SURF 101

Getting Started with SimGrid Models

Da SimGrid Team

April 22, 2014

About this Presentation

Goals and Contents

- CPU and Network Models used in SimGrid (and elsewhere)
- Some realism considerations
- Choosing the models used in practice

The SimGrid 101 serie

- This is part of a serie of presentations introducing various aspects of SimGrid
- SimGrid 101. Introduction to the SimGrid Scientific Project
- SimGrid User 101. Practical introduction to SimGrid and MSG
- SimGrid User::Platform 101. Defining platforms and experiments in SimGrid
- SimGrid User::SimDag 101. Practical introduction to the use of SimDag
- SimGrid User::Visualization 101. Visualization of SimGrid simulation results
- SimGrid User::SMPI 101. Simulation MPI applications in practice
- SimGrid User::Model-checking 101. Formal Verification of SimGrid programs
- SimGrid Internal::Models. The Platform Models underlying SimGrid
- SimGrid Internal::Kernel. Under the Hood of SimGrid
- Get them from `http://simgrid.gforge.inria.fr/documentation.html`

Outline

- Requirements
- CPU Model
- Network Models
  - Max-Min Fairness
  - TCP Key Features

Validity: Community Requirements

Networking Protocol design requires accurate packet-level simulations
$\leadsto$ overly detailed network; no CPU

P2P DHT geographic diversity, jitter, churn
$\leadsto$ no need for contention, only delay

P2P streaming network proximity, asymmetry, interference on the edge
$\leadsto$ ignore the core

Grid heterogeneity, complex topology, contention w. large transfers
$\leadsto$ no need to focus on packets

Volunteer Computing dynamic availability, heterogeneity
$\leadsto$ little need for networking

HPC complex communication workload, protocol peculiarities
$\leadsto$ build on regularity and homogeneity

Cloud mixture of previous requirements

Consequence: most simulators are $\underbrace{\text{ad hoc and domain-specific}}_{\text{read “dead within a year or so”}}$

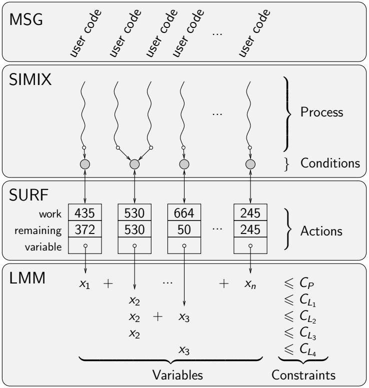
Outline

- Requirements
- CPU Model
- Network Models
  - Max-Min Fairness
  - TCP Key Features

Interactions between the user code and the models

SimGrid Functional Organization

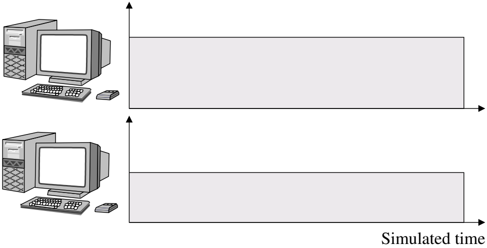
- MSG: User-friendly syntaxic sugar
- Simix: Processes, synchro (SimPosix)
- SURF: Resources usage interface
- Models: Action completion computation



The CPU model in a nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R



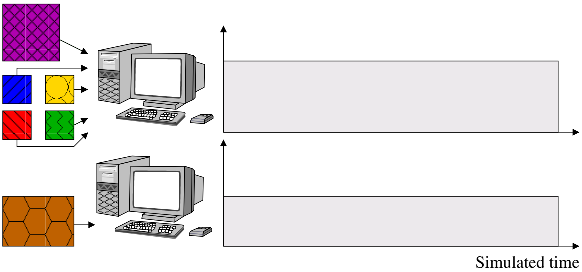
The CPU model in a nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources



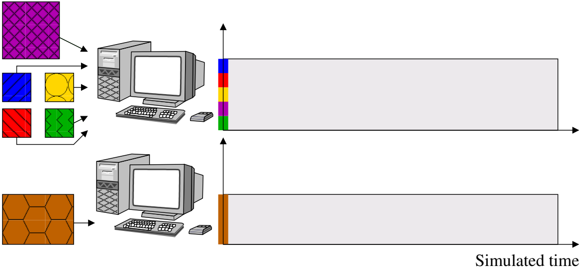
The CPU model in a nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)



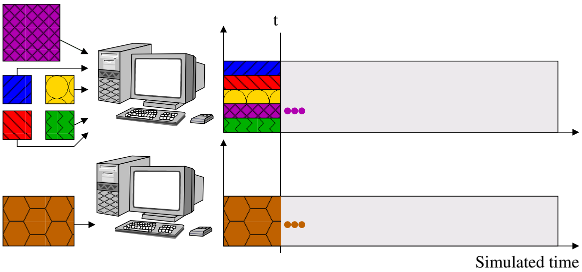
The CPU model in a nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time



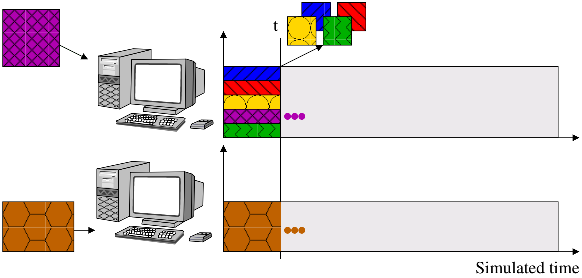
The CPU model in a nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions



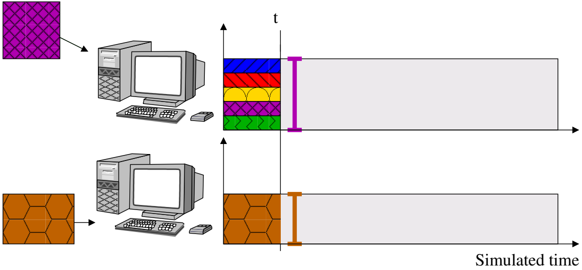
The CPU model in a nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



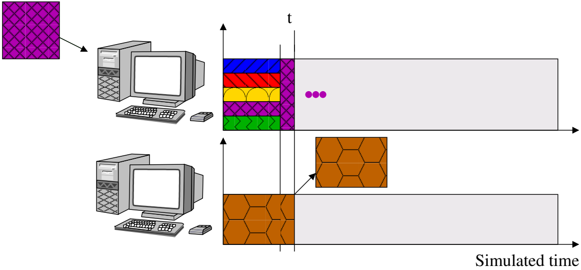
The CPU model in a nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\rightsquigarrow$ Time $= A/R$

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



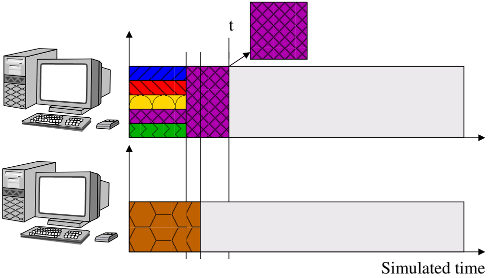
The CPU model in a nutshell

Modeling computations in SimGrid

CPU = rate R in Mflop/s $\oplus$ Computation = amount A of Flops $\leadsto$ Time = A/R

Simulation kernel main loop

1. Some actions get created (by application) and assigned to resources
2. Compute share of everyone (resource sharing algorithms)
3. Compute the earliest finishing action, advance simulated time to that time
4. Remove finished actions
5. Loop back to 2



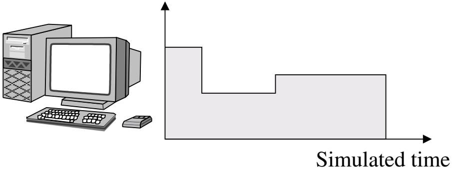
Adding Dynamic Availabilities to the Picture

Trace definition

- List of discrete events where the maximal availability changes
- $t_0 \rightarrow 100\%$, $t_1 \rightarrow 50\%$, $t_2 \rightarrow 80\%$, *etc.*

Adding traces doesn't change kernel main loop

- Availability changes: simulation events, just like action ends



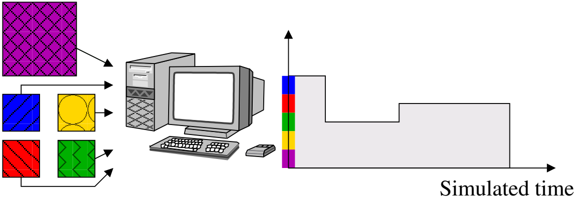
Adding Dynamic Availabilities to the Picture

Trace definition

- List of discrete events where the maximal availability changes
- $t_0 \rightarrow 100\%$, $t_1 \rightarrow 50\%$, $t_2 \rightarrow 80\%$, *etc.*

Adding traces doesn't change kernel main loop

- Availability changes: simulation events, just like action ends



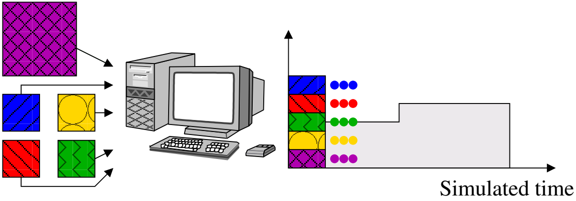
Adding Dynamic Availabilities to the Picture

Trace definition

- List of discrete events where the maximal availability changes
- $t_0 \rightarrow 100\%$, $t_1 \rightarrow 50\%$, $t_2 \rightarrow 80\%$, *etc.*

Adding traces doesn't change kernel main loop

- Availability changes: simulation events, just like action ends



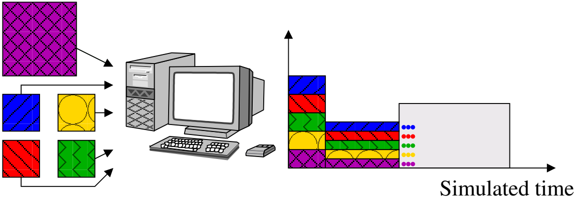
Adding Dynamic Availabilities to the Picture

Trace definition

- List of discrete events where the maximal availability changes
- $t_0 \to 100\%$, $t_1 \to 50\%$, $t_2 \to 80\%$, *etc.*

Adding traces doesn't change kernel main loop

- Availability changes: simulation events, just like action ends



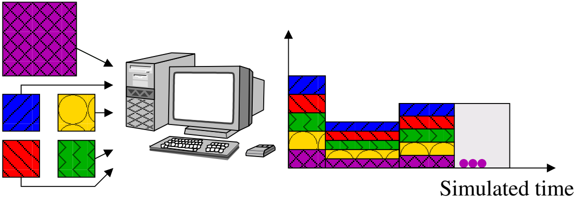
Adding Dynamic Availabilities to the Picture

Trace definition

- List of discrete events where the maximal availability changes
- $t_0 \rightarrow 100\%$, $t_1 \rightarrow 50\%$, $t_2 \rightarrow 80\%$, *etc.*

Adding traces doesn't change kernel main loop

- Availability changes: simulation events, just like action ends



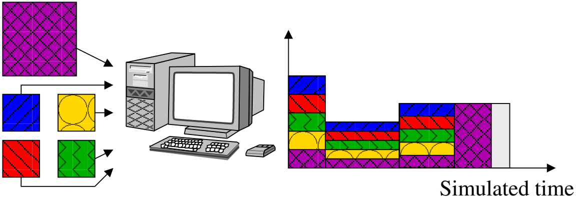
Adding Dynamic Availabilities to the Picture

Trace definition

- List of discrete events where the maximal availability changes
- $t_0 \rightarrow 100\%$, $t_1 \rightarrow 50\%$, $t_2 \rightarrow 80\%$, *etc.*

Adding traces doesn't change kernel main loop

- Availability changes: simulation events, just like action ends
- Efficient implementation thanks to trace integration



SimGrid also accept state changes (on/off)

Pros and Cons

Pros

- Simple and accounts for speed heterogeneity
- Trace integration $\leadsto$ very efficient implementation
- Simple multi-core extension where each process receives $min(R, pR/N)$

Cons

- Too simple:
  - does not account neither for affinity/memory nor compiler/OS peculiarities
    $\leadsto$ rate is bound to a specific kernel
  - (a priori) bad modeling of inter-core communications
  - does not account for cache sharing between cores
    $\leadsto$ neither trashing nor symbiosis
- The failure mechanism has been here for 8 years but people barely use it
- No GPU model in SimGrid yet

Outline

Network Communication Models

Packet-level simulation Networking community has standards, many popular open-source projects (NS, GTneTS, OmNet++,. . .)

- full simulation of the whole protocol stack
- complex models $\leadsto$ hard to instantiate
- inherently **slow**
- beware of simplistic packet-level simulation

Along the same lines: Weaver and MsKee, *Are Cycle Accurate Simulations a Waste of Time?*, Proc. of the Workshop on Duplicating, Deconstruction and Debunking, 2008

Network Communication Models

Packet-level simulation Networking community has standards, many popular open-source projects (NS, GTneTS, OmNet++,. . .)

- full simulation of the whole protocol stack
- complex models $\rightsquigarrow$ hard to instantiate
- inherently **slow**
- beware of simplistic packet-level simulation

Along the same lines: Weaver and MsKee, *Are Cycle Accurate Simulations a Waste of Time?*, Proc. of the Workshop on Duplicating, Deconstruction and Debunking, 2008

Delay-based models The simplest ones. . .

- communication time = constant delay, statistical distribution, LogP
  $\rightsquigarrow$($\Theta(1)$ footprint and $O(1)$ computation)
- coordinate based systems to account for geographic proximity
  $\rightsquigarrow$($\Theta(N)$ footprint and $O(1)$ computation)

Although very scalable, these models ignore network congestion and typically assume large bissection bandwidth

Network Communication Models (cont'd)

Flow-level models A communication (flow) is simulated as a single entity:

$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

Estimating $B_{i,j}$ requires to account for interactions with other flows

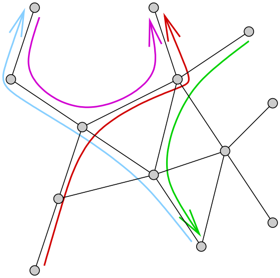
Network Communication Models (cont'd)

Flow-level models A communication (flow) is simulated as a single entity:

$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

Estimating $B_{i,j}$ requires to account for interactions with other flows

Assume steady-state and **share bandwidth** every time a new flow appears or disappears



Setting a set of flows $\mathcal{F}$ and a set of links $\mathcal{L}$

Constraints For all link j: $\sum_{\textbf{if flow i uses link j}} \varrho_i \leqslant C_j$

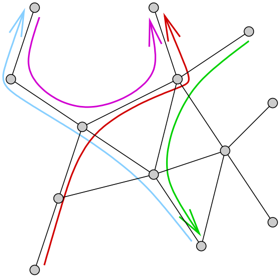
Network Communication Models (cont'd)

Flow-level models A communication (flow) is simulated as a single entity:

$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

Estimating $B_{i,j}$ requires to account for interactions with other flows

Assume steady-state and **share bandwidth** every time a new flow appears or disappears



Setting a set of flows $\mathcal{F}$ and a set of links $\mathcal{L}$

Constraints For all link j: $\sum_{\textbf{if flow i uses link j}} \varrho_i \leqslant C_j$

Objective function

- Max-Min $\max(\min(\varrho_i))$
- or other fancy objectives
  e.g., Reno $\sim \max(\sum \log(\varrho_i))$

Outline

- Requirements
- CPU Model
- Network Models
  - Max-Min Fairness
  - TCP Key Features

Max-Min Fairness

Objective function: maximize $\min_{f \in \mathcal{F}}(\varrho_f)$

- Equilibrium reached if increasing any ϱ_f decreases a ϱ'_f (with $\varrho_f > \varrho'_f$)
- Very reasonable goal: gives fair share to anyone
- Optionally, one can add prorities w_i for each flow i
  $\rightsquigarrow$ maximizing $\min_{f \in \mathcal{F}}(w_f \varrho_f)$

Bottleneck links

- For each flow f, one of the links is the limiting one l
  (with more on that link l, the flow f would get more overall)
- The objective function gives that l is saturated, and f gets the biggest share

$$\forall f \in \mathcal{F}, \exists l \in f, \quad \sum_{f' \ni l} \varrho_{f'} = C_l \quad \text{and} \quad \varrho_f = \max\{\varrho_{f'}, f' \ni l\}$$

L. Massoulié and J. Roberts, *Bandwidth sharing: objectives and algorithms*, IEEE/ACM Trans. Netw., vol. 10, no. 3, pp. 320-328, 2002.

Implementation of Max-Min Fairness

Bucket-filling algorithm

- Set the bandwidth of all flows to 0
- Increase the bandwidth of every flow by ε. And again, and again, and again.
- When one link is saturated, all flows using it are limited ($\leadsto$ removed from set)
- Loop until all flows have found a limiting link

Implementation of Max-Min Fairness

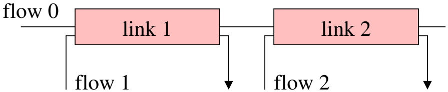
Bucket-filling algorithm

- Set the bandwidth of all flows to 0
- Increase the bandwidth of every flow by ε. And again, and again, and again.
- When one link is saturated, all flows using it are limited ($\leadsto$ removed from set)
- Loop until all flows have found a limiting link

Efficient Algorithm

1. Search for **the** bottleneck link l so that: $\dfrac{C_l}{n_l} = min\left\{\dfrac{C_k}{n_k},\ k \in \mathcal{L}\right\}$
2. $\forall f \in l,\ \ \varrho_f = \frac{C_l}{n_l}$;
   Update all n_l and C_l to remove these flows
3. Loop until all ϱ_f are fixed

Max-Min Fairness on Homogeneous Linear Network

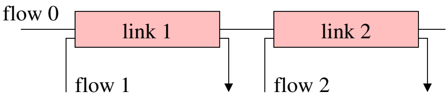


$$C_1 = C \qquad n_1 = 2$$
$$C_2 = C \qquad n_2 = 2$$

$$\varrho_0 =$$
$$\varrho_1 =$$
$$\varrho_2 =$$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

Max-Min Fairness on Homogeneous Linear Network



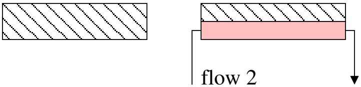
$$C_1 = C \qquad n_1 = 2$$
$$C_2 = C \qquad n_2 = 2$$

$$\varrho_0 = C/2$$
$$\varrho_1 = C/2$$
$$\varrho_2 =$$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\varrho_0 = C/2$ and $\varrho_1 = C/2$

Max-Min Fairness on Homogeneous Linear Network



$C_1 = 0 \qquad n_1 = 0$
$C_2 = C/2 \qquad n_2 = 1$

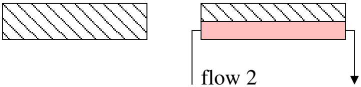
$\varrho_0 = C/2$
$\varrho_1 = C/2$
$\varrho_2 =$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\varrho_0 = C/2$ and $\varrho_1 = C/2$

- Remove flows 0 and 1; Update links' capacity

Max-Min Fairness on Homogeneous Linear Network



$C_1 = 0 \qquad n_1 = 0$
$C_2 = 0 \qquad n_2 = 0$

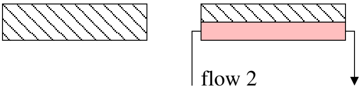
$\varrho_0 = C/2$
$\varrho_1 = C/2$
$\varrho_2 = C/2$

- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\varrho_0 = C/2$ and $\varrho_1 = C/2$

- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\varrho_1 = C/2$

Max-Min Fairness on Homogeneous Linear Network



$C_1 = 0$ $n_1 = 0$
$C_2 = 0$ $n_2 = 0$

$\varrho_0 = C/2$
$\varrho_1 = C/2$
$\varrho_2 = C/2$

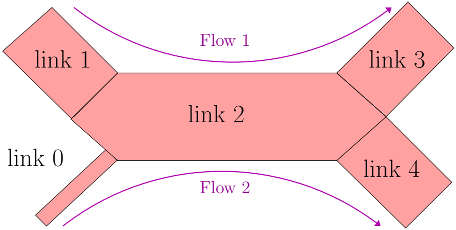
- All links have the same capacity C
- Each of them is limiting. Let's choose link 1

$\Rightarrow$ $\varrho_0 = C/2$ and $\varrho_1 = C/2$

- Remove flows 0 and 1; Update links' capacity
- Link 2 sets $\varrho_1 = C/2$

We're done computing the bandwidth allocated to each flow

Max-Min Fairness on Backbone



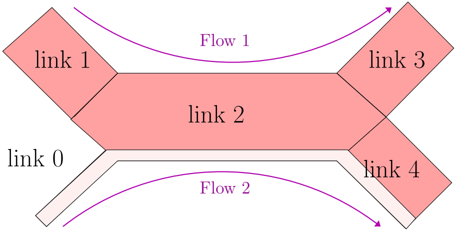
$$
\begin{array}{ll}
C_0 = 1 & n_0 = 1 \\
C_1 = 1000 & n_1 = 1 \\
C_2 = 1000 & n_2 = 2 \\
C_3 = 1000 & n_3 = 1 \\
C_4 = 1000 & n_4 = 1
\end{array}
$$

$\varrho_1 =$

$\varrho_2 =$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$

Max-Min Fairness on Backbone

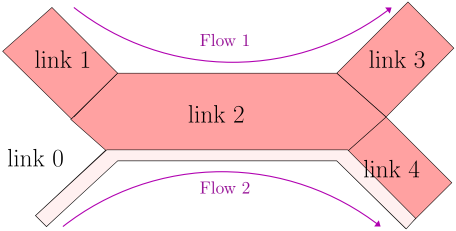


$C_0 = 0$ $\quad n_0 = 0$
$C_1 = 1000$ $\quad n_1 = 1$
$C_2 = 999$ $\quad n_2 = 1$
$C_3 = 1000$ $\quad n_3 = 1$
$C_4 = 999$ $\quad n_4 = 0$

$\varrho_1 =$
$\varrho_2 = 1$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\varrho_2 = 1$. Update the links

Max-Min Fairness on Backbone

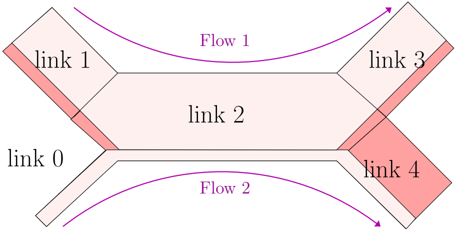


$C_0 = 0$ $\quad n_0 = 0$
$C_1 = 1000$ $\quad n_1 = 1$
$C_2 = 999$ $\quad n_2 = 1$
$C_3 = 1000$ $\quad n_3 = 1$
$C_4 = 999$ $\quad n_4 = 0$

$\varrho_1 =$
$\varrho_2 = 1$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\varrho_2 = 1$. Update the links
- The limiting link is link 2

Max-Min Fairness on Backbone

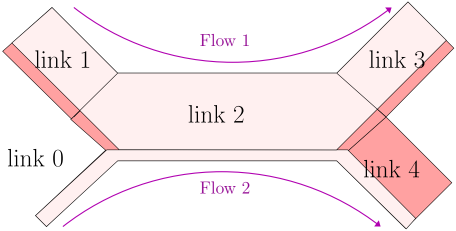


$C_0 = 0$ $n_0 = 0$
$C_1 = 1$ $n_1 = 0$
$C_2 = 0$ $n_2 = 0$
$C_3 = 1$ $n_3 = 0$
$C_4 = 999$ $n_4 = 0$

$\varrho_1 = 999$
$\varrho_2 = 1$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\varrho_2 = 1$. Update the links
- The limiting link is link 2
- This fixes $\varrho_1 = 999$

Max-Min Fairness on Backbone



$C_0 = 0$ $\quad n_0 = 0$
$C_1 = 1$ $\quad n_1 = 0$
$C_2 = 0$ $\quad n_2 = 0$
$C_3 = 1$ $\quad n_3 = 0$
$C_4 = 999$ $\quad n_4 = 0$

$\varrho_1 = 999$
$\varrho_2 = 1$

- The limiting link is link 0 $\left(\text{since } \frac{1}{1} = min\left(\frac{1}{1}, \frac{1000}{1}, \frac{1000}{2}, \frac{1000}{1}, \frac{1000}{1}\right)\right)$
- This fixes $\varrho_2 = 1$. Update the links
- The limiting link is link 2
- This fixes $\varrho_1 = 999$
- Done. We know ϱ_1 and ϱ_2

Side note: OptorSim 2.1 on Backbone

OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using simplistic packet-level routing

Unfortunately, "strange" resource sharing:

1. For each link, compute the share that each flow may get: $\frac{C_l}{n_l}$
2. For each flow, compute what it gets: $\varrho_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$

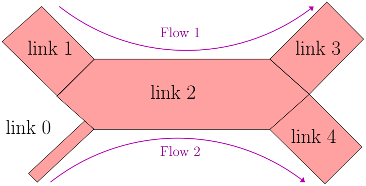
Side note: OptorSim 2.1 on Backbone

OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using simplistic packet-level routing

Unfortunately, “strange” resource sharing:

1. For each link, compute the share that each flow may get: $\frac{C_l}{n_l}$
2. For each flow, compute what it gets: $\varrho_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$$
\begin{array}{lll}
C_0 = 1 & n_1 = 1 & \text{share} = \\
C_1 = 1000 & n_1 = 1 & \text{share} = \\
C_2 = 1000 & n_2 = 2 & \text{share} = \\
C_3 = 1000 & n_3 = 1 & \text{share} = \\
C_4 = 1000 & n_4 = 1 & \text{share} =
\end{array}
$$

$\varrho_1 =$

$\varrho_2 =$

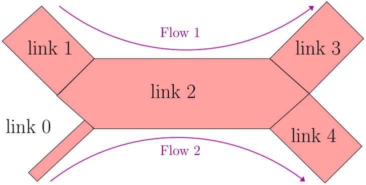
Side note: OptorSim 2.1 on Backbone

OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using simplistic packet-level routing

Unfortunately, “strange” resource sharing:

1. For each link, compute the share that each flow may get: $\frac{C_l}{n_l}$
2. For each flow, compute what it gets: $\varrho_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$$\begin{array}{lll} C_0 = 1 & n_1 = 1 & \text{share} = 1 \\ C_1 = 1000 & n_1 = 1 & \text{share} = 1000 \\ C_2 = 1000 & n_2 = 2 & \text{share} = 500 \\ C_3 = 1000 & n_3 = 1 & \text{share} = 1000 \\ C_4 = 1000 & n_4 = 1 & \text{share} = 1000 \end{array}$$

$$\varrho_1 = min(1000, 500, 1000)$$
$$\varrho_2 = min(\ 1\ \ \ , 500, 1000)$$

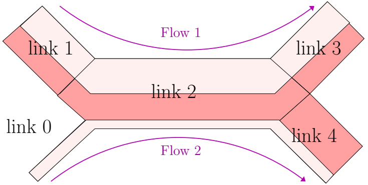
Side note: OptorSim 2.1 on Backbone

OptorSim (developped @CERN for Data-Grid)

- One of the rare ad-hoc simulators not using simplistic packet-level routing

Unfortunately, "strange" resource sharing:

1. For each link, compute the share that each flow may get: $\frac{C_l}{n_l}$
2. For each flow, compute what it gets: $\varrho_f = \min_{l \in f} \left(\frac{C_l}{n_l} \right)$



$$
\begin{array}{lll}
C_0 = 1 & n_1 = 1 & \text{share} = 1 \\
C_1 = 1000 & n_1 = 1 & \text{share} = 1000 \\
C_2 = 1000 & n_2 = 2 & \text{share} = \mathit{500} \\
C_3 = 1000 & n_3 = 1 & \text{share} = 1000 \\
C_4 = 1000 & n_4 = 1 & \text{share} = 1000
\end{array}
$$

$\varrho_1 = min(1000, 500, 1000) = \mathbf{500}!!$

$\varrho_2 = min(\ 1\ \ , 500, 1000) = 1$

ϱ_1 limited by link 2, but 499 still unused on link 2

This "unwanted feature" is even listed in the README file...

Side note: GridSim and CloudSim

GridSim 5.2 on a single link

- Packet-level: latency paid for every packet, not only the first one
  $\rightsquigarrow$ actually wormhole of packets
- Flow sharing: buggy, only subsequent flows share the link
  The code intend seems to be on reevaluating the sharings, but it fails on tests

CloudSim

- Flow sharing, but no sharing between flows starting at t and $t + \varepsilon$
- Consequence:
  - 1 message of size S takes time t
  - 2 flows take time T and 2T.
    Not because it's FIFO but because bandwidth allocation is not reavaluated upon flow arrival and departure...

Proportional Fairness

Max-Min validity limits

- MaxMin gives a fair share to everyone
- Reasonable, but TCP does not do so
- Congestion mecanism: Additive Increase, Muplicative Decrease (AIMD)
- Complicates modeling, as shown in literature

Other sharing methods

- MaxMin gives more to long flows (resource-eager), TCP known to do opposite
- TCP Vegas achieves weighted proportionnal fairness and maximizes:

$$\sum_{f \in \mathcal{F}} L_f \log(\varrho_f) \qquad (L_f \text{ being the latency})$$

- TCP Reno maximizes $\displaystyle\sum_{f \in \mathcal{F}} \frac{\sqrt{3/2}}{L_f} \arctan\left(\frac{\sqrt{3/2}}{L_f} . \varrho_f\right)$

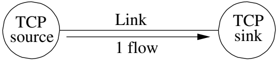
Kelly, *Charging and rate control for elastic traffic*, in European Transactions on Telecommunications, vol. 8, 1997, pp. 33-37.

Outline

- Requirements
- CPU Model
- Network Models
  - Max-Min Fairness
  - TCP Key Features

Wanted Feature (1): Flow Control Limitation

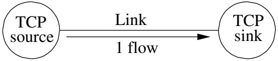
Experimental settings



- Flow throughput as function of L and B
- Fixed size (S=100MB) and window (W=20KB)

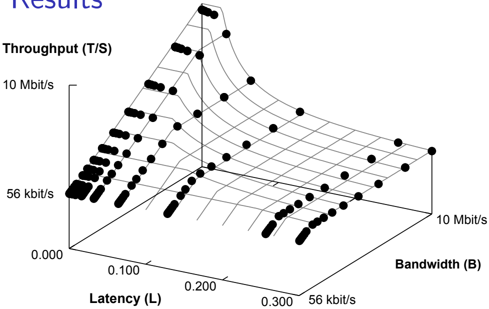
Wanted Feature (1): Flow Control Limitation

Experimental settings



- Flow throughput as function of L and B
- Fixed size (S=100MB) and window (W=20KB)

Results



Legend

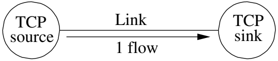
- Mesh: SimGrid results

$$\frac{S}{S/min(B, \frac{W}{2L}) + L}$$

- •: GTNetS results

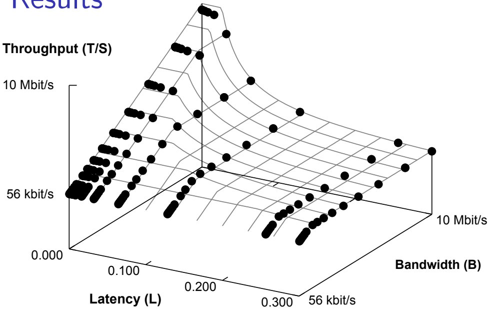
Wanted Feature (1): Flow Control Limitation

Experimental settings



- Flow throughput as function of L and B
- Fixed size (S=100MB) and window (W=20KB)

Results



Legend

- Mesh: SimGrid results

$$\frac{S}{S/min(B, \frac{W}{2L}) + L}$$

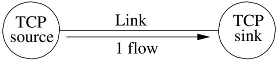
- •: GTNetS results

Conclusion

- SimGrid estimations close to packet-level simulators (when S=100MB)
  - When $B < \frac{W}{2L}$ (B=100KB/s, L=500ms), $|\varepsilon_{max}| \approx \overline{|\varepsilon|} \approx 1\%$
  - When $B > \frac{W}{2L}$ (B=100KB/s, L= 10ms), $|\varepsilon_{max}| \approx \overline{|\varepsilon|} \approx 2\%$

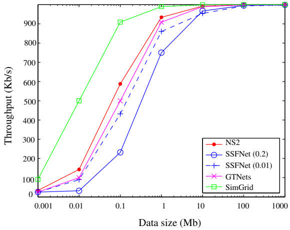
Wanted Feature (2): Slow Start

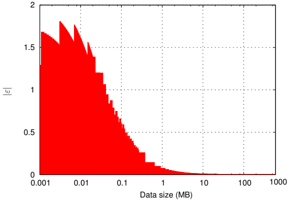
Experimental settings



- Compute achieved bandwidth as function of S
- Fixed L=10ms and B=100MB/s

Evaluation of the SimGrid fluid model

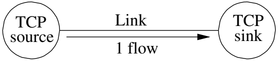




- Packet-level tools don't completely agree
- SSFNet `TCP_FAST_INTERVAL` bad default

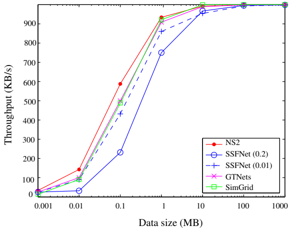
Wanted Feature (2): Slow Start

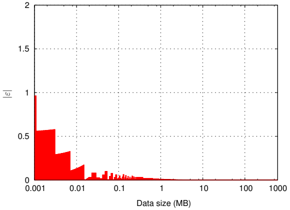
Experimental settings



- Compute achieved bandwidth as function of S
- Fixed L=10ms and B=100MB/s

Evaluation of the SimGrid fluid model





- Packet-level tools don't completely agree
- SSFNet `TCP_FAST_INTERVAL` bad default
- Statistical analysis of GTNetS slow-start
- New SimGrid model (LV08: MaxMin based)
  - Bandwidth decreased (97%)
  - Latency changed to $13.1 \times L$
  - Hence: $Time = \frac{S}{min\left(0.97 \times B, \frac{W}{2L}\right)} + 13.1 \times L$

- This dramatically improve validity range compared to using raw L and B

S	$\overline{\lvert\varepsilon\rvert}$	$\lvert\varepsilon_{max}\rvert$
$S < 100KB$	$\approx 12\%$	$\approx 162\%$
$S > 100KB$	$\approx 1\%$	$\approx 6\%$

Wanted Feature (3): RTT-unfairness

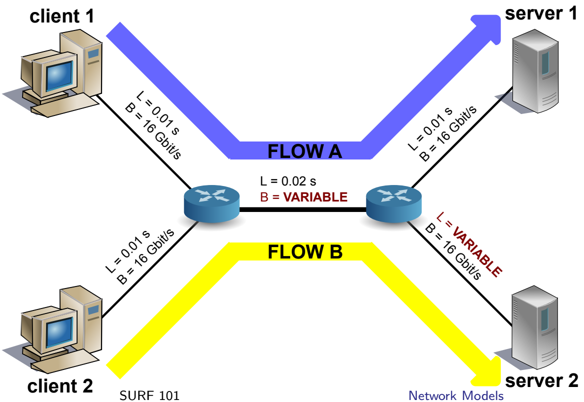
Hypothesis: **Bottleneck links are proportionally shared with respect to flow RTT**

Wanted Feature (3): RTT-unfairness

Hypothesis: **Bottleneck links are proportionally shared with respect to flow RTT**

$$RTT_A.\varrho_A = RTT_B.\varrho_B \text{ where } RTT_i \approx \sum_{\text{flow i uses link j}} (L_j) \text{ (naive model)}$$

- Longer flows (higher latency) will receive slightly less bandwidth

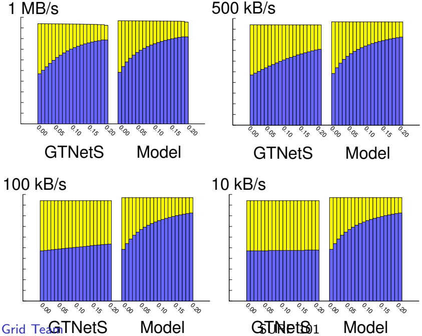


Wanted Feature (3): RTT-unfairness

Hypothesis: **Bottleneck links are proportionally shared with respect to flow RTT**

$$RTT_A.\varrho_A = RTT_B.\varrho_B \text{ where } RTT_i \approx \sum_{\text{flow i uses link j}} (L_j) \text{ (naive model)}$$

- Longer flows (higher latency) will receive slightly less bandwidth

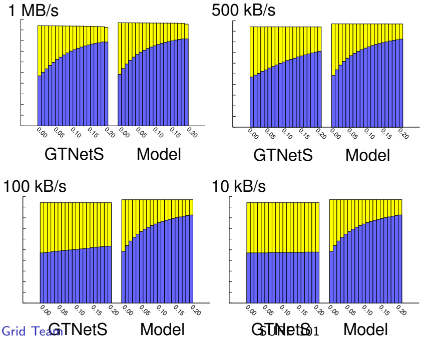


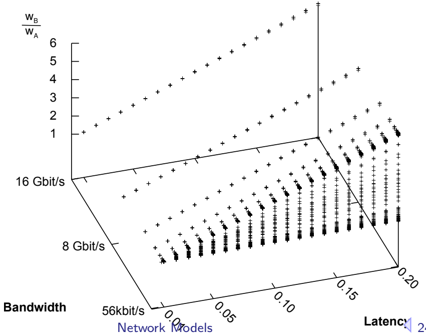
Wanted Feature (3): RTT-unfairness

Hypothesis: Bottleneck links are proportionally shared with respect to flow RTT

$$RTT_A.\varrho_A = RTT_B.\varrho_B \textbf{ where } \cancel{RTT_i \approx \sum_{\text{flow i uses link j}} (L_j)} \text{ (naive model)}$$

- Longer flows (higher latency) will receive slightly less bandwidth
- However, bandwidth also matters



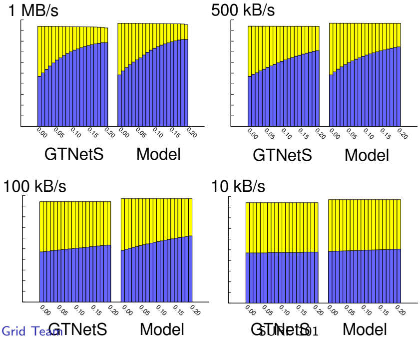


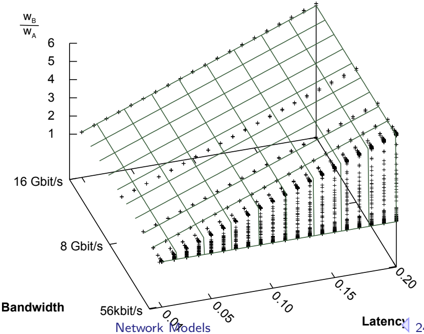
Wanted Feature (3): RTT-unfairness

Hypothesis: **Bottleneck links are proportionally shared with respect to flow RTT**

$$RTT_A.\varrho_A = RTT_B.\varrho_B \textbf{ where } \cancel{RTT_i \approx \sum_{\text{flow i uses link j}} (L_j)} \text{ (naive model)}$$

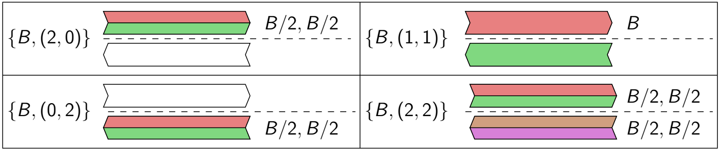
- Longer flows (higher latency) will receive slightly less bandwidth
- However, bandwidth also matters
- Simple fix: $RTT_i \approx \sum_{\text{flow i uses link j}} \left(\frac{M}{B_j} + L_j \right)$





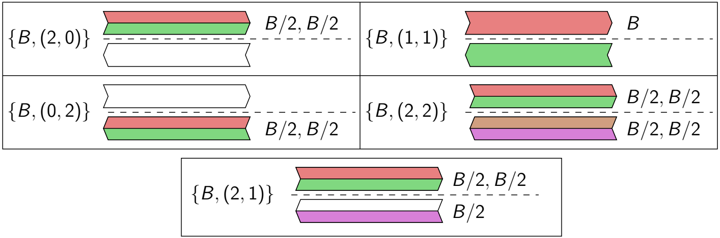
Wanted Feature (4): Cross-Traffic Interference

Take two machines connected by a full-duplex ethernet link.



Wanted Feature (4): Cross-Traffic Interference

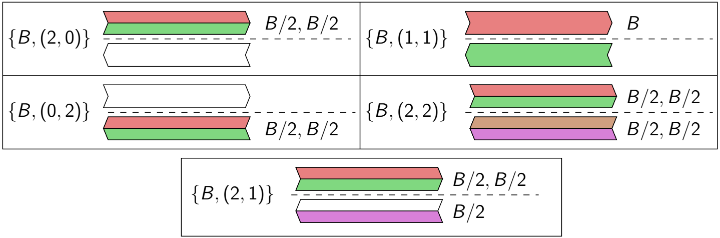
Take two machines connected by a full-duplex ethernet link.



This is a well-known phenomenon when you are using ADSL

Wanted Feature (4): Cross-Traffic Interference

Take two machines connected by a full-duplex ethernet link.



This is a well-known phenomenon when you are using ADSL

Burstiness at micro-scale severely impact macro-scale properties

Modeling such burstiness is ongoing research and resorts to complex differential algebraic equations

Tang et al., *Window Flow Control: Macroscopic Properties from Microscopic Factors*, in INFOCOM 2008

Wanted Features!

Key characteristics of TCP

- Flow-control limitation
- Slow start
- RTT-unfairness
- Cross Traffic Interference

That's messy. Have fluid models a chance ?

- Most previous models (delay, $\sum \log$, $\sum \arctan$, ...) are available in SimGrid
- When well-instantiated, max-min based model can account for all these well-known phenomenon
- The default SimGrid model is LV08: a pragmatic max-min based that is far from perfect but seems reasonnable according to our invalidation studies

Invalidation studies

Wanted Features!

Key characteristics of TCP

- Flow-control limitation
- Slow start
- RTT-unfairness
- Cross Traffic Interference

That's messy. Have fluid models a chance ?

- Most previous models (delay, $\sum$ log, $\sum$ arctan, ...) are available in SimGrid
- When well-instantiated, max-min based model can account for all these well-known phenomenon
- The default SimGrid model is LV08: a pragmatic max-min based that is far from perfect but seems reasonnable according to our invalidation studies

Invalidation studies

Wanted Features!

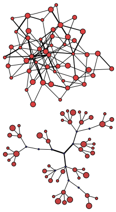
Key characteristics of TCP

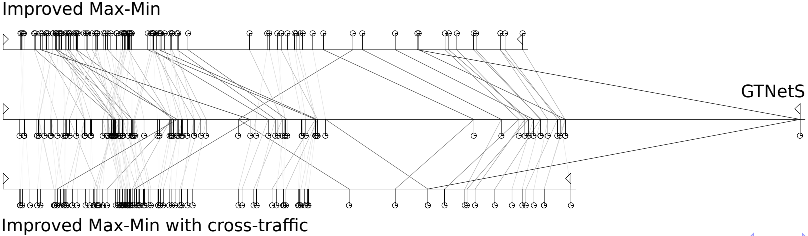
- Flow-control limitation
- Slow start
- RTT-unfairness
- Cross Traffic Interference

That's messy. Have fluid models a chance ?

- Most previous models (delay, $\sum \log$, $\sum \arctan$, ...) are available in SimGrid
- When well-instantiated, max-min based model can account for all these well-known phenomenon
- The default SimGrid model is LV08: a pragmatic max-min based that is far from perfect but seems reasonnable according to our invalidation studies

Invalidation studies





Wanted Features!

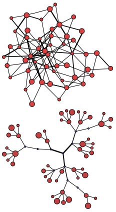
Key characteristics of TCP

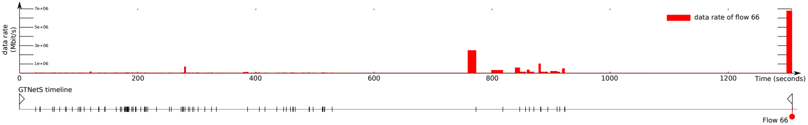
- Flow-control limitation
- Slow start
- RTT-unfairness
- Cross Traffic Interference

That's messy. Have fluid models a chance ?

- Most previous models (delay, $\sum \log$, $\sum \arctan$, ...) are available in SimGrid
- When well-instantiated, max-min based model can account for all these well-known phenomenon
- The default SimGrid model is LV08: a pragmatic max-min based that is far from perfect but seems reasonnable according to our invalidation studies

Invalidation studies





So, what is the model used in SimGrid?

"`network/model:`" command line argument

- `CM02` $\rightsquigarrow$ Old max-min fairness
- `Vegas` / `Reno` $\rightsquigarrow$ Vegas/Reno TCP fairness (Lagrange approach)
- By default since SimGrid v3.3: "smart" max-min LV08
- Example: `./my_simulator --cfg=network_model:Vegas`

  Hint: try the `./my_simulator --cfg=network_model:Vegas` command

So, what is the model used in SimGrid?

"`network/model:`" command line argument

- `CM02` $\leadsto$ Old max-min fairness
- `Vegas` / `Reno` $\leadsto$ Vegas/Reno TCP fairness (Lagrange approach)
- By default since SimGrid v3.3: "smart" max-min LV08
- Example: `./my_simulator --cfg=network_model:Vegas`

  Hint: try the `./my_simulator --cfg=network_model:Vegas` command

CPU sharing policy (`--cfg=network/model:`)

- Default max-min Cas01 is sufficient for most cases
- Different implementations though (Lazy/TI/Full)

So, what is the model used in SimGrid?

"network/model:" command line argument

- CM02 $\rightsquigarrow$ Old max-min fairness
- Vegas / Reno $\rightsquigarrow$ Vegas/Reno TCP fairness (Lagrange approach)
- By default since SimGrid v3.3: "smart" max-min LV08
- Example: `./my_simulator --cfg=network_model:Vegas`

  Hint: try the `./my_simulator --cfg=network_model:Vegas` command

CPU sharing policy (--cfg=network/model:)

- Default max-min Cas01 is sufficient for most cases
- Different implementations though (Lazy/TI/Full)

Want more?

- `network_model:gtnets` or `ns3` $\rightsquigarrow$ use GTNetS or NS3 for network Accuracy of a packet-level network simulator without changing your code (!)
- Compose with `workstation/model:compound` or use `ptask_L07` $\rightsquigarrow$ model specific to parallel tasks

So, what is the model used in SimGrid?

"network/model:" command line argument

- CM02 $\leadsto$ Old max-min fairness
- Vegas / Reno $\leadsto$ Vegas/Reno TCP fairness (Lagrange approach)
- By default since SimGrid v3.3: "smart" max-min LV08
- Example: `./my_simulator --cfg=network_model:Vegas`

  Hint: try the `./my_simulator --cfg=network_model:Vegas` command

CPU sharing policy (--cfg=network/model:)

- Default max-min Cas01 is sufficient for most cases
- Different implementations though (Lazy/TI/Full)

Want more?

- `network_model:gtnets` or `ns3` $\leadsto$ use GTNetS or NS3 for network Accuracy of a packet-level network simulator without changing your code (!)
- Compose with `workstation/model:compound` or use `ptask_L07` $\leadsto$ model specific to parallel tasks
- Plug your own model in SimGrid!!

Conclusion and Future Work

Conclusion

- SimGrid is among the few projects that made (in)validation studies
- One can choose and tweak existing models but you should stick with the default one...
- ...unless it does not fulfill your needs and you know of a less generic but more accurate one!

Future Work

- Network models for HPC networks
- Various storage models (in collaboration with the CERN)
- Improve the cpu model, add a notion of GPU, ...
- Use stochastic models with a careful management of randomness