So you think you want to simulate a network?

Mark Handley
Professor of Network Systems
UCL

Why do you want to do network simulation?

- Understand a problem.
- Demonstrate your ideas work.
- Demonstrate your ideas are better than the competition.

- Publish papers.
- Finish your PhD.
- Fame and fortune.

- But seriously… what do you think you will get from simulation?

The limits of simulation

- Understand a problem.
- Demonstrate your ideas work.
- Demonstrate your ideas are better than the competition.

- How good a programmer are you?
  - Can you write bug free code?

Simulating a bug…

- You're doing network simulation because you don't understand the system you're simulating.
  - If you understood it completely, there's no need to simulate.
- When you get results, what do they mean?
  - Did you get what you expected?
  - Usually no.
  - Was your intuition wrong, or did you simulate a bug?

Software engineering.

- With most computer software, the end results are part of the specification.
  - Payroll gets processed, Airbus doesn't crash, etc.
- With network simulation, you're usually simulating something novel that you don't completely understand.
  - The end results are unspecified.
  - How do you know when you've succeeded?

Software engineering.

1. **Write unit tests.**
   - Test every single part of your code in isolation.
   - Automate your unit tests. Make sure you run them after every change.
   - Yes, it's tedious, but debugging a complete simulation is worse.

2. **Use a version control system.**
   - CVS, subversion, whatever.
   - When you get an interesting result, tag it in CVS, and note down the tag in your log book.
   - You *will* want to go back at some point later, when you've broken the code again.

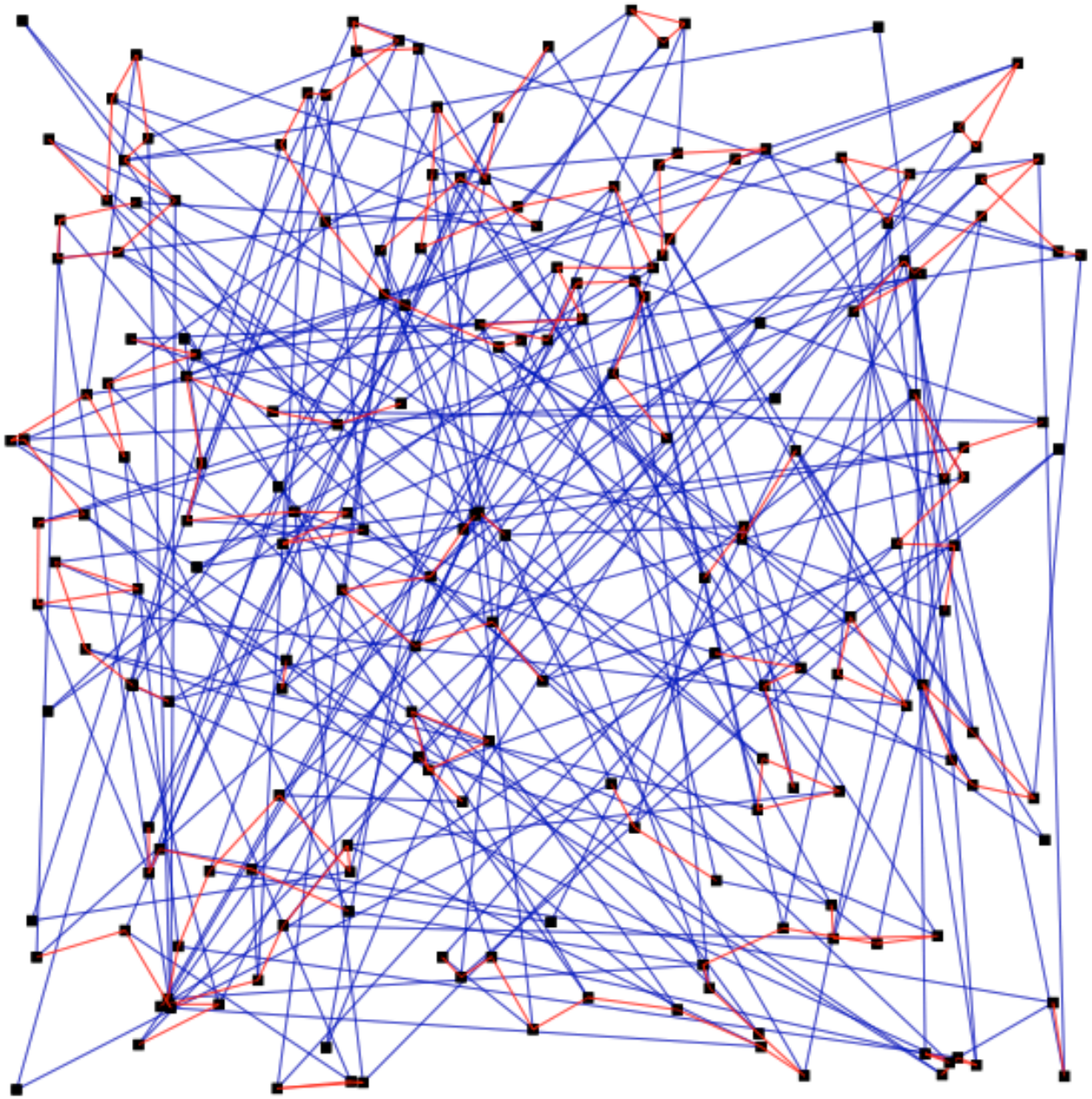
Software engineering.

3. **Start simple.**
   - Run a two-node simulation.
   - Run a three-node simulation.
   - Completely understand these before moving on.

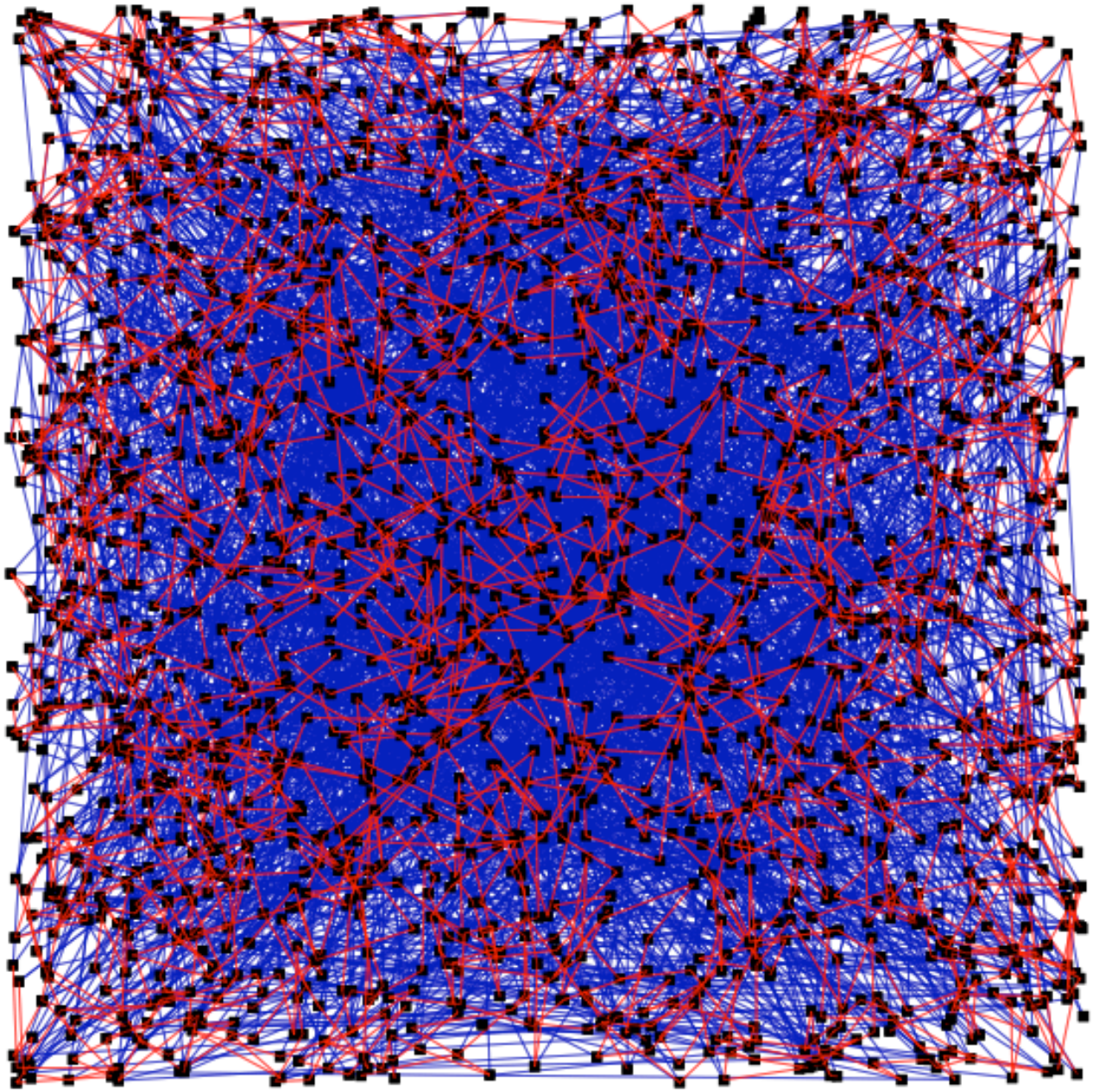
4. **Start simple.**
   - Leave out all the optional features of your protocol on the first pass.
   - Use very simple solutions that don't scale but that you can understand. Eg a linked list, rather than some super fancy efficient hash-table-balanced-tree-thingummybob.
   - Then you'll have a baseline for comparison when you write the scalable efficient version.

Understand your simulation.

- If you're simulating complex topologies, usually you'll be simulating buggy topologies.
  - You need to get an intuitive feel for the topology.
  - Print it out, or view it in nam.



200 nodes



2000
nodes

Simple postscript driver

```
%!
0.001 setlinewidth
36 36 translate
72 7.5 mul 72 7.5 mul scale
/drawbox { newpath 0 0 0 setrgbcolor moveto -0.005 -0.005 rmoveto
  0.01 0 rlineto 0 0.01 rlineto -0.01 0 rlineto 0 -0.01 rlineto
  fill } def
/redline { newpath 1 0 0 setrgbcolor moveto lineto stroke } def
/blueline { newpath 0 0 0.7 setrgbcolor moveto lineto stroke } def
0.469309 0.816928 drawbox
0.469309 0.816928 0.043651 0.769113 blueline
0.469309 0.816928 0.604013 0.968648 blueline
0.577478 0.242383 drawbox
0.577478 0.242383 0.642284 0.473466 blueline
0.577478 0.242383 0.069395 0.407594 blueline
...
0.532045 0.122970 drawbox
0.532045 0.122970 0.540032 0.040943 redline
0.532045 0.122970 0.629494 0.101863 redline
showpage
```

Software engineering summary.

- Writing simulation code is harder than writing normal applications.
- You will *definitely* be simulating bugs for the first six months if you don't start simple!

Abstraction

Abstraction and Modelling

- Simulation is about modelling a system.
- Big problem:
  - *Real Internet is too complex to model.*
  - Too big.
  - Too heterogeneous.
  - Not stationary.
  - Effects are often local.
  - Topologies are secret.
  - Configuration is secret.
  - Real systems don’t conform to the specs (if there are any specs)

Essential Reading

- S. Floyd and V. Paxson, *Difficulties in Simulating the Internet,* IEEE/ACM Transactions on Networking, Vol.9, No.4, pp. 392-403, August 2001.

  http://www.icir.org/vern/papers.html

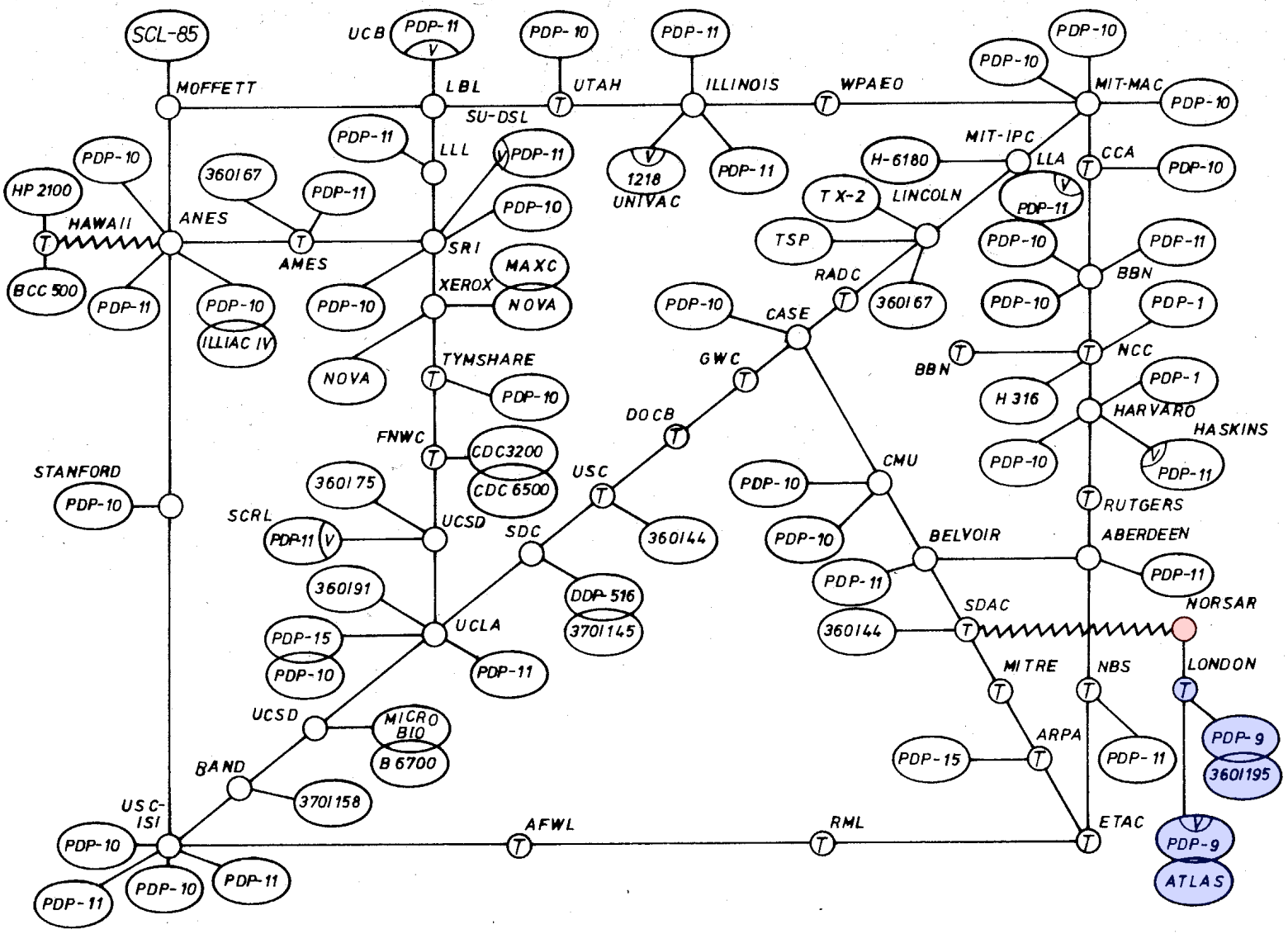
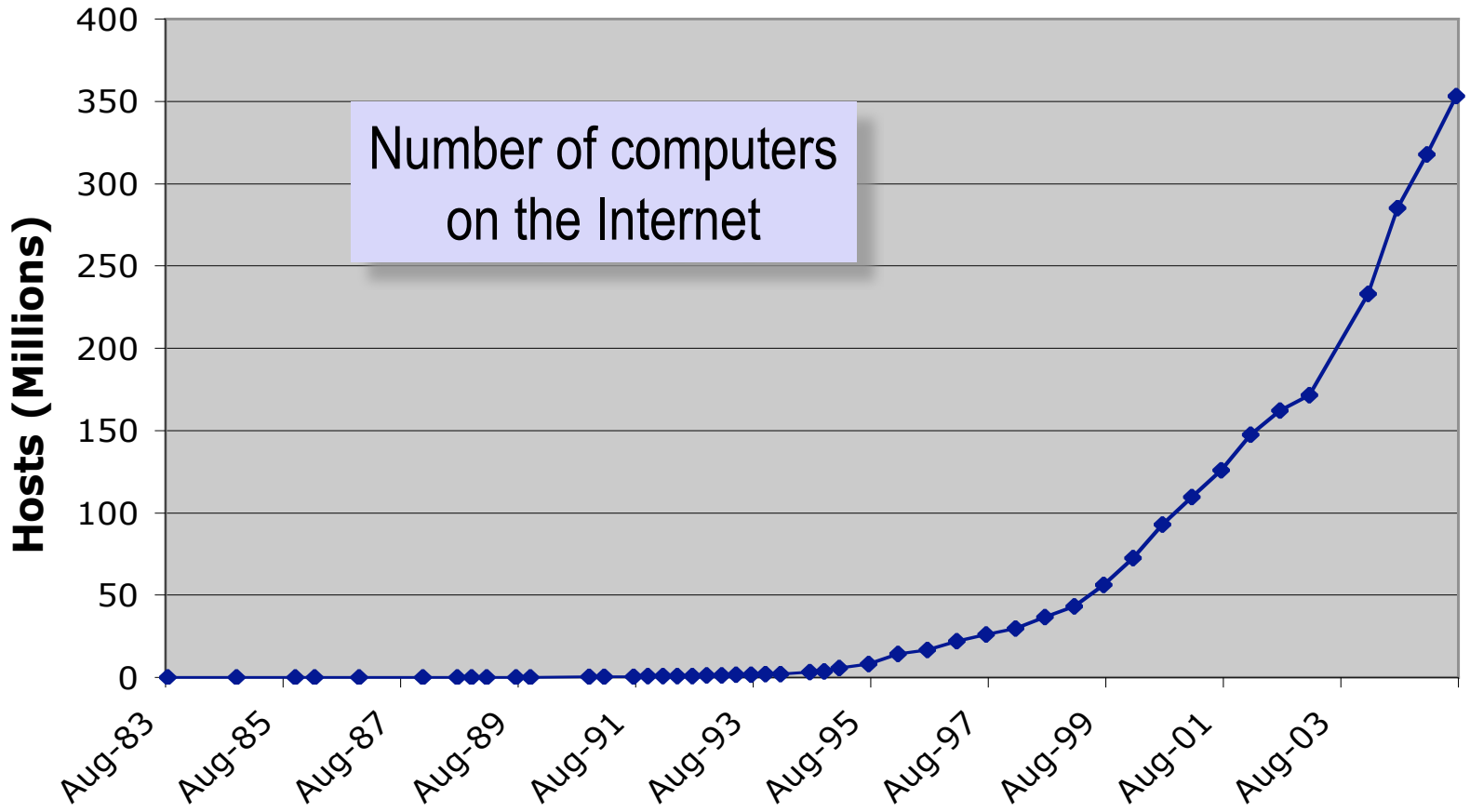


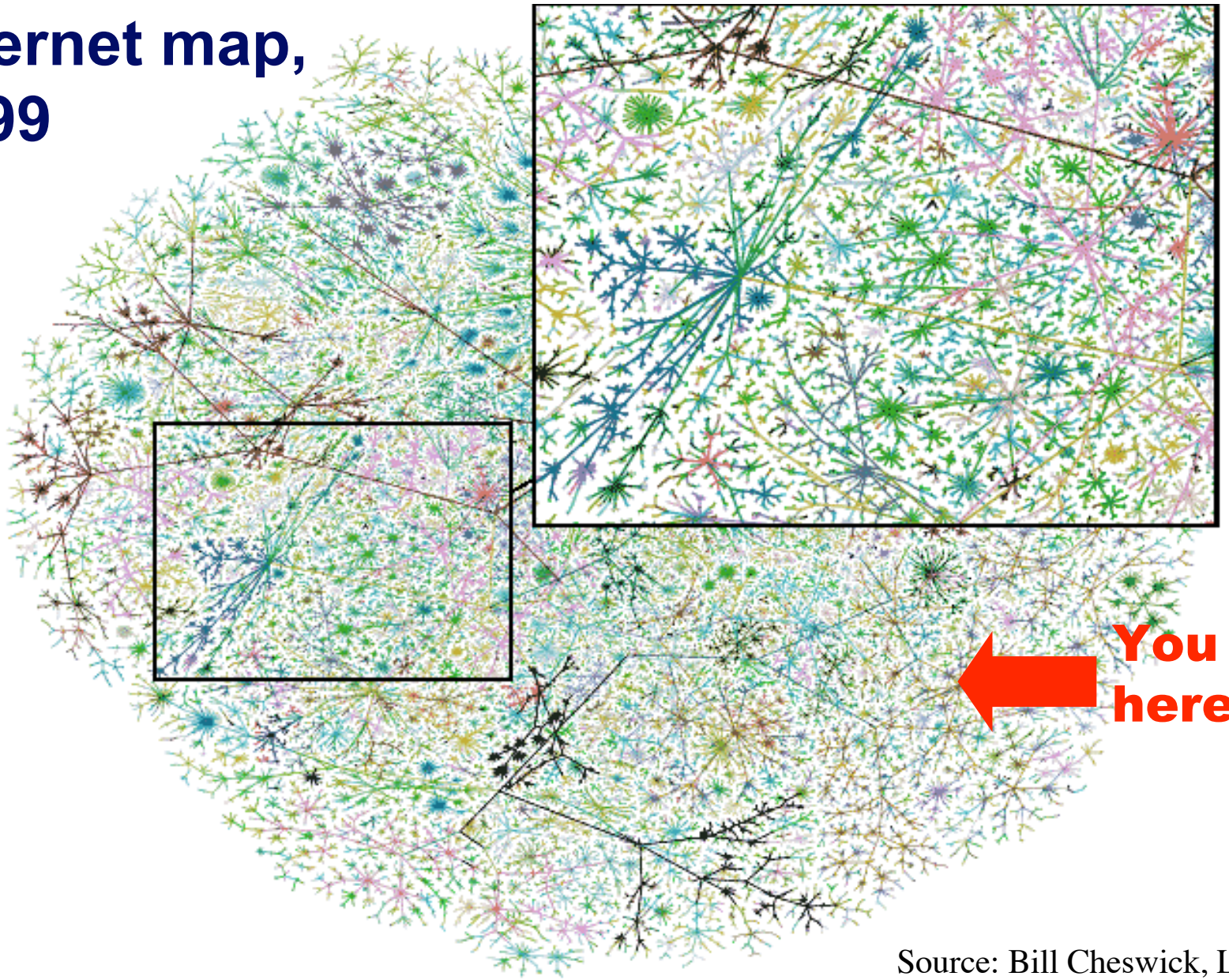
Abb. 4 ARPA NETwork, topologische Karte. Stand Juni 1974.

Scale Problems



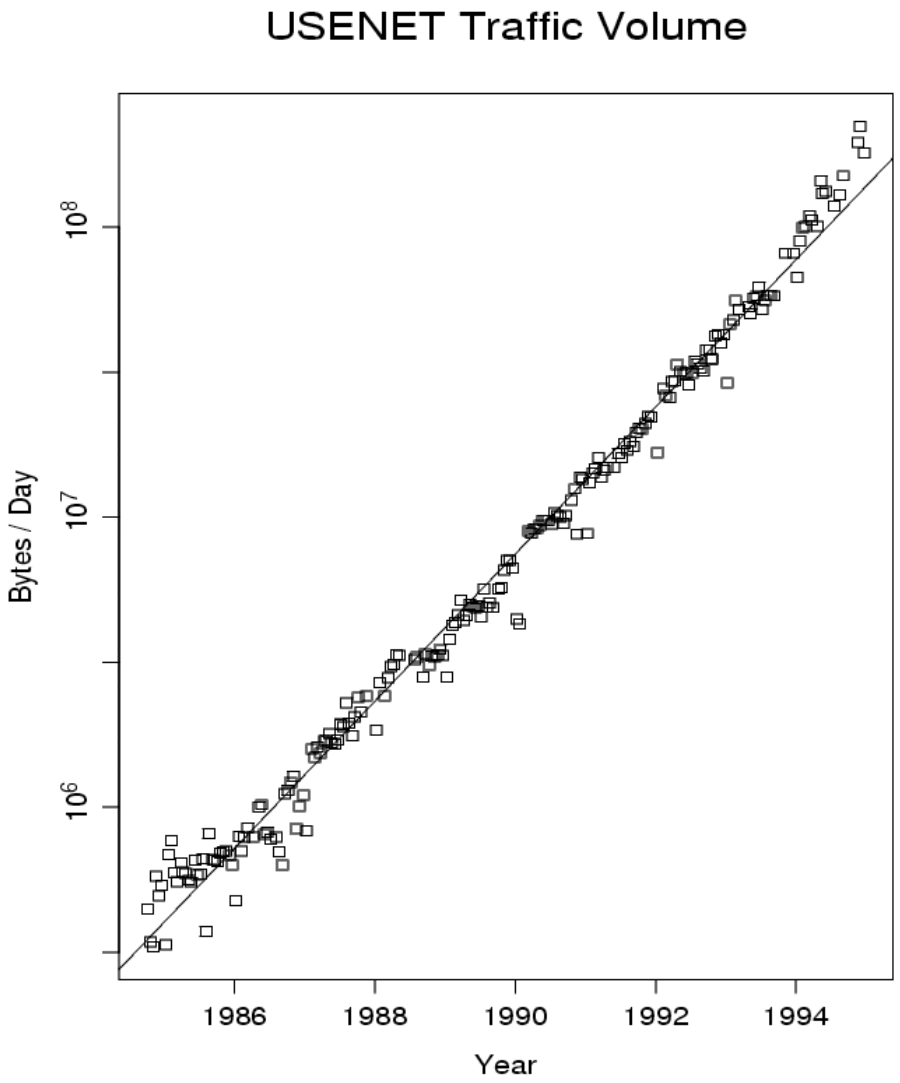
Source:Internet Software Consortium (http://www.isc.org/)

Internet map, 1999



Source: Bill Cheswick, Lumeta

Usenet traffic growth



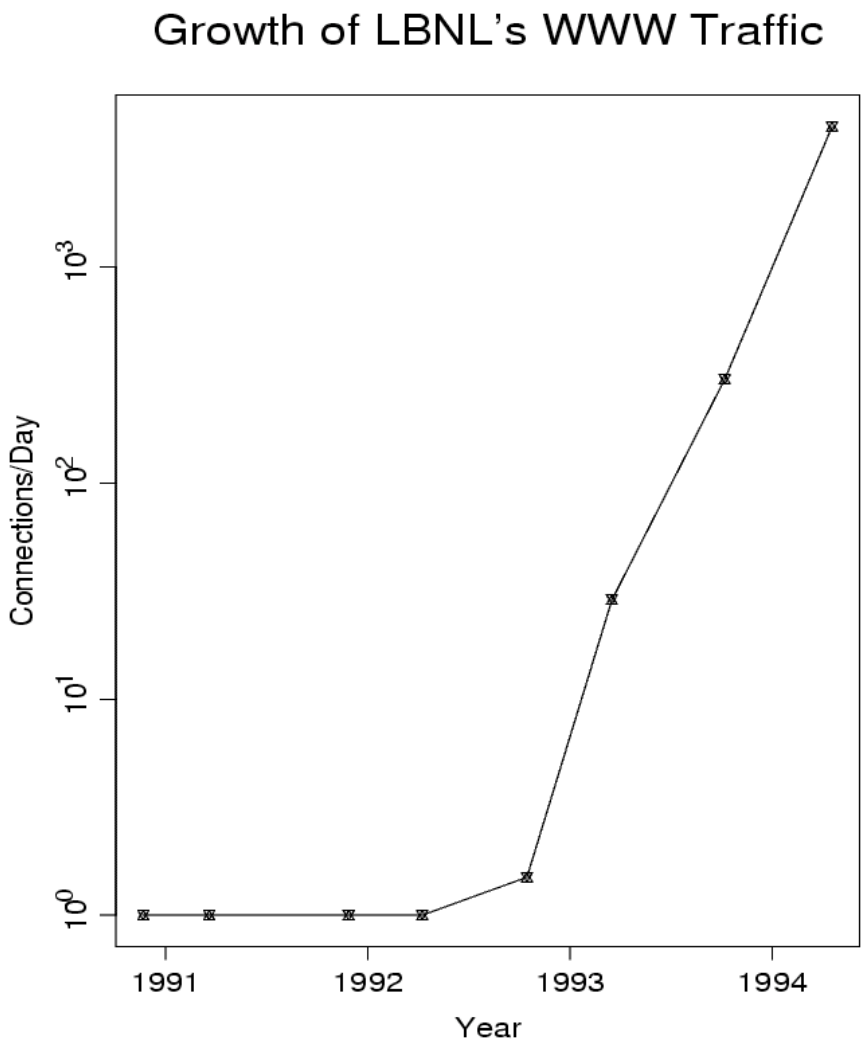
- Some future properties of the Internet are predictable

Median size of FTP transfers at LBNL

Oct 1992	4,500 bytes
Mar 1993	2,100 bytes
Mar 1998	10,900 bytes
Dec 1998	5,600 bytes
Dec 1999	10,900 bytes
June 2000	62,000 bytes
Nov 2000	10,000 bytes

- Median is normally considered a robust statistic.
- Need to take care not to assume that properties of the net are predictable over time.

Web traffic at Lawrence Berkeley Lab



- If you were simulating in 1992, you'd simulate the wrong Internet.

Abstraction

- You can't model the Internet, so you need to abstract out only the parts of the Internet that are relevant to your problem.

- Problem:
  - Which parts are relevant?

Analysis vs. Simulation

- Analysis gives you complete control of a model.
- Gives a greater understanding of the basic forces at play.
- Risk: to make problem tractable, you've simplified the model to the point where the results are useless.

- Simulation *complements* analysis.
  - Provides a validity check.
  - Allows more detail to be modelled.
  - However:
    - May provide less fundamental understanding.
    - Only serves as a validity check if you didn't make the same oversimplifications.
    - Harder than analysis to verify that it implements your model.

Role of Simulation

Simulation is most useful for:

- Understanding dynamics.
- Illustrating a point.
- Searching for unexpected behaviour.

Simulation is less useful (or downright dangerous) for:

- Generating absolute results.
- Comparing two solutions.

  "*Solutions A performs 23% better than solution B*"

  Both solutions may be sensitive to parameters.

  At best, your parameters, topologies, traffic mix, etc, are a rough approximation to the real world.

Simulation for Comparison

- If you can carefully define the model, you *may* be able to use simulation for comparison.
  - Need to simulate a *wide range of parameters* to demonstrate lack of parameter sensitivity.

- *Release your source code*, so others can validate your results with different parameters.
  - If you do not trust your simulation enough to publish the source code, you've no right to publish the results obtained from your simulator.

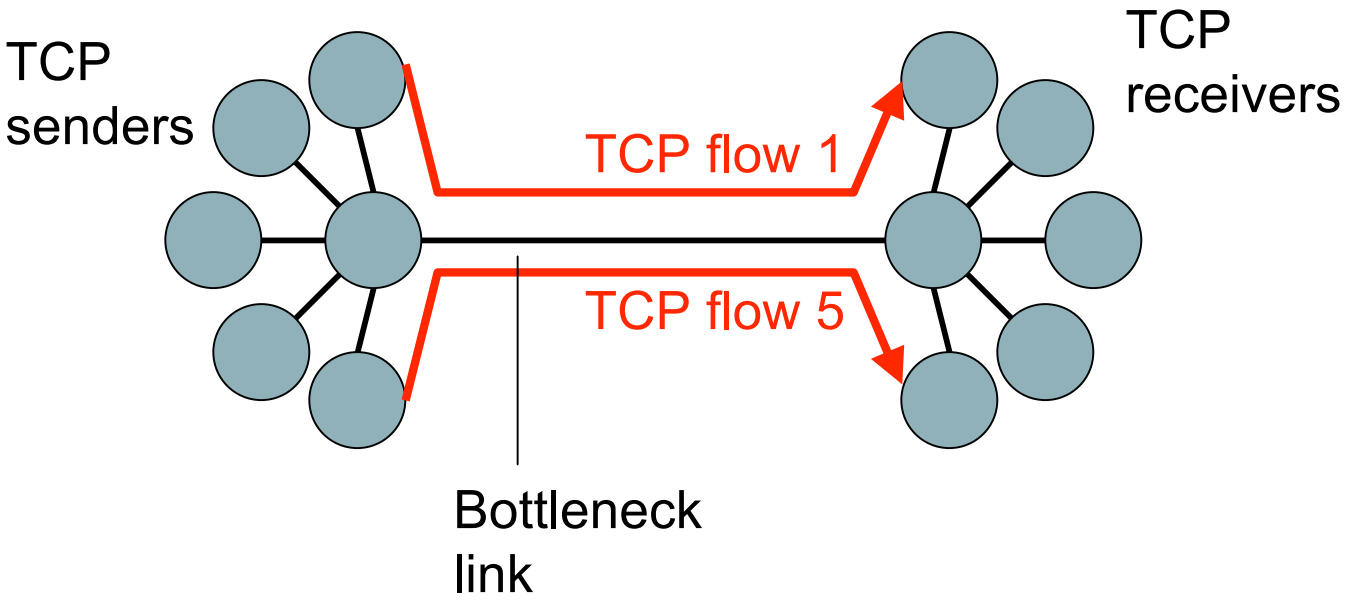
Abstraction

- We know we don't completely understand the Internet.
  - How do you choose a model that keeps the important properties and abstracts away the rest?

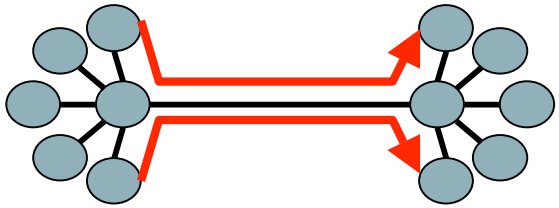
- Best you can hope for is to explore the simplified model and learn from it.
  - Simulation *cannot* demonstrate that a system will perform well in the real world.
  - Simulation *can* demonstrate that system performance is not especially sensitive to parameters.

Example: Congestion Control

- Simple “dumbell” topology is commonly used:



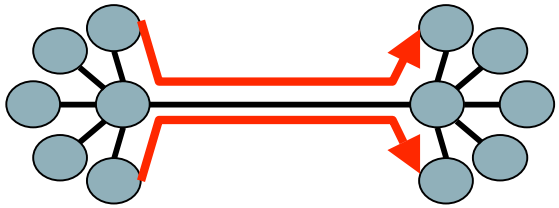
Dumbell Topology



- Is this a good model of the real world.
  - No.
- Does it capture the important properties for understanding congestion control dynamics?
  - Maybe, if you're careful.
  - It's simple enough you *might* understand your results.

- It misses out the effects of multiple congested links.
  - Need to simulate this separately.

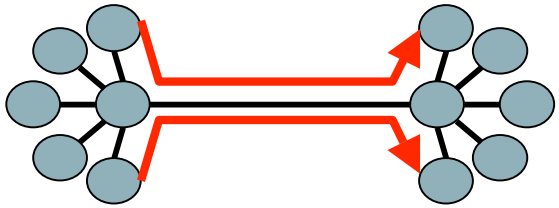
Dumbell Topology



Basic parameters:

- Number of flows.
- Link speeds
- Propagation delays
- Queues:
  - Droptail, RED, other.
  - Queue size
  - RED parameters
- TCP variants used

Dumbell Topology

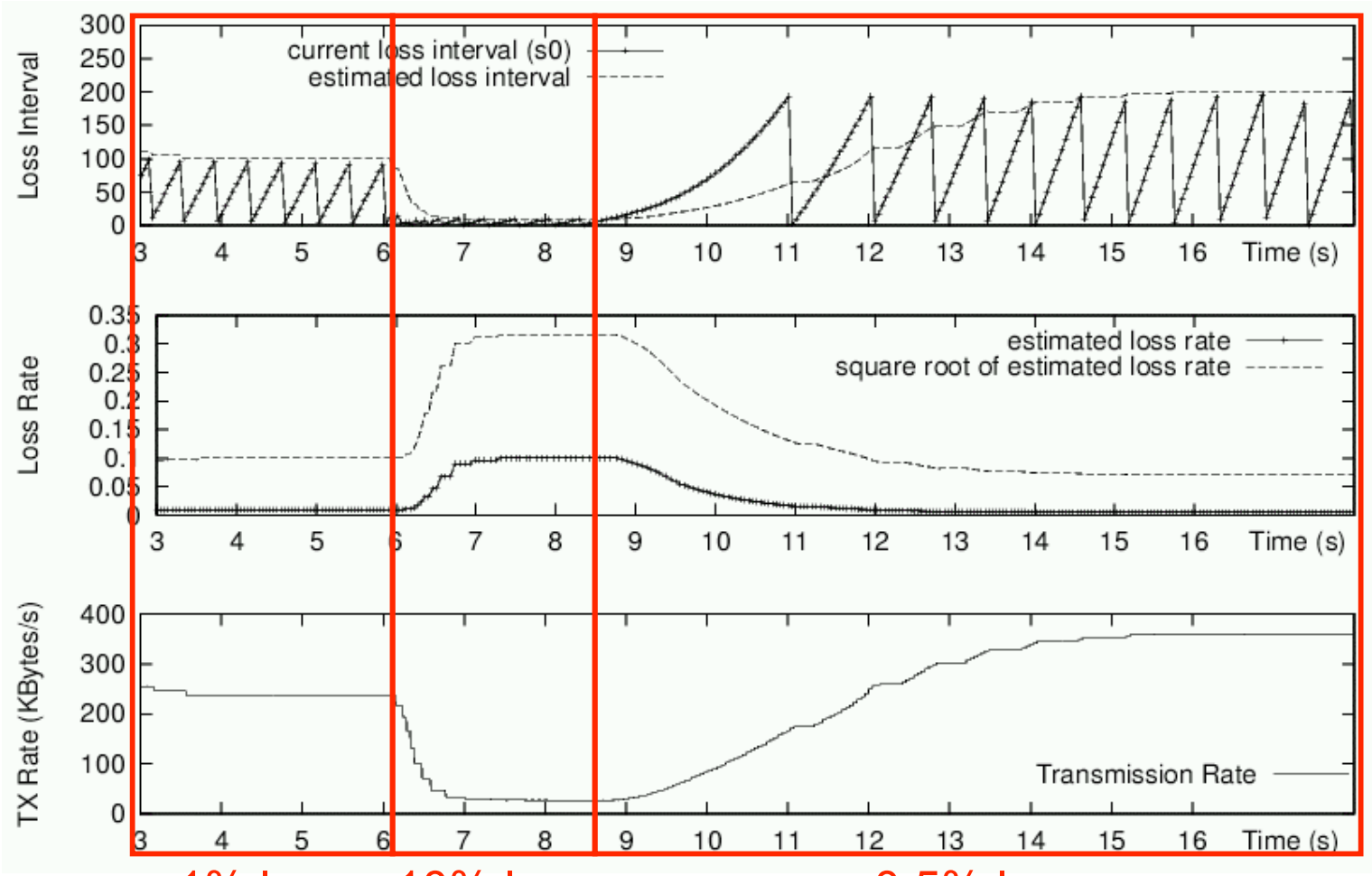


Basic parameters:

- Number of flows (1-1000 flows?).
- Link speeds (28Kb/s - 1Gb/s?).
- Propagation delays (1ms - 500ms?).
- Queues:
  - Droptail, RED, other.
  - Queue size (0.25 - 2 RTT*BW?).
  - RED parameters (min_th, max_th, max_p???).
- TCP variants used (tahoe, reno, newreno, sack?)

Dumbell: start simple

Example: TFRC controlled experiment

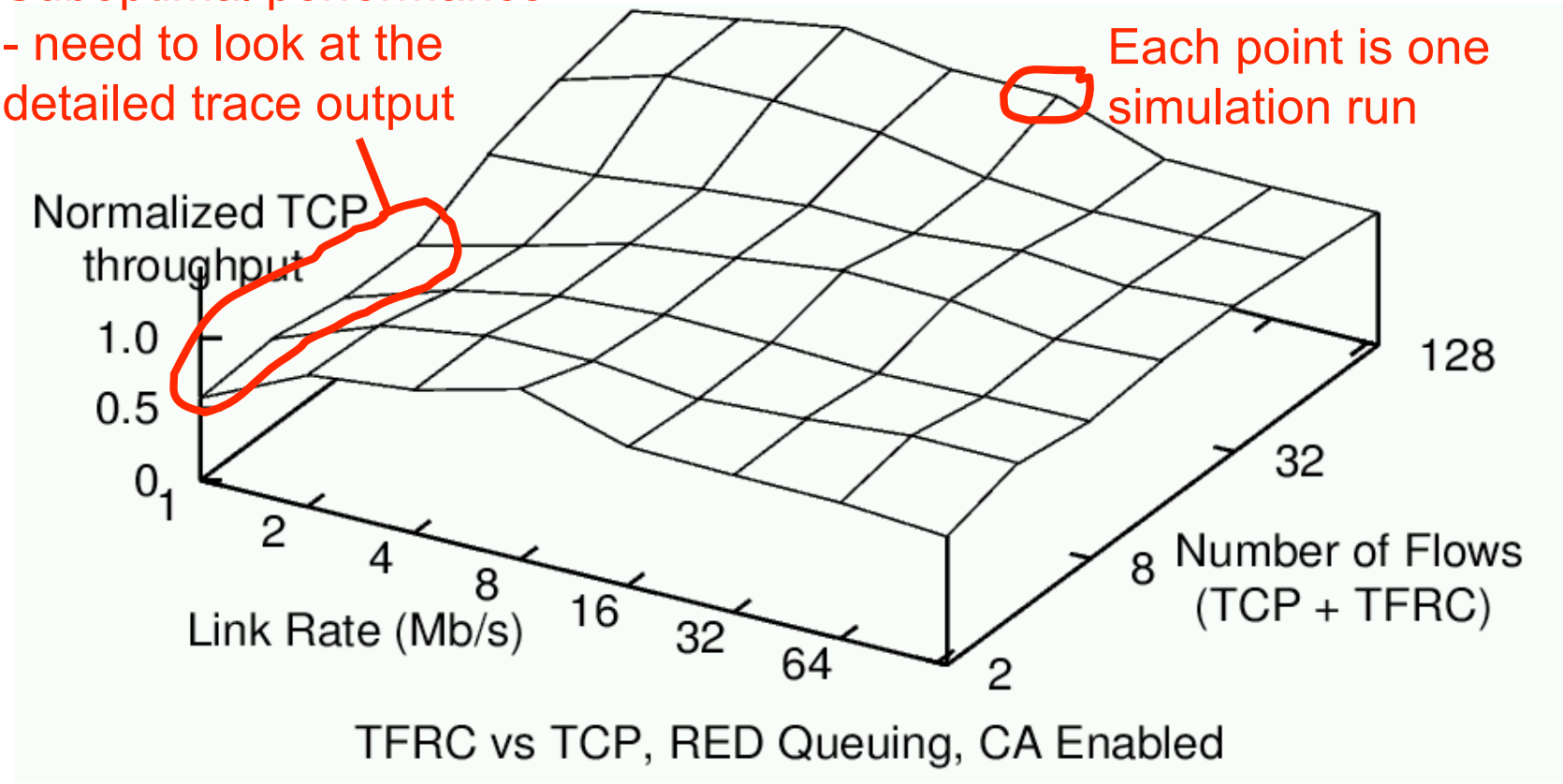


1% loss 10% loss 0.5% loss

Dumbell: Exploring the parameter space

Suboptimal performance
- need to look at the
detailed trace output

Each point is one
simulation run



TFRC vs TCP, RED Queuing, CA Enabled

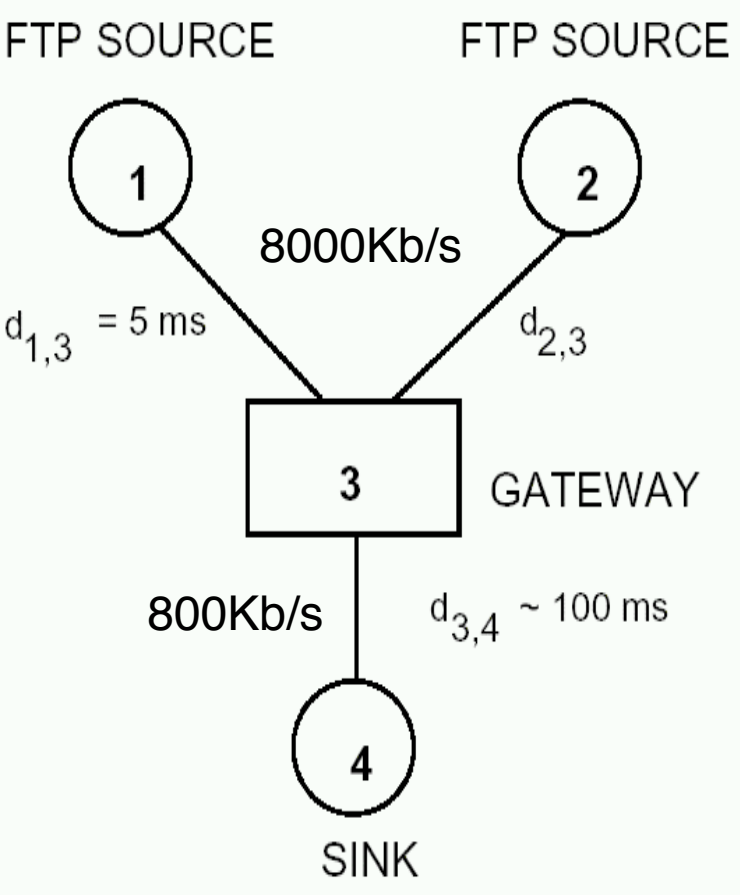
Phase Effects.

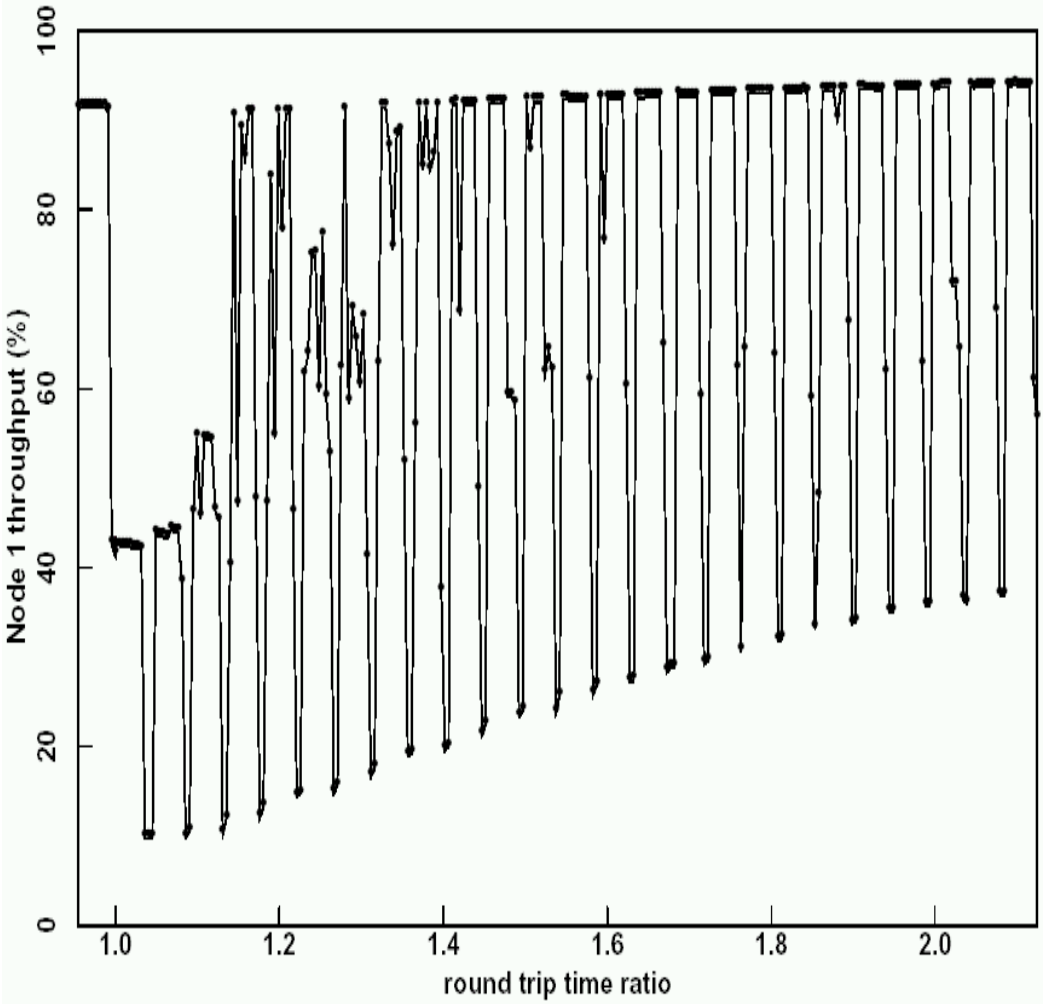
Essential reading:

- Sally Floyd, Van Jacobson, *On traffic phase effects in packet switched gateways*, Journal of Internetworking: Practice and Experience, Vol 3, No. 3, Sept 1992.

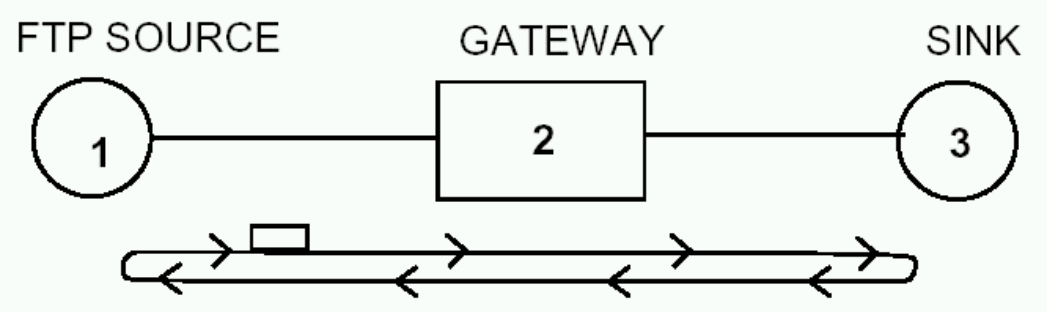
  http://www.icir.org/floyd/papers.html

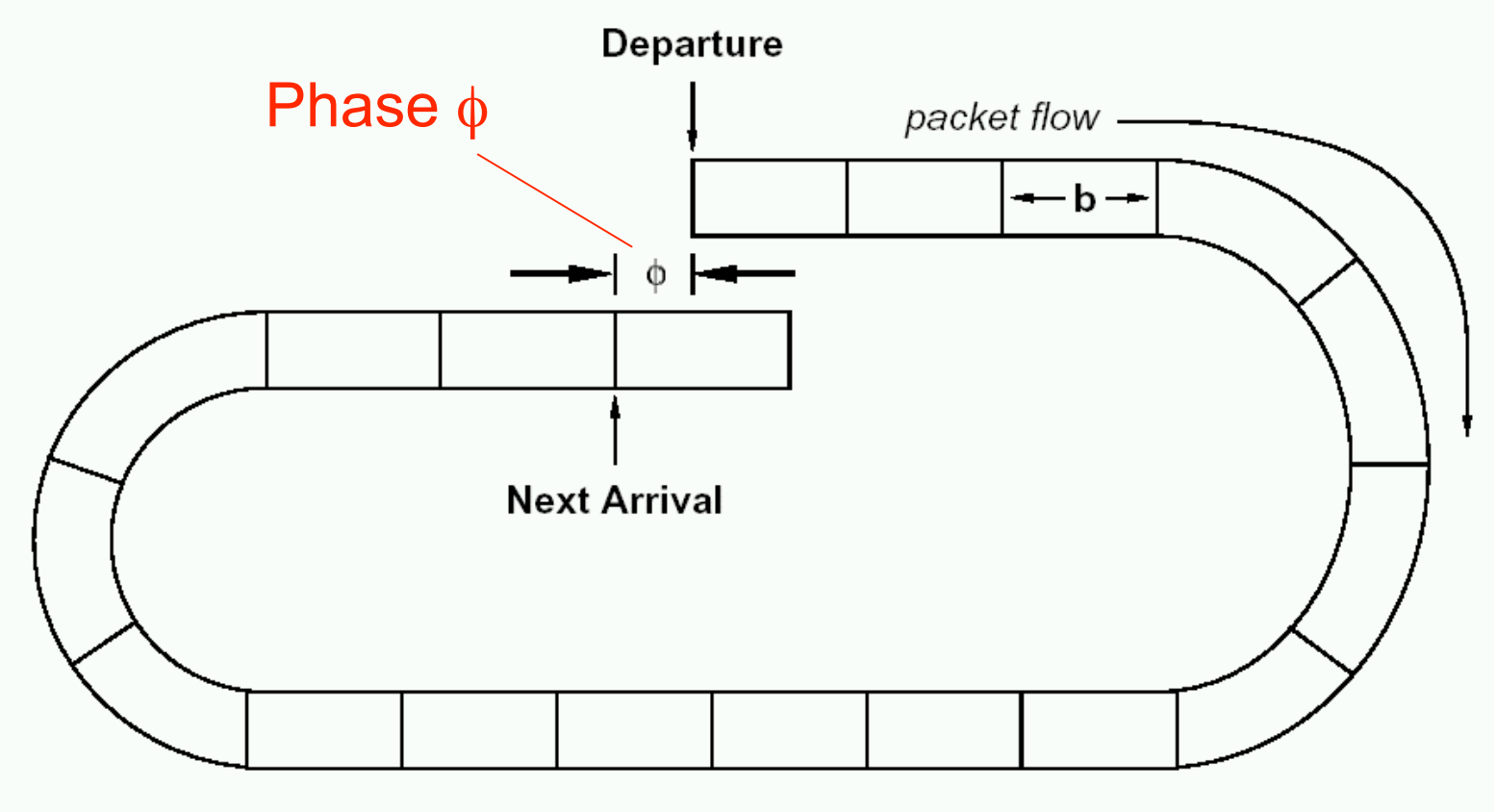
Phase effects



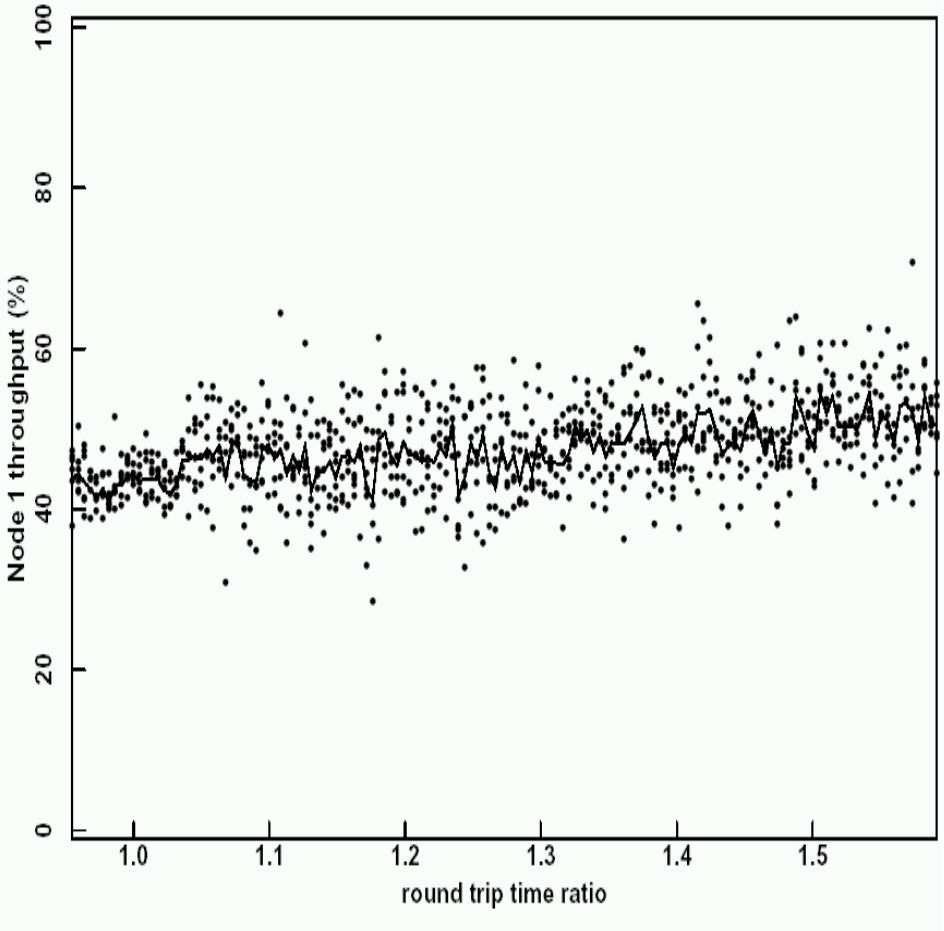


Phase


FTP SOURCE
GATEWAY
SINK
1
2
3

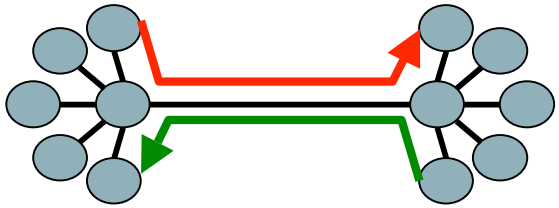

Departure
Phase ϕ
packet flow
b
ϕ
Next Arrival

Random Drop to Avoid Phase Effects



- RED is very effective at avoiding phase effects.
- But you really need to understand RED to set the parameters properly.
- Few real routers deploy RED, so does this make your simulations unrealistic?

Reverse Traffic



- Adding some TCP flows in the reverse path can break up phase effects.
  - Small ACK packets in forward path help reduce periodicity.
  - ACK-compression in reverse path increases TCP's burstiness.

- In general, simulating without reverse path traffic is risky, as ACK-compression can often be a non-trivial side-effect.

Ok, so it works now. That's good!

- This only modelled steady state congestion behaviour.
- Now we need to look at behaviour when background load changes.
  - Impulse behaviour.
  - "Realistic" background traffic.
- Now we need to look at behaviour with multiple bottlenecks.

Background traffic

- Real traffic stats vary greatly from place to place, and time to time.
  - Could gather many real world traces to drive your simulations.
  - That would add realism.

- Problem:
  - Real world protocols are adaptive.
  - Cannot just replay a trace from one environment in another environment.

Background traffic.

- Can use real traffic traces to model source behaviour.
  - I.e., model when a flow starts.
  - Then use ns's TCP model to do the packet-level behaviour.
- Problem:
  - Real world sometimes has a second feedback loop.
  - User doesn't click on next web link before current on has finished loading.
  - May need to model this too.

Background traffic:

Simplistic example.

A new TCP flow starts every second, sends 1,000,000 bytes, then stops.

- If the link is *greater than 8Mb/s*, the first connection may complete before the second connection starts.
- If the link is *less than 8Mb/s*, the first connection will not complete before the second connection starts.
  - Second connection will take more time that the first one, as it competes with the first one.
  - Number of active connections grows linearly with time.

- Not a viable model.
  - Trace-driven simulation can suffer the same problem.

Background Traffic: some invariants.

- What should you model?
  - **Poisson session arrivals.**
    - The arrival of user sessions (eg web browsing session) is well described using Poisson models.
  - **Log-normal connection sizes.**
    - Distribution of the logarithm of connection sizes is well approximated with a Gaussian distribution.
  - **Heavy tailed distributions.**
    - Although the body of the distribution is log-normal, the upper tails of the distribution may be better modelled as a heavy tailed distribution.

OK, done all that. I'm done!

Can I have a PhD please?

What have you actually learned?

- Explored a large parameter space in steady state.
- Explored the dynamics.
- Explored simple and richer topologies.

Did you find where it breaks?

- If not, go back and try harder - you need to understand the limits.

And you *still* can't say it will work well in the Internet.

- But it may be worth real-world experiments now to validate your simulations.

Mobility

- A great deal of simulation today concerns mobile systems.
- There is almost no data on how mobile systems move.
  - The results in mobility simulations often critically depend on the mobility model.
  - This means your mobility simulations have almost no predictive power for real-world deployment.

- Explore the parameter space.
  - Try to design algorithms that are robust, even if they aren't optimal.

Writing a paper.

- When you write a paper, no doubt you'll include a lot of pretty graphs.
  - Each graph has a lot of data points from a lot of simulation runs.
  - It's very each to lose track of which simulation version produced which graph.
  - End up with the best graphs in your paper, but no one version of your code has all these good features.

- "`make paper`"
  - Consider using a Makefile to check out the relevant version of your simulator from CVS, re-run all the simulations, re-generate all the graphs, and latex the final paper.
  - We did this with the TFRC Sigcomm paper.

When to use ns

- Ns is a pretty good simulator for packet-level simulation.
- Tcp models are good.
- Useful traffic models.
- Useful topology generators.
- Models are improving, but beware:
  - If you don't understand a model, don't trust it.

When to write your own simulator.

Not a packet-level simulation.

- A lot of peer-to-peer simulation can be done at the flow level. Ns doesn't help you.

Scale.

- Ns doesn't scale to very high link speeds or very large numbers of flows. To do this you need a packet-level simulator that makes additional assumptions.
- Ns can't simulate huge topologies. To do this, you need to simulate at a higher level of abstraction.

Routing.

- Ns can do routing, but it's not great at it. Unless you need a packet-level simulation of routing, don't use ns.
- Inter-domain routing is usually simulated at an AS-level, not at a router level.

Why do you want to do network simulation?

- **Understand a problem.**
  - Good reason.
  - Did you understand all the factors of the system?
- **Demonstrate your ideas work.**
  - OK, but work under what circumstances?
  - Did you model the important properties?
  - Did you predict the future correctly?
- **Demonstrate your ideas are better than the competition.**
  - Risky.
  - Need to explore a large parameter space.
  - Need to debug their code too.
  - Robust solutions better than optimal ones.