A Different Kind of Flow Analysis

David M Nicol

University of Illinois at Urbana-Champaign
What Am I Doing Here???

Invite for “ICASE Reunion”

Did research on “Performance Analysis Supporting Supercomputing”
- many problems supporting HPC CFD

TODAY’S TALK
- Simulation, modeling flows, HPC,
What Am I Doing Here???

Invite for “ICASE Reunion”

Did research on “Performance Analysis Supporting Supercomputing”
• many problems supporting HPC CFD

TODAY’S TALK
• Simulation, modeling flows, HPC,

And Now For Something Completely Different..
Motivation

Large-scale network simulations with
- “background” traffic where details aren’t needed
- Congestion affecting results
- Traffic where principal interest is delivered volume
  - e.g. worm scans, flooding attack
- Our specific motivation is for cyber-defense training (RINSE)

Possible solution: simulate such traffic as “flows” at a coarse time-scale
- Inject flow rates at edge of network
- Compute delivered volume for each flow
- Compute link utilization throughout network

Challenges:
- Capture interactions between flows, routing infrastructure, fine scale traffic
Big Picture

Define time-step larger than end-to-end latency (e.g. 1 sec)

Each time-step

• Define (src,dest,rate) triples
  – At all network ingress points
  – Rate can depend on feedback

• “Push” flows through network
  – fine time-scale traffic viewed in aggregate with its own (historical) flow rates
  – routing based on forwarding tables
  – loss at router ports where aggregate input rate exceeds port bandwidth
  – record bandwidth consumption
**Modeling Congestion**

Even though flows are acyclic, dependency cycles may form in definition of flow rates.

```
Define \( l^* = l_1 + \cdots + l_n \)
```

\[
* = \begin{cases} 
\frac{l_i}{L} & \text{when } L \leq r \\
\min\{1, \frac{l_i}{L}\} & \text{otherwise}
\end{cases}
\]

No congestion

congestion

* depends on
1
* depends on
2
* depends on
3

\[ l_1 + l_2 + l_3 > r \]

\[ l_1 + l_2 + l_3 \leq r \]
Resolution and Transparency

Try to *resolve* final output flow values based on upper bounds.

All of a port’s final output flows can be resolved once all of its input flow values are resolved.

*But to break cycles we need to be smarter....*

**Notice that every output flow is bounded from above by input flow rate.** .... Every flow can be bounded by its ingress rate.

A port is *transparent* if the sum of input rate bounds is no greater than the output bandwidth.

Example: Suppose $l_1 + l_3 \leq r$

Then $l_1 + l_3 \leq r$ so that $l_1 = l_1$

1. Port becomes resolved
2. Flows become resolved
3. Repeat.

Flow rate becomes resolved

Port becomes resolved

Port becomes transparent
Dependency Reduction

Formalization

Flow states are \{settled, bounded\}
Port states are \{resolved, transparent, unresolved\}

A port’s state may change, depending on input flows
An output flow state may settle, when the port state becomes resolved or transparent

Iterate: {
1. Select port or flow whose state may change
2. Process state/value change
3. Identify ports/flows affected by the change
}
State Change Rules

Port states are \{resolved, transparent, unresolved\}

Flow states are \{settled, bounded\}

**Rule 1: port resolution**

**Pre-condition**
Port state is not *resolved* and all input flow states are settled

**Action**
Mark port state as resolved, compute all output flow values, mark each as settled
State Change Rules

Port states are \{resolved, transparent, unresolved\}
Flow states are \{settled, bounded\}

Rule 2: port transparency

Pre-condition

Port state is \textit{unresolved} and sum of input rate bounds is less than bandwidth,

Action

Mark port state as \textit{transparent}. For every input rate that is \textit{settled}, mark corresponding output rate as \textit{settled}. 
State Change Rules

Port states are \{resolved, transparent, unresolved\}

Flow states are \{settled, bounded\}

Rule 3: settle state transition

Pre-condition

Port state is transparent, some input flow is settled, and corresponding output flow is not settled.

Action

Mark corresponding output flow as settled, with value equal to input flow value.
State Change Rules

Port states are \{resolved, transparent, unresolved\}

Flow states are \{settled, bounded\}

Rule 4: flow bound transition #1

Pre-condition

Port state is unresolved, the fair proportion relative to settled flows of an input flow rate exceeds bound on output flow

\[
\left( \frac{\text{in}}{f} \right) < \text{out}
\]

Action

Lower corresponding output flow bound to be equal to fair proportion of input flow bound

\[
\text{out} = \left( \frac{\text{in}}{f} \right)
\]

is sum of settled flow rates
State Change Rules

Port states are \{resolved, transparent, unresolved\}

Flow states are \{settled, bounded\}

Rule 5: flow bound transition #2

Pre-condition

Port state is not resolved, the flow rate bound of an input flow is less than the corresponding output flow bound

Action

Set bound of output flow equal to bound on input rate

\[ \text{in} < \text{out} \]

\[ \text{out} = \text{in} \]
Cycle Resolution

After all that, we may still be left with cycles of unresolved ports. The general problem is the solution of a system of non-linear equations. Solution methods generally are iterative:

- The number of iterations, and cost of iterations is a principle issue.
- We explore “fixed-point” iteration.

Each iteration:
- freeze all input rates
- compute output rates based on frozen input rates
- compare new solutions with old for convergence

Our experiments define convergence when the relative difference between successive flow value solutions is less than \((1/10)\)% for all flow values.
Experiments

Topologies obtained from Rocketfuel database of observed Internet topologies

Traffic loads derived from Poisson-Pareto Burst Processes

We ask

– How many cycles form, as a function of load?
– How many iterations needed to converge, as a function of load?
– How fast does it run?
– What is speedup relative to pure packet simulation?
– What is the accuracy?
Results

Convergence behavior
- Examine # ports in cycle and iterations for convergence
- Vary topology
- 50% average link utilization

<table>
<thead>
<tr>
<th>Topology</th>
<th>#routers</th>
<th>#links</th>
<th>#flows</th>
<th>Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1</td>
<td>27</td>
<td>88</td>
<td>702</td>
<td>100</td>
</tr>
<tr>
<td>Top-2</td>
<td>244</td>
<td>1080</td>
<td>12200</td>
<td>2488</td>
</tr>
<tr>
<td>Top-3</td>
<td>610</td>
<td>3012</td>
<td>61000</td>
<td>2488</td>
</tr>
<tr>
<td>Top-4</td>
<td>1126</td>
<td>6238</td>
<td>168900</td>
<td>2488</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Topology</th>
<th>median #ports in cycles</th>
<th>#median iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-2</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>Top-3</td>
<td>40</td>
<td>9</td>
</tr>
<tr>
<td>Top-4</td>
<td>125</td>
<td>11</td>
</tr>
</tbody>
</table>

Dependency reduction is effective
Fixed point algorithm converges quickly
Results

We ask
– How fast does it run?
– What is speedup relative to pure packet simulation?
– What is the accuracy relative to packet simulation?

Topologies

<table>
<thead>
<tr>
<th>Topology</th>
<th>#routers</th>
<th>#links</th>
<th>#flows Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1</td>
<td>27</td>
<td>88</td>
<td>702 100</td>
</tr>
<tr>
<td>Top-2</td>
<td>244</td>
<td>1080</td>
<td>12200 2488</td>
</tr>
<tr>
<td>Top-3</td>
<td>610</td>
<td>3012</td>
<td>61000 2488</td>
</tr>
<tr>
<td>Top-4</td>
<td>1126</td>
<td>6238</td>
<td>168900 2488</td>
</tr>
</tbody>
</table>

Experiments run on PC
• 1.5 GHz CPU
• 3Gb memory
• Linux OS

For 1 sec time-step, faster than real-time on a model equivalent to 1.9G pkt-evts/sec (1K bytes/pkt)
Results

We ask
- How fast does it run?
- What is speedup relative to pure packet simulation?
- What is the accuracy relative to packet simulation?

Topologies

<table>
<thead>
<tr>
<th>Topology</th>
<th>#routers</th>
<th>#links</th>
<th>#flows</th>
<th>Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1</td>
<td>27</td>
<td>88</td>
<td>702</td>
<td>100</td>
</tr>
<tr>
<td>Top-2</td>
<td>244</td>
<td>1080</td>
<td>12200</td>
<td>2488</td>
</tr>
<tr>
<td>Top-3</td>
<td>610</td>
<td>3012</td>
<td>61000</td>
<td>2488</td>
</tr>
<tr>
<td>Top-4</td>
<td>1126</td>
<td>6238</td>
<td>168900</td>
<td>2488</td>
</tr>
</tbody>
</table>

Experiments run on PC
- 1.5 GHz CPU
- 3Gb memory
- Linux OS

Directly compare packet-oriented simulation, using exactly same input flow rates, on Top-1

\( \Omega(1000) \) speedup over wide range of loads

Results

<table>
<thead>
<tr>
<th>Link util.</th>
<th>speedup</th>
<th>Link util.</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>213</td>
<td>50%</td>
<td>3436</td>
</tr>
<tr>
<td>20%</td>
<td>1665</td>
<td>60%</td>
<td>3725</td>
</tr>
<tr>
<td>30%</td>
<td>2112</td>
<td>70%</td>
<td>1023</td>
</tr>
<tr>
<td>40%</td>
<td>2728</td>
<td>80%</td>
<td>1135</td>
</tr>
</tbody>
</table>
Results

We ask

– How fast does it run?
– What is speedup relative to pure packet simulation?
– What is the accuracy relative to packet simulation?

Experiments gather statistics of foreground UDP and TCP flows, comparing equivalent packet and fluid based background flows.

UDP foreground traffic is largely insensitive to difference in background flows.

TCP foreground traffic is insensitive to difference in background flows when link utilization is either low, or high. Significant variability observed in middle region.

Accuracy is sufficient for real-time training exercises that motivate this work.
Results

We ask

- How fast does it run?
- What is speedup relative to pure packet simulation?
- What is the accuracy relative to packet simulation?

Experiment: run on 3.2GHz Xeon cluster, 1,2,4,8,16,32 processors
# flows = 118,828 x # procs

Results

Phase III delay grows due to irregular load

32 processor problem finishes in 2.3 x the 1 processor problem
Conclusions

• Coarse scale simulation of network flows is a necessary component of large-scale network simulation
  – We’ve shown how to do it efficiently
    • Faster than real-time on large problems
    • Accurate enough for the training context for which it was designed
  – Parallelization is a different talk...