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

<table>
<thead>
<tr>
<th>Probe</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>4216</td>
<td>11010</td>
</tr>
<tr>
<td>4316</td>
<td>10101</td>
</tr>
<tr>
<td>1249</td>
<td>01010</td>
</tr>
<tr>
<td>312</td>
<td>01100</td>
</tr>
<tr>
<td>124</td>
<td>01010</td>
</tr>
<tr>
<td>349</td>
<td>00011</td>
</tr>
<tr>
<td>421</td>
<td>00101</td>
</tr>
<tr>
<td>431</td>
<td>10100</td>
</tr>
<tr>
<td>216</td>
<td>11000</td>
</tr>
<tr>
<td>249</td>
<td>00010</td>
</tr>
<tr>
<td>213</td>
<td>01100</td>
</tr>
<tr>
<td>316</td>
<td>10100</td>
</tr>
<tr>
<td>134</td>
<td>00101</td>
</tr>
<tr>
<td>1349</td>
<td>00111</td>
</tr>
</tbody>
</table>

All possible probes of length >1

Sufficient for unique fault diagnosis:
- Four probes suffice for detection and single fault diagnosis: all columns ("signatures" of problems) are unique, all links are covered
Isolating Bottlenecks: Problem Formulation

\[ Y = AX + \text{noise} \]

- **Vector X of link delays**
- **Routing matrix A**
- **Vector Y of end-to-end observations**

\[
\begin{array}{cccccc}
\hline
\text{X1} & \text{X2} & \text{X3} & \text{X4} & \text{X5} & \text{X6} \\
\hline
4 & 2 & 1 & 6 & 0 & 1 & 0 & 0 \\
4 & 3 & 1 & 6 & 0 & 1 & 0 & 0 \\
1 & 2 & 4 & 9 & 0 & 1 & 0 & 1 \\
3 & 1 & 2 & 0 & 1 & 1 & 0 & 0 \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
\hline
\text{Y1} & \text{Y2} & \text{Y3} & \text{Y4} \\
\hline
0 & 1 & 1 & 0 & 0 \\
\hline
\end{array}
\]

- **M \leq 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: $X_i \geq 0$
  - May add regularization
    - $\|X\|_0 \leq k$, or
    - minimize $\|X\|_0$ subject to $\|Y - AX\|_2 \leq \delta$, or
    - convexify: minimize $\|X\|_1$ subject to $\|Y - AX\|_2 \leq \delta$
      - 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

<table>
<thead>
<tr>
<th>1) Initialization:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the residual to $Y$ and the approximation to 0.</td>
</tr>
</tbody>
</table>

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 1 6 is slow

<table>
<thead>
<tr>
<th></th>
<th>1 6</th>
<th>1 2</th>
<th>1 3</th>
<th>2 4</th>
<th>3 4</th>
<th>4 9</th>
<th>mean normal delay</th>
<th>mean stressed delay</th>
<th>difference $dY_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 2 1 6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>7,762</td>
<td>15,912</td>
<td>8,166</td>
</tr>
<tr>
<td>4 3 1 6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7,562</td>
<td>15,275</td>
<td>7,691</td>
</tr>
<tr>
<td>1 2 4 9</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>7,846</td>
<td>7,453</td>
<td>-408</td>
</tr>
<tr>
<td>3 1 2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5,917</td>
<td>5,621</td>
<td>-308</td>
</tr>
</tbody>
</table>

In microseconds

L2 loss

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

$X$’s minimizing the loss

Decode to the column k minimizing

<table>
<thead>
<tr>
<th>0.6</th>
<th>10.4</th>
<th>9.9</th>
<th>9.8</th>
<th>9.9</th>
<th>11.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>*10^3</td>
<td>*10^3</td>
<td>*10^3</td>
<td>*10^3</td>
<td>*10^3</td>
<td>*10^3</td>
</tr>
<tr>
<td>7928</td>
<td>2483</td>
<td>3691</td>
<td>3879</td>
<td>7691</td>
<td>-408</td>
</tr>
</tbody>
</table>
## Incident 2: Links 1-2 and 1-3 are slow

Double fault gives a significantly better explanation of observed Y

<table>
<thead>
<tr>
<th></th>
<th>1-6</th>
<th>1-2</th>
<th>1-3</th>
<th>2-4</th>
<th>3-4</th>
<th>4-9</th>
<th>mean normal delay</th>
<th>mean stressed delay</th>
<th>difference dY_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-2-1-6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>7,762</td>
<td>14,360</td>
<td>6,598</td>
</tr>
<tr>
<td>4-3-1-6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7,562</td>
<td>16,031</td>
<td>8,469</td>
</tr>
<tr>
<td>1-2-4-9</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>7,846</td>
<td>16,084</td>
<td>8,238</td>
</tr>
<tr>
<td>3-1-2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5,917</td>
<td>18,875</td>
<td>12,958</td>
</tr>
</tbody>
</table>

### L2 loss for columns individually

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

### 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

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

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

Where  \( \alpha_k, \beta_m \) are coefficients for the linear combination.
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
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”

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

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

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.

Scenario 1: Link (1,6) is stressed by pinging it with 30K 1400-byte ECHO_REQUEST packets. Stressing link (1,6) has no effect as expected.

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

Normal behavior (~30 minutes)
Advantage over the “hard” decoding approach of SMARTs

2 bottlenecks

<table>
<thead>
<tr>
<th></th>
<th>16</th>
<th>12</th>
<th>13</th>
<th>24</th>
<th>34</th>
<th>49</th>
<th>mean normal delay</th>
<th>mean stressed delay</th>
<th>difference $dY_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4216</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>7,762</td>
<td>14,360</td>
<td>6,598</td>
</tr>
<tr>
<td>4316</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7,562</td>
<td>16,031</td>
<td>8,469</td>
</tr>
<tr>
<td>1249</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>7,846</td>
<td>16,084</td>
<td>8,238</td>
</tr>
<tr>
<td>312</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5,917</td>
<td>18,875</td>
<td>12,958</td>
</tr>
</tbody>
</table>

The codebook approach **cannot distinguish** the case when 12 and 13 fail from ANY of the following cases: 12 and 34, 12 and 16, 13 and 24, etc.

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