Simulation

ICS632: Principles of High-Performance Computing

Henri Casanova (henric@hawaii.edu)

Fall 2015

Acknowledgments

- Joint work with a LOT of people:

  http://simgrid.gforge.inria.fr

Why Simulation?

- Many "HPC" 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 perfor mance than approximation algorithms
- Sadly, we rarely have theorems to compare heuristics
- Typically, given 20 (reasonable) heuristics and 10,000 random problem instance s, each heuristic could be best on some of the instances
- Question: How do we find out which heuristics are best?

Why Simulation?

- Many "HPC" 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 perfor mance than approximation algorithms
- Sadly, we rarely have theorems to compare heuristics
- Typically, given 20 (reasonable) heuristics and 10,000 random problem instance s, each heuristic could be best on some of the instances
- Question: How do we find out which heuristics are best?

Why Simulation?

- Many "HPC" 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 perfor mance than approximation algorithms
- Sadly, we rarely have theorems to compare heuristics
- Typically, given 20 (reasonable) heuristics and 10,000 random problem instance s, 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?

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...

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...

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 (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 (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 (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

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 to the rescue

- 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

Simulation to the rescue

- 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

Simulation to the rescue

- 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

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:
  - $compute\_time = \#inst / inst\_per\_second$
  - $compute\_time = [\#inst_{load\_store} \times (CPI_{load\_store} + cache\_miss\_rate \times cache\_penalty) + \#inst_{other} \times CPI_{other}] / clock\_rate$
  - Accounting for instruction-level parallelism...

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:
  - $compute\_time = \#inst / inst\_per\_second$
  - $compute\_time = [\#inst_{load\_store} \times (CPI_{load\_store} + cache\_miss\_rate \times cache\_penalty) + \#inst_{other} \times CPI_{other}] / clock\_rate$
  - Accounting for instruction-level parallelism...

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:
  - $compute\_time = \#inst / inst\_per\_second$
  - $compute\_time = [\#inst_{load\_store} \times (CPI_{load\_store} + cache\_miss\_rate \times cache\_penalty) + \#inst_{other} \times CPI_{other}] / clock\_rate$
  - 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*

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 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

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

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

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

Outline

Parallel/Distributed Computing Simulation

- Many results in parallel/distributed computing research are obtained in simulation
- Simulation has been used for decades in various areas of computer science
  - Network protocol design, microprocessor design
- By comparison, current practice in parallel/distributed computing is in its infancy
- Part of the problem is the lack of a standard tool, likely due to research being partitioned in different sub-areas
  - cluster computing, grid computing, volunteer computing, peer-to-peer computing, cloud computing, ...
- Part of the problem is the lack of open methodology...

Lack of open methodology

- The key to open science is that results can be reproduced
- Unfortunately, open science is not yet possible in parallel/distributed computing simulations
- Example: "Towards Yet Another Peer-to-Peer Simulator", Naicken et al., *Proc. of HET-NETs*, 2006
  - Surveyed 141 papers that use simulation
  - 30% use a custom simulator
  - 50% don't say which simulator is used (!!)
  - Only few authors provide downloadable software artifacts / datasets
- Researchers don't trust simulators developed by others?
- They feel they can develop the best simulator themselves?
- Ironic consequence: published simulation results are often not reproducible

Lack of open methodology

- The key to open science is that results can be reproduced
- Unfortunately, open science is not yet possible in parallel/distributed computing simulations
- Example: "Towards Yet Another Peer-to-Peer Simulator", Naicken et al., *Proc. of HET-NETs*, 2006
  - Surveyed 141 papers that use simulation
  - 30% use a custom simulator
  - 50% don't say which simulator is used (!!)
  - Only few authors provide downloadable software artifacts / datasets
- Researchers don't trust simulators developed by others?
- They feel they can develop the best simulator themselves?
- Ironic consequence: published simulation results are often not reproducible

Lack of open methodology

- The key to open science is that results can be reproduced
- Unfortunately, open science is not yet possible in parallel/distributed computing simulations
- Example: "Towards Yet Another Peer-to-Peer Simulator", Naicken et al., *Proc. of HET-NETs*, 2006
  - Surveyed 141 papers that use simulation
  - 30% use a custom simulator
  - 50% don't say which simulator is used (!!)
  - Only few authors provide downloadable software artifacts / datasets
- Researchers don't trust simulators developed by others?
- They feel they can develop the best simulator themselves?
- Ironic consequence: published simulation results are often not reproducible

And yet....

- There are parallel/distributed computing simulators that are intended to be used by others!
- These simulators all attempt to do the same 3 things:
  - Simulate CPUs
  - Simulate networks
  - Simulate storage
- The two main concerns are:
  - Simulation accuracy
  - Simulation speed / scalability
    - Ability to simulate large/long-running applications/platforms fast and without too much RAM
- Simulators make choices to trade off one for the other
- Let's look at typical such choices...

Outline

Simulating computation (I)

- There are two main approaches for simulating computations: microscopic or macroscopic
- Microscopic approach: cycle-accurate simulation
  - Simulate micro-architecture components at the clock cycle resolution based on instructions from real (compiled) code or for synthetic instruction mixes
  - Typically used by computer architecture researchers
  - ☺ Arguably very accurate (bus contention, cache effects, instruction-level parallelism, multi-core, GPUs, ...)
  - ☹ Very slow (ratio of simulation/simulated time > 100)
  - ☹ Arbitrary unscalable as the volume of computation increases, which is a problem for simulating long-running applications (e.g., grid computing, cluster computing)

Simulating computation (II)

- Macroscopic approach: (scaled) delays
  - Defined for each compute resource a *computation speed*
  - Define a computation as a *computation volume*
  - The simulated time is computed as the *ratio* of the two, plus an optional random component
  - ☺ Reasonably accurate for compute-bound applications
    - Compute times can also be sampled from real application or benchmark executions on a reference computer architecture, and then scaled
  - ☺ Scalable: simulation of a computation in $O(1)$ space/time
  - ☹ Can be wildly inaccurate for memory-bound applications

Simulating communication

- Each link: *latency* and *bandwidth*
- Protocol-accurate packet-level simulation
  - Used by network protocol researchers
  - ☺ Accurate
  - ☹ slow and not scalable
- Non-protocol-accurate store-and-forward of packets
  - ☹ An attempt to be more scalable than the above, but can be made arbitrarily inaccurate (packet size?)
- Ad-hoc fluid models
  - ☺ Scalable
  - ☹ Not protocol-realistic
- TCP fluid models
  - ☺ Scalable and accurate within limits

Simulating storage

- Microscopic approach: detailed discrete-event simulation (e.g., DiskSim)
  - ☺ Accurate
  - ☹ Just like cycle-accurate, and packet-level: unscalable
    - Arbitrarily high simulated/simulation ratio for large data
- Most simulators provide very little
  - Notion of storage capacity and stored data
  - Each transfer has a latency and bandwidth, with optional random components
  - ☹ Fails to capture caching effects, file system effects, ...

Specifying the simulated application (I)

- There are three main approaches: Finite automata, Event traces, Concurrent Sequential Processes (CSP)

- Finite automata:
  - Each simulated process is described as an automaton or a Markov chain
  - Each state is an action that lasts for some number of (simulated) seconds
    - Example: compute for 10s, then with probability 0.5 communicate for 1s, then with probability 1.0 do nothing for 30s, ...
  - ☺ Very scalable since each process is described with only a few bytes and fast algorithms can do state transitions
  - ☹ Limited expressive power, no/little application logic

Specifying the simulated application (II)

- Event traces:
  - Each simulated process is described as a sequence of compute, communicate, store events
    - At $t = 0$ compute for 10s, at time $t = 10$ send a message for 2s, at $t = 12$ receive a message for 20s, ...
  - Events obtained from real-world executions and "replayed", while scaling delays
  - ☺ Fast since event replay is algorithmically simple
  - ☹ Not always scalable (traces can be large, obtaining large traces on dedicated platforms can be hard)
  - ☹ Difficult to extrapolate traces to simulate more/fewer processes

Specifying the simulated application (III)

- Concurrent Sequential Processes (CSP):
  - Application implemented as fragments of arbitrary code that call simulation API functions
    - `sim_compute`, `sim_send`, `sim_recv`, ...
  - ☺ Maximum expressive power, arguably
  - ☹ Not scalable
    - Number of threads is limited (e.g., "only" a few thousands Java threads)
    - Context switching and synchronization overhead can be high
    - The user can implement expensive logic/computation, which may be useful but also unscalable

State-of-the-art simulators

Simulator	Community of Origin					Application			Network						CPU		Disk		
	high perf. comp.	grid comp.	cloud comp.	volunteer comp.	peer-to-peer comp.	execution trace	abstract spec.	programmatic spec.	latency	bandwidth	store-and-forward	ad-hoc flow	TCP fluid model	packet-level	scaled real delay	scaled delay	capacity	seek + transfer	block-level
PSINS	✓					✓			✓	✓		✓			✓	✓			
LogGOPSim	✓					✓			✓	✓						✓			
BigSim	✓					✓			✓	✓				✓	✓	✓			
MPI-SIM	✓							✓	✓	✓					✓	✓			
OptorSim		✓						✓	✓	✓		✓				✓	✓		
GridSim		✓						✓	✓	✓	✓	✓				✓	✓	✓	
GroudSim		✓	✓					✓	✓	✓		✓				✓			
CloudSim			✓					✓	✓	✓	✓					✓	✓	✓	
iCanCloud			✓					✓	✓	✓				✓		✓	✓	✓	✓
SimBA				✓			✓	✓	✓							✓			
EmBOINC				✓			✓	✓	✓							✓			
SimBOINC				✓				✓	✓	✓			✓			✓			
PeerSim					✓		✓		✓										
OverSim					✓			✓	✓	✓				✓					
SIMGRID		✓				✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	

SimGrid: a counter-intuitive design approach

- Accepted wisdom: to be both accurate and fast/scalable, a simulator must be highly *specialized* to a target domain
- Typical rationale: to achieve scalability one must "cut corners" and reduce accuracy in ways that are hopefully ok for the target domain
  - Example: when simulating a p2p application, no need to simulate network contention, or compute times
  - Example: when simulating a cluster computing application, no need to simulate external load on the system
- Instead, with SimGrid we target multiple domains:
  - Grid, cloud, cluster/HPC, volunteer, peer-to-peer
- We claim/demonstrate that we can be accurate/scalable across domains!

SimGrid: a counter-intuitive design approach

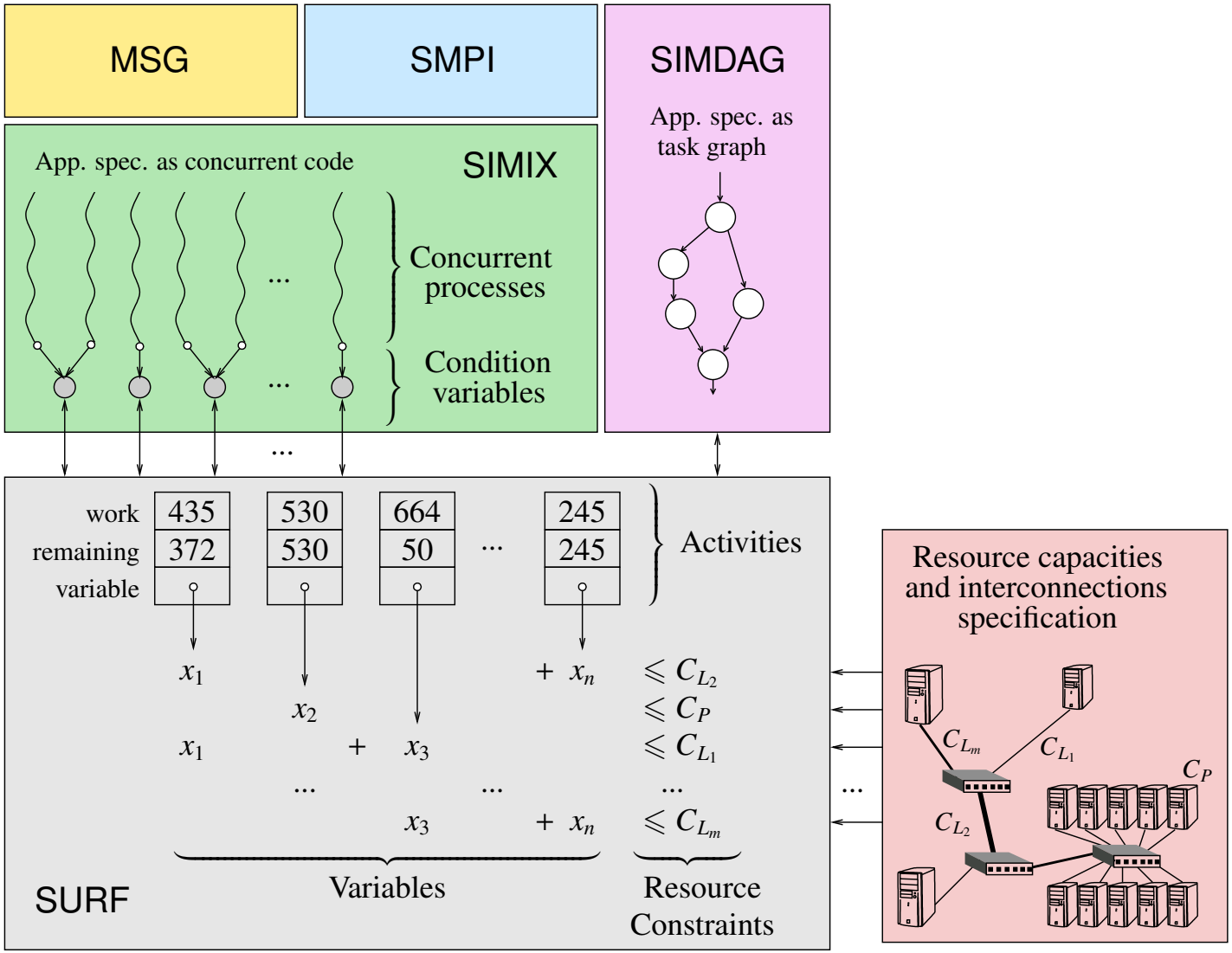
- Accepted wisdom: to be both accurate and fast/scalable, a simulator must be highly *specialized* to a target domain
- Typical rationale: to achieve scalability one must "cut corners" and reduce accuracy in ways that are hopefully ok for the target domain
  - Example: when simulating a p2p application, no need to simulate network contention, or compute times
  - Example: when simulating a cluster computing application, no need to simulate external load on the system
- Instead, with SimGrid we target multiple domains:
  - Grid, cloud, cluster/HPC, volunteer, peer-to-peer
- We claim/demonstrate that we can be accurate/scalable across domains!

Outline

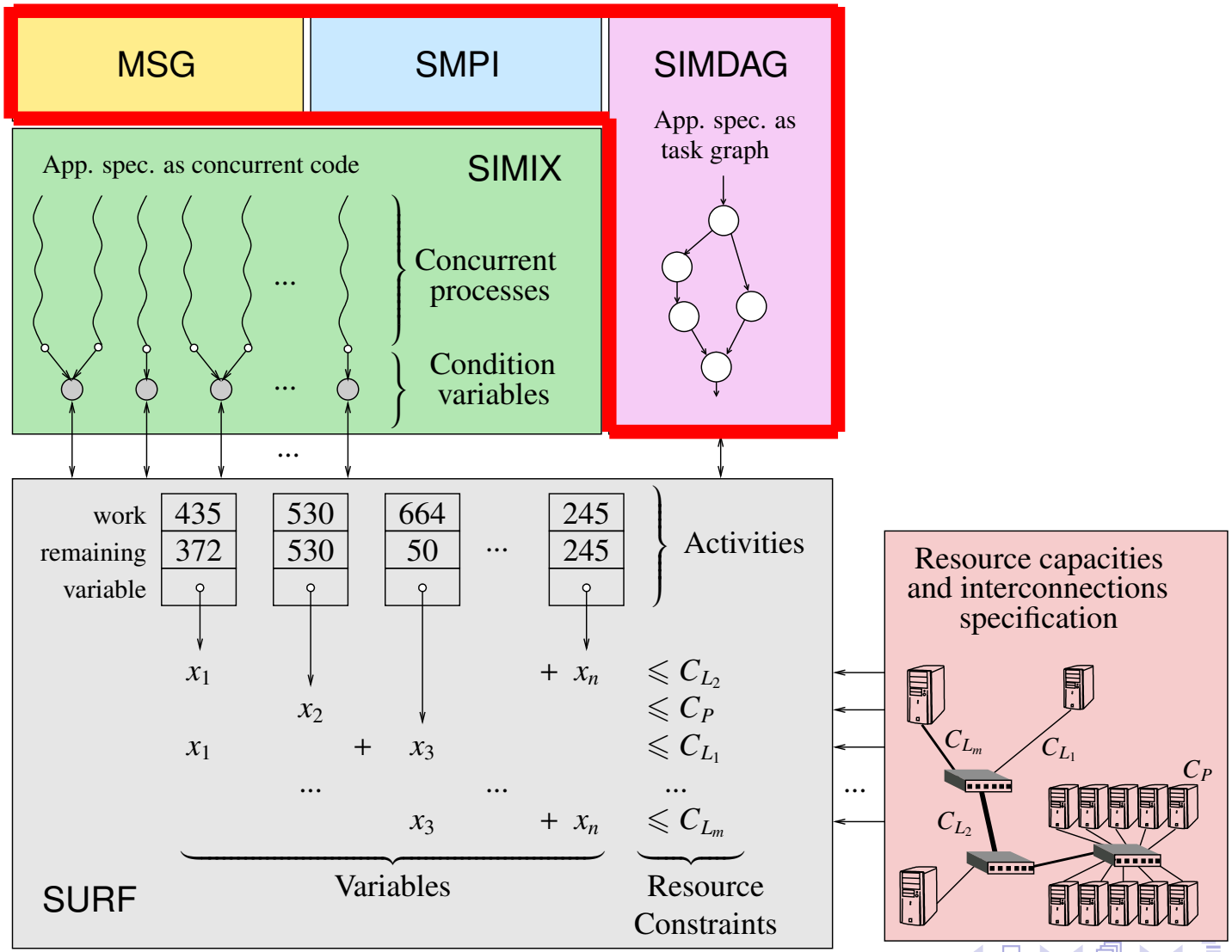
SimGrid: history

- A 14-year old open source project
- First release: 1999
  - A tool to prototype/evaluate scheduling heuristics, with naïve resource models
  - Essentially: a way to not have to draw Gantt charts by hand
- Second release: 2000
  - Addition of TCP fluid network models
  - Addition of an API to describe simulated app as CSPs
- Third release: 2005
  - Stability, documentation, packaging, portability, ...
- Release v.3.3: 2009
  - Complete rewrite of the simulation core for better scalability
  - Possible to describe transient resource behavior via traces
  - Addition of an "Operating System"-like layer
  - Two new APIs

Software stack



APIs


MSG
SMPI
SIMDAG
App. spec. as task graph
App. spec. as concurrent code
SIMIX
Concurrent processes
Condition variables
work
remaining
variable
435
372
530
530
664
50
245
245
Activities
x_1
x_2
$x_1 + x_3$
x_3
$+ x_n$
$\leqslant C_{L_2}$
$\leqslant C_P$
$\leqslant C_{L_1}$
$\leqslant C_{L_m}$
Variables
Resource Constraints
SURF
Resource capacities and interconnections specification
C_{L_m}
C_{L_1}
C_P
C_{L_2}

The SIMDAG API

- Application is described as a task graph
  - SD_TASK_CREATE(), SD_TASK_DEPENDENCY_ADD(), ...
  - SD_TASK_SCHEDULE() (on a host)
  - All types of API functions to get/set task properties/parameters
- One call to SD_SIMULATE() launches the simulation:
  - While there are ready tasks, run them
    - Computation + communication
  - Resolve dependencies
  - Repeat
- A very simple API designed for users who don't need the full power of the CSP abstraction
  - Looks a lot like SimGrid v1.0

The SMPI API

- Designed to simulate Message Passing Interface (MPI) applications
  - Standard way to implement communication in parallel applications
- The (almost unmodified) application is compiled so that MPI processes run as threads
- MPI calls are intercepted and passed to SimGrid's simulation core
- "Tricks" are used to allow simulation on a single computer
  - CPU burst durations are samples a few times and "replayed" for free
  - Arrays are shared among threads (wrong data-dependent application behavior but small memory footprint)

The MSG API

- This is the most commonly used API: basic CSP abstraction
- It has bindings in C, Java, Lua, Ruby
- Let's go through a full (but simple) *master-worker* example
  - The master process has tasks to send to workers
  - Each worker "processes" the tasks until it receive a termination signal
- Let's look at:
  - Master code
  - Worker code
  - Main function
  - XML platform description file
  - XML application deployment description file

Master

int master(int argc, char *argv[ ]) {

```
  int number_of_tasks = atoi(argv[1]); double task_comp_size = atof(argv[2]);
  double task_comm_size = atof(argv[3]); int workers_count = atoi(argv[4]);
  char mailbox[80]; int i;
  char buff[64];
  msg_task_t task;

  /* Dispatching tasks (dumb round-robin algorithm) */
  for (i = 0; i < number_of_tasks; i++) {
    sprintf(buff, "Task_%d", i);
    task = MSG_task_create(buff, task_comp_size, task_comm_size, NULL);
    sprintf(mailbox,"worker-%d",i % workers_count);
    print("Sending task %s to mailbox %s\n",buff,mailbox);
    MSG_task_send(task, mailbox);
  }
  /* Send finalization message to workers */
  for (i = 0; i < workers_count; i++) {
    sprintf(mailbox,"worker-%ld",i % workers_count);
    MSG_task_send(MSG_task_create("finalize", 0, 0, 0), mailbox);
  }
  return 0;
}
```

Worker

int worker(int argc, char *argv[ ]) {

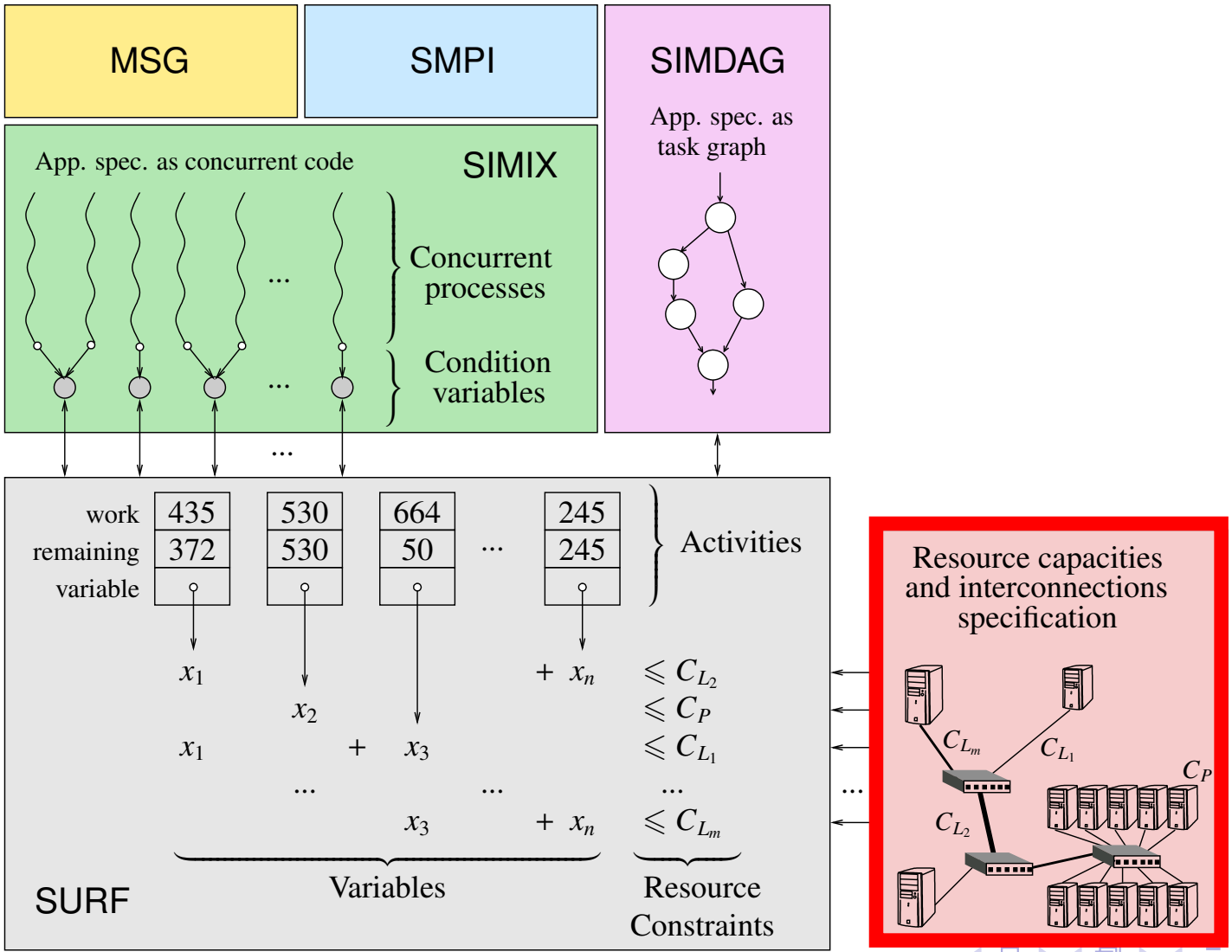
```
  msg_task_t task; int errcode; int id = atoi(argv[1]);
  char mailbox[80];
  sprintf(mailbox,"worker-%d",id);
  while(1) {
    /* Receive a task */
    errcode = MSG_task_receive(&task, mailbox);
    if (errcode != MSG_OK) {
      print("Error");  return errcode;
    }
    if (!strcmp(MSG_task_get_name(task),"finalize")) {
      MSG_task_destroy(task);
      break;
    }
    print("Processing %s", MSG_task_get_name(task));
    MSG_task_execute(task);
    print("Task %s done", MSG_task_get_name(task));
    MSG_task_destroy(task);
  }
  print("Worker done!");
  return 0;
}
```

Main program

int main(int argc, char *argv[ ]) {

```
  char *platform_file = "my_platform.xml";
  char *deployment_file = "my_deployment.xml";

  MSG_init(&argc,argv);

  /* Declare all existing processes, binding names to functions */
  MSG_function_register("master", &master);
  MSG_function_register("worker", &worker);

  /* Load a platform description */
  MSG_create_environment(platform_file);

  /* Load an application deployment description */
  MSG_launch_application(deployment_file);

  /* Launch the simulation (until its terminates) */
  MSG_main();

  print("Simulated execution time %g seconds",MSG_get_clock());
}
```

Platform and app deployment description



Platform description file (XML)

my_platform.xml

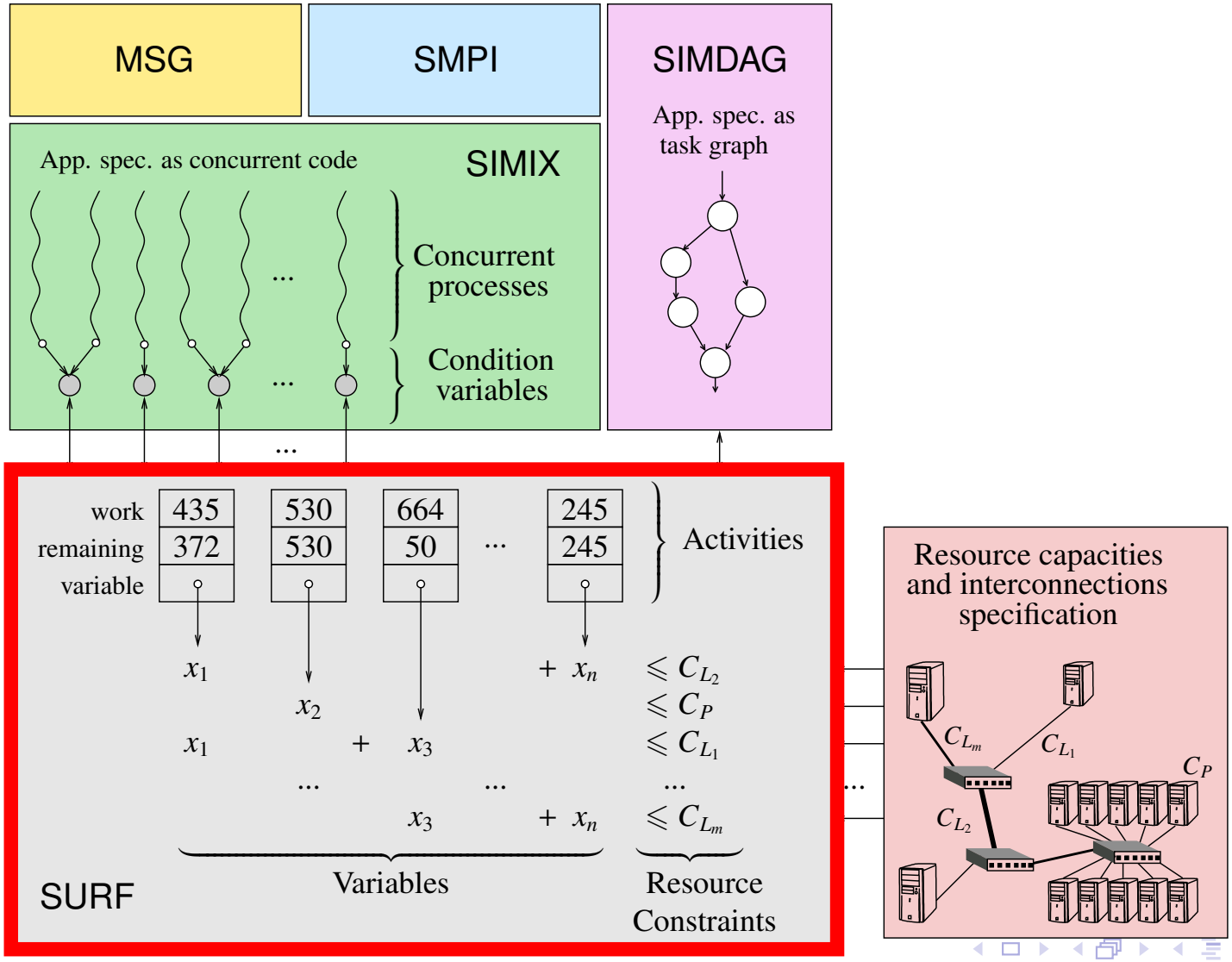
```
<?xml version="1.0"?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
  <platform version="3">
    <AS id="mynetwork" routing="Full">
      <host id="host1" power="1E6" />
      <host id="host2" power="1E8" />
      <host id="host3" power="1E6" />
      <host id="host4" power="1E9" />

      <link id="link1" bandwidth="1E4" latency="1E-3" />
      <link id="link2" bandwidth="1E5" latency="1E-2" />
      <link id="link3" bandwidth="1E6" latency="1E-2" />
      <link id="link4" bandwidth="1E6" latency="1E-1" />

      <route src="host1" dst="host2"> <link id="link1"/> <link id="link2"/> </route>
      <route src="host1" dst="host3">  <link id="link1"/> <link id="link3"/> </route>
      <route src="host1" dst="host4"> <link id="link1"/> <link id="link4"/> </route>
    </AS>
  </platform>
```

Application deployment description file (XML)

my_deployment.xml

```
<?xml version="1.0"?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
  <!-- The master (with some arguments) -->
  <process host="host1" function="master">
    <argument value="6"/> <!-- Number of tasks -->
    <argument value="50000000"/> <!-- Computation size of tasks -->
    <argument value="1000000"/> <!-- Communication size of tasks -->
    <argument value="3"/> <!-- Number of workers -->
  </process>

  <!-- The workers (argument: mailbox number to use) -->
  <process host="host2" function="worker"><argument value="0"/></process>
  <process host="host3" function="worker"><argument value="1"/></process>
  <process host="host4" function="worker"><argument value="2"/></process>
</platform>
```

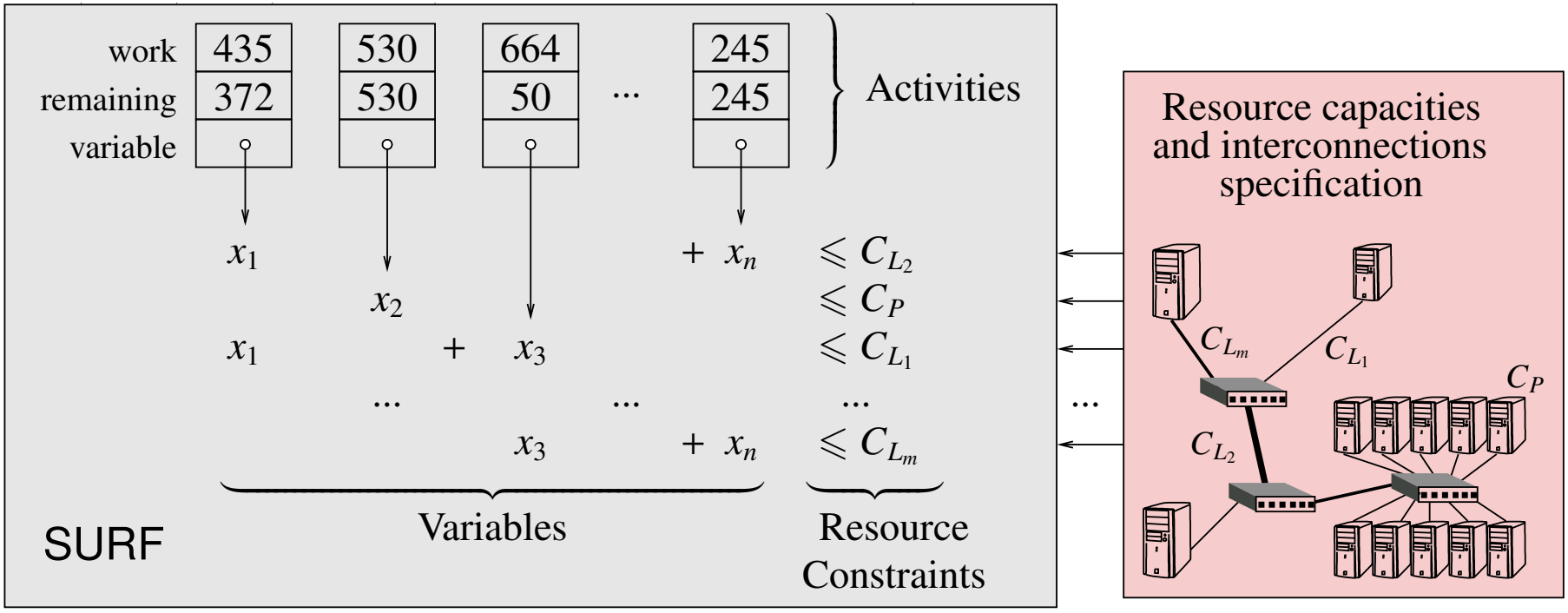
Simulation core



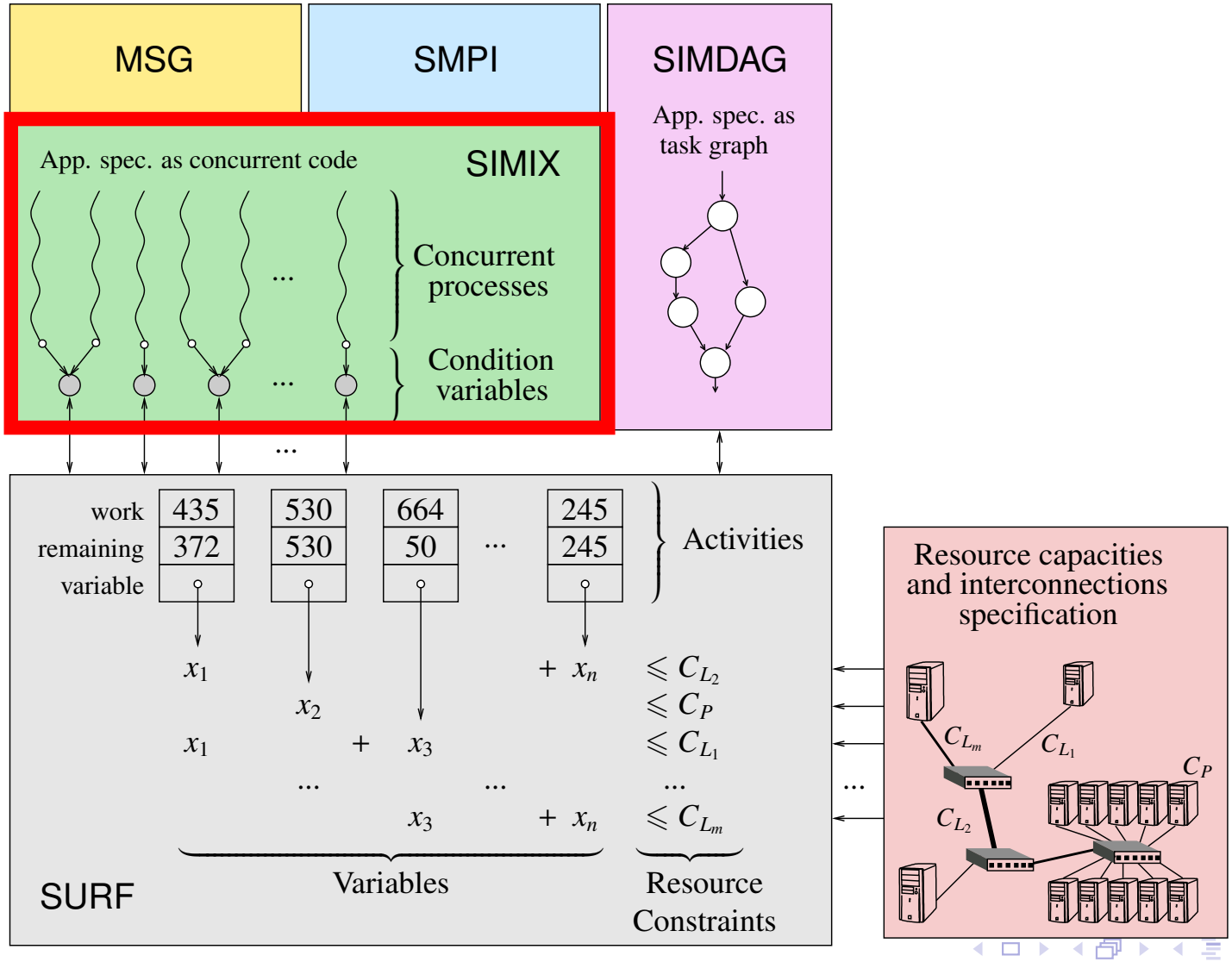
Simulation core

- The SURF component implements all simulation models
- All application activities are called actions
- SURF keeps track of all actions:
  - Work to do
  - Work that remains to be done
  - Link to a set of variables
- All action variables occur in constraints
  - Capture the fact that actions use one of more resources
- SURF solves the a (modified) Max-min constrained optimization problem betwen each simulation event
  - See previous seminar for more details
- Let's explain the example in the figure...

Simulation core



The SIMIX simcall interface



The SIMIX simcall interface

- SIMIX: an "OS kernel" on top of the simulation core
- Each simulated processes is a "thread" (more on this later)
- These threads run in mutual exclusion, round-robin, as controlled by SIMIX
- Each time a thread places an API call, translated to a simcall (a simulated syscall), it blocks on a condition variable in SIMIX
- When all threads are blocked in this way, SIMIX tells the simulation core computes the simulation models
- Threads are then unblocked and proceed until they all enter the SIMIX "kernel" again
- Total separation: application / synchronization / models

Outline

Scalability

- We have spent many years trying to increase scalability
- The first step was fast analytical resource modeling
  - Solving a weighted Max-min problem as opposed to packet-level, cycle-accurate simulation
  - Implementing the solver with cache-efficient data structures
- Scalability issues in SimGrid don't come from the models!

- Four limits to scalability:
  - X Running the simulation models too often
  - X Too large platform descriptions
  - X Too many simulated processes
  - X Simulation limited to a single core

Scalability

- We have spent many years trying to increase scalability
- The first step was fast analytical resource modeling
  - Solving a weighted Max-min problem as opposed to packet-level, cycle-accurate simulation
  - Implementing the solver with cache-efficient data structures
- Scalability issues in SimGrid don't come from the models!

- Four limits to scalability:
  - X Running the simulation models too often
  - X Too large platform descriptions
  - X Too many simulated processes
  - X Simulation limited to a single core

Running models too often

- SimGrid was originally intended for simulating tightly-coupled parallel apps on hierarchical networks
  - e.g., a grid platforms with 3 clusters on a fast wide-area network for running parallel scientific applications
- In this setting, every simulated action can have an impact on every other simulated action
  - A data transfer completion frees up some bandwidth usable by many other transfers
  - A computation completion can lead to a message that will unblock many other computations
- As a result, SimGrid was implemented in the typical loop:
  - Run all simulation models
  - Determining the next event (e.g., action completions)
  - Update all actions remaining work amounts
  - Advance the simulated time and repeat

Running models too often

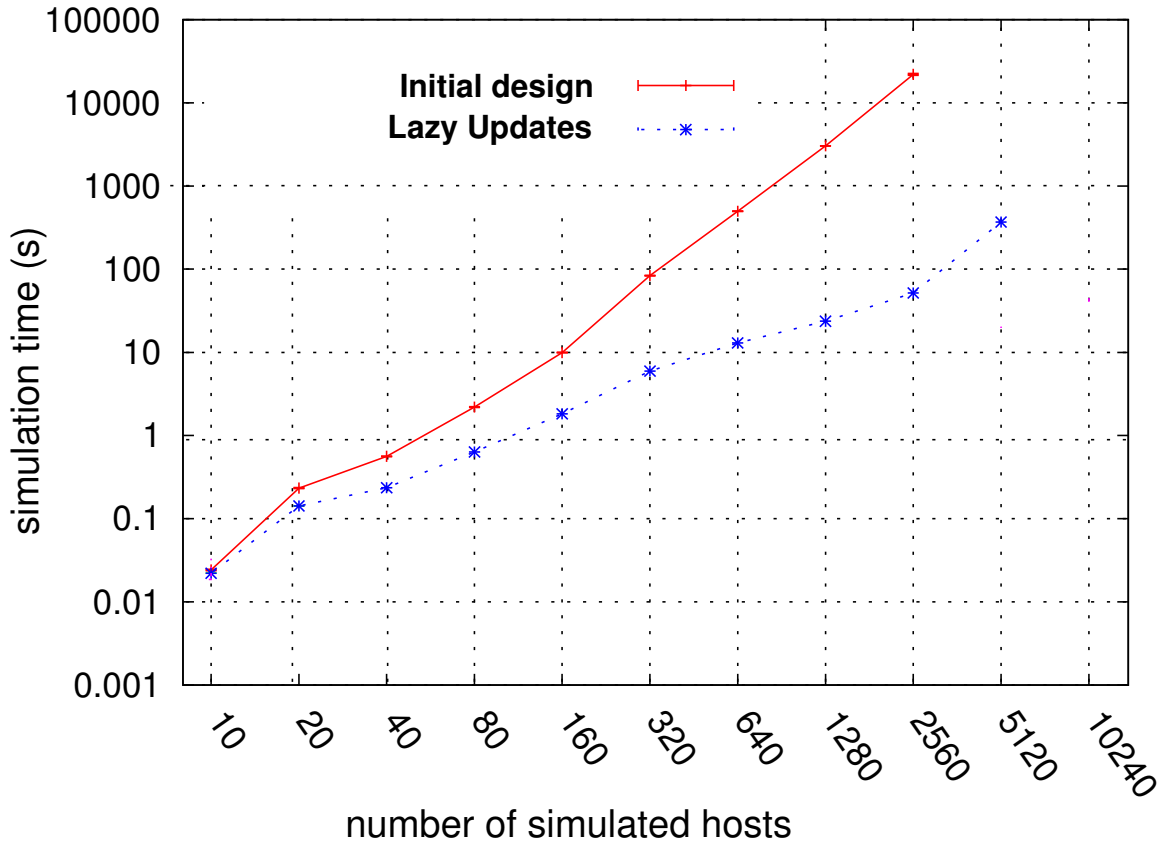
- As SimGrid gained popularity, we and users tried to apply it to different domains
- One such domain: volunteer computing
  - Donated compute cycles and disk space at the edge of the network to contribute to public-interest projects
  - e.g., SETI@Home, AIDS@Home, BOINC, etc.
- In this setting, many actions are independent
  - There is little resource contention among participating hosts
  - Computations are independent and long-running
- Yet there are many events (thousands of simulated hosts)
- Essentially, SimGrid keeps decreasing the remaining work amounts of all actions by ε over and over
- The result: slooooooow simulations at large-scale

Lazy action updates

- Modified "Lazy Updates" simulation loop:
  - All actions are stored in a heap, sorted by their current completion dates
  - When a resource state is modified, we remove relevant actions (those that use the resource) from the heap, we update their remaining work amounts and completion dates, and we re-insert them into the heap
  - Removing/Inserting from/to a heap: $O(\log n)$
  - Finding the next action that completes: $O(1)$
- Not a revolutionary idea of course
  - Large simulation literature on efficient *future event sets*
  - But not seen in parallel/distributed computing simulators
- If the application is tightly coupled, then lazy updates are slower because all actions are removed/inserted
- Lazy updates enabled by default but optional

Lazy updates in action

- Lazy updates for the motivating volunteer computing scenario [Heien et al., 2008]



SimGrid better than specialized simulator?

- Lazy updates are really effective
  - Example: from 3h to 1min for a simulation with 2,560 hosts on a 2.2GHz processor
- And in fact, SimGrid, even though it implements more sophisticated (network) models is $\sim$25 times faster than the SimBA volunteer computing simulator
- SimBA was optimized for scalability in a different way
  - It uses finite automata to describe simulated processes
- And yet, a more versatile simulator can "out-scale" it thanks to careful design

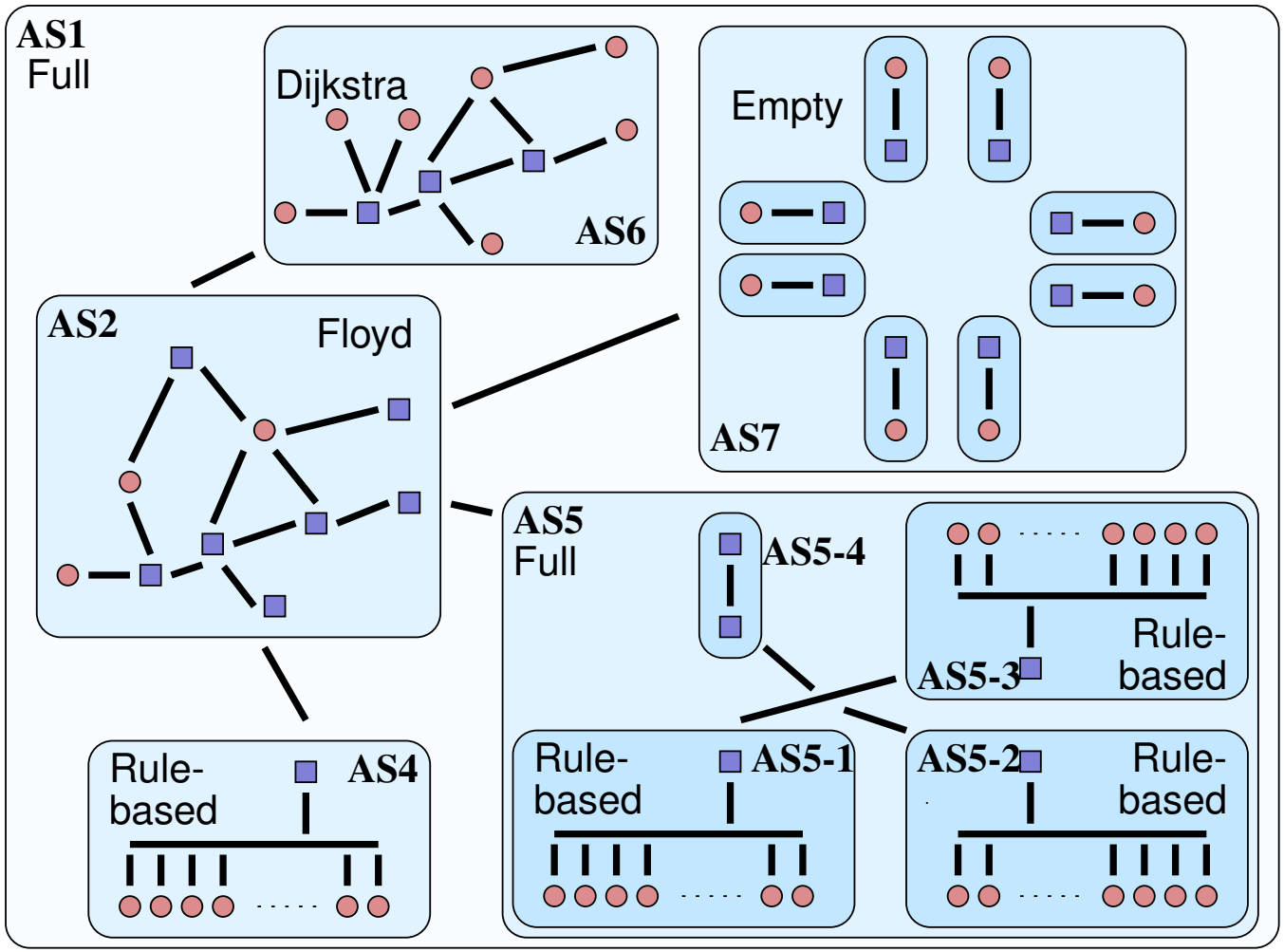
Four limits to SimGrid's scalability

- ✓ Running the simulation models too often
- X Too large platform descriptions
- X Too many simulated processes
- X Simulation limited to a single core

Large platform descriptions

- Users who used SimGrid for truly large-scale platform simulations often found themselves stuck
  - Long XML parse time
  - Out-of-memory errors
  - Long time to compute network routes, especially because we need $N \times (N-1)$ routes for N hosts!
- To enable large-scale simulation we must have hierarchical platform descriptions
- A platform is an Autonomous System (AS), that can contain interconnected ASes
- Each AS has its own routing scheme
  - Full routing tables, Dijkstra, Floyd, no routing, routing based on rules encoded as regular expressions

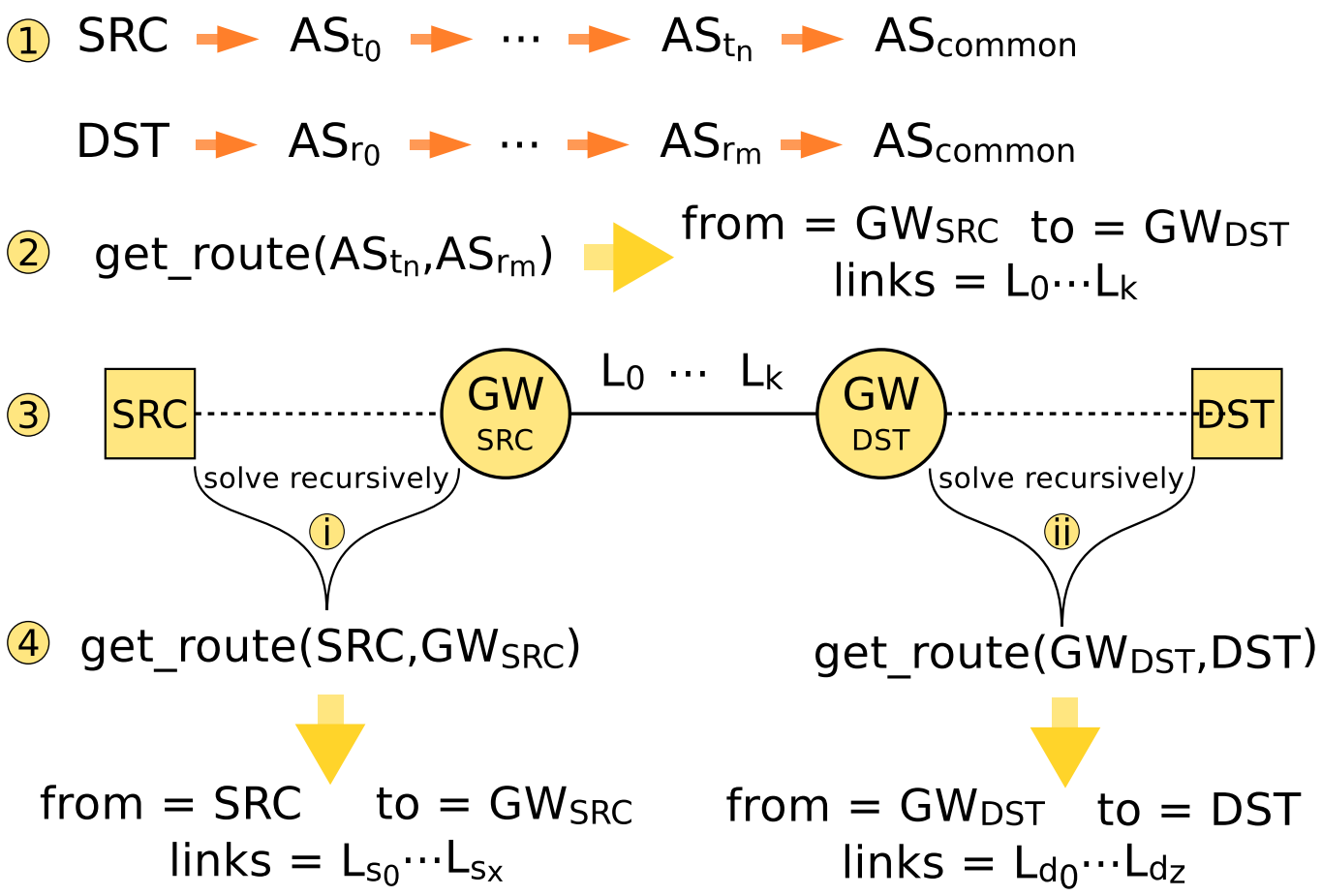
Platform description example



Recursive route computation

- Each AS declared *gateways* to other ASes, and that are used to compute routes
- To determine a route between two hosts:
  1. Search for the common ancestor AS
  2. Search for the route between the two relevant AS children
  3. Repeat until the full route is determined
- Let's see this on a figure...

Recursive route computation



Generating platform descriptions

- The SimGrid user can either generate an XML file, or used SimGrid's platform generation API
- The overhead of (recursively) computing the route is negligible in our implementation
- The memory footprint of the platform description is small
- XML parsing is fast
- Example:
  - The Grid'5000 testbed (10 sites, 40 clusters, 1,500 nodes)
  - Described with 22KiB of XML, parsed in < 1s
  - Previous SimGrid versions: 520MiB, parsed in 20min

Four limits to SimGrid's scalability

- ✓ Running the simulation models too often
- ✓ Too large platform descriptions
- ✗ Too many simulated processes
- ✗ Simulation limited to a single core

Too many threads

- SimGrid allows users to described simulated apps as sets of CSPs
- Great for flexibility and expressivity
- Not scalable if implemented as processes/threads:
  - Thread creation/management overhead in the kernel
  - Thread memory footprint in the kernel
  - Thread synchronization overhead (locks + condvar)
- **But** in a SimGrid simulation threads run in mutual exclusion and in a round-robin fashion
- Therefore, we don't need the full power/flexibility of kernel threads since we do our own scheduling and our own synchronization

Scalable CSPs (I)

- As opposed to having threads each with a bunch of locks and condition variables we take a different approach
- A single "core context":
  - Since simulation models are fast, a single thread does all model computations (i.e., it run the SURF code)
  - All simulated processes place SIMIX simcalls, and all these simcalls are resolved by the core context: no shared state among threads
    - Two simulated processes waiting on each other don't really wait on each other
    - i.e., no multi-step process-to-process interactions
    - Instead, they place wait/notify-like simcalls to the core context

Scalable CSPs (I)

- Lightweight "continuations":
  - Since we don't need full threads we can use cooperative, light-weight, non-preemptive threads
    - Known as continuations
    - No actual context-switching by the kernel
  - Windows: *fibers*
  - Linux, Mac OSX: *ucontexts*
  - We actually re-implemented them in assembly to avoid a costly system call that is not needed for our purpose

How scalable is it?

- With all three scalability improvements so far, we can now compare SimGrid to "competitors"

- Case study #1: Grid computing
  - Master-worker scenario
  - Comparison to GridSim (implemented in Java)
- Case study #2: Peer-to-peer computing
  - The Chord protocol [Stoica et al., 2003]
  - Comparison to PeerSim and OverSim

How scalable is it?

- With all three scalability improvements so far, we can now compare SimGrid to "competitors"
- Case study #1: Grid computing
  - Master-worker scenario
  - Comparison to GridSim (implemented in Java)
- Case study #2: Peer-to-peer computing
  - The Chord protocol [Stoica et al., 2003]
  - Comparison to PeerSim and OverSim

How scalable is it?

- With all three scalability improvements so far, we can now compare SimGrid to "competitors"
- Case study #1: Grid computing
  - Master-worker scenario
  - Comparison to GridSim (implemented in Java)
- Case study #2: Peer-to-peer computing
  - The Chord protocol [Stoica et al., 2003]
  - Comparison to PeerSim and OverSim

Scalability case study #1

- One master, N workers, P tasks, round-robin scheduling
- Simulation on a 2.4GHz core and 8GiB of RAM

- GridSim:
  - No network topology simulated (simply latency+bandwidth communication costs)
- SimGrid:
  - Grid'5000 topology simulated (with TCP flow-level modeling, etc.)

Scalability case study #1

- Polynomial fits based on measured values

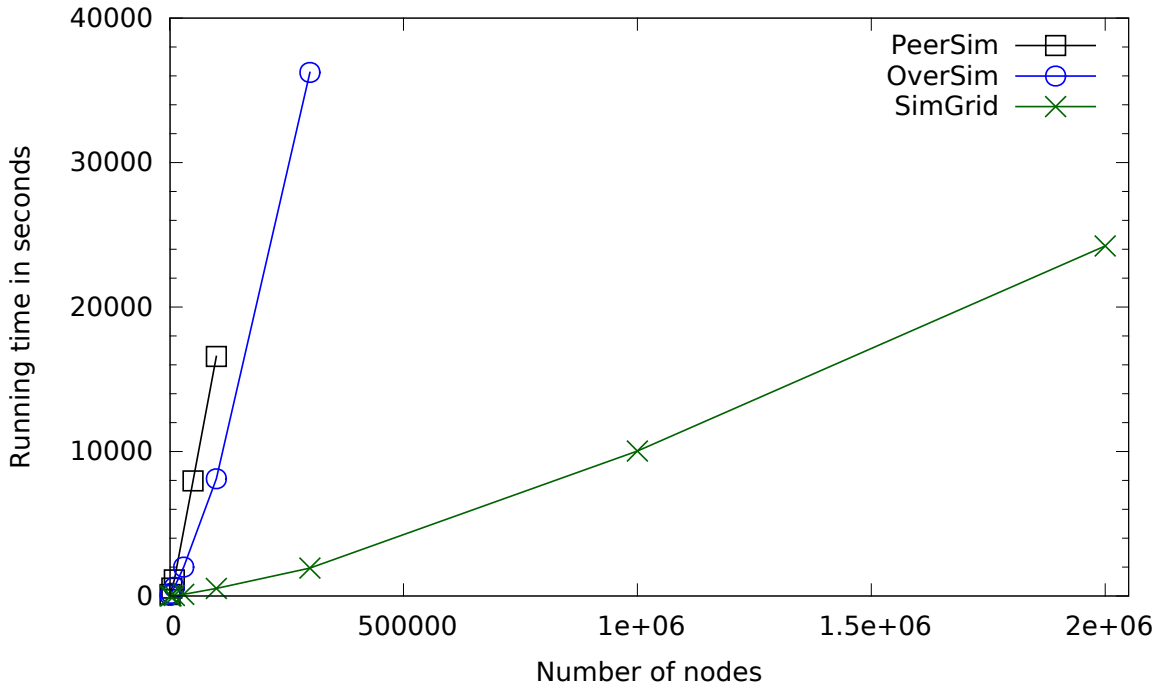
	Simulation Time (s)	Peak Memory Footprint (byte)
GridSim	$5.599 \times 10^{-2}P + 1.405 \times 10^{-8}N^2$	$2.457 \times 10^{6} + 226.6P + 3.11N$
SimGrid	$1.021 \times 10^{-4}P + 2.588 \times 10^{-5}N$	$5188 + 79.9P$

- Example: $N = 2,000$, $P = 500,000$
  - GridSim: 4.4GiB of RAM, > 1hour
  - SimGrid: 165MiB of RAM, < 14s
- And SimGrid uses more sophisticated models!

Scalability case study #2

- Implementation of the Chord protocol for N hosts
- Simulations on a 1.7Ghz core with 48GiB of RAM
- SimGrid
  - TCP flow-level modeling on a full topology
- OverSim [Baumgart et al., 2007]
  - Communication delays based on Euclidian distance between peers
  - Implemented in C++
- PeerSim [Montresor et al., 2009]
  - Constant communication delays
  - Implemented in Java

Scalability case study #2



- PeerSim: 100,000 peers in 4h36min
- OverSim: 200,000 peers in 10h
- SimGrid: 2,000,000 peers in 32min

Scalability results

- We have shown SimGrid to be faster than specialized simulators, even when it uses more sophisticated network simulation models!
- Some of these simulators were designed specifically for scalability (especially p2p simulators)
  - Of course, they may suffer from implementation inefficiencies, while we have spent hours trying to optimize our implementation
- Nevertheless, we claim that it is not necessary to be specialized to be scalable, at least for parallel/distributed computing simulations
- Can we go further?

Scalability results

- We have shown SimGrid to be faster than specialized simulators, even when it uses more sophisticated network simulation models!
- Some of these simulators were designed specifically for scalability (especially p2p simulators)
  - Of course, they may suffer from implementation inefficiencies, while we have spent hours trying to optimize our implementation
- Nevertheless, we claim that it is not necessary to be specialized to be scalable, at least for parallel/distributed computing simulations
- Can we go further?

Four limits to SimGrid's scalability

- ✓ Running the simulation models too often
- ✓ Too large platform descriptions
- ✓ Too many simulated processes
- **X** Simulation limited to a single core

Parallelizing SimGrid

- Because of all the optimizations we've talked about, often most of the compute time is spent in user code!
  - What simulated processes do outside of SimGrid
- There is thus no need to parallelize SimGrid's internals
  - Which would be very difficult anyway since Parallel Discrete Event Simulation is difficult
- We are thus able to run concurrent user processes easily on multiple cores
- Experiments for "difficult cases" (e.g., peer-to-peer Chord) show that achieved speedup is minimal (13%) but non-zero
- Experiments for "easy cases" (e.g., simulated processes that do complex logic in between calls to SimGrid) show that achieved speedup up is large

Parallelizing SimGrid

- Because of all the optimizations we've talked about, often most of the compute time is spent in user code!
  - What simulated processes do outside of SimGrid
- There is thus no need to parallelize SimGrid's internals
  - Which would be very difficult anyway since Parallel Discrete Event Simulation is difficult
- We are thus able to run concurrent user processes easily on multiple cores
- Experiments for "difficult cases" (e.g., peer-to-peer Chord) show that achieved speedup is minimal (13%) but non-zero
- Experiments for "easy cases" (e.g., simulated processes that do complex logic in between calls to SimGrid) show that achieved speedup up is large

Outline

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: 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

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

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?

Some simulators

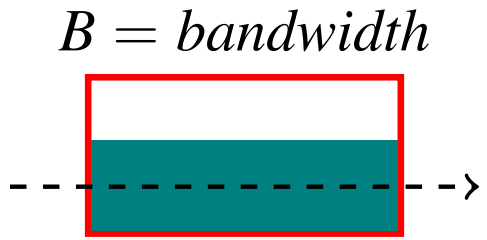
- A research area in which non-packet-level network simulation has been used for decades is “Grid computing” (and lately “Cloud computing”)
  - Clusters federated via wide-area networks, many processes, large data
- Four well-known, and/or popular, and/or widely used simulators that employ flow-level 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?

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?

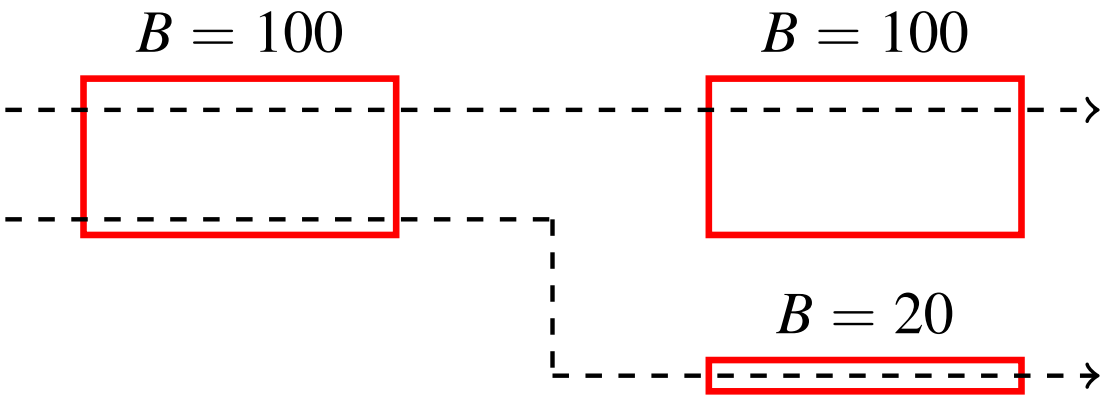
Invalidating experiments

- 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:

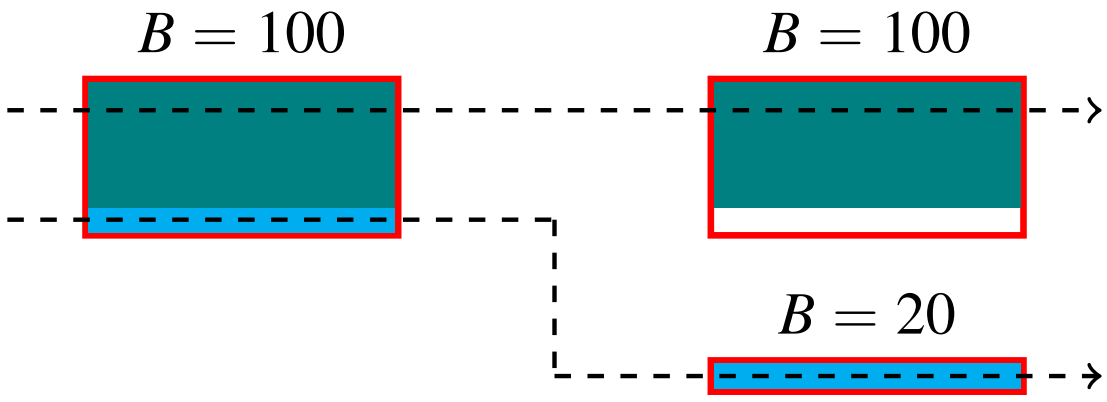


A flow receiving a bandwidth share of a link of bandwidth B

Invalidating OptorSim and GroudSim

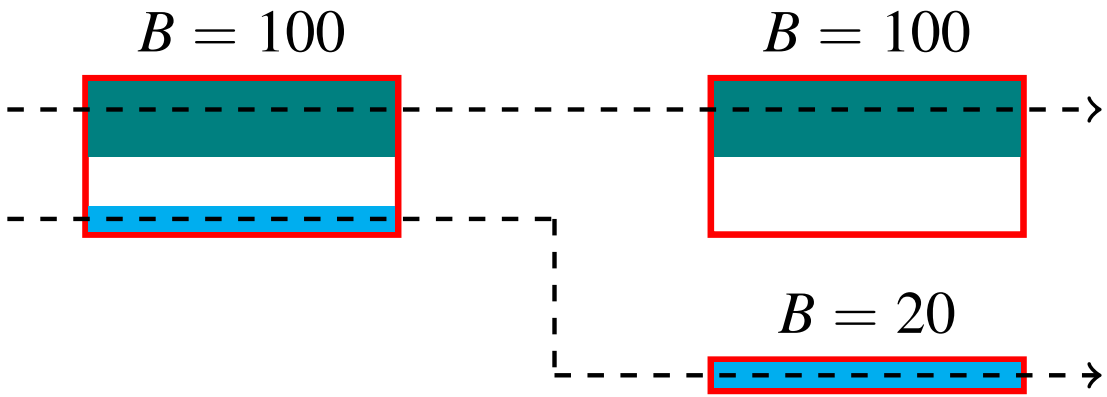


Invalidating OptorSim and GroudSim



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

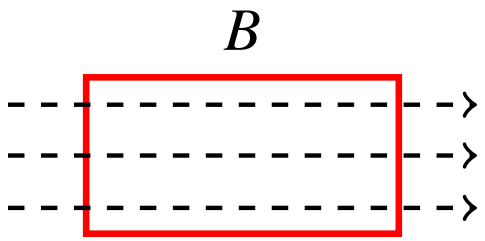
Invalidating OptorSim and GroudSim



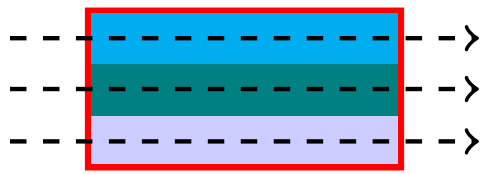
What OptorSim and GroudSim do
(links are shared 50% in spite of bottlenecks)

This "weakness" is document by the authors

Invalidating GridSim

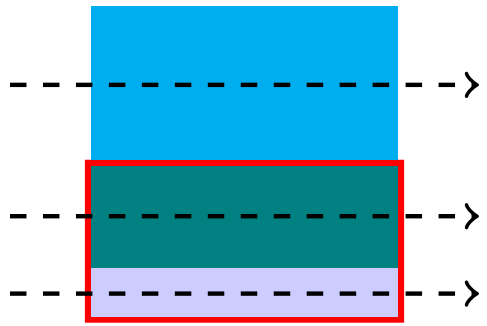


Invalidating GridSim



What a real network would do

Invalidating GridSim

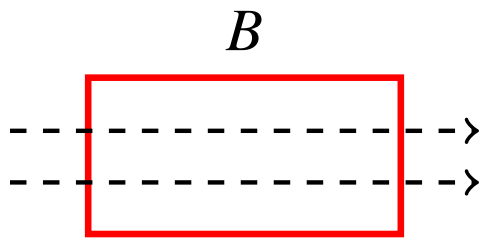


What GridSim does
(first flow receives B, second $B/2$, etc.)

Likely a bug (which we reported)

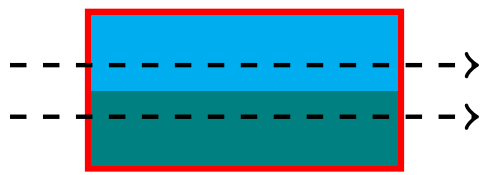
Invalidating CloudSim

One flow starts ϵ seconds after the other



Invalidating CloudSim

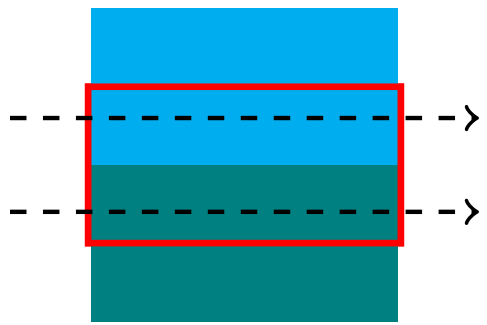
One flow starts ϵ seconds after the other



What a real network would do

Invalidating CloudSim

One flow starts ϵ seconds after the other

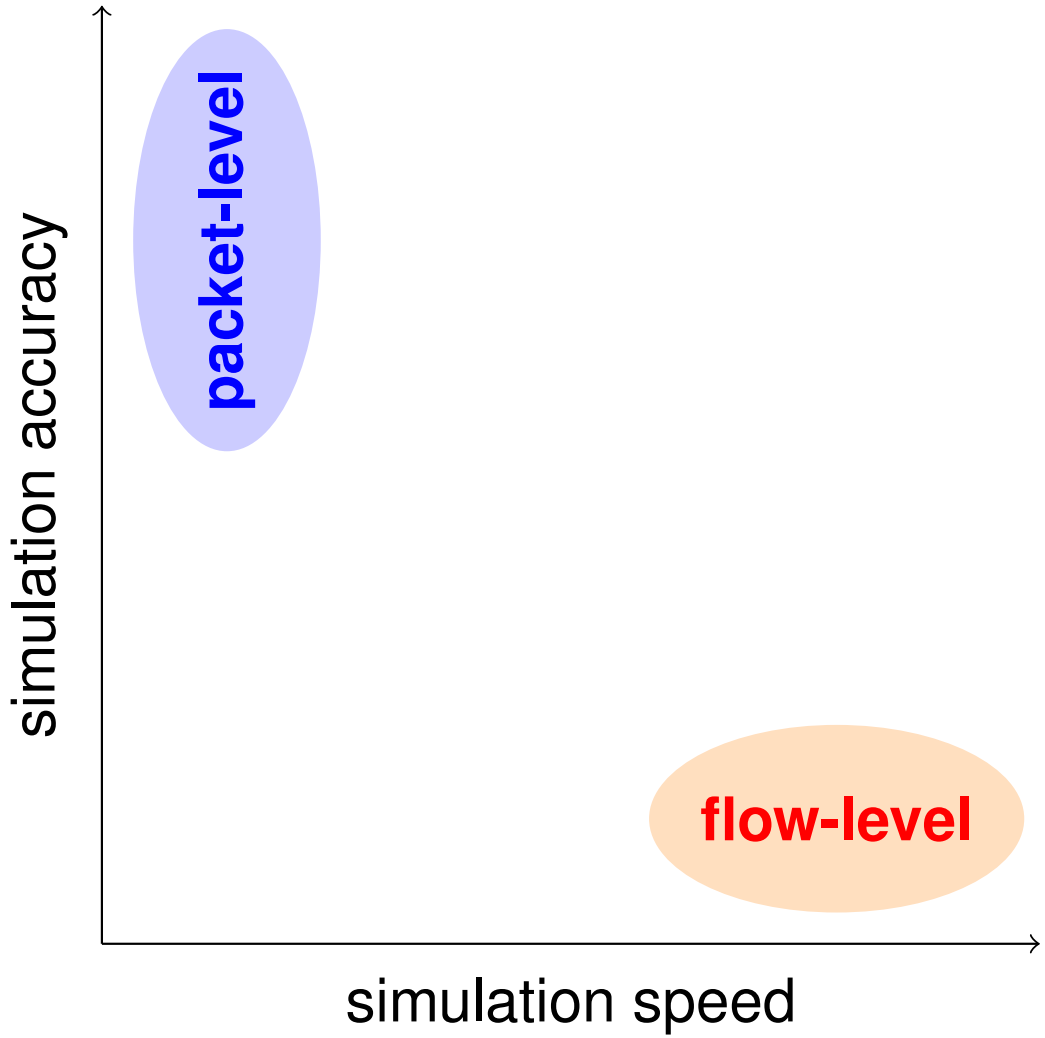


What CloudSim does
(each flow receives B)

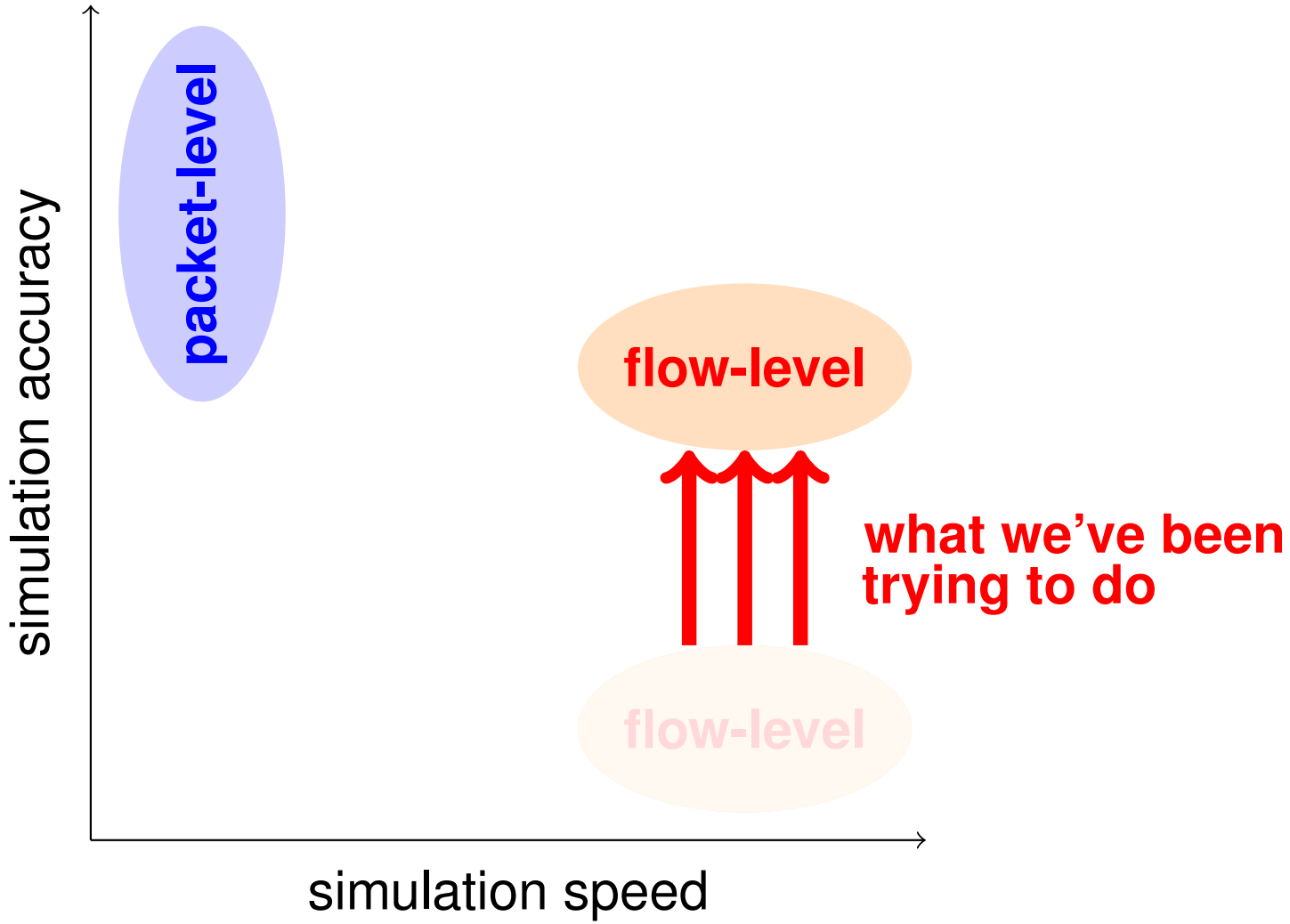
Invalidation is easy???

- 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?



Is there no hope then?



Outline

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 $f \in \mathcal{F}$

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

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

Outline

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]

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]

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

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

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

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...

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 ☺

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 ☺

Outline

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

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

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

- Makes the least happy flow as happy as possible

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

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

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

- Makes the least happy flow as happy as possible

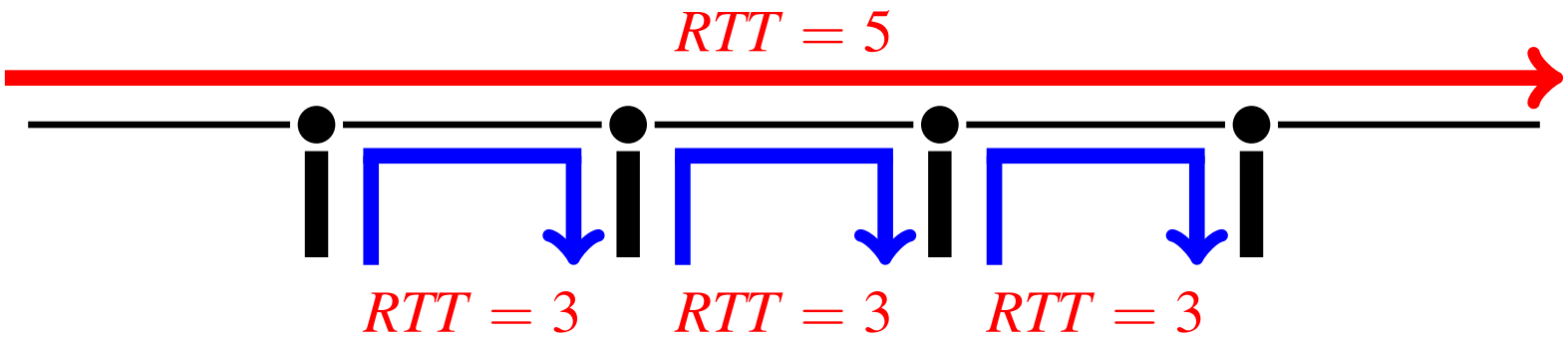
Max-min fairness

- 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**

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 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?

Outline

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

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)

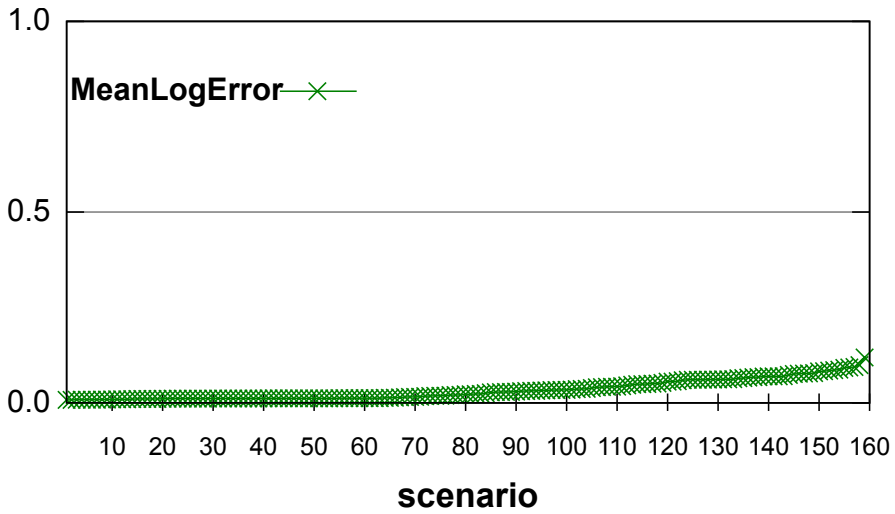
Accuracy metric

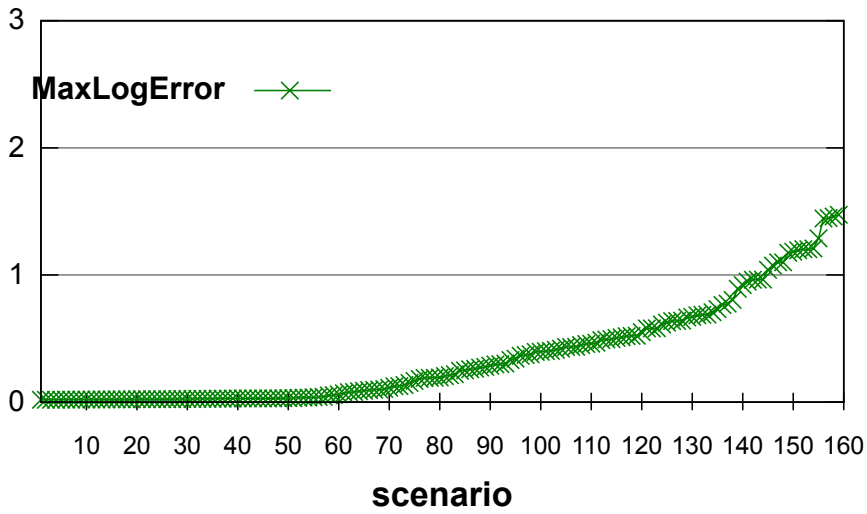
$$\text{MeanLogErr} = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} \left| \log\left(\rho_f^{\textit{flow}}\right) - \log\left(\rho_f^{\textit{packet}}\right) \right| \text{, and}$$

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

- 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



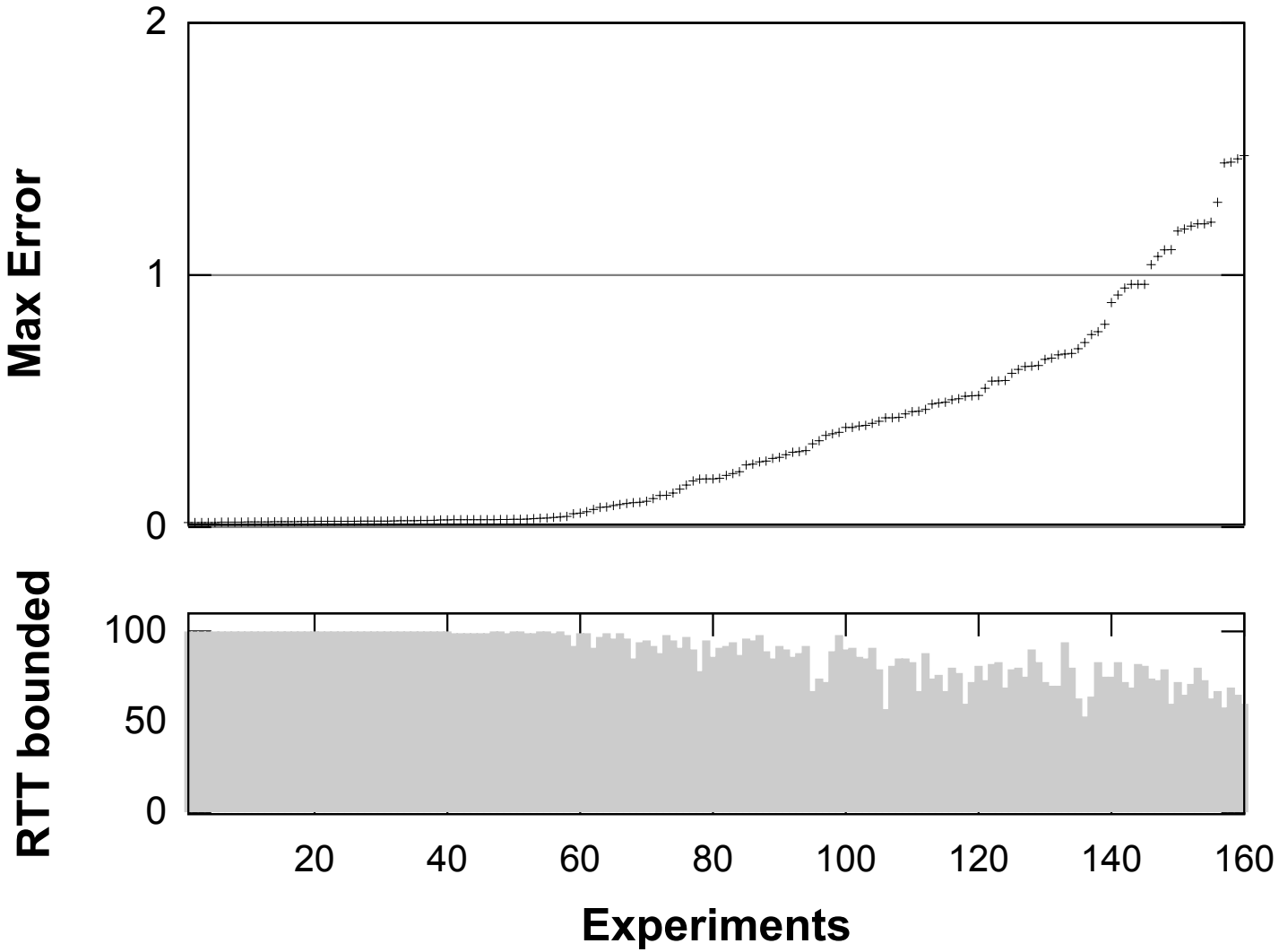


- A few scenarios show high error
- Many show low error: they're **not** "crucial experiments"

Making scenarios more crucial

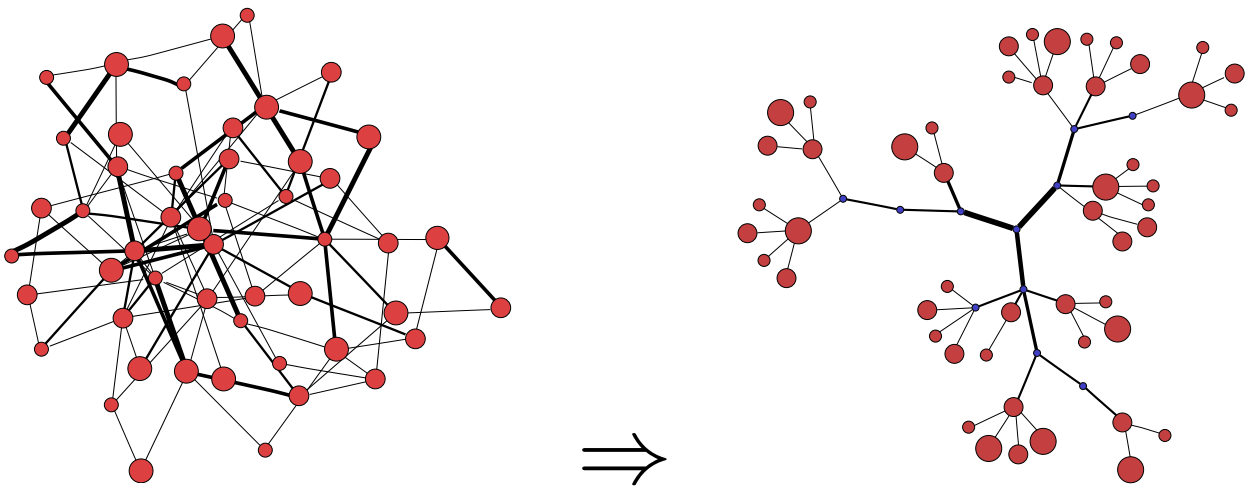
- Inspecting the results, we find that many of the low-error scenarios have mostly RTT-bound flows
- They are governed by the $\rho_f \leq W/(2L_f)$ constraint
  - The "real" bandwidth sharing model is bypassed
- They are easy cases for our flow-level model, hence the low error
- We can see this trend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

More RTT-bounded flow, less crucial

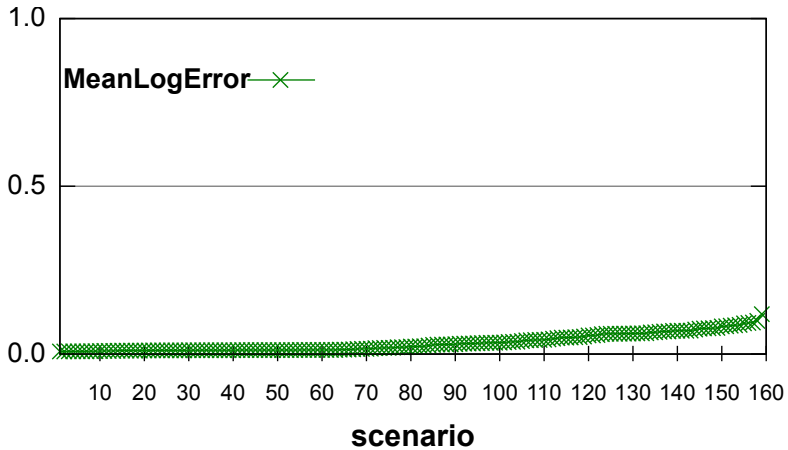


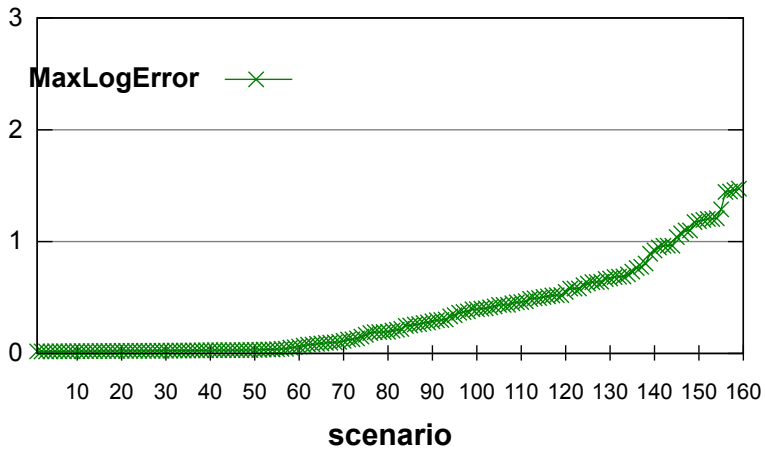
Making experiments more critical

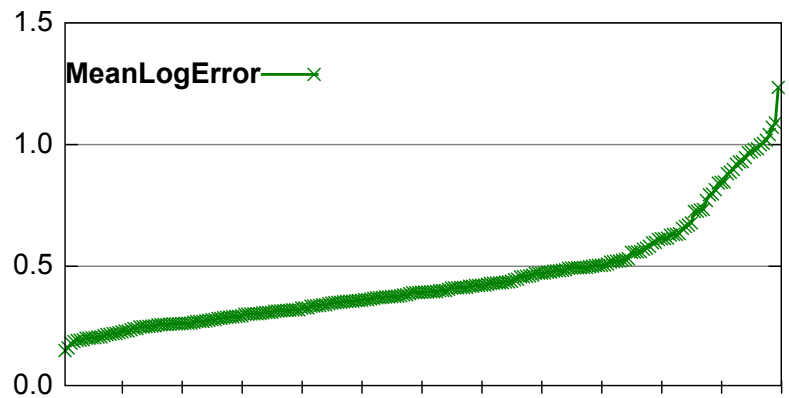
- We sample bandwidths in $[10, 12.8]$ MiB/s
  - Likely rare in practice, but we use the critical method
- We no longer use BRITE, but instead use Tiers

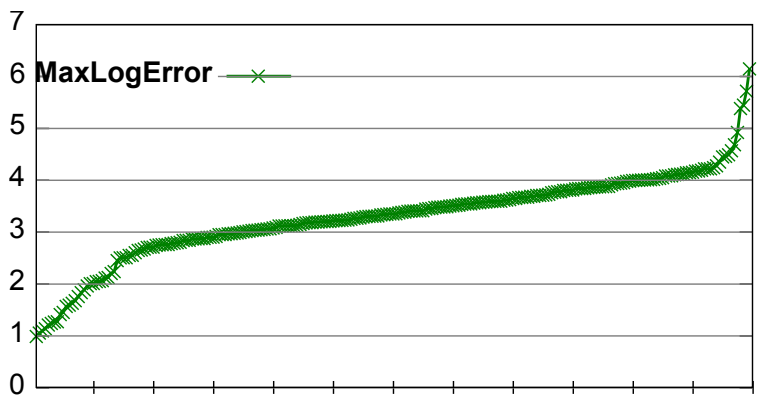


New crucial workload





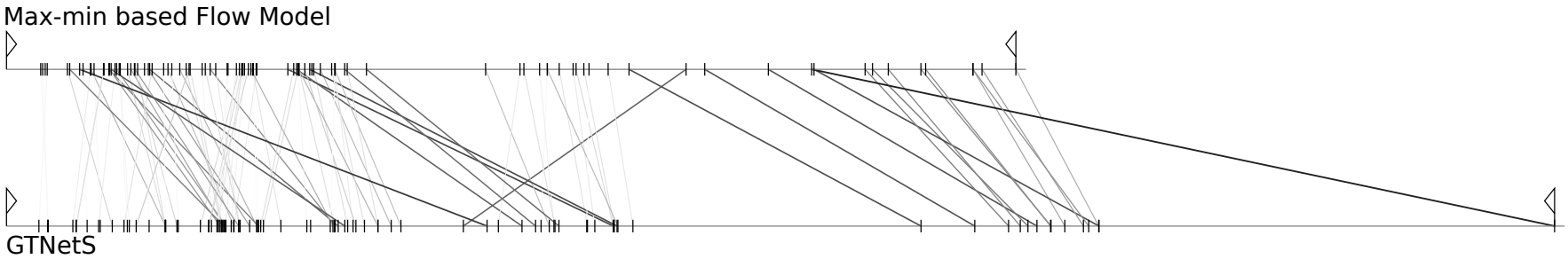




Finding sources of error

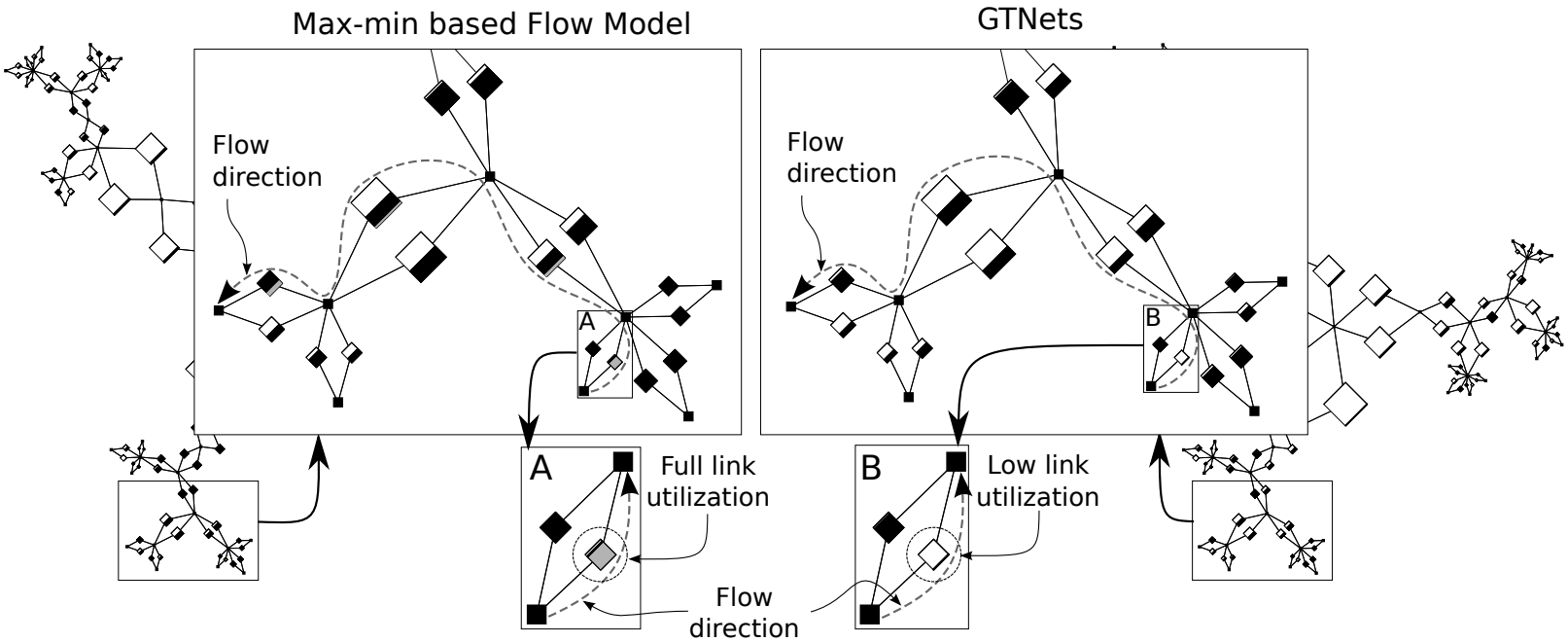
- Now we have a lot of bad results, which is great because now it's easier to try to understand what's wrong with the model
- We pick a particularly bad workload and visualize flow completion time in flow-level simulation and packet-level simulation...

Visualization of completion times



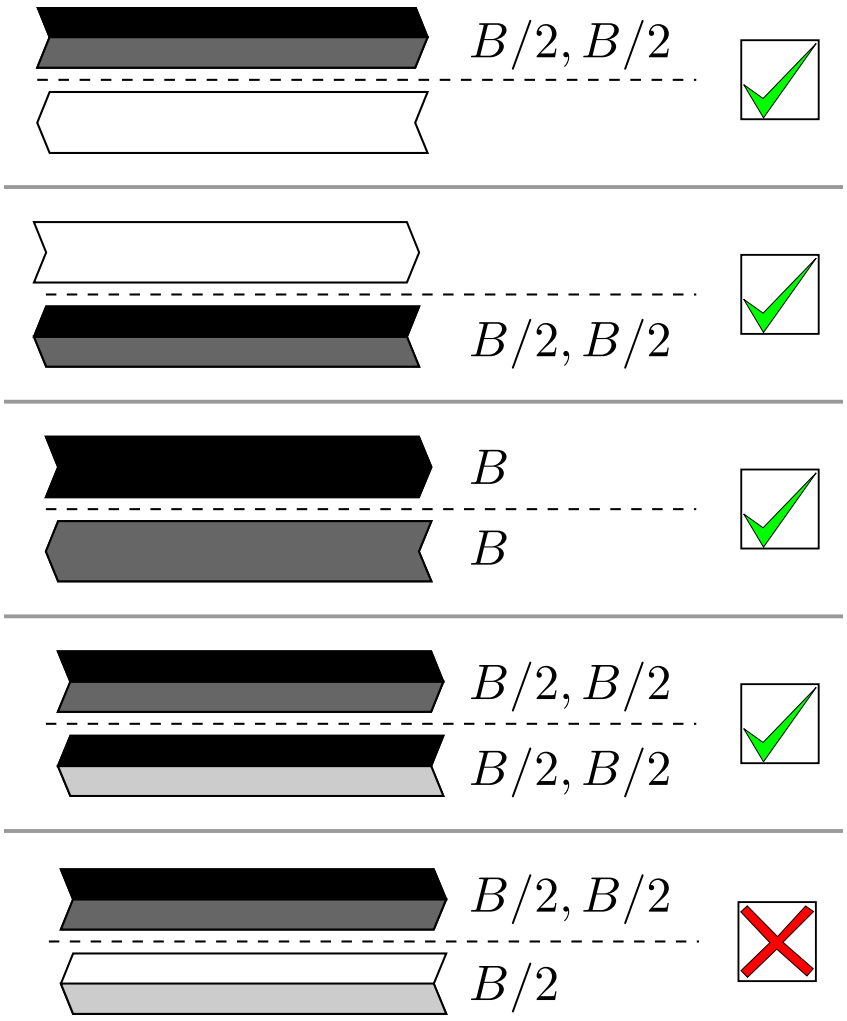
- Many almost vertical lines (light)
- Several really non-vertical lines (dark)
- Flow-level simulation almost always underestimates completion times
  - We have no idea how to explain the one odd flow that is faster with packet-level simulation
- We now use another visualization to inspect the "bad"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

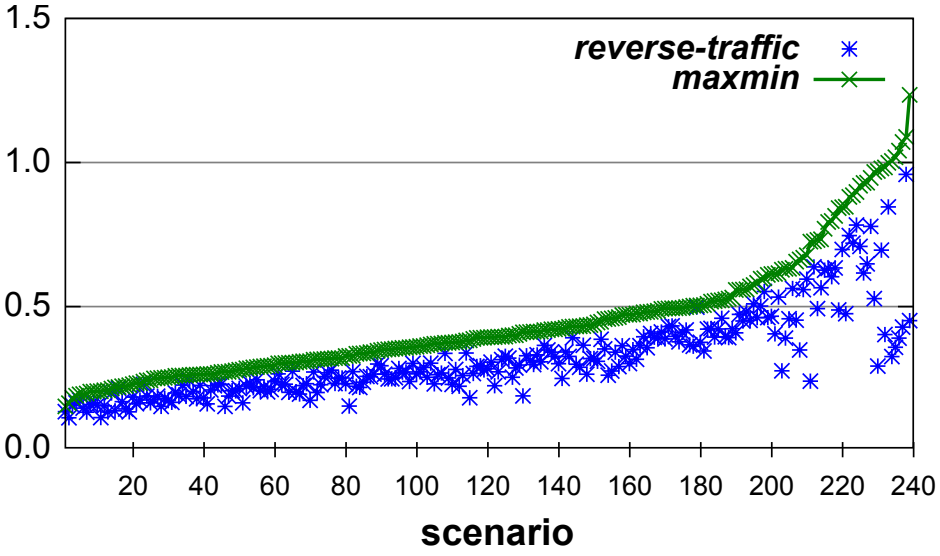
$$\forall f \in \mathcal{F}, \; L_f = \sum_{l \text{ traversed by } f} L_l + \frac{\gamma}{B_l}$$

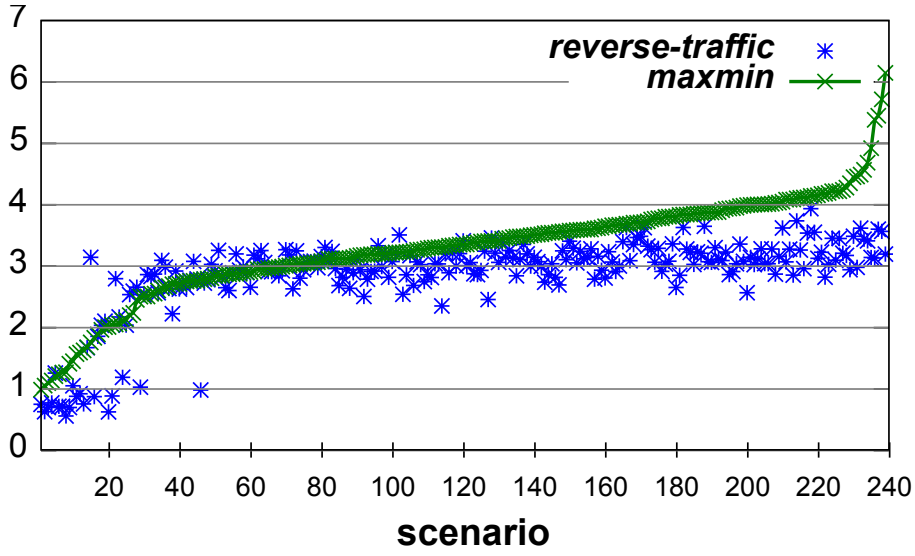
$$\forall l \in \mathcal{L}, \sum_{f \text{ going through } l} \rho_f + \epsilon \left(\sum_{f\text{'s ack going through } l} \rho_f \right) \leq \beta \times B_l$$

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

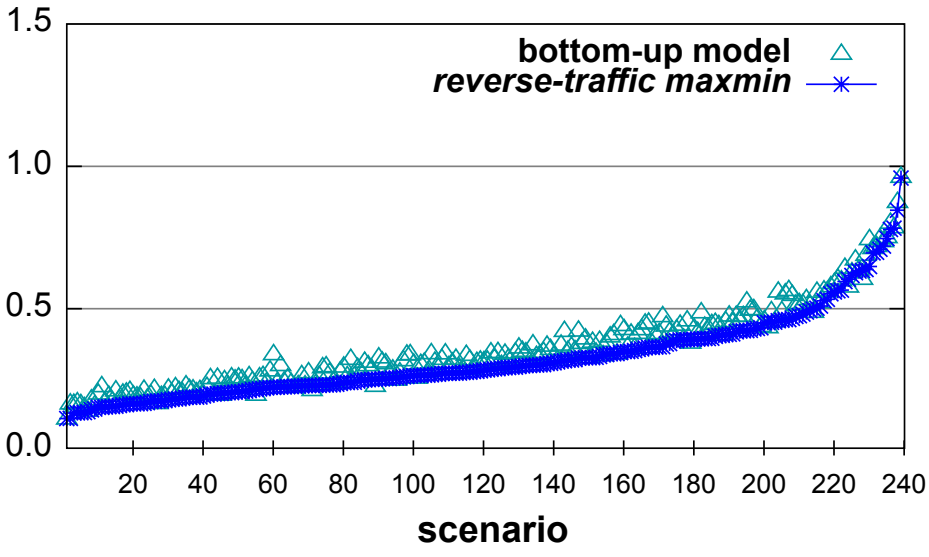
$$t_f = \alpha \, L_f + S_f \, / \, \rho_f$$

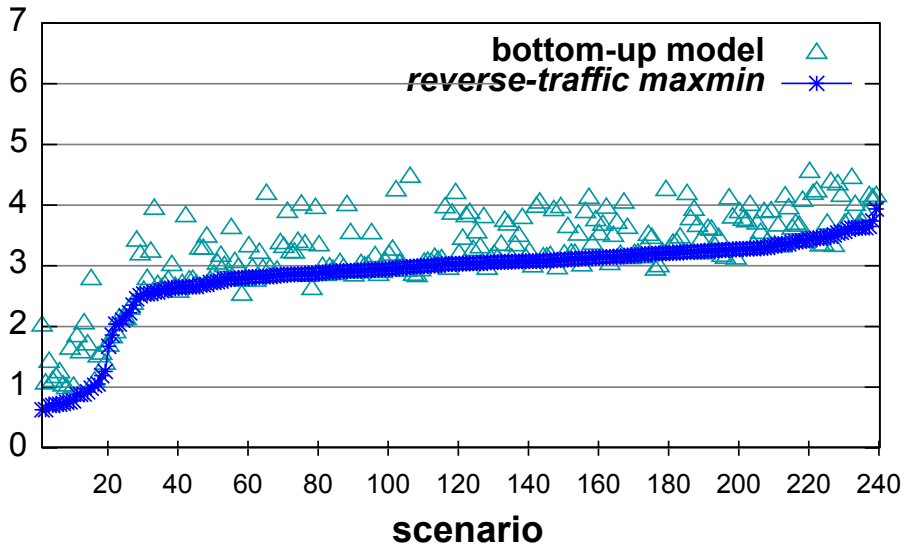
Improved results





Comparison to bottom-up model





Could bottom-up be improved?

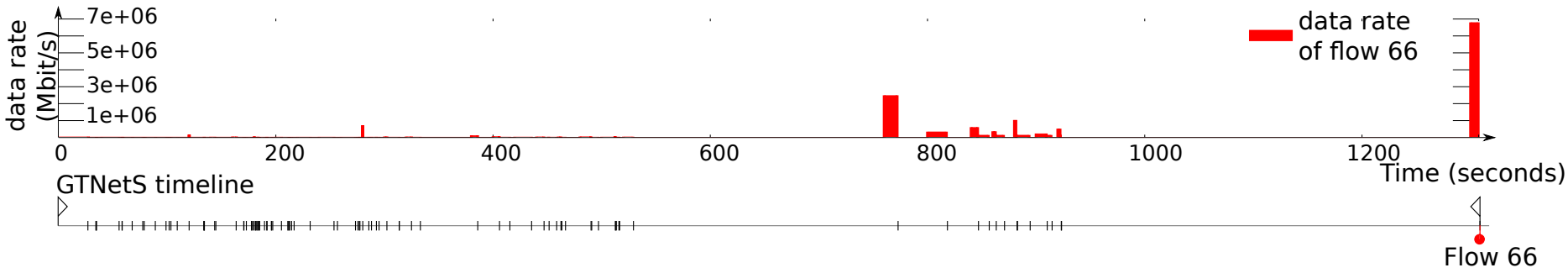
- The model by Low et al. does not account for reverse traffic
- Since the work by Low et al., new advances have been made
  - Differential Algebraic Equation (DAE) model in [Tang et al., 2008, 20010]
  - Ack-clocking dynamics in [Jacobsson et al., 2009]
- These models are not designed for use in simulation
  - 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

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