	1	0	1	0	**7,562**	**15,275**	**7,691**
❶❷❹❾	0	1	0	1	0	1	**7,846**	**7,453**	**-408**
❸❶❷	0	1	1	0	0	0	**5,917**	**5,621**	**-308**

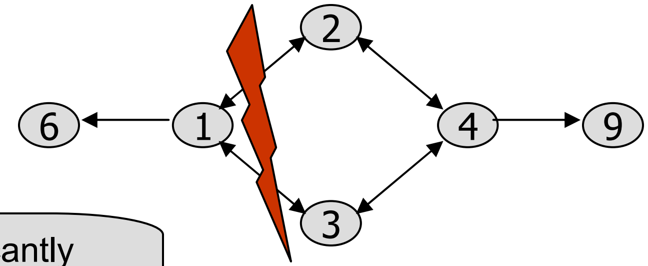
L2 loss

X's minimizing the loss

0.6 *10^3	**10.4 *10^3**	**9.9 *10^3**	**9.8 *10^3**	**9.9 *10^3**	**11.2 *10^3**
7928	**2483**	**3691**	**3879**	**7691**	**-408**

Decode to the column k minimizing $\sum_i (\alpha_k A(i,k) - dY_i)^2$

Incident 2: Links ❶❷ and ❶❸ are slow



2.185*10^3

Double fault gives a significantly better explanation of observed Y

	❶❻	❶❷	❶❸	❷❹	❸❹	❹❾	mean normal delay	mean stressed delay	difference dY_i
❹❷❶❻	1	1	0	1	0	0	**7,762**	**14,360**	**6,598**
❹❸❶❻	1	0	1	0	1	0	**7,562**	**16,031**	**8,469**
❶❷❹❾	0	1	0	1	0	1	**7,846**	**16,084**	**8,238**
❸❶❷	0	1	1	0	0	0	**5,917**	**18,875**	**12,958**
	15.4 *10^3	**12.6 *10^3**	**11.0 *10^3**	**15.5 *10^3**	**16.7 *10^3**	**16.8 *10^3**			

L2 loss for each column individually (hypothesizing single fault)

L2 loss for pairs (best linear combination for each pair)

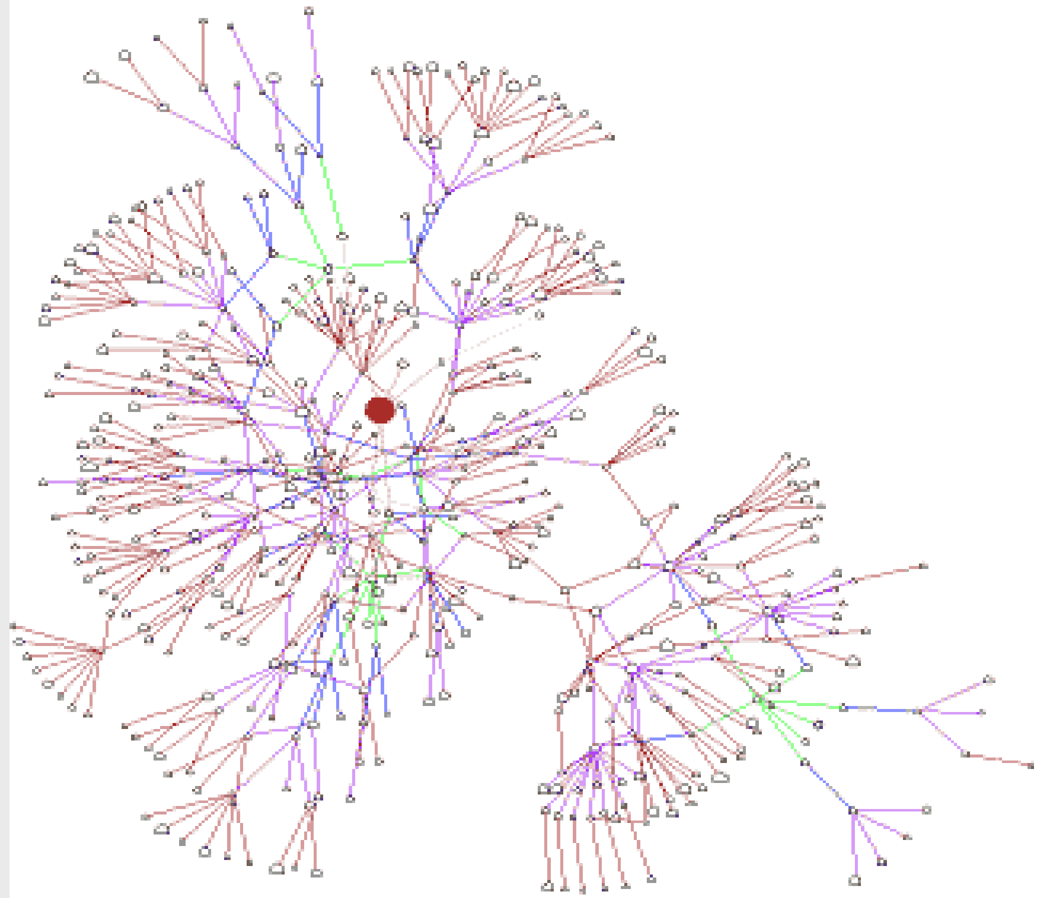
(1,2)+(1,3): 2.185*10^3
(1,6)+(1,3): 10.43*10^3
(2,4)+(3,4): 13.01*10^3

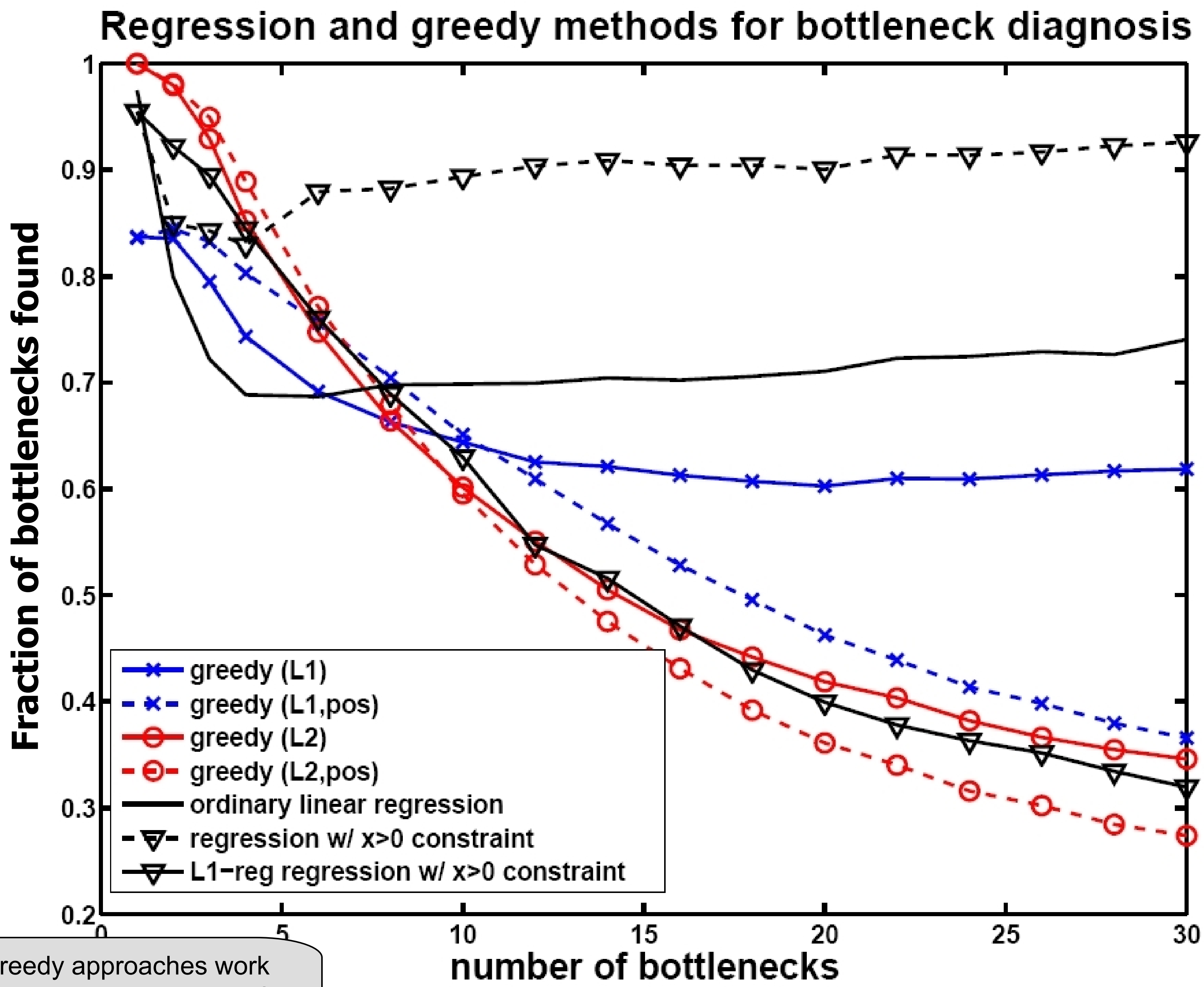
Decode to a linear combination of a column pair (k,m) minimizing

$$\sum_i (\alpha_k A(i,k) + \beta_m A(i,m) - dY_i)^2$$

Evaluating the approach in larger, realistically simulated environments

- Generate dependency matrices from LimeWire snapshots of Gnutella
  - Generate shortest-path spanning trees from a random set of “sources”
  - Run probe selection for single fault diagnosis on the set of resulting paths to generate several dozen probes
- Induce K bottlenecks with random delays, simulated end-to-end delays using $Y = AX + N(0,\sigma)$
- Evaluate how well the algorithms identify the induced bottlenecks

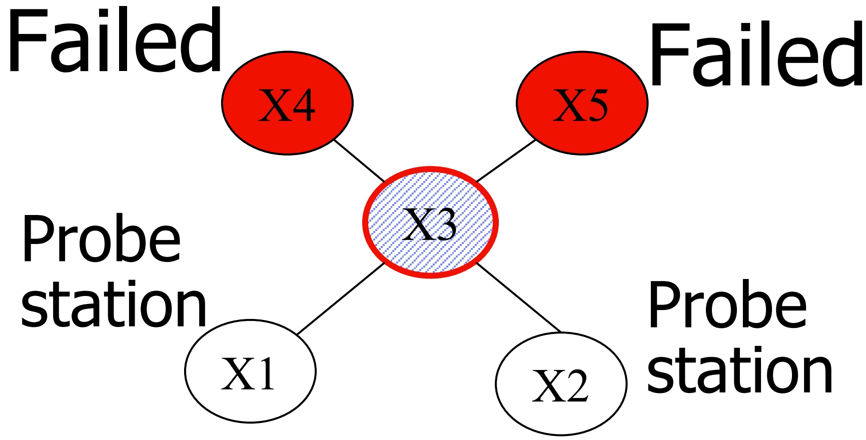




Greedy approaches work well when the number of bottlenecks is small

Conclusions

- Preliminary results suggest that:
  - If you expect a small number of bottlenecks, use Greedy with L2 norm
    - Positivity constraint hurts Greedy; less so if coefficients are re-adjusted at each iteration
  - If you expect more than 5-6 bottlenecks, use ordinary least-square regression with positivity constraint



	X1	X2	X3	X4	X5
Probe1	1	0	1	0	1
Probe2	0	1	1	1	0
Probe3	0	1	0	0	0
Probe4	1	0	0	0	0

Probe=1 denotes failure, Probe=0 denotes success

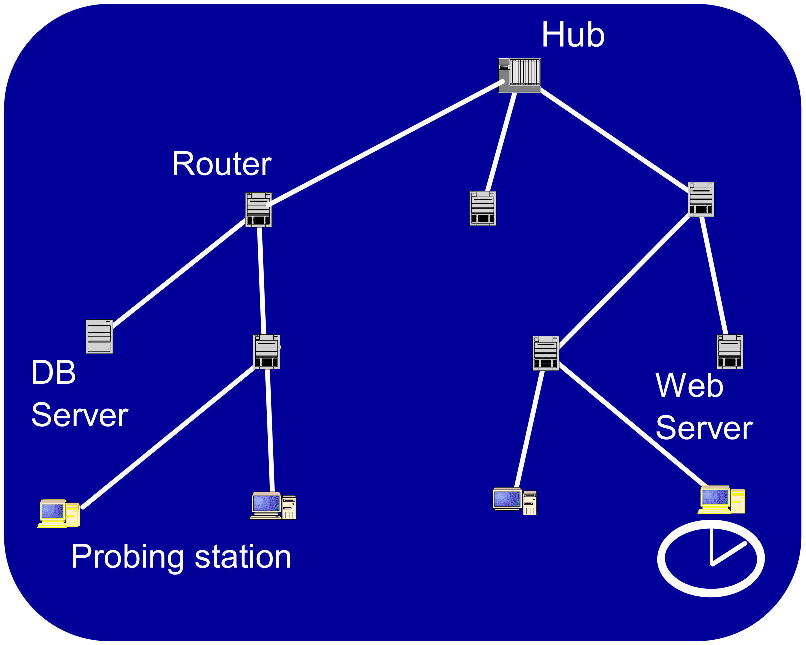
1. Multiple faults confused with a single-fault

Truth: 'X4 and X5 failed', but symptom: (1, 1, 0, 0) in single-fault diagnosis indicates that X3 failed

2. multi-fault symptom mistreated as 'noise'

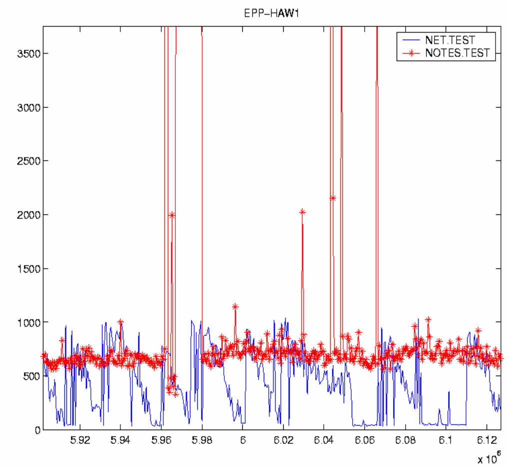
Truth: 'X1 and X4 failed', symptom: (1, 1, 0, 1) Does not exist in 'codebook' => treated as 'noise'

Real-time Problem Diagnosis



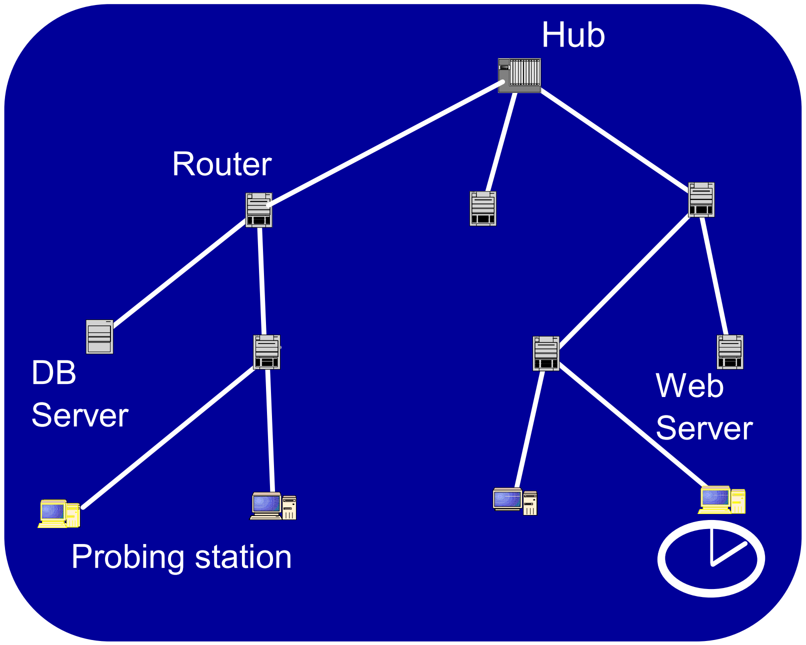
Available data:

End-to-end transactions or probes: ping, traceroute, application transactions (DB- and web-access probes, etc.)



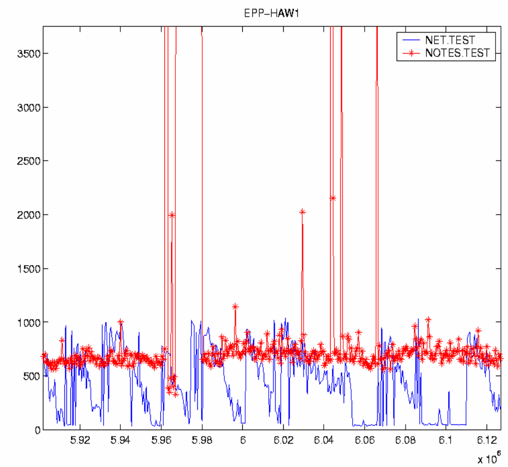
1. Real-time problem diagnosis
2. End-to-end performance prediction
3. Discovering partially known or unknown routing, especially in wireless and mobile networks

Real-time Problem Diagnosis



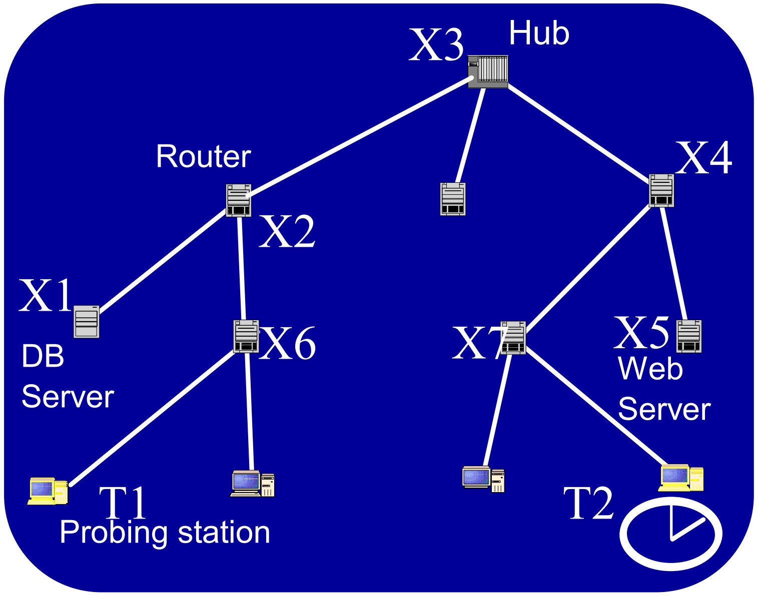
Available data:

End-to-end transactions or probes: ping, traceroute, application transactions (DB- and web-access probes, etc.)



1. Real-time problem diagnosis
2. End-to-end performance prediction
3. Discovering partially known or unknown routing, especially in wireless and mobile networks

Inference about Hidden Causes in Networks: "Network Tomography"

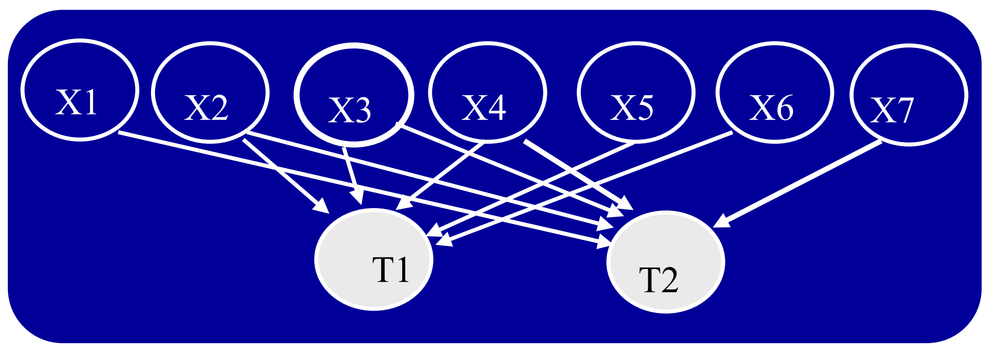


General Task: find the unobserved states of network components (node/link failures or delays)

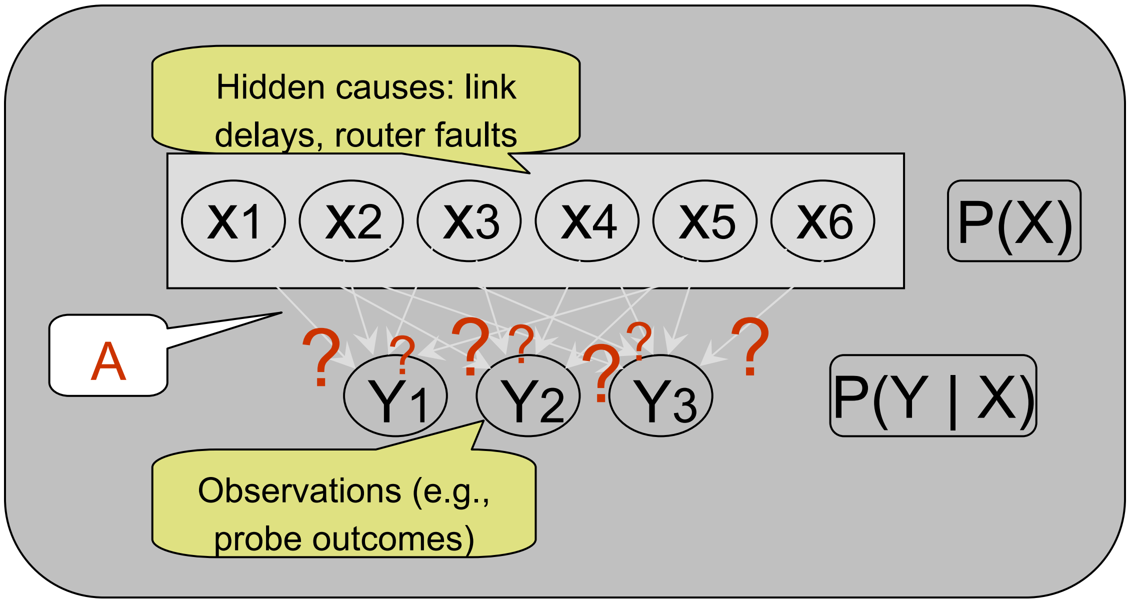
Available measurements: end-to-end transactions ('probes'): ping, traceroute, email- and web-access, e-business transactions

Challenges:

- Minimize probing costs
- Maximize diagnostic speed
- Maximize diagnostic accuracy



Inference about Hidden Causes in Networks: "Network Tomography"

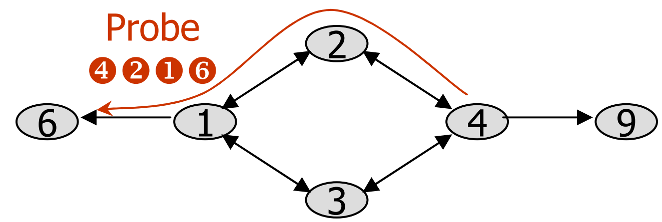


Case 2: **unknown** or partially known dependencies (routing table)

- learning dependency/routing matrix
- diagnosis, prediction

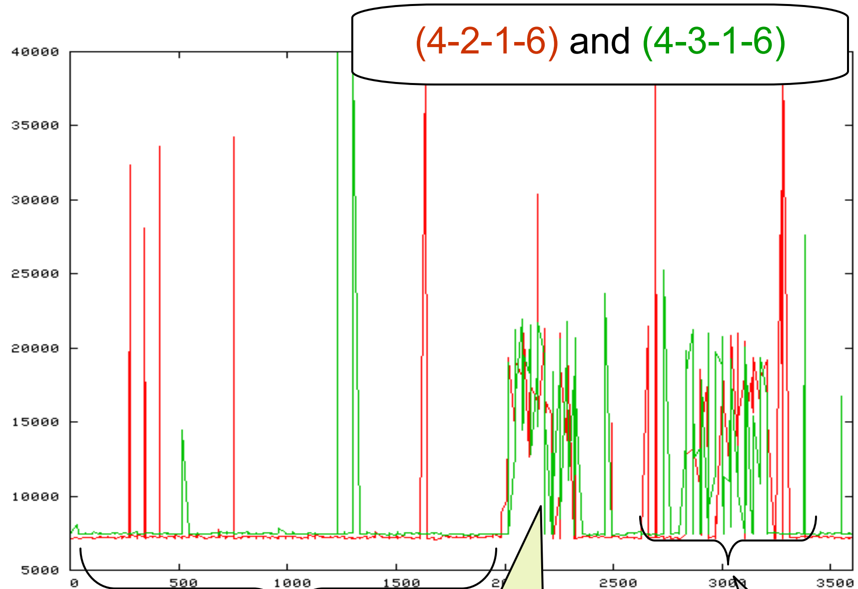
[Chandalia & Rish, submitted to IMC07]

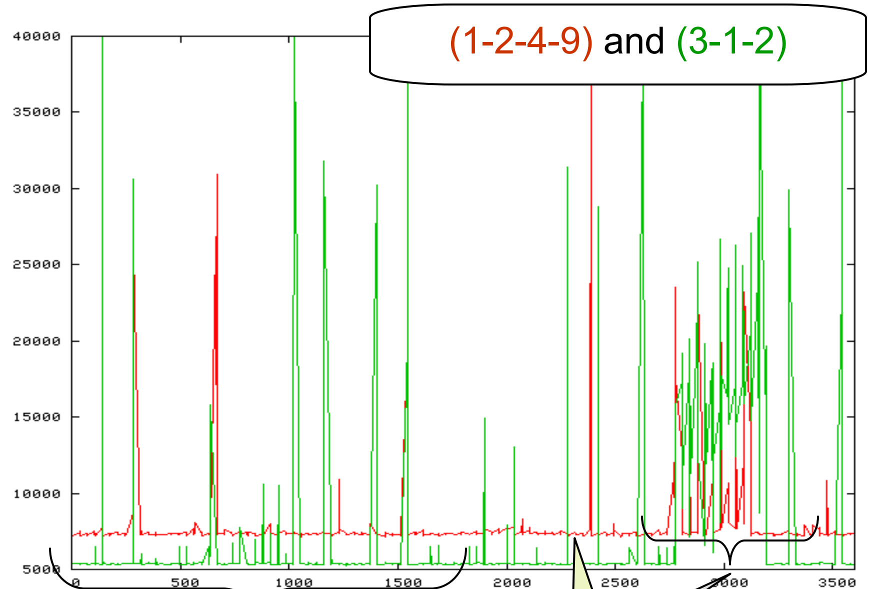
Controlled experiments: Stressing links (using pings)



Testbed: Micromuse's Lab Environment in NYC

We can remotely use monitoring probes (SAA/rtr on Cisco routers and RPM on Juniper) to measure delay, jitter, and packet loss





Normal behavior (~30 minutes)

Normal behavior (~30 minutes)

Scenario 1: Link (1,6) is stressed by pinging it with 30K 1400-byte ECHO_REQUEST packets

Scenario 2: Links (1,2) and (1,3) are stressed by pinging with 30K 1400-byte ECHO_REQUEST packets each

Stressing link (1,6) has no effect as expected

Advantage over the “hard” decoding approach of SMARTs

$\overbrace{\quad\quad\quad\quad}^{\text{2 bottlenecks}}$

	❶❻	❶❷	❶❸	❷❹	❸❹	❹❾	mean normal delay	mean stressed delay	difference dY_i
❹❷❶❻	1	1	0	1	0	0	**7,762**	**14,360**	**6,598**
❹❸❶❻	1	0	1	0	1	0	**7,562**	**16,031**	**8,469**
❶❷❹❾	0	1	0	1	0	1	**7,846**	**16,084**	**8,238**
❸❶❷	0	1	1	0	0	0	**5,917**	**18,875**	**12,958**

The codebook approach **cannot distinguish** the case when ❶❷ and ❶❸ fail from **ANY** of the following cases: ❶❷ and ❸❹, ❶❷ and ❶❻, ❶❸ and ❷❹, etc.

We can, since there is more information in the real-valued performance “signal” than in binary fault/no fault “signal”