Flow Networking in Grid Simulations

James Broberg and Rajkumar Buyya

Grid Computing and Distributed Systems (GRIDS) Lab
Dept. of Computer Science and Software Engineering
The University of Melbourne, Australia
Emial: {brobergj, raj}@csse.unimelb.edu.au

1. Introduction

Simulation tools play an essential role in the evaluation of emerging peer-to-peer, computing, service and content delivery networks. Given the scale, complexity and operational costs of such networks, it is often impossible to analyse the low-level performance, or the effect of new scheduling, replication and organisational algorithms on actual test-beds. As such, practitioners turn to simulation tools to allow them to rapidly evaluate the efficiency, performance and reliability of new algorithms on large topologies before considering their implementation on test-beds and production systems.

In particular, the study of Grids is significantly aided by robust and rapid prototyping via simulation, due to the sheer scale and complexities that arise when operating over many administrative domains, which precludes easy prototyping on real test-beds. Grid computing [1] has been integral in enabling knowledge breakthroughs in fields as diverse as climate modelling, drug design and protein analysis, through the harnessing of computing, network, sensor and storage resources owned and administered by many different organisations. These fields (and other so-called Grand Challenges) have benefited from the economies of scale brought about by Grid computing, tackling difficult problems that would be impossible to feasibly solve using the computing resources of a single organisation. However, when prototyping such applications and services that harness the power of the Grid, it is beneficial to test their operation via simulation in order to optimise their behaviour, and avoid placing strain on Grid resources during the development phase.

Despite the obvious advantages of simulation when prototyping applications and services that run on Grids, realistically simulating large topologies and complicated scenarios can take significant amounts of memory and computational power. For statistically significance, large numbers of simulation runs are needed to increase our confidence in the results we obtain from simulation platforms. This is particularly the case when studying applications and services that store and move significant volumes of data over the grid, such as data-grids or content and service delivery networks. Simulators that attempt to model the full complexity of TCP/IP networking in such environments scale poorly and often run significantly slower than real-time, practically defeating the purpose of simulating such environments in the first place.

In this chapter we look at incorporating flow-level (or ‘fluid’) networking models into Grid simulators, in order to improve the scalability and speed of Grid simulations by reducing the overhead of data and network intensive experiments, and improving their accuracy. Network flow models are used that closely approximate actual steady-state

TCP/IP networking. We utilise the GridSim toolkit as a candidate implementation, and fully replace the existing packet-level networking model in GridSim with a flow-level networking stack. However, the principles outlined in this chapter could be applied to other simulation platforms.

The remainder chapter is organised as follows; Section 2 describes the GridSim Toolkit, and gives a brief overview of its feature set. In Section 3, the existing packet-level networking implementation for the GridSim toolkit is described, and some inefficiencies are identified that arise when doing large scale network and data centric simulations. In Section 4 we outline the basic principles behind modelling network traffic and transfers as flows or 'fluid', rather than discrete packets. The bandwidth-sharing model utilised in our flow-level networking model is described in Section 5. The new flow-level networking implementation for the GridSim toolkit is introduced in Section 6, highlighting the additions made to GridSim in order to support the flow-based networking paradigm. Section 7 describes the flow tracking and management algorithms required to compute the durations of network flows, and to update them when conditions change during a simulation run. The performance improvements gained from the flow-networking model over existing packet-based implementation are highlighted in Section 8. Finally, we conclude this chapter in Section 9, taking a macroscopic view of the potential applications of flow-level networking in large scale grid simulations.

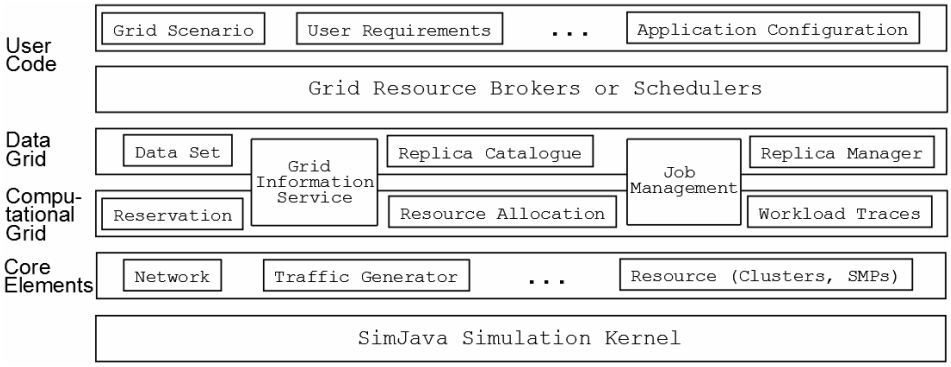

Figure 1: The GridSim Architecture

2. The GridSim Toolkit

GridSim is a grid simulation toolkit for resource modelling and application scheduling for parallel and distributed computing [2]. The GridSim toolkit has been used extensively by researchers across the globe [3] to model and simulate data grids [4], failure detection [5], differentiated service [6], auction protocols [7], advanced reservation of resources [8] and computational economies in grid marketplaces.

GridSim has been designed as an extensible framework by following a multi-layer architecture as shown in Figure 1. This allows new components or layers to be added and integrated into GridSim easily. GridSim implementations use SimJava [9], a general purpose discrete-event simulation package for handling the interaction or events among GridSim components.

At basic level, all components in GridSim communicate with each other through message passing operations defined by SimJava. The second layer models the core elements of the distributed infrastructure, namely Grid resources such as clusters, storage repositories and network links. These core components are absolutely essential to create simulations in GridSim. The third and fourth layers are concerned with modelling and simulation of services specific to Computational and Data Grids [4] respectively. Some of the services provide functions common to both types of Grids such as information about available resources and managing job submission.

From networking perspective, the current version supports packet-based routing including background network traffic modelling based on a probabilistic distribution [6]. This is useful for simulating data-intensive jobs over a public network where the network is congested. The limitations of this network model are highlighted in the next section.

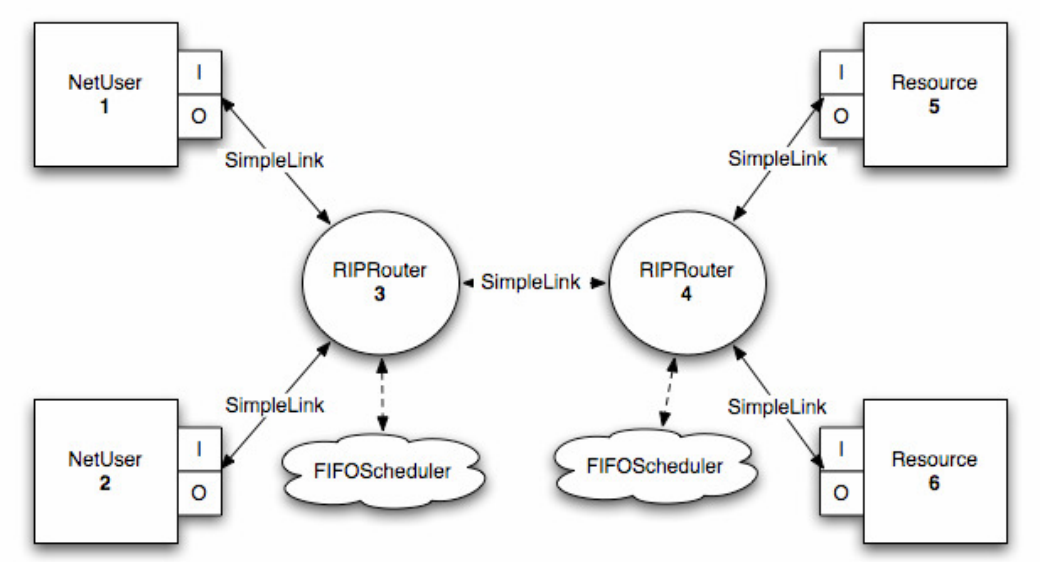

Figure 2: GridSim Packet Networking Architecture

3. The GridSim Packet Networking Architecture

A typical dog-bone topology is shown in Figure 2, for a GridSim experiment using the existing packet-level network framework. Consider a user at user node 1 that wishes to send a 10Mb file to resource node 6. In the current GridSim network model [6] the file would be packetised into MTU-sized packets by the *Output* class of the *NetUser* GridSim entity and sent over the links. Every packet but the last is an empty packet (*GridSimTags.EMPTY_PKT*), with the last packet containing the actual data (*IO_data*). If the Maximum Transmission Unit (MTU) was 1500 on all elements between the source and destination, sending a 10MB file would result in approximately 34,952 packets being generated. In GridSim, each packet is represented by a *NetPacket* Java object, thus creating a considerable amount of overhead for large data transfers. This can lead to lengthy simulation execution times for data or network dependent simulations. The magnitude of this overhead will be quantified later in this chapter. In the next section we describe the new flow

networking implementation that seeks to minimise the overhead of network dependant simulations.

4. Flow Networking Concepts

Rather than modelling each network transfer using packets, we wish to consider a network flow model that captures the steady-state behaviour of network transfers. For convenience we will denote our Grid topology (such as that depicted in Figure 2) as a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, consisting of 2-element subsets of V. For instance, if vertices x and y are connected, then $\{x, y\} \in E$. In the system there exist flows $f = 1,2,\ldots,F$, with each flow f having a source and destination. Each flow f describes a simple path of length k represented by a set of edges $\{(v_1, v_2),\ldots,(v_k, v_{k+1})\}$. The number of bytes in each flow f is denoted as $SIZE^f$.

Let us consider a simple topology where the two entities, node u and node v are *directly* connected by an edge (u,v), with available bandwidth $BW_{u,v}$ (in bytes per second) and latency $BW_{u,v}$ (in seconds). Calculating the duration of a single network flow f with size $SIZE^f$ from u to v can be trivially computed as follows:

$$T_f = LAT_{u,v} + \frac{SIZE^f}{BW_{u,v}}$$

Equation 1

As an interesting aside, the above equation can be tested in a rudimentary fashion by utilising the first networking example in the GridSim distribution (NetEx01)[1]. An extremely coarse approximation of basic flow networking can be achieved with the current packet-level network framework in GridSim by setting the MTU to equal the size of the network flow to be transferred, causing only a single *NetPacket* to be generated, which is held at the *Output* of the *NetUser* GridSim entity for the appropriate duration. However, this does not model bandwidth sharing on the links in any way.

More generally, a flow f with a source u and destination v that is not directly connected has an expected duration of:

$$T_f = \sum_{(u',v') \in f} LAT_{u,v} + \frac{SIZE^f}{\min BW^f}$$

Equation 2

where $\min BW^f$ is the smallest bandwidth available on any edge on the path f between u and v (i.e. the bottleneck link), and latency $LAT^f = \sum_{(u',v') \in f} LAT_{u,v}$ is the sum of the latency of all edges (u',v') that connect the source u to the destination v.

[1] Available at http://www.gridbus.org/gridsim/example/net_index.html

We note that the above equations and discussions are only valid for a single active flow at a time, as it does not account for any bandwidth sharing between multiple flows on common (overlapping) links. Where multiple flows are active over links, then $\min BW^f$ is the smallest bandwidth allocated by edge (based on some bandwidth sharing model) on the path f between u and v. The implications of this will be discussed in the following section.

5. Bandwidth Sharing Models

In Section 4 we examined a simple theoretical model to compute the duration of each flow in a system based on the bottleneck bandwidth. This approach significantly improves the speed of Grid simulations by avoiding the need to packetise large network transfers, instead taking a macro or fluid view of network traffic in a given topology.

In order for this approach to be effective we need to calculate the appropriate bandwidth given to flows on each segment of their respective route. More importantly, we must model how the bandwidth is shared when many flows are active over one or many links. As a proof of concept for the GridSim flow networking implementation, we have implemented simple MIN-MAX bandwidth fair sharing, where each flow that shares a link is allocated an equal portion of the bandwidth. That is, an edge (u,v), with available bandwidth $BW_{u,v}$ that has n active flows will allocate each flow $\frac{BW_{u,v}}{n}$ bandwidth. Whilst it has been found that other bandwidth sharing models are closer to actual TCP/IP behaviour [10], MIN-MAX bandwidth sharing is a useful candidate model with minimal state to track in the implementation.

We intend to include other bandwidth sharing models that more closely approximate TCP/IP in the near future, such as proportional bandwidth sharing that considers latency, round-trip times and class-based priorities [11, 12].

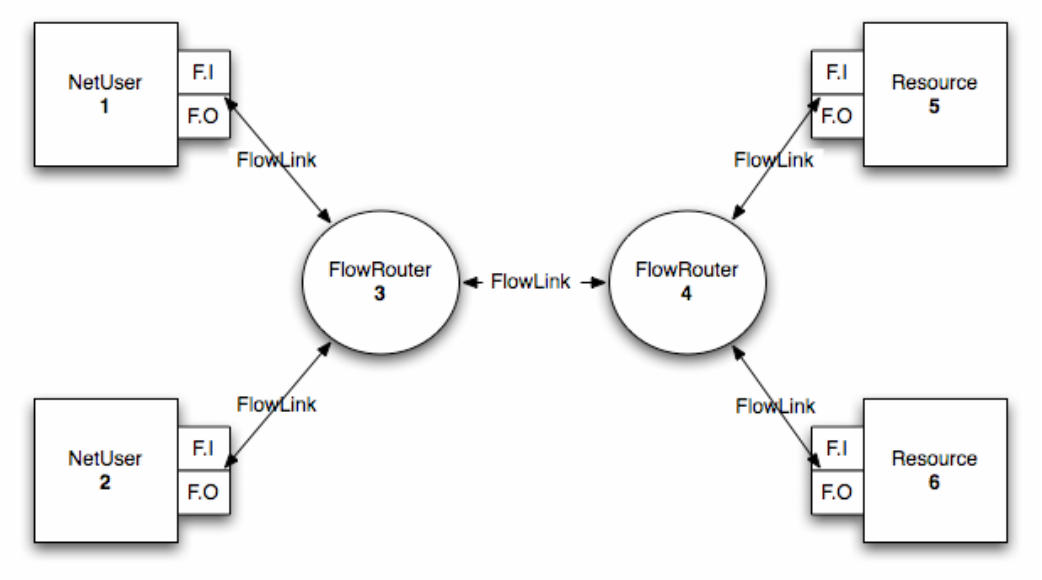

Figure 3: GridSim Flow Networking Architecture

6. The New GridSim Flow Networking Architecture

In order to implement the flow-level networking model described in Section 4, we need to make some fundamental changes to the existing packet-level network implementation in GridSim. More specifically, we need to replace the entire networking stack with flow-aware components due to the significant differences between the two approaches.

Figure 4 depicts a high-level class diagram showing the flow aware networking stack that is to be added to GridSim to enable flow-level network functionality. The new support components are shown as dotted boxes to differentiate them from the existing packet networking stack. A summary of these additions (and changes) is listed in Table 1. Figure 3 shows an example GridSim topology that utilises the new flow model

To keep the flow-level network functionality logically separated, a new package was added, namely `gridsim.net.flow`. This will encapsulate all of the flow-level networking functionality to be added. A new interface, `NetIO`, was created to provide a common set of functions for the existing `Input` and `Output` classes, as well as the new flow-aware `FlowInput` and `FlowOutput` classes. These flow-aware input and output classes are automatically generated for GridSim entities by calling `GridSim.initNetworkType(GridSimTags.NET_FLOW_LEVEL)`, before initialising a GridSim simulation.

The `FlowOutput` class performs a similar function to the existing `Output` class, but instead of packetising data than is sent by GridSim entities into MTU sized chunks (as described in Section 3), it creates a single `FlowPacket` that will represent an active flow for its lifetime. The `FlowOutput` class also supports background traffic, creating junk flows to simulate load on links, and the Grid Information Service (GIS), which is an entity that provides grid resource registration, indexing and discovery services. These two features were available in the `NetPacket` implementation and are depended on by GridSim users worldwide for

simulating complex scenarios and topologies, and thus were supported in the flow implementation.

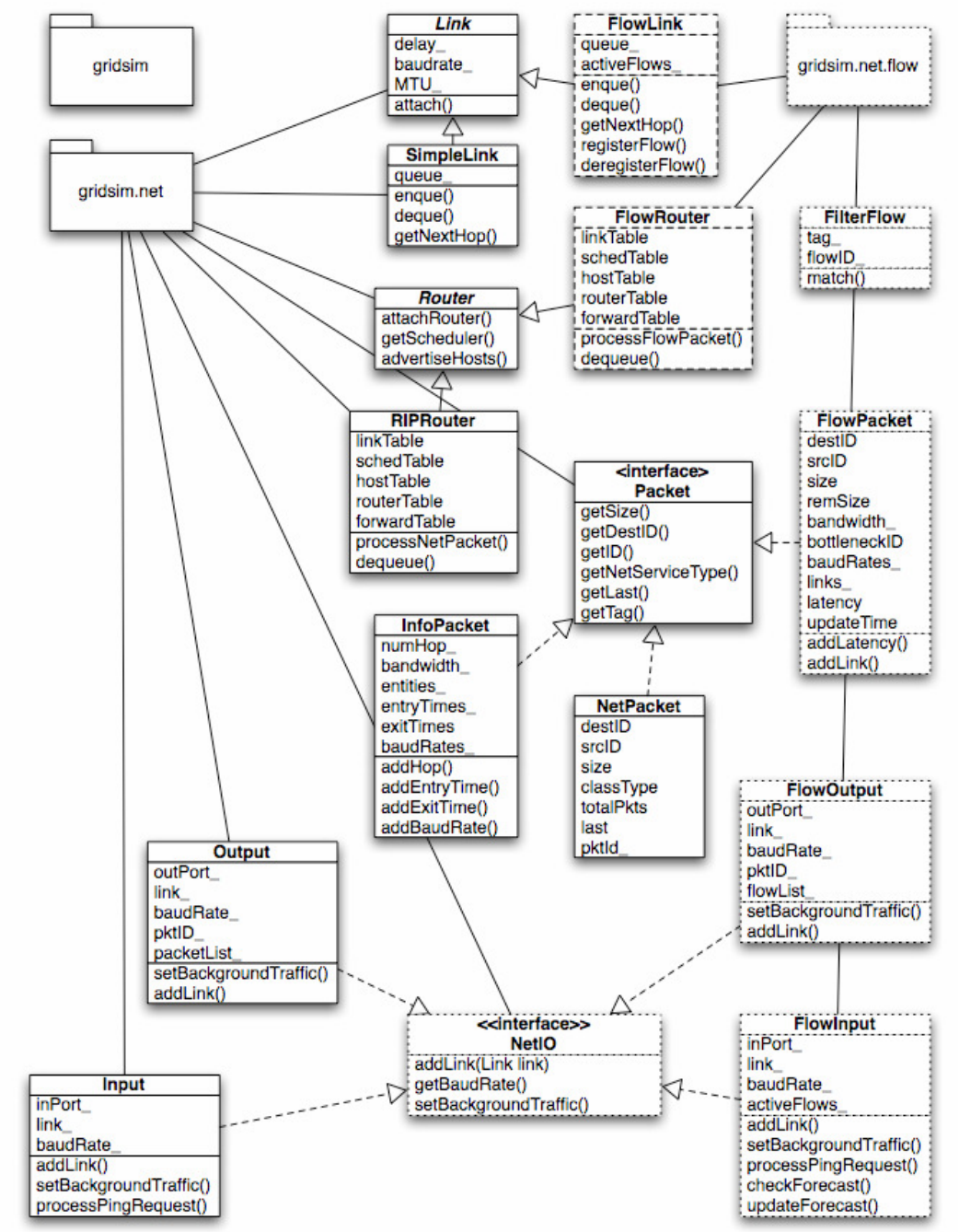

Figure 4: GridSim Flow Networking class diagram

We still require an entity to represent the network flow. As such, for convenience we will leverage a subset of the existing `Packet` implementation, extending it to create a `FlowPacket` class. This allows us to utilise the existing features of the `Packet` class, whilst adding logic that will support an accurate flow-networking model for GridSim. A flow is then simply represented by a single `FlowPacket`, which exists

as long as the flow is active. As it traverses along a GridSim topology from its source to destination, it maintains a list of the `FlowLink` entities it passes over, and more specifically the latency and bandwidth available on each of these links.

Table 1: Summary of changes between GridSim packet and flow implementations

Component	Packet Model	Flow Model
GridSim Network type	`GridSimTags.NET_PACKET_LEVEL`	`GridSimTags.NET_FLOW_LEVEL`
Input / Output	`Input / Output` *`extends`* `Sim_entity`	`FlowInput / FlowOutput` *`extends`* `Sim_entity` *`implements`* `NetIO`
Packet	`NetPacket` *`extends`* `Packet`	`FlowPacket` *`extends`* `Packet`
Link	`SimpleLink` *`extends`* `Link`	`FlowLink` *`extends`* `Link`
Router	`RIPRouter / FloodingRouter / RateControlledRouter` *`extends`* `Router`	`FlowRouter` *`extends`* `Router`
Scheduler	`SCFQScheduler / FIFOScheduler / RateControlledScheduler` *`implements`* `PacketScheduler`	`N / A`
Event filter	`N /A`	`FilterFlow`
Package	`gridsim.net`	`gridsim.net.flow`

As a `FlowPacket` traverses a `FlowLink`, it is registered as an active flow on that link for the purpose of computing the bandwidth a `FlowLink` allocates a given `FlowPacket`, when multiple flows are active on a given link. If the bottleneck bandwidth of an existing flow is affected by a new flow becoming active or an existing flow becoming de-active), then `FlowLink` notifies the remaining active flows (which are held at the `FlowInput` of their destination) of their new bottleneck bandwidth.

Routers for flow-level networking are significantly less complicated than those supporting the packet networking model, as they have minimal responsibility in the flow model. A new class, `FlowRouter`, has been added, which enables many-to-many (*m:m*) connections from GridSim entities but performs no actual scheduling itself. As such, there is no equivalent to `PacketScheduler` needed for the flow model.

Finally, the `FlowInput` holds the `FlowPacket` for the appropriate duration, based on Equation 2 and an appropriate bandwidth sharing model that determines the bandwidth assigned to each flow (as described in Section 5). As stated previously, the bottleneck bandwidth of a flow can change during its lifetime due to the arrival of a new overlapping flow or the termination of an existing flow. When this occurs, affected flows are notified and the duration is updated, potentially being brought forward as available bandwidth increases or pushed back as available bandwidth decreases. This process is explained in more detail in the next section.

7. High Level Flow Management Algorithms

When running any non-trivial GridSim scenario, it is obvious that more than one flow will be active at a given time, and that flows will overlap, begin and end at different times. As such, the bandwidths assigned to each flow can change frequently. Therefore, flow management algorithms are employed to make an initial forecast based on the bottleneck bandwidth when the flow begins, and to update the forecast when the bottleneck bandwidth increases or decreases, decreasing or increasing the expected duration accordingly.

Let us consider a GridSim system connected in a dog-bone topology like that depicted in Figure 2. We wish to compute the end time T_{end}^{f} of a new flow f that arrives into a system, created at time T_{NOW}:

Algorithm 1: Initial flow forecast

$BW_{\min}^{f} \leftarrow \infty$

$T_{start}^{f} \leftarrow T_{NOW}$

$T_{end}^{f} \leftarrow \infty$

$REM\_SIZE^{f} \leftarrow SIZE^{f}$

for each $BW_{u,v} \in BW^{f}$

do {**if** $BW_{u,v} < BW_{\min}^{f}$ **then** $BW_{\min}^{f} = BW_{u,v}$}

$$T_{dur}^{f} = LAT^{f} + \frac{REM\_SIZE^{f}}{BW_{\min}^{f}}$$

$T_{end}^{f} \leftarrow T_{start}^{f} + T_{dur}^{f}$

If a the bottleneck link of an active flow f changes (i.e. it becomes larger or smaller) at time T_{NOW}, then the expected duration of that flow must be updated:

Algorithm 2: Update flow forecast

$BW_{old\,\min}^{f} \leftarrow BW_{\min}^{f}$

$BW_{\min}^{f} \leftarrow \infty$

$T_{elap}^{f} \leftarrow T_{NOW} - T_{start}^{f}$

$T_{start}^{f} \leftarrow T_{NOW}$

$T_{end}^{f} \leftarrow \infty$

$REM\_SIZE_{old}^{f} \leftarrow REM\_SIZE^{f}$

$$REM\_SIZE^{f} = REM\_SIZE_{old}^{f} - \left(T_{elap}^{f} - BW_{old\,\min}^{f}\right)$$

for each $BW_{u,v} \in BW^{f}$

do {**if** $BW_{u,v} < BW_{\min}^{f}$ **then** $BW_{\min}^{f} = BW_{u,v}$}

$$T_{dur}^{f} = LAT^{f} + \frac{REM\_SIZE^{f}}{BW_{\min}^{f}}$$

$T_{end}^{f} \leftarrow T_{start}^{f} + T_{dur}^{f}$

8. Performance comparison

In this section we quantify the performance improvements gained from modelling networking traffic as *flows* instead of packets. Using our candidate implementation of a flow model for the GridSim toolkit (described in Section 6), we perform some numerical comparisons of specific scenarios. In each scenario we compare the existing NetPacket implementation and the new flow-level FlowPacket implementation described in this chapter. All tests are run using on a Macbook with a 2GHZ Intel Core 2 Duo and 2GB of ram. Each data point represents the average of 30 runs.

The first scenario is a classic dogbone topology like that depicted in Figure 2 and Figure 3. `NetUser1/FlowUser1` and `NetUser2/FlowUser2` each send 3 identically sized files to `Resource6` and `Resource5` respectively. The size of the files is varied from 0.5MB to 500MB. The links between the Users and the first router are rated at 10MB/sec. The link between the two routers has a capacity of 1.5MB/sec, and is clearly the bottleneck link. The link between the second router and the resources is 10MB/sec. The files transfers are initiated in 10-second intervals. The latencies from the users to the router, between the routers, and from the router to the resources are 45, 25 and 30 milliseconds respectively.

In Figure 5 we see a comparison of the CPU time taken for the simulation to execute when utilising either the NetPacket or FlowPacket networking stack. The size of the files sent by each user is varied from 0.5MB to 500MB. We can clearly see that as the size of the files increase, the overhead of the NetPacket implementation is demonstrated as the CPU time explodes. The plot is presented on a log-scale on the y-axis to highlight the huge difference in simulation running time. The FlowPacket implementation is totally impervious to the size of the files being transferred, as it has no effect whatsoever on the amount of state it maintains. When transmitting several 500MB files, the FlowPacket implementation takes 0.432 seconds to execute, while the NetPacket implementation takes a staggering 6699 seconds, or approximately 111 minutes.

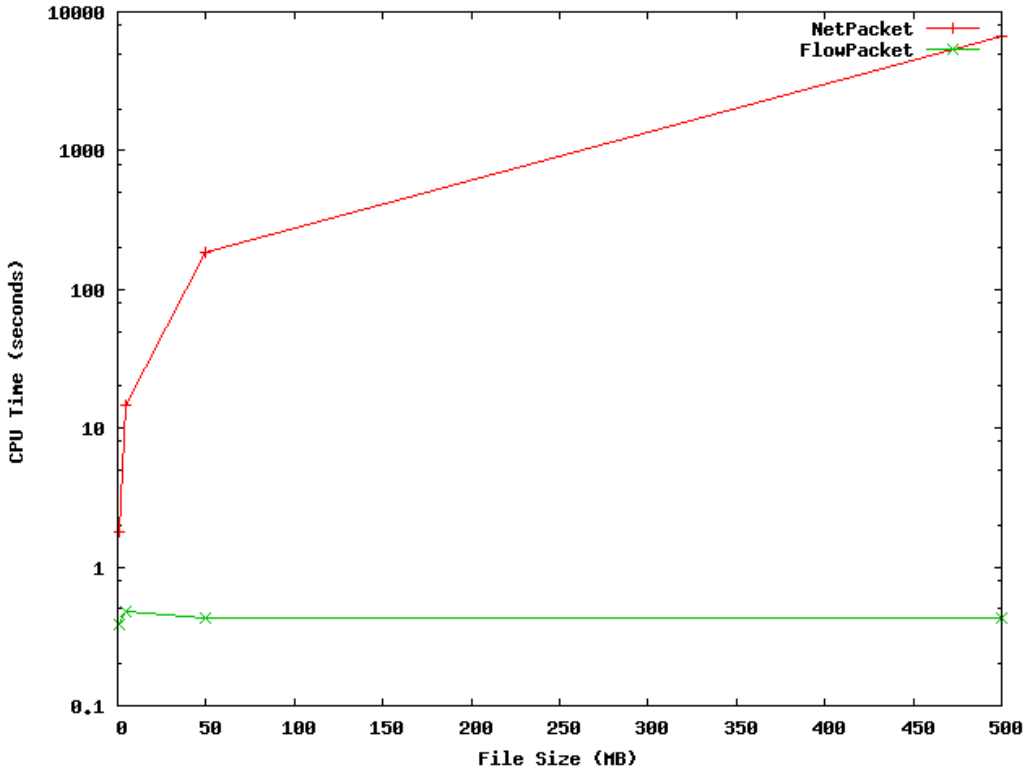

Figure 5: Comparison of CPU time needed for "dogbone" simulation run

We can see a linear relationship between the amount of memory consumed and the size of the files being transmitted by the GridSim simulator when using the NetPacket implementation in Figure 6. When utilising the FlowPacket implementation, the size of the files being transferred has no effect on the peak memory consumption, as it stores the same amount of state regardless of whether it is sending a 0.5MB file or a 500MB file.

In the second scenario we examine a similar dogbone topology where users submit Gridlets for processing, instead of sending files. A Gridlet is a construct that contains all the information related to a grid job and its execution management details such as job length expressed in MI (Millions Instruction), and the size of input and output files. Individual users can model their application by creating Gridlets for processing on Grid resources. These basic parameters are utilised to determine the execution time, the time required to transport input and output files between users and remote resources, and returning the processed Gridlets back to the originating user along with the results. We have two users submitting Gridlets with 5000 byte input file, an service requirement of 5000MI and return a 5000 byte output file to 10 available resources. The number of Gridlets sent by each user is varied from 5 to 40.

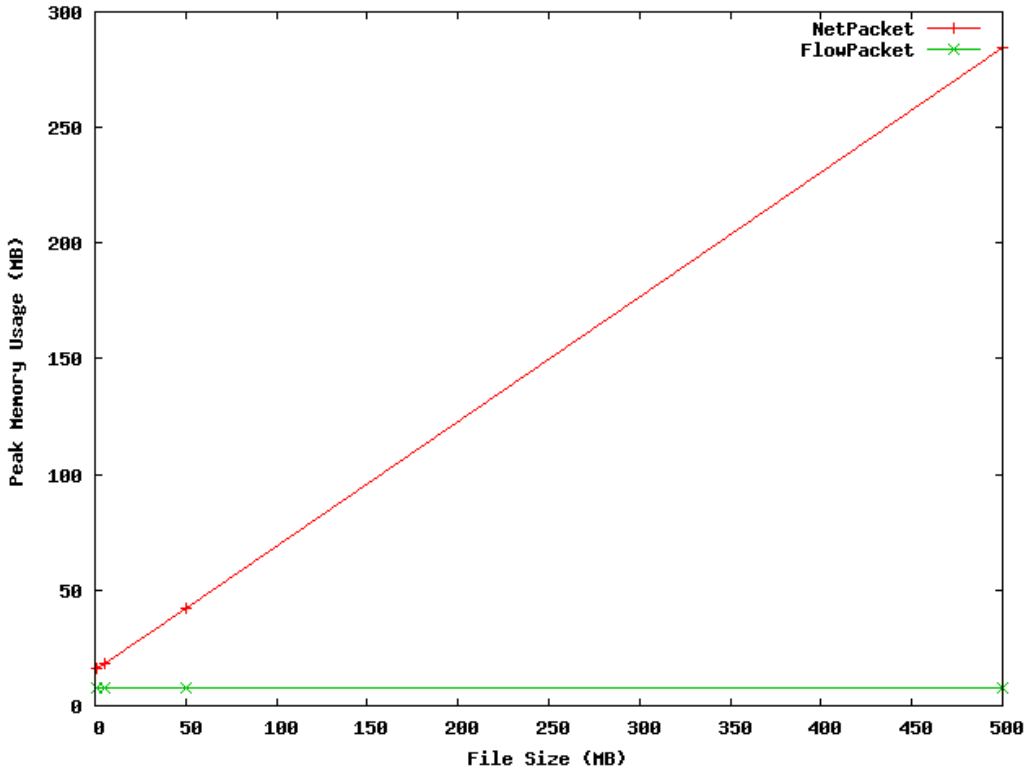

Figure 6: Comparison of peak memory usage for "dogbone" simulation run

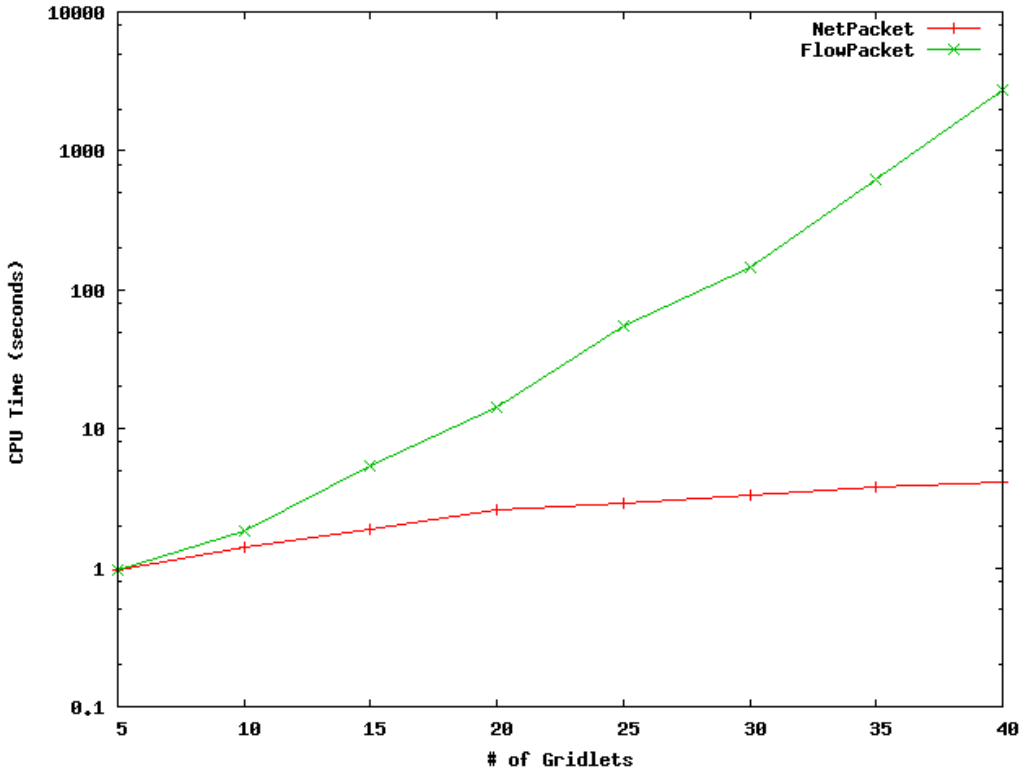

Figure 7: Comparison of CPU time needed for Gridlet simulation run

From Figure 7 can see as the number of Gridlets being sent by each user increases, the CPU time taken to execute the simulation increases exponentially (highlighted by the near-linear line on the plot, where the y axis is log-scale). On the other hand, the FlowPacket implementation only sees a nominal increase in simulation time as the number of Gridlets sent by each user increases.

In Figure 8 we see an examination of the peak memory usage of the NetPacket and FlowPacket implementations for GridSim, running the Gridlet scenario described previously. As the number of Gridlets sent by each user increases, we see a significant increase in memory utilisation by the NetPacket implementation after an initial flat response. Conversely, the FlowPacket sees only negligible increases in memory utilisation as the number of Gridlets increases.

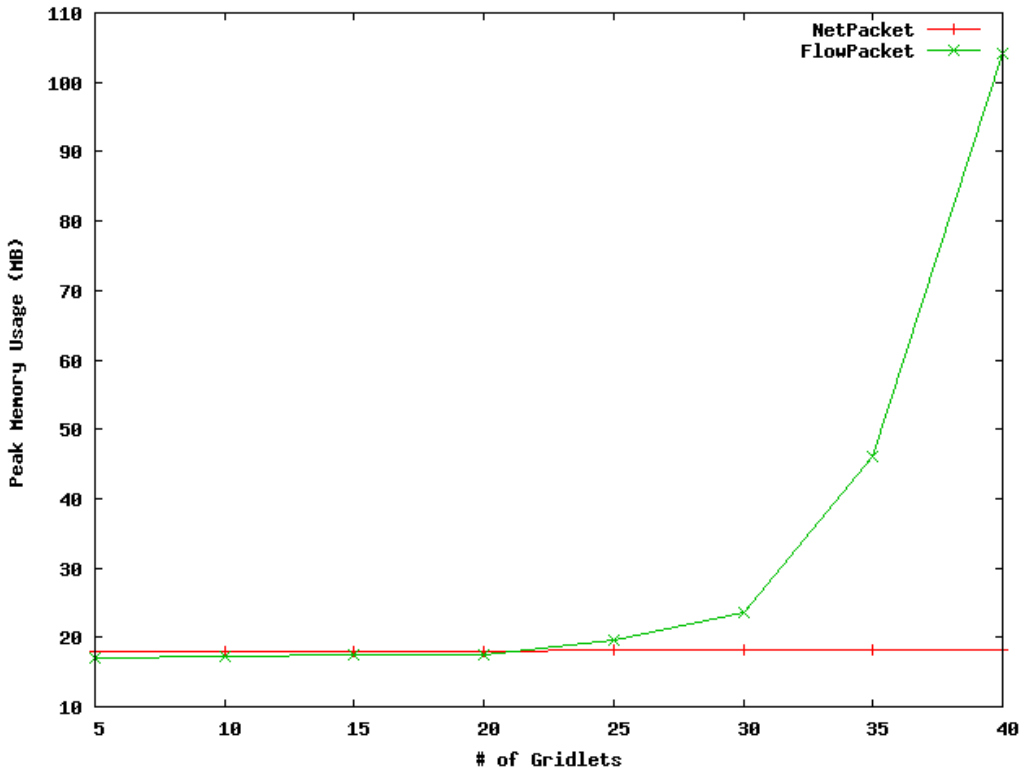

Figure 8: Comparison of peak memory usage for Gridlet simulation run

9. Conclusion

We have explored the improvements in accuracy and reduction in complexity that that be achieved by utilising flow-based networking in grid simulation, instead of packet based networking, when attempting to model real TCP/IP networks. From the results obtained it is clear that significant improvements, in the order of many magnitudes, can be made in terms of speed and scalability when executing complex Grid Simulations involving large data transfers and large numbers of Grid job submissions. It is clear that practitioners will not be limited in the size or complexity of the scenarios they wish to model, allowing them to simulate complex computational Grid topologies, content and service delivery networks, and data Grids, to name a few examples. This will hopefully lead to greater advances in solving the so-called Grand

Challenges in areas such as climate modelling, drug design and protein analysis, by allowing practitioners to prototype their solutions trivially and rapidly before committing time and resources in building complex software and network systems that operate on global Grids.

10. Acknowledgements

This work was supported in part by the Department of Industry, Innovation, Science, and Research (DIISR) and the Australian Research Council (ARC) through their respective International Science Linkages (ISL) and Discovery Projects. We thank Anthony Sulistio for adding support classes to GridSim to enable this work, and to Srikumar Venugopal for valuable discussions regarding the implementation.

11. Bibliography

1. I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*. 1998: Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.
2. R. Buyya and M. Murshed, *GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing.* Concurrency and Computation: Practice and Experience, 2002. **14**(13-15): p. 1175-1220.
3. R. Buyya and A. Sulistio. *Service and Utility Oriented Distributed Computing Systems: Challenges and Opportunities for Modeling and Simulation Communities.* in *41st Annual Simulation Symposium*. 2008. Ottawa, Canada.: IEEE CS Press.
4. A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, *A toolkit for modelling and simulating Data Grids: An extension to GridSim*, in *Concurrency and Computation: Practice and Experience*. 2007, Wiley Press, New York, USA.
5. A. Caminero, A. Sulistio, B. Caminero, C. Carrion, and R. Buyya, *Extending GridSim with an architecture for failure detection*, in *International Conference on Parallel and Distributed Systems*. 2007.
6. A. Sulistio, G. Poduval, R. Buyya, and C.K. Tham, *On incorporating differentiated levels of network service into GridSim.* Future Generation Computer Systems, 2007. **23**(4): p. 606-615.
7. M. Dias De Assuncao and R. Buyya, *An Evaluation of Communication Demand of Auction Protocols in Grid Environments*, in *GECON 2006: Proceedings of the 3rd International Workshop on Grid Economics & Business*. 2006, World Scientific Press: Singapore.
8. A. Sulistio and R. Buyya. *A Grid Simulation Infrastructure Supporting Advance Reservation.* in *Parallel and Distributed Computing and Systems*. 2004. Cambridge, USA: MIT.
9. F. Howell and R. McNab, *SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling*, in *Proceedings of the First International Conference on Web-based Modelling and Simulation*. 1998: San Diego, CA.
10. D.M. Chiu, *Some Observations on Fairness of Bandwidth Sharing*, in *Fifth IEEE Symposium on Computers and Communications (ISCC 2000)*. 2000, IEEE Computer Society: Los Alamitos, CA, USA. p. 125.

11. S. Floyd and K. Fall, *Promoting the use of end-to-end congestion control in the Internet.* IEEE/ACM Transactions on Networking (TON), 1999. **7**(4): p. 458-472.
12. M. Matthew, S. Jeffrey, and M. Jamshid, *The macroscopic behavior of the TCP congestion avoidance algorithm.* SIGCOMM Computer Communication Review, 1997. **27**(3): p. 67-82.