Fast and accurate network simulation

Henri Casanova[1,2]

[1]Associate Professor
Department of Information and Computer Science
University of Hawai'i at Manoa, U.S.A.
[2]Visiting Associate Professor
National Institute of Informatics, Japan

NII Seminar Series, October 2013

Acknowledgments

- This is joint work with
  - P. Velho, UFRGS, Brazil
  - L. Mello Schnorr, UFRGS, Brazil
  - A. Legrand, CNRS Grenoble, France

Introduction

- In the previous seminars we've seen theoretical results
  - Optimal alg., approx. alg., $\mathcal{NP}$-completeness
- Theory is key to understanding problems better
- Theoretical results also provide motivation and inspiration for developing non-guaranteed *heuristics*
  - e.g., when there is no optimal algorithm
  - e.g., non-guaranteed algorithms may achieve much better average performance than approximation algorithms
- Sadly, we rarely have theorems to compare heuristics
- Typically, given 20 (reasonable) heuristics and 10,000 random problem instances, each heuristic could be best on some of the instances
- Question: How do we find out which heuristics are best?

Unrealistic assumptions

- In many cases the "view of the world" of the algorithm is not the real world
  - Example: a heuristic is designed for homogeneous hosts in a cluster, but in practice hosts are a bit heterogeneous
  - Example: network latencies would make the problem $\mathcal{NP}$-complete, an optimal algorithm is known when there are no network latencies, and it can be applied in real-world networks in which there are latencies
  - Example: network topologies are so complex that designing algorithms that truly exploit them is too challenging. Furthermore, in practice one often doesn't know the network topology!
- Question: How do we find out how algorithms behave in the real world?

Experiment!

- We can't compare heuristics in theory, and in general we don't really know how they would behave in the real world
- So we have to run empirical experiments
  - Create a set of problem instances in the real world
  - Run the heuristics for these instances
  - Do some statistics and try to obtain useful conclusions
- Unfortunately, running experiments is not easy, especially at large scale...

The trouble with experiments (I)

- Experiments are labor-intensive
- To run an experiment on a real-world platform, we need a full-fledge implementation of the studied application/system
- We do not always have such an implementation!
  - e.g., we want to run simulations to decide how to implement the application/system!
  - e.g., we want to answer research questions without necessarily having a particular application at hand with all the required input datasets, etc.
- Developing a full implementation "just" to study scheduling algorithms it not necessarily something one wants to do
  - Or at least not until much later

The trouble with experiments (II)

- Experiments can be costly in time, ¥, and Watts
- As the target platform increases in scale (e.g., large number of hosts) and/or the application increases in scale (e.g., larger data, larger amounts of computations), so does the time for each experiment
- Longer experiments imply larger cost and larger power consumption
- To make matters worse, we typically need large numbers of experiments to achieve reasonable statistical significance

The trouble with experiments (III)

- Experiments are limited in scope
- The set of experimental scenarios is constrained by the platform configurations at hand, and exploring a wide range of configurations may not be possible
- In fact, production platforms may not be available at all, limiting experiments to limited testbeds
- It it difficult to explore hypothetical "what if?" scenarios
    - "What if the network was twice as congested?", "What if network paths were twice as long?", "What if all hosts were 16-core instead of 8-core?"
    - Enabled to some extent by emulation/virtualization techniques, but not possible on all platforms

The trouble with experiments (IV)

- Experiments are not always perfectly repeatable
- As soon as the target platform becomes large (e.g., several large clusters interconnected over wide-area networks) it is typically subject to external and/or unpredictable load conditions
  - A shared network, perhaps even shared hosts
  - Unscheduled downtimes or performance bugs
  - Changing software configurations
- As a result, the same experiment may lead to different results on different days
- The simple approach that consists in running experiments "back-to-back" is not satisfying if each experiment takes a long time

Simulation

- Given all these difficulties with real-world experiments, a popular approach is simulation
- Simulation: the use of a computer program that implements mathematical and algorithmic models of the behavior of an application running on a platform
- The input to the simulation:
  - A specification of the platform's characteristics
  - A specification of the application's characteristics
- The output of the simulation:
  - A time-stamped list of relevant events throughout (simulated) time
  - From this list can be gathered statistics, visualizations

The trouble with simulation

- Every simulation introduces a bias, or error
  - The simulation is instantiated with unrealistic parameters
  - The simulation models' implementation is buggy
  - The simulation models are implemented correctly but do not correspond to the real world
  - The simulated application does not correspond to the real-world application because not based on a real implementation
  - The simulation ignores transient real-world behaviors
  - ...
- The larger the simulation error the less useful the simulation
- If the simulation error is not bounded/quantified, then the simulation is even less useful

Reducing simulation error (I)

- A (widely accepted) way to reduce simulation error is to have the simulation be *highly detailed*
- More details can be achieved by making analytical models *more complex* (i.e., more parameters)
- Example:
  - $compute\_time = \#inst / inst\_per\_second$
  - $compute\_time = [\#inst_{load\_store} \times (CPI_{load\_store} + cache\_hit\_rate \times cache\_penalty) + \#inst_{other} \times CPI_{other}] / clock\_rate$
  - Accounting for instruction-level parallelism...

Reducing simulation error (II)

- More details can be achieved by *replacing* analytical simulation models by complex programmatic models
- Example:
  - Don't use a formula to estimate the compute time
  - Instead write a program that generates control signals in a hypothetical micro-architecture, simulating what happens at each clock cycles (register set, ALUs, caches, etc.)
  - The input is the actual list of instructions and operands
  - The output is a time-stamped trace of instruction completions from which one can compute statistics
  - So-called *cycle-accurate simulation*

The trouble with complex simulation

- The notion that "more complex = better" is not without its problems [Gibson et al., ASPLOS'00]
  - And more complex models may be poorly instantiated
- But making models more complex/sophisticated/complete is still the most common approach
- One problem is: more complex is slower
- The ratio of simulation time to simulated time grows and can become very large (e.g., > 10, > 100)
  - Hardware is fast, (simulation) software is slow
- If simulated scenarios are large/long, then simulation time is prohibitively long
  - Remember that we may need to run thousands of simulations to compare scheduling heuristics

The sweet spot?

- We want simulation to be fast and accurate
- It would seem that simulation can be either
  - slow and accurate, or
  - fast and erroneous
- The goal is to push the limits:
  - Improve the speed of "complex" simulation models
  - Improve the accuracy of "simple" simulation models
  - Improve the accuracy of "simple" simulation models

Network simulation

- In this seminar: network simulation
- An important aspect of the execution of a parallel and distributed application is the time spent doing network activities
  - Sending messages
  - Waiting for messages
- Many proposed scheduling algorithms attempt to carefully schedule both computation and communication
- But the "view of the network" of the algorithms is notoriously different from the reality of deployed networks

Packet-level simulation

- The network community, i.e., researchers concerned with the design of network protocols, routing schemes, etc., typically use packet-level simulation
- A packet-level simulator is a discrete-event simulator
  - packet emission/reception events
  - network protocol events
- The life-cycle of every individual packet is simulated
  - e.g., from the TCP stack down the the IP level
- Popular simulators: NS2, NS3, GTNetS, OMNet++, ...
  - Some use simplified protocol stacks (GTNeTS, NS2)
  - Some can use real TCP implementations (NS3)
- In this seminar: simulation of wired TCP/IP networks

Packet-level simulation: accurate

- Since packet-level simulation simulates every packet and implements protocol stack, it is often accepted as accurate
  - Or at least, published packet-level simulation results are typically trusted
- It is outside the scope of this seminar to discuss to which extent this is true
  - It turns out there not many validation studies
  - And the word "accurate" means different things to different people
    - e.g., matching the spec of how we design the network
    - vs. matching what actually happens in a real-world environment
- We will simply assume that "packet-level = perfectly accurate"

Packet-level simulation: slow

- High simulation/simulated ratio
  - Using GTNetS, which is known for being reasonable fast
  - Simulating 200 communications each transferring 100MiB between two random end-points in a random 200-node topology
  - 125 sec of simulated time, 1,500 sec of simulation time on a 3.2GHz processor
  - The ratio can be made arbitrarily large by increasing the number of communications
- This is because each packet leads to many network events to be simulated in software
- Not usable to run (many) simulations in which there are large numbers of communications that involve large numbers of packets

An alternative to packet-level simulation

- To make simulation fast, we must avoid simulating individual packets
- Simple idea: simulate *flows*, abstracting away packets, and determine flow completion times via equations
  - flow: source + network path + destination + #bytes (S)
- The simplest of flow models: $T = latency + bandwidth/S$
  - latency = end-to-end latency (sec)
  - bandwidth = *bottleneck* bandwidth (byte/sec)
- There are, unfortunately, several problems with the above
  - Network protocol overhead?
  - Network protocol effects (e.g., congestion window)?
  - How do we find out the bottleneck bandwidth???
- Let's look at some flow-level simulators

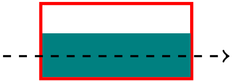
Some simulators

- A research area in which non-packet-level network simulation has been used for decades is “Grid computing” (and lately “Cloud computing”)
  - Clusters federated via wide-area networks, many processes, large data
- Four well-known, and/or popular, and/or widely used simulators that employ flow-level models
  - GridSim [Buyya et al., CCPE’02] ($\sim$ 10 download/day)
  - OptorSim [Bell et al., IJHPCA’03]
  - GroudSim [Ostermann at al., Grid’10]
  - CloudSim [Calheiros et al., SPE’11]
- These are MUCH faster than packet-level simulators
- But are they accurate?

Invalidating experiments

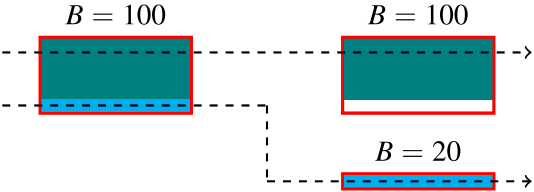
- It turns out these simulators are not accurate
  - In spite of perhaps convincing published results in some cases
- Very easy to come up with invalidating experiments
- Let's exhibit such experiments with the following visuals:

$B = bandwidth$



A flow receiving a bandwidth share of a link of bandwidth B

Invalidating OptorSim and GroudSim

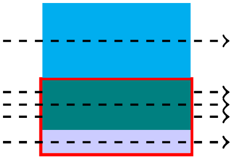


What OptorSim and GroudSim do
(links are shared 50% in spite of bottlenecks)

What a real network would do
(link utilization is maximized)

This "weakness" is document by the authors

Invalidating GridSim



What a real network would do

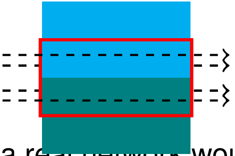
What GridSim does

(first flow receives B, second $B/2$, etc.)

Likely a bug (which we reported)

Invalidating CloudSim

One flow starts ϵ seconds after the other



What a real network would do

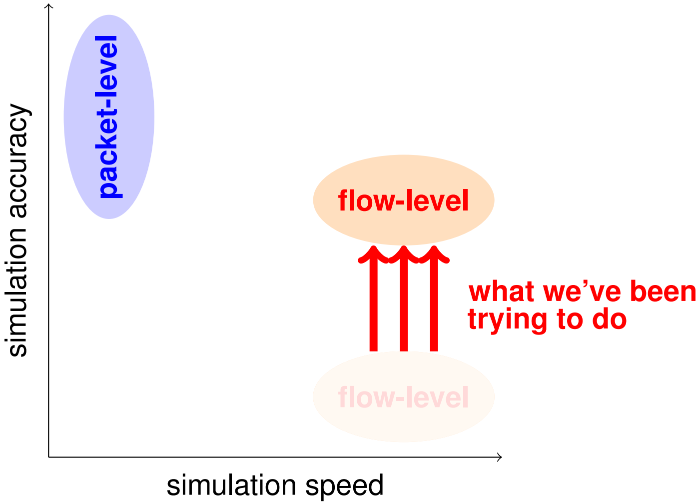
What CloudSim does

(each flow receives B)

Invalidation is easy???

- We took four well-known simulators
- Suggested by our own experience and by inspecting the code of those simulators, we came up with trivial invalidating experiments
- It is quite shocking how easy it was to perform the invalidation
- One may wonder: how is this possible?
- Answer: validation studies are rare (and invalidation studies are more rare)
  - Published results typically show "good cases"

Is there no hope then?



The two main questions

- Consider a network topology (end-points, routers, links), latency and bandwidth values for the links, and a set of flows each with some start time
- The two questions we need to answer are:
    1. What bandwidth is allocated to each flow throughout its execution?
    2. What is the completion time of each flow?
- Question 1 can be answered (approximately) if we take a *steady-state* simulation approach

Steady-state bandwidths

- We assume that each flow f present at time t run forever, and that no new flow arrives in the system
- Given these assumptions, we compute the constant bandwidth allocated to each flow, B_f
  - In a real network the bandwidths would converge reasonably quickly to pseudo-constant values
- For each flow f, given S_f (*remaining* amount of data) and B_f, we compute the flow's hypothetical completion time t_f
- Say the next new flow arrives at time t_{new}
- Between time t and time $t' = \min(t_{new}, \min_f t_f)$ the number of flows is constant, and we know their bandwidths!
- We "advance" the simulation to time t', updating $S_f \leftarrow S_f - B_f/(t' - t)$

Bandwidth constraints

- With the steady-state approach, i.e., no notions of time and flow sequencing, we can formalize the bandwidth computation problem nicely

Problem (Constraints)

- $\mathcal{F}$ a set of flows, $\mathcal{L}$ a set of links
- B_l is the bandwidth of $l \in \mathcal{L}$
- ρ_f is the bandwidth allocated to $f \in \mathcal{F}$

$$\forall l \in \mathcal{L}, \quad \sum_{f \in \mathcal{F} \text{ going through } l} \rho_f \leq B_l$$

Objective: compute realistic ρ_f's

Computing realistic bandwidths

- The question of computing realistic bandwidths has been studied in the context of TCP
- Two main approaches have been developed

1. Bottom-up:
   - Reason on the microscopic behavior of TCP
   - Infer its macroscopic behavior
2. Top-down:
   - Propose a reasonable model of macroscopic behavior
   - Instantiate it based on (in)validation experiments

A bottom-up model

- TCP uses a congestion window that bounds the number of in-flight packets for a flow, so as to perform congestion control at the end-points
- The window size is tuned via *additive increase and multiplicative decrease*
  - Ramp up slowly, back down fast
- Several authors have proposed bottom-up “bandwidth sharing” models by reasoning on window size dynamics
  - Mo et al. [INFOCOM'99], [IEEE TN 2000]
  - Yaïche et al. [IEEE TN 2010]
  - Low et al. [J. ACM 2002], [IEEE TN 2003]
  - Low et al. [J. ACM 2002], [IEEE TN 2003]

The bottom-up model by Low et al.

- $w_f(t)$: the window size for flow $f \in \mathcal{F}$ at time t
- d_f: the equilibrium RTT (propagation plus equilibrium queuing delay) of f, which is assumed to be constant
- f's instantaneous data transfer rate in packets per time unit is $\rho_f(t) = w_f(t)/d_f$
- $q_f(t)$: the packet loss probability for flow f at time t
- At time t, f's emitter transmits $\rho_f(t)$ packets per time units
- For these packets it receives $\rho_f(t)(1 - q_f(t))$ positive acks and $\rho_f(t)q_f(t)$ negative acks
- Additive increase: the window increased by $1/w_f(t)$
  - $w_f(t)$ is increased by 1 for each $w_f(t)$ packets sent
- Multiplicative decrease: the window is halved

The bottom-up model by Low et al.

- The net change in window size per time unit is:

$$w_f(t+1) - w_f(t) = \rho_f(t)\frac{1}{w_f(t)}(1 - q_f(t)) - \rho_f(t)\frac{w_f(t)}{2}q_f(t)$$

- Since $w_f(t) = \rho_f(t) \times d_f$, we obtain

$$\rho_f(t+1) - \rho_f(t) = \frac{1}{d_f^2}(1 - q_f(t)) - \frac{1}{2}\rho_f(t)^2 q_f(t)$$

- As an approximation we obtain a differential equation:

$$\frac{\partial \rho_f}{\partial t}(t) = \frac{1}{d_f^2}(1 - q_f(t)) - \frac{1}{2}\rho_f(t)^2 q_f(t)$$

The bottom-up model by Low et al.

- Given the differential equation, it is possible to prove (via a Lyapunov function) that the bandwidths satisfy the following constrained optimization problem

Problem

$$\text{maximize} \quad \sum_{f \in \mathcal{F}} \frac{\sqrt{2}}{d_f} \arctan\left(\frac{d_f \rho_f}{\sqrt{2}}\right)$$

$$\text{subject to} \quad \forall l \in \mathcal{L}, \quad \sum_{f \in \mathcal{F} \text{ going through } l} \rho_f \leq B_l$$

The bottom-up model by Low et al.

- The previous model is in fact for TCP Reno with RED
- With TCP Vegas and Droptail, Low et al. obtain instead

Problem

$$\text{maximize} \quad \sum_{f \in \mathcal{F}} d_f \log(\rho_f)$$

$$\text{subject to} \quad \forall l \in \mathcal{L}, \sum_{f \in \mathcal{F} \text{ going through } l} \rho_f \leq B_l$$

Bottom-up simulation models?

- These models come from the network protocol community
- They're elegant and meant to enlighten us about the fundamental properties of network protocols
- Nowhere in the articles by Low et al. is it suggested to use these models for simulation
- Some parameters may be hard to instantiate for simulation, since users see/care only about the macroscopic behavior
- And in fact, there is a problem with the d_f parameter: equilibrium RTT...

Equilibrium RTT???

- The assumption by Low et al. is that d_f is constant
- What this means is that d_f is constant for a given traffic, i.e., a set of flows and their transfer rates
- But if the transfer rates are computed according to the model (for simulation purposes), then we have a *circular dependency*
  - d_f tells us what the traffic looks like
  - The traffic defines the value of d_f
- Question: how can we pick d_f for simulation purposes?
- Our best guess: $d_f = \sum_{l \in \mathcal{L} \text{ traversed by } l} L_l$
  - L_l: latency of link l
  - We can guess, we're not network protocol people ☺

A top-down bandwidth sharing model

- With top-down modeling one *hypothesizes* a bandwidth sharing model that seems reasonable, and then one attempts to prove it is valid
- Such a model is Max-Min fairness:

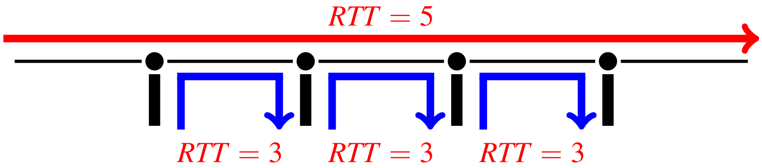
Problem

$$\begin{array}{ll} \text{maximize} & \min_{f \in \mathcal{F}} \rho_f \\ \text{subject to} & \forall l \in \mathcal{L}, \displaystyle\sum_{f \in \mathcal{F} \text{ going through } l} \rho_f \leq B_l \end{array}$$

- Makes the least happy flow as happy as possible

Max-min fairness

- Max-min fairness is convenient and makes sense
  - what a network *should* do to be "as fair as possible"
  - Easily computed via a recursive simple algorithm
- Problem: TCP is *RTT-unfair*



- On each shared *bottleneck* link: **3/7**, **5/7**

RTT-unfair Max-min fairness

- Weighted Max-Min fairness to account for RTT-unfairness

Problem

$$\text{maximize} \quad \min_{f \in \mathcal{F}} L_f \; \rho_f$$

subject to:

$$\forall f \in \mathcal{F},\; L_f = \sum_{l \text{ traversed by } f} L_l$$

$$\forall l \in \mathcal{L},\; \sum_{f \text{ going through } l} \rho_f \leq B_l$$

RTT-unfair, windowed Max-min fairness

- TCP uses a congestion window, W, that bounds the number of in-flight bytes

Problem

$$\text{maximize} \quad \min_{f \in \mathcal{F}} L_f \; \rho_f$$

subject to:

$$\forall f \in \mathcal{F},\; L_f = \sum_{l \text{ traversed by } f} L_l$$

$$\forall l \in \mathcal{L},\; \sum_{f \text{ going through } l} \rho_f \leq B_l$$

$$\forall f \in \mathcal{F},\; \rho_f \leq W/(2L_f)$$

Validity of the top-down model

- It is known that TCP does not implement (RTT-unfair) Max-min fairness
- But authors have argued it's a reasonable approximation in some cases
  - [Chiu, 1999], [Casanova and Marchal, 2002]
- The big question: is it good enough to be use as a simulation model?
- In [Fujiwara and Casanova, 2007] some good results are presented for a set of topology/flow scenarios
  - But the validity limits are not explored much
  - They do report that for transfers < 100 MiB results are poor (more on this in the next few slides)
- Velho and Legrand went further...

Parameterizing the model

- In [Velho and Legrand, Simutools'09], the authors have evolved this basic model by adding several parameters
- Their approach:
  - Generate a bunch of test topologies
    - Dumbbell topologies with 4 endpoints and 2 routers (>-<)
    - Random with 50 or 200 end-points generated with BRITE
    - Random latencies or latencies computed by BRITE
    - Random bandwidths in various ranges
  - Generate a bunch of flows between random end-points
  - Compare flow-level results to packet-level results
  - Gain insight into what tuning parameters should be added
  - Estimate parameter values of maximum likelihood
- Let's review their findings...

High congestion and RTT-unfairness

- In highly congested scenarios, the RTT is impacted by bandwidth
- New parameter: γ

$$\forall f \in \mathcal{F},\ L_f = \sum_{l \text{ traversed by } f} L_l + \frac{\gamma}{B_l}$$

- Velho and Legrand find that good results are achieved by $\gamma \approx 8775$ or $\gamma \approx 20537$ depending on the TCP version

Protocol overhead

- When using a network protocol, not every transferred byte is a byte of useful data
- Every protocol has an overhead, which decreases the achievable data transfer rate on a link of given bandwidth
- New parameter: β

$$\forall l \in \mathcal{L}, \sum_{f \text{ going through } l} \rho_f \leq \beta \times B_l$$

- Velho and Legrand find that good results are achieved by $\gamma \approx 0.92$ or $\gamma \approx 0.97$ depending on the TCP version (corresponds to the fraction of useful payload)

Simulating finite flows

- Regardless of the steady-state bandwidth sharing model, we want to simulate finite flows
- We must compute for each flow its execution time t_f
  - to advance the simulated time to $t' = \min(t_{new}, \min_f t_f)$
- A simple model:

$$t_f = L_f \;+\; S_f / \rho_f$$

- Problem: TCP's slow start behavior!
  - Due to additive increase, the steady-state bandwidth is not reached immediately, even when there is no congestion

Simulating slow-start

- Because of slow-start, a simulation could only be realistic for large transfer sizes (i.e., ≈ 10 MiB)
  - See the results by Fujiwara and Casanova
- In [Velho et al., 2009] the authors propose a simple modification with a new parameter α:

$$t_f = \alpha\, L_f + S_f/\rho_f$$

- The author find that $\alpha \approx 10.40$ or $\alpha \approx 13.01$ leads to good results depending on the TCP version
- This makes the simulations reasonably accurate down to ~ 100 KiB data sizes
- For smaller transfers: forget flow-level models and go back to packet-level since each flow is only a few packets!

The full model in [Velho and Legrand, Simutools'09]

Problem

$$\text{maximize} \quad \min_{f \in \mathcal{F}} L_f \; \rho_f$$

subject to:

$$\forall f \in \mathcal{F},\; L_f = \sum_{l \text{ traversed by } f} L_l + \frac{\gamma}{B_l}$$

$$\forall l \in \mathcal{L}, \quad \sum_{f \text{ going through } l} \rho_f \;\leq\; \beta \times B_l$$

$$\forall f \in \mathcal{F}, \quad \rho_f \;\leq\; W \,/\, (2L_f)$$

$$t_f \;=\; \alpha\, L_f \;+\; S_f \,/\, \rho_f$$

The results in [Velho and Legrand, Simutools'09]

- Drastic improvements are demonstrated over the basic Max-Min
  - The addition of each parameter helps in its own way
- Low discrepancy with packet-level simulation shown on hundreds of test cases
- So overall, a very convincing paper with convincing results

- And yet, occasionally, some flow in some test case will experience a discrepancy by up to a factor 20!
- Question: To which extent is the (modified) Max-Min fairness model valid?

The critical method

- By contrast with bottom-up modeling, top-down modeling is much more of an *empirical science*
  - (Re)formulate a model as an hypothesis
  - Try to invalidate it via experiments
  - Repeat
- This is the *critical method* [Karl Popper, philosopher]
  - Don't keep showing "good" cases in which the model works (although it's the typically publication strategy!)
  - Instead keep looking for "bad" cases to evolve the models
  - These bad cases are called *crucial experiments* (i.e., invalidation experiments)
- Invalidation should be at the core of the modeling activity

Invalidation

- Let us follow the critical method to invalidate the flow-level model in [Velho and Legrand, 2009]
- Essentially, we try to break the model
- So that we can either:
  - Improve it, or
  - Quantify validity limits
- All results are for TCP Reno + RED
  - Some results are discussed with TCP Vegas + Droptail in our recent paper in IEEE TOMACS

Experimental procedure in [Velho and Legrand]

- The results in [Velho and Legrand] are for 4 sets of 10 topologies generated with BRITE, using the Waxman model
  - Number of nodes: 50 or 200
  - Link bandwidths: uniformly distributed in $[100, 128]$ MiB/s or $[10, 128]$ MiB/s
  - Link latencies: computed by BRITE based on Euclidian distance
- For each of the topologies:
  - 100 flows are generated between random end-points
  - Each flow transfers 100 MiB of data
  - TCP congestion window is set to 60 KiB
- In total: 160 experimental scenarios (4×40)

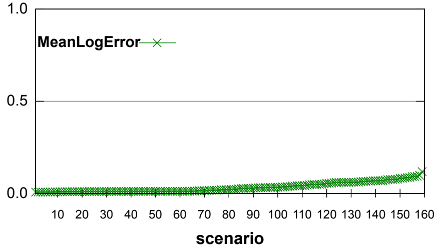
Accuracy metric

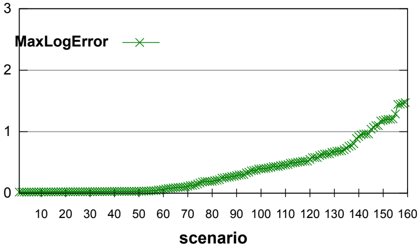
$$\text{MeanLogErr} = \frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}}\left|\log\left(\rho_f^{\textit{flow}}\right) - \log\left(\rho_f^{\textit{packet}}\right)\right| \text{, and}$$

$$\text{MaxLogErr} = \max_{f\in\mathcal{F}}\left|\log\left(\rho_f^{\textit{flow}}\right) - \log\left(\rho_f^{\textit{packet}}\right)\right| .$$

- The smaller the error, the better
- The logarithms are used for symmetry (doesn't matter whether the "reference" is the smaller or the larger value)
- Computed over the time window from $t = 0$ up to the completion of the first flow

Initial results



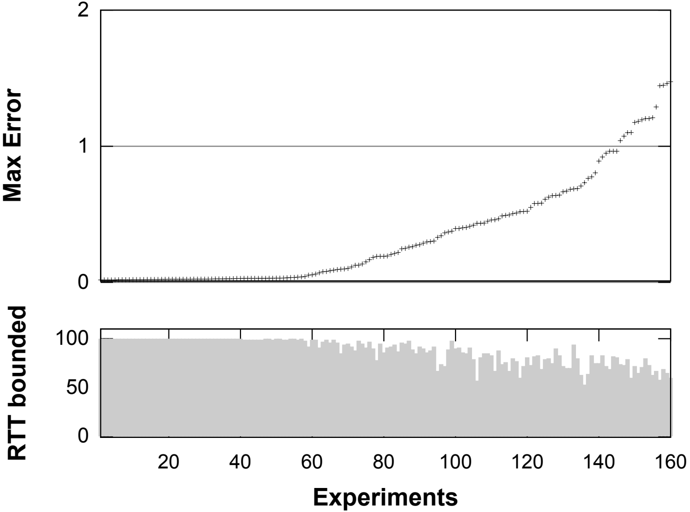


- A few scenarios show high error
- Many show low error: they're **not** "crucial experiments"

Making scenarios more crucial

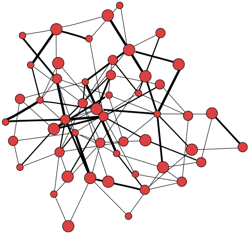
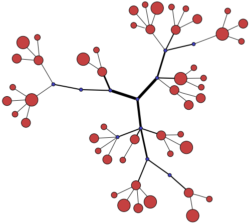
- Inspecting the results, we find that many of the low-error scenarios have mostly RTT-bound flows
- They are governed by the $\rho_f \leq W/(2L_f)$ constraint
  - The "real" bandwidth sharing model is bypassed
- They are easy cases for our flow-level model, hence the low error
- We can see this trend easily if we plot % of RTT-bounded flow vs. MaxError...

More RTT-bounded flow, less crucial

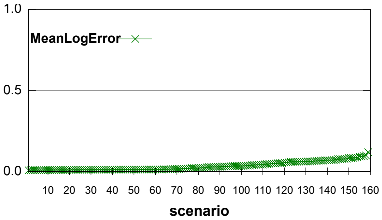


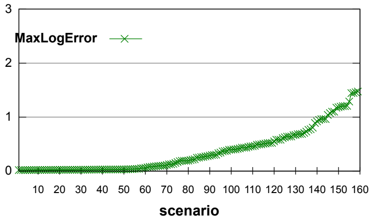
Making experiments more critical

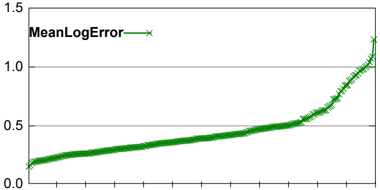
- We sample bandwidths in $[10, 12.8]$ MiB/s
  - Likely rare in practice, but we use the critical method
- We no longer use BRITE, but instead use Tiers

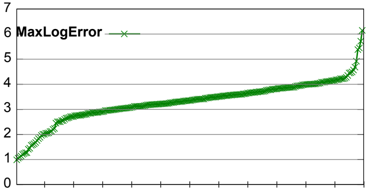
 $\Rightarrow$ 

New crucial workload





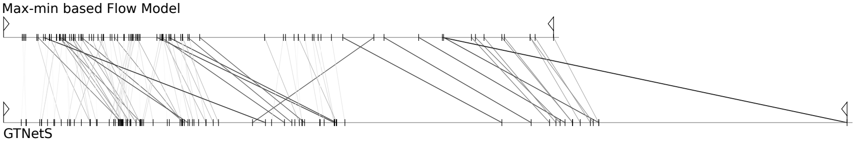




Finding sources of error

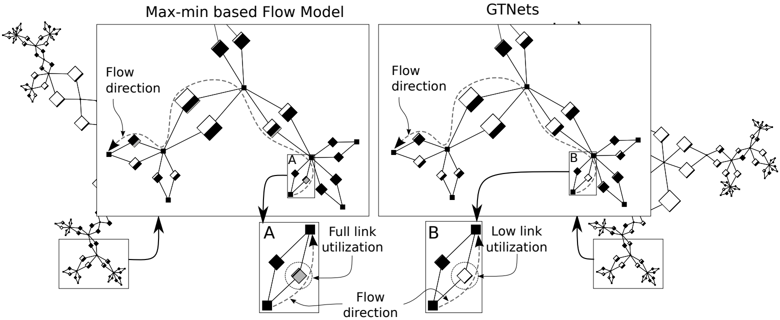
- Now we have a lot of bad results, which is great because now it’s easier to try to understand what’s wrong with the model
- We pick a particularly bad workload and visualize flow completion time in flow-level simulation and packet-level simulation...

Visualization of completion times



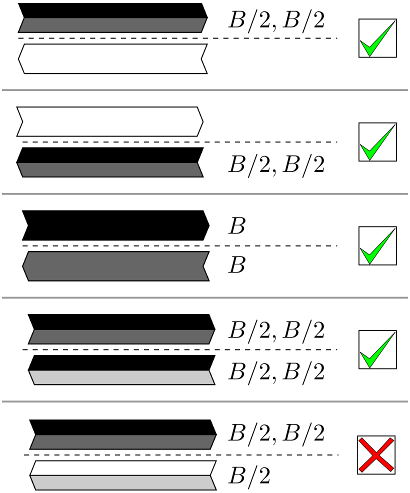
- Many almost vertical lines (light)
- Several really non-vertical lines (dark)
- Flow-level simulation almost always underestimates completion times
  - We have no idea how to explain the one odd flow that is faster with packet-level simulation
- We now use another visualization to inspect the "bad" flows

Visualization of link usage



- There is no hope for our flow-level model to capture the underutilization of that link...

Link underutilization explained



- Underutilization is due to reverse traffic
- One explanation: *Network compression* [Zhang 1991]
  - Ack packets are queued with the data packets of reverse flows
- Another: *Ack-clocking* [Heusse, 2011]
  - Data and ack segments alternatively fill only one link buffer
- We modify our problem...

Accounting for reverse traffic in our model

Problem

$$\text{maximize} \quad \min_{f \in \mathcal{F}} L_f \; \rho_f$$

subject to:

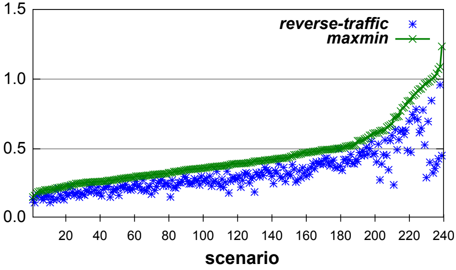
$$\forall f \in \mathcal{F},\; L_f = \sum_{l \text{ traversed by } f} L_l + \frac{\gamma}{B_l}$$

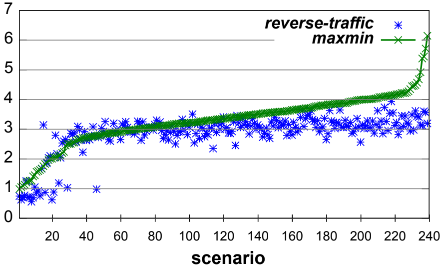
$$\forall l \in \mathcal{L}, \sum_{f \text{ going through } l} \rho_f + \epsilon \left(\sum_{f\text{'s ack going through } l} \rho_f \right) \leq \beta \times B_l$$

$$\forall f \in \mathcal{F},\; \rho_f \leq W \,/\, (2L_f)$$

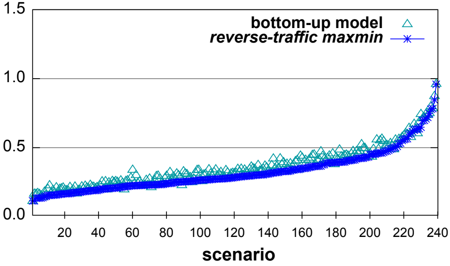
$$t_f = \alpha\, L_f + S_f \,/\, \rho_f$$

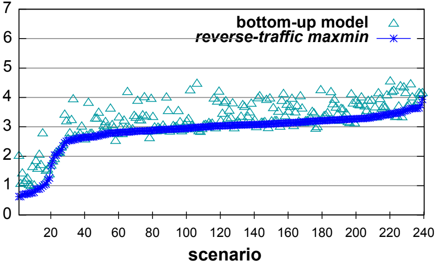
Improved results





Comparison to bottom-up model



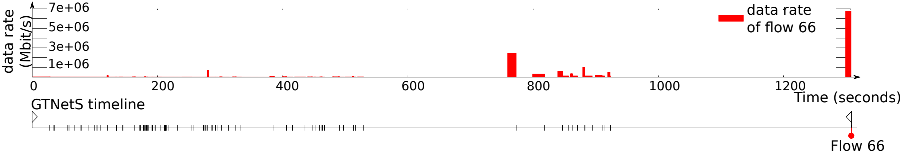


Could bottom-up be improved?

- The model by Low et al. does not account for reverse traffic
- Since the work by Low et al., new advances have been made
  - Differential Algebraic Equation (DAE) model in [Tang et al., 2008, 20010]
  - Ack-clocking dynamics in [Jacobsson et al., 2009]
- These models are not designed for use in simulation
  - They require solving complex differential equations
  - Models not yet validated for more than three flows and three nodes
- Conclusion: At this point, our top-down modified Max-Min model looks pretty good

Some things, we just can't do...

- Some flow just experience data rates that cannot be modeled by a flow-level model



- This flows stalls for 380 seconds!
  - Results reproduced with NS-3 and GTNetS
- This is for a very (unlikely) high-contention scenario

What we found out

- Flow-level models used in state-of-the-art grid/cloud simulators can be vastly improved upon
- Results are close to packet-level results unless
  - Transferred data sizes are small (a few KiB)
  - Congestion is really high
- Using the critical method was key to obtained improvements in the model
- We feel that we have reached the limits of what we can do
- All our work is implemented in the SIMGRID simulator
  - See next seminar

Sources and acknowledgments

- *A Duality Model of TCP and Queue Management Algorithms*, [IEEE TN, 2003]

  S. H. Low
- *Accuracy Study and Improvement of Network Simulation in the SimGrid Framework*, [Simutools 2009]

  P. Velho, A. Legrand
- *On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations* [IEEE TOMACS 2013]

  P. Velho, L. Mello Schnorr, H. Casanova, A. Legrand