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 “Peformance 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 “Peformance 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:

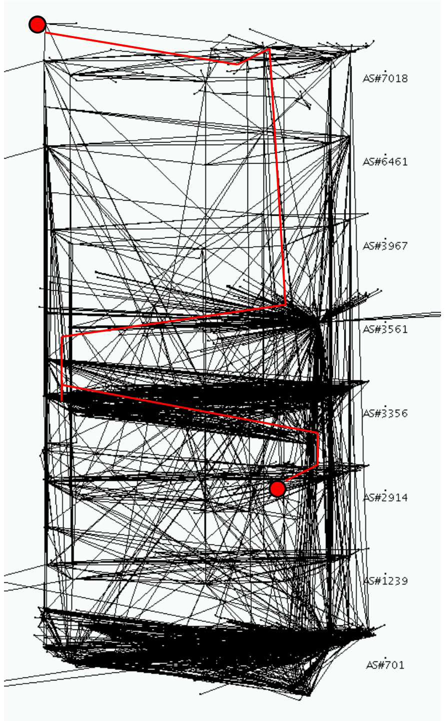
- Capture interactions between flows, routing infrastructure, fine scale traffic

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



AT&T

AboveNet

Exodus

Cable&Wireless

Level3

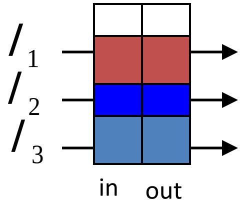
Verio

Sprint

UUNet

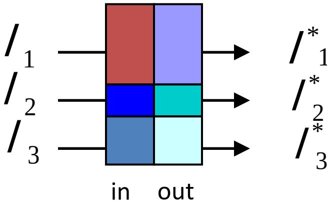
Modeling Congestion

$/_1 + /_2 + /_3 £ r$



No congestion

$/_1 + /_2 + /_3 > r$



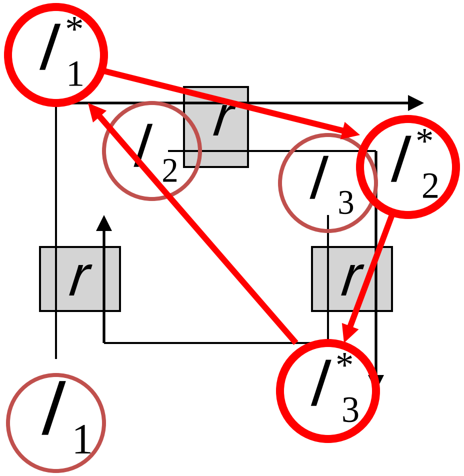
congestion

$\boxed{\text{Define } L = /_1 + \cdots + /_n}$

$$/^*_i = \begin{cases} /_i & \text{when } L £ r \\ r ´ /_i / L & \text{otherwise} \end{cases}$$
$$= /_i ´ \min\{1, r / L\}$$

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

$/^*_1$ depends on $/^*_3$
$/^*_2$ depends on $/^*_1$
$/^*_3$ depends on $/^*_2$



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….

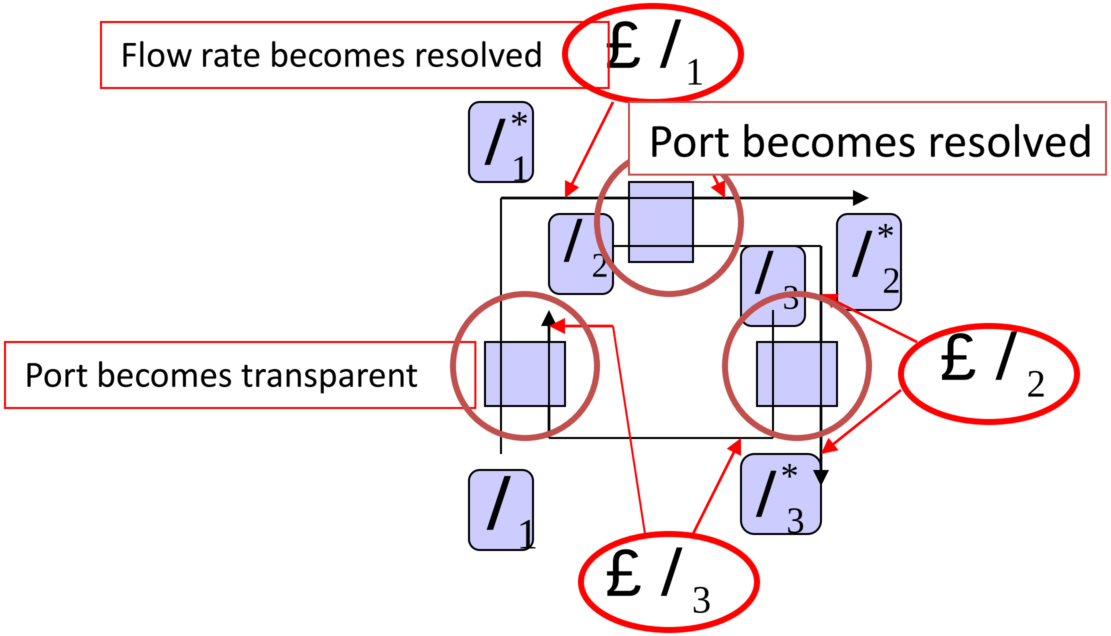
Notice that every output flow is bounded from above by input flow rate …. Every flow can be bounded by its ingress rate

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

Example : Suppose $/_1 + /_3 \pounds r$

Then

1. $/_1 + /^*_3 \pounds r$ so that $/^*_1 = /_1$
2. Port becomes resolved
3. Flows become resolved
4. 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

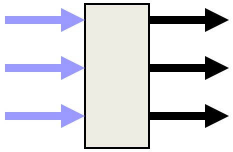
Port states are {*resolved, transparent, unresolved*}

Flow states are {*settled, bounded*}

Rule 1: port resolution

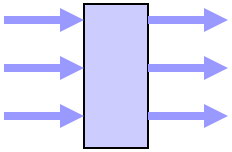
Pre-condition

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

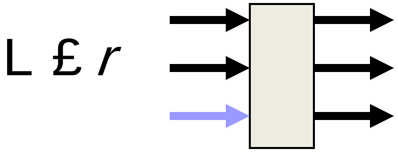
Port states are {*resolved, transparent, unresolved*}

Flow states are {*settled, bounded*}

Rule 2: port transparency

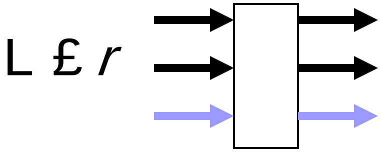
Pre-condition

Port state is *unresolved* and sum of input rate bounds is less than bandwidth,



Action

Mark port state as *transparent*. For every input rate that is *settled*, mark corresponding output rate as *settled*



State Change Rules

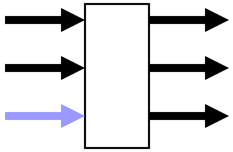
Port states are {*resolved, transparent, unresolved*}

Flow states are {*settled, bounded*}

Rule 3: settle state transition

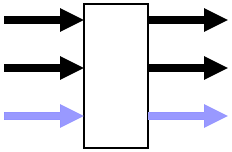
Pre-condition

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



Action

Mark corresponding output flow as settled, with value equal to input flow value



State Change Rules

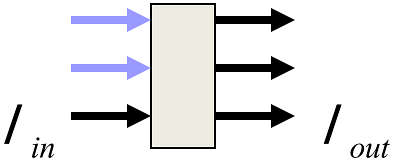
Port states are {*resolved, transparent, unresolved*}

Flow states are {*settled, bounded*}

Rule 4: flow bound transition #1

Pre-condition

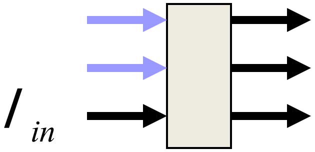
Port state is *unresolved,* the fair proportion relative to settled flows of an input flow rate exceeds bound on output flow



$$r´\,(l_{in}/f) < l_{out}$$

Action

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



$$l_{out} = r´\,(l_{in}/f)$$

f is sum of settled flow rates

State Change Rules

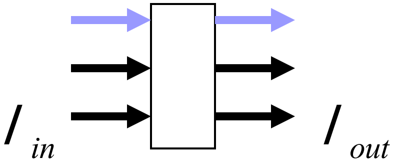
Port states are {*resolved, transparent, unresolved*}

Flow states are {*settled, bounded*}

Rule 5: flow bound transition #2

Pre-condition

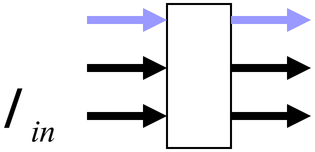
Port state is not *resolved,* the flow rate bound of an input flow is less than the corresponding output flow bound



$/_{in} < /_{out}$

Action

Set bound of output flow equal to bound on input rate



$/_{out} = /_{in}$

Cycle Resolution

After all that, we may still be left with cycles of unresolved ports

General problem is solution of a system of non-linear equations

- Solution methods generally iterative
    - The number of iterations, and cost of iterations is 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

Topology	#routers	#links	#flows	Mbps
Top-1	27	88	702	100
Top-2	244	1080	12200	2488
Top-3	610	3012	61000	2488
Top-4	1126	6238	168900	2488

Topology	median #ports in cycles	#median iterations
Top-2	20	5
Top-3	40	9
Top-4	125	11

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

Topology	#routers	#links	#flows	Mbps
Top-1	27	88	702	100
Top-2	244	1080	12200	2488
Top-3	610	3012	61000	2488
Top-4	1126	6238	168900	2488

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

Results

Topology	secs/time-step (20% link util.)	secs/time-step (50% link util.)
Top-1	0.0026	0.0026
Top-2	0.051	0.051
Top-3	0.283	0.285
Top-4	0.852	0.907

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

Topology	#routers	#links	#flows	Mbps
Top-1	27	88	702	100
Top-2	244	1080	12200	2488
Top-3	610	3012	61000	2488
Top-4	1126	6238	168900	2488

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

Results

Link util.	speedup	Link util.	speedup
10%	213	50%	3436
20%	1665	60%	3725
30%	2112	70%	1023
40%	2728	80%	1135

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

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

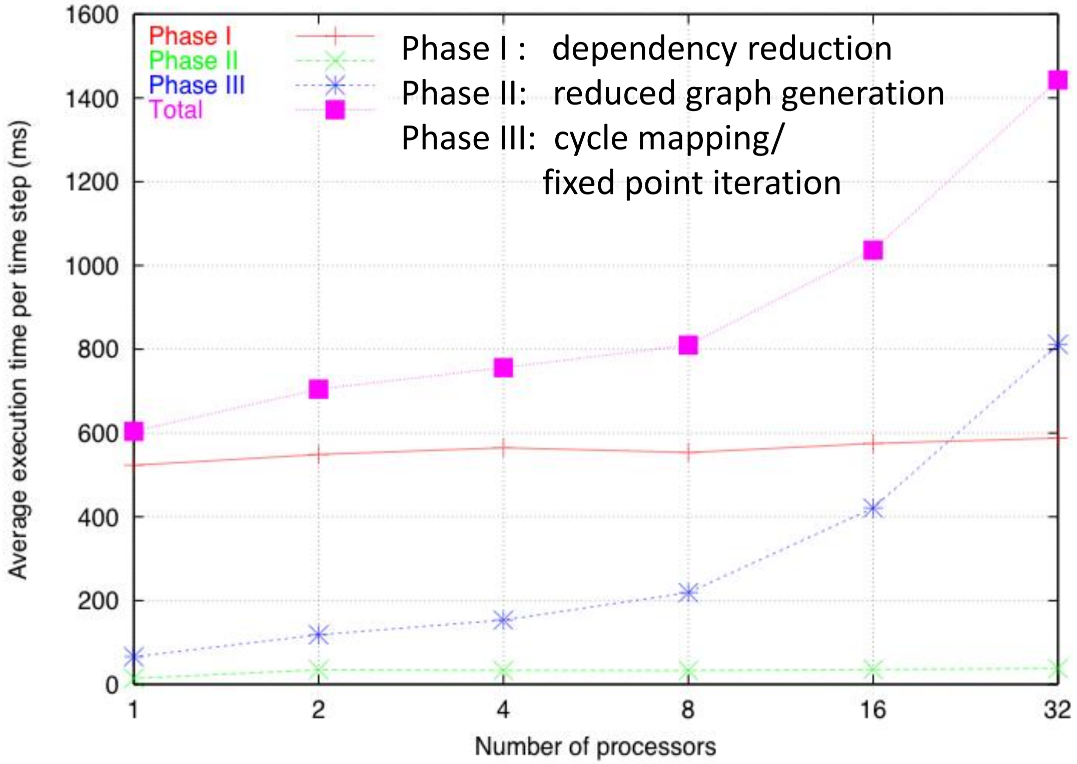
- How fast does it run?
- What is speedup relative to pure packet simulation?
- What is the accuracy relative to packet simulation?

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



Phase I : dependency reduction
Phase II: reduced graph generation
Phase III: cycle mapping/
fixed point iteration

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…