

Simulating Computer Networks using Clusters of PCs

David Abramson †
Kieran Power †

Rok Sosic ‡

†School of Computer Science and Software Engineering
Monash University
Clayton, Vic 3168.
davida@csse.monash.edu.au

‡Griffith University
Nathan, Qld 4111.

Abstract

Simulation has become the dominant technique for evaluating the performance of many products before real hardware is produced. In computer networks, this is particularly true, and can avoid expensive hardware and software development. Using simulation it is possible to explore different protocols and design options, under different loads, long before the design is frozen. However, such simulations can be time consuming, especially if a large number of scenarios are to be considered. In this paper, we show how it is possible to perform distributed execution of a network simulation written for the OpNet package. This is performed using a distributed computing tool called Cluster. Very high efficiency has been experienced on a variety of workstation clusters. A case study is presented in which a real network protocol is evaluated using OpNet and Cluster.

Keywords: Computer Network Simulation, Parallel Discrete Event Simulation, Clusters.

1 Introduction

Today's network products are becoming increasingly complex. The wide variety of networking protocols and transmissions methods, means that a single product may have to support many different protocols concurrently. Importantly, as new protocols are developed for different transmission mediums, it is important to simulate the behavior of these algorithms before they are implemented in real products. Simulation has emerged in this area as an extremely important technique for both verifying the correctness of a protocol, but also for studying how it behaves under different operational parameters, such as network delays and user demand.

Simulation can be expensive. In order to model a given real world scenario accurately, the simulation must be performed with sufficient detail to capture real hardware behaviour and user demand. This is usually achieved by a discrete event simulation at a low level, and it may take a

long time to gain a realistic summary across different scenarios, particularly if monte-carlo methods are employed to study stochastic behaviour.

There have been many attempts to accelerate discrete event simulation by parallel processing. Because the basic process is sequential, there has only been limited success in accelerating the execution of a single scenario. However, when multiple independent scenarios are simulated, it is possible to execute them concurrently – this is regarded as an “embarrassingly parallel” application. In spite of this, there has not been wide acceptance of parallel and distributed platforms for simulation systems. This is most likely because the cost of adding and maintaining the parallel code in the already large simulation system is too high for the expected returns.

In this paper we describe a way of achieving the benefits of parallel computing for evaluating independent scenarios, without any modification to the underlying simulation engine. We make use of the Cluster [12] package, which can execute a program multiple times on many platforms, changing parameter values between executions. Because Cluster is generic, it can actually be applied to a wide range of applications, thus reducing the cost of parallel computing infrastructure for any one applications.

The paper contains a case study in which a new network protocol is considered, and the results of both the simulations, and the parallel computation efficiency, are presented.

2 Parametric Modelling on Clusters of PCs using Cluster

2.1 Affordable Supercomputing

Traditionally, supercomputers have been built with high performance processors, which use techniques to execute as many computations concurrently as possible. These machines have employed a number of arithmetic units, and sufficient hardware to allow them to be fed with data and instructions at their peak rate. Supercomputers built as parallel processors have also contained special shared

memory or message passing hardware as an integral part of the machine design – this special hardware, in combination with the high speed processors has meant that they have been expensive. However, the rapid change in the price performance characteristics of PCs and high speed networking hardware has made it possible to assemble a super-computing platform from large clusters of PCs. A number of projects around the world have been investigating this approach [3] [8][11], in which PCs are simply racked together and a high speed switch, such as a Myrinet [23][14] is provided for inter-processor communication. In this paper we explore one particular application of machine clusters, namely “parametric modelling”.

2.2 Parametric Modelling on PC Clusters

Parametric modelling involves running the same application many times, varying one or more parameters each time the code is executed. Because this type of application is highly parallel, it is possible to run more than one copy of the code at the same time using a machine cluster. However, managing an experiment of many thousands of runs can be time consuming and error prone. Figure 1 shows a sample machine configuration for a machine cluster. The processors are configured without display monitors, and each one contains a reasonable amount of local disk. A “root” machine is responsible for controlling the cluster, and may contain the user file system. We do not assume that the cluster processors share a file system, although this is not excluded.

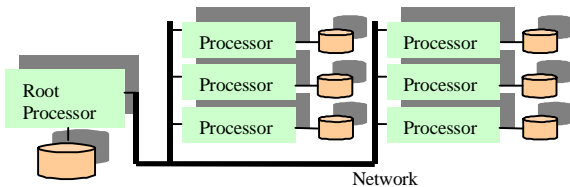


Figure 1 – Cluster configuration

Performing a computational experiment involves 5 distinct phases:

1. Experiment pre-processing
2. Execution pre-processing
3. Execution
4. Execution post-processing
5. Experiment post-processing

Phases 1 and 5 are performed once per experiment. In these, data is set up for the experiment and results are processed. In phase 5 it is often necessary to run data interpretation or visualisation software. Phases 2, 3 and 4

are run for each distinct parameter set. Phase 2 prepares data for a particular execution, and phase 4 reduces the data from an individual execution. Phase 3 actually executes the program given a set of parameter values. During phases 2 and 4 files may be moved between the root machine and the cluster processes – unlike general parallel computing this is the only communication that occurs between tasks.

Because of the repetitive nature of these computations, it is desirable that they are performed automatically. Nimrod [7][4][6][5] and Clustor [12] are general purpose tools which automate this process. Thus, it is possible to run thousands of jobs per experiment without consideration of the underlying computational resources, and without concern for management of the data.

2.3 Active Tools Cluster

Cluster is a tool which manages the execution of parametric studies across distributed computers. It takes responsibility for the overall management of an experiment, as well as the low level issues of distributing files to remote systems, performing the remote computation and gathering the results. Cluster is a commercial version of the research system Nimrod, and has been applied to a number of case studies.

Cluster supports the five phases of an experiment discussed in the previous section. A user develops a short “plan” file which describes the parameters, their default values, and the commands necessary for performing the work. The system then uses this information to transport the necessary files and schedule the work on the first available machine.

When the user invokes Cluster on a workstation, the machine becomes known as the “root” machine because it controls the experiment. When the dispatcher executes code on remote platforms, each of these is known as a computational “node”. Thus, as shown in Figure 2, a given experiment can be conducted with one root and multiple nodes, each of a different architecture if required.

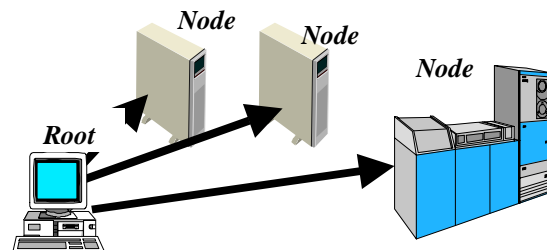


Figure 2 – Cluster Configuration

A plan file is composed of two main sections, the parameter section and the tasks section. Figure 3 shows a sample plan used for some of the simulation work discussed in this paper. The experiment consists of 4 parameters, “grpval”, “conval”, “seedval” and “mttfval”. “grpval” and “mttfval” are selected from a list of possible legal values. “conval” is taken from a range of values, varying from 0.5 to 1.0. “seedval” is a random integer between 0 and 10,000 – in this case each execution receives a different seed value. Cluster generates one job for each unique combination of the parameter values, by taking the cross product of all values. In the plan in Figure 3, up to 162 jobs would be generated.

In this example there are two tasks – “main” and “rootfinish”. The “main” task is executed for each set of parameters. It runs an OpNet simulation on a node, passing the parameter values to the program through the command line. It then copies a number of result files back to the root machine, appending each name with a unique identifier. This task corresponds to the 3rd phase discussed in the previous section. The “rootfinish” task executes once per experiment, and corresponds to the 5th phase described in the previous section. It catenates all of the output files together to produce a single list of results for each observed variable.

```
parameter grpval label "Group Value" float select
  anyof 0.7 0.5 0.2;
parameter conval label "Connectivity" float range
  from 0.5 to 1.0 step .1;
parameter seedval label "Seed Value" integer random
  from 0 to 10000;
parameter mttfval label "MTTF" integer select
  anyof 200 500 800 1100 1400 1700 2000 2300 2600 2900;
task main
  node:execute op_runsim -net_name adhoc -duration 30000 \
  -seed $seedval -ef attrs_1 optimize_simulation \
  -mttf $mttfval -connectivity $conval -grp_size $grpval \
  -join_rate 0.1111 -core_node 5-static/dynamic static
  copy node:average_delay average_delay.$jobname
  copy node:hop_count hop_count.$jobname
  copy node:overhead_result overhead_result.$jobname
  copy node:throughput_result throughput_result.$jobname
endtask
task rootfinish
  execute cat average_delay.* > average_delay
  execute cat hop_count.* > hop_count
  execute cat overhead_result.* > overhead_result
  execute cat throughput_result.* > throughput_result
endtask
```

Figure 3 – Sample Plan file

The plan file is processed by a tool called the “generator”. The generator takes the parameter values, and gives the user the choice of actual values. It then builds a “run” file, which contains a description for each job. The “run” file is processed by another tool called the “dispatcher”, which is responsible for managing the computation across the nodes. The dispatcher implements the file transfer commands, as well as the execution of the model on the remote node.

3 Simulation of Network Protocols using OpNet

The OpNet [13] simulation package allows rapid modelling of complex communication systems by providing features such as: a graphical user interface (GUI), a comprehensive library of contemporary protocols (e.g., TCP/IP, ATM, and Ethernet) and an object oriented development environment.

The process of modelling a communication system is distributed over a three level hierarchy. The lowest level is referred to as the *process model* and is the point at which a protocol’s finite state machine (FSM) is defined. The middle level is called the *node model* and contains one or more process models. When multiple process models are present they are usually configured to represent a layered protocol stack. Finally, the *network model* is the highest level and contains the individual elements that make up a real network (e.g., routers, clients and servers). Each of these elements is associated with a single node model that determines its behavior in the network.

Currently OpNet provides two methods for executing simulation runs: firstly, from within the OpNet environment and secondly, as an independent executable. The GUI of the OpNet environment provides a convenient method for subjecting a simulation model to a range of input parameter values. The downside associated with the OpNet environment is that simulations can only be executed sequentially on a single machine. When the range of input parameter values is small the time required to complete all simulations maybe acceptable, however for a large range of input parameter values the time required to complete all simulations may become prohibitively long.

By using the independent execution technique, a reduction in simulation time can be obtained. To create an independent executable, OpNet compiles the simulation model into an executable program that is run like any other program on a computer. By running the program on different computers (using unique input parameter values) concurrent execution of the simulation can be carried out. Obviously, the more computers that are used the greater the reduction in simulation time. Unfortunately, using independent executions does have some drawbacks, probably the most significant of which is that it makes inefficient use of computer resources.

Typically, a number of simulation runs are distributed equally across available computers. However, as is often the case with computers, their performance can vary considerably due to differences in hardware (e.g., CPU and memory) or, in the case of shared systems, resource usage. This means that some computers will complete their quota of the simulations prior to others and will remain idle even though the slower computers may have numerous simulation runs pending. Cluster distributes the load across the machines using a FIFO scheduling algorithm, which accounts for the varying run times by using the next available processor when a new task is executed.

Another shortcoming associated with independent executions is that each simulation on a different computer normally needs to be managed manually. This requires a user to log into each of the computers being used to run simulations and provide it with a copy of the simulation executable, then initiate the simulation run. In a situation where only a few computers are being used this process may be manageable, but if many computers are being used the process of manually logging into each of the computers may become tedious. Cluster avoids these problems by automating the process completely.

4 Simulating the 'Ad hoc Multicast Network Protocol': A Case Study

4.1 Prior Work

With the demand for multiparty services such as audio/video conferencing and replicated databases, it is imperative that today's computer networks support multicast routing. Multicast routing provides an efficient technique for the dissemination of a single packet to multiple destinations. The past decade has seen an enormous amount of research into multicast routing, culminating in the development of numerous protocols: DVMRP [9], MOSPF [16], CBT [1], PIM [25] and MIP [21]. These protocols have successfully been deployed in the Internet and currently form the basis of the MBONE [2].

Unfortunately, the majority of today's multicast routing protocols have been designed to operate in networks such as the Internet where nodes (routers) are interconnected via relatively high bandwidth hardwired links (e.g., coaxial and fiber optical cable) and changes in the network topology occur infrequently. In contrast, an ad hoc network contains no fixed infrastructure and nodes are interconnected via relatively low bandwidth wireless links. In addition, nodes are free to roam in a random

fashion resulting in a network topology that changes in a rapid and unpredictable manner.

In an ad hoc network, the time and communication complexity associated with protocols developed for the Internet may lead to the generation of excessive control overhead and/or the inability to quickly re-establish routes. These problems have prompted the recent development of two multicast routing protocols [24][22] specifically for use in an ad hoc network. However, these two proposals lack several desirable properties and in certain situations may not perform particularly well.

The AMP protocol has been developed to address shortcomings associated with current ad hoc multicast routing protocols. Its operation is distributed, provides both source and receiver initiated tree construction, allows dynamic group membership and is loop and deadlock free.

4.2 The AMP Protocol

While a complete description of AMP is beyond the scope of this paper, the following provides an overview of AMP's operation. The interested reader can consult [17][18] for further details.

To provide an efficient multicast routing service AMP uses diffusing computations [10] to structure the underlying network as a directed acyclic graph (DAG) [20], upon which a multicast distribution tree is constructed. Additionally, protocol state maintenance is performed using a hard-state mechanism. The main benefits to be gained from these techniques is that topology update packets are only exchanged when required (as opposed to periodically), and in the event that topology update packets are exchanged, they are generally confined to a small region and need not be forwarded throughout the entire network.

Nodes running AMP maintain several parameters: height, DAG link array, multicast link array and a list of multicast groups active on the underlying DAG. Each of the links interconnecting a node with its neighbouring nodes are assigned a direction of either 'upstream' or 'downstream'.

The direction assigned to a link forming the DAG is obtained by comparing a node's height to the height of its neighbours. If a node's height is greater than its neighbour's then a link is assigned a direction of downstream. On the other hand, if the node's height is lower than its neighbour's then a link is assigned a direction of upstream. Once a link forming the DAG has

been assigned a direction it is then stored in the DAG link array. The direction of a multicast link is simply determined by the exchange of multicast control packets. When a node sends the appropriate multicast control packet to a neighbour, the corresponding multicast link is assigned a direction of downstream. The recipient of the multicast control message assigns the direction of upstream to the multicast link. The direction of a link forming part of the multicast tree is stored in the multicast link array.

To commence a multicast session a node first sets its height to 0 and then forwards a **join request** packet to each of its neighbours; the **join request** contains several parameters among which is the height of the node sending it. On receiving a **join request**, a node first determines if the packet has previously been received, if so the packet is simply discarded. If not the node updates its height to that contained in the **join request** packet plus one ($\text{height} + 1$). The direction of each of the DAG links is then updated as discussed previously.

Once a node has received a **join request** packet it is free to join the multicast session. To do so, a node first selects a downstream link from the DAG link array and then proceeds to forward over it a **join response** packet. The node sending the **join response** assigns a multicast direction of upstream to the link over which the **join response** was sent. On receiving a **join response** packet a node first assigns a multicast direction of downstream to the link over which the **join response** was received. The node then determines whether it is already a member of the group specified in the **join response** packet. If so, the packet is discarded. If not, the node simply joins the multicast session as previously discussed.

On receiving multicast data a node determines the direction of the multicast link over which the data was received. If the direction is downstream then the data is discarded, however if the direction is upstream then the data is forwarded to the application and/or interested end systems (hosts). In addition, if the node contains downstream multicast links the data is forwarded onto its neighbours.

When a change in the network topology occurs, more specifically when a link fails, a node reacts in several ways. If the failed link does not result in either: a node losing its last downstream DAG link or loss of an upstream multicast link, then the node takes no action. However, if the failed link results in a node losing its last downstream DAG link then the node sets its height to

a level that is higher than the neighbour with the highest height (this technique is referred to as link reversal). The node then assigns the direction of downstream to all DAG links and informs its neighbours of its new height by forwarding an **update** packet. On receiving an **update** packet a node updates the DAG direction of the link over which the **update** packet was received. On receiving an **update** it is possible that a node will also lose its last downstream DAG link. When this does occur, a similar procedure to the link reversal is employed, however the DAG direction of the link over which the **update** was received is not altered.

When the link reversal process is carried out it is possible that a link will have a DAG direction and multicast direction that are the same. To ensure the multicast tree remains correct (loop free) a link's DAG and multicast direction must be opposite. During the link reversal process a node removes any entries from the multicast link array. On receiving the **update** packet a node follows a similar procedure and removes any multicast link entries that have an equivalent DAG direction. In doing this it is more than likely that a node will no longer remain part of the multicast group. To rectify this situation a node simply rejoins the multicast session.

4.3 A Simulation model of AMP

Using the OpNet simulation package, a comparative simulation study of AMP and RBM was conducted. The simulated network consists of 48 nodes (routers) interconnected via point-to-point links with a bandwidth of 1024 bits/sec, resulting in a mesh topology as shown in Figure 4. The size of packets exchanged between nodes is 1024 bits for data while control packets vary between 10 and 60 bits depending on the type of control packet. Data packets are generated at a constant bit rate, the value of which is a simulation input parameter.

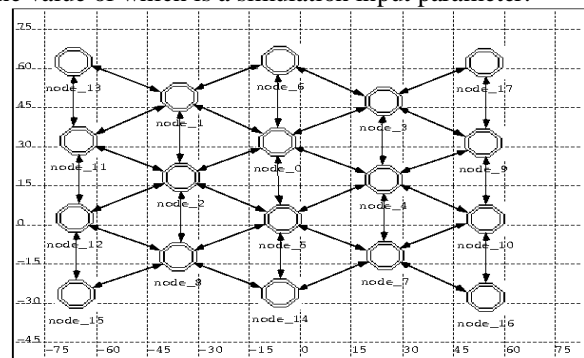


Figure 4 – Example Network.

To simulate the dynamic nature of an ad hoc network each point-to-point link alternates between two states

(active and inactive) according to a two-state Markov process. In the active state packets are transferred in order and error free. Any packets in the process of being delivered when an active link becomes inactive are destroyed. In the inactive state all packets are destroyed.

The duration a link spends in the active state is exponentially distributed with the mean being a simulation input parameter. A high mean value corresponds to a relatively static topology while a low mean corresponds to a highly dynamic topology. The time a link spends in the inactive state is determined using the following equation:

$$1/\lambda = ((1 - \alpha) / \alpha) * 1/\mu$$

where $1/\mu$ is the average duration a link remains in the active state and α (a simulation input parameter) is the average fraction of total time a link remains in the active state. A high α corresponds to a densely connected network, while a low α corresponds to a sparsely connected network.

At the being of a simulation run both the core node and multicast group members are randomly selected. To observe the scaling properties of both protocols the number of group members is varied, the value of which is a simulation input parameter. Each simulation is run for a duration of 30000 seconds. Table 1 summarizes the simulation input parameters.

<i>Input Parameter</i>	<i>Description</i>	<i>Value</i>
Link Active Time (Sec)	Represents the average time a link remains active.	200 to 4200 in steps of 500
Connectivity	The average fraction of links which are active at any given instant in time.	0.3, 0.5, 0.7 and 0.9
Group Size	The fraction of nodes which are members of the multicast group.	0.1, 0.4 and 0.7
Data Rate (Pkts/Min)	The rate at which data packets are generated.	6, 3, 1.5 and 1

Table 1 – Simulation Input Parameters.

5 AMP Simulation Results

The simulation model described in the previous section (4.3.A) was used to collect data in relation to the performance metrics summarized in Table 2.

The following results show the performance of AMP and RBM for a connectivity of 0.9 and a data rate of 6 Pkts/Min.

<i>Metric</i>	<i>Description</i>
Throughput (Bits/Sec)	Average number of data bits received by multicast group members per second.
Overhead (Bits/Sec)	Average number of control bits generated by all network nodes per second.
Hop Count	Average number of hops required to distribute data packets to multicast destinations.
Packet Delay (Sec)	Average time required to deliver data packets to multicast destinations.

Table 2 – Performance Metrics.

5.1 Throughput

By examining Figures 5 and 6 it can be seen that for a small group size (0.1) there is little difference between the throughput of AMP and RBM. However, for larger group sizes (0.4 and 0.7) AMP's throughput is significantly higher than that of RBM. More importantly, AMP's throughput remains relatively constant with respect to the rate of topological change (link active time). In contrast, RBM's throughput drops significantly as the rate of topological change increases (link active time < 1500).

5.2 Overhead

Figures 7 and 8 show that the overhead generated by AMP is considerably less than that generated by RBM. Interestingly, the overhead generated by AMP is largely independent of the group size whereas the overhead generated by RBM is closely related to the group size. This characteristic has important implications in determining the protocol's ability to scale.

5.3 Hop Count

As shown in Figures 9 and 10, AMP requires less network resources than that of RBM. For a large group size (0.7) the number of links forming a multicast tree constructed by AMP is roughly equal to the number of links used by RBM in a small group (0.1). As the group size increases, RBM uses significantly more links than AMP.

5.4 Packet Delay

The average delay encountered in delivering data packets to multicast destinations is shown in Figures 11 and 12. In a relatively static network (link active time > 2000) with a small group (0.1) AMP and RBM have similar delays. However, as the rate of topological change increases (link active time < 2000) AMP's delay becomes less than RBM's. For larger group sizes (0.4 and 0.7)

AMP's delay is significantly less than RBM's over the entire range of link active times.

6 Cluster Performance

We have experimented with two different configurations in the application of Cluster to the AMP simulation reported in the last section. Initially, the work was performed on a small number of SUN Ultra Sparc workstations, connected by 10 Mbit ethernet and a NFS mounted file system. The results presented in this paper were produced using this environment. We also experimented with running the simulations on a 20 processor Windows NT cluster consisting of 10 dual processor, 350 MHz, Pentium IIs. Unfortunately, at the time the paper was written the OpNet binaries were not running correctly on Windows NT, and so it was not possible to perform the same simulation runs. To gauge the effectiveness of the platform, however, we performed the same experiment without the actual network simulation code. All of the input files were copied to a node, and the output results were copied back to the root. However, instead of the OpNet simulation another null process was executed lasting 25 minutes – well short of the 8 hours for the full simulation on the Sparc Stations. This scheme simulates the effect of running the actual experiment, shows the effect of the file copying on the performance and gives a worst case performance. Table 3 shows the performance of the resulting simulations. In both cases the efficiency is very high, mainly because the simulation time far exceeded the setup and management times.

System	Time on 1 processor	Time on N processors	Speedup/Efficiency
SUN Ultra Sparc	480 mins	3 procs 160 mins	3/ 100%
Pentium II Cluster	1750 mins	20 procs 100 mins	17.5/ 87.5%

Table 3 – Speedup using Cluster

7 Conclusions

This paper has described how it is possible to accelerate the parametric simulation of computer networks using a sequential commercial-off-the-shelf simulation package, OpNet, and a cluster of inexpensive PCs. The key is to use a software tool, called Cluster, for distributing the individual runs for each parameter combination concurrently to as many machines as possible. Cluster administers the generation of individual jobs and their distribution in a seamless way, so the user is unaware that the jobs are not running on the one machine. Using this technique it is possible to reduce the execution time dramatically.

The paper also presented some simulation results for a particular network protocol, AMP, which has been designed to support mobile communication multicast operations. The paper shows some results of the simulations that have been performed, and shows how the protocol performs as key parameters, such as rate of topological change, traffic load, connectivity and group size are altered.

A new research version of Cluster, called Nimrod/G is currently being prepared [5]. Nimrod/G addresses issues of how to access wide area resources using a meta-computing toolkit, called Globus [15]. Nimrod/G also presents an interface which allows the user to specify soft real time deadlines for the experiment. In this way the user can request that a given experiment will be completed by a particular time. The system then searches for sufficient resources to perform the work, and schedules the execution on the machines. This type of tool will make it possible for users to perform very large parametric modelling experiments using widely distributed computers. Experiences with such a tool will be reported at a later date.

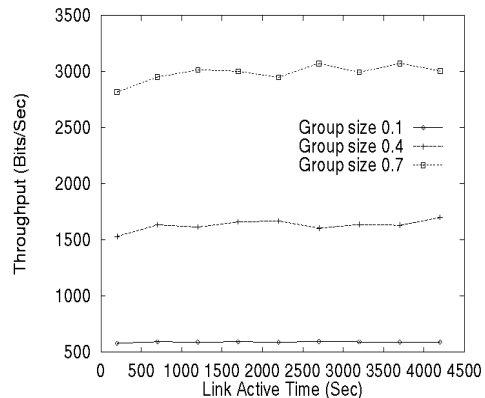


Figure 5 – AMP Throughput.

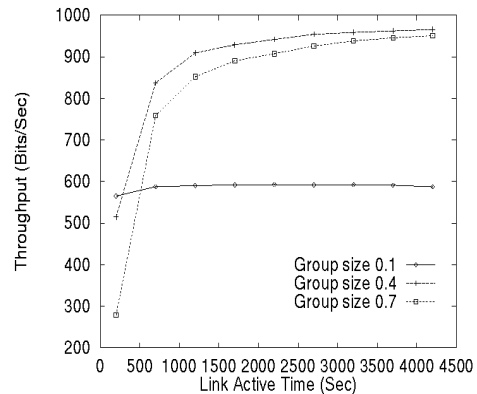


Figure 6 – RBM Throughput.

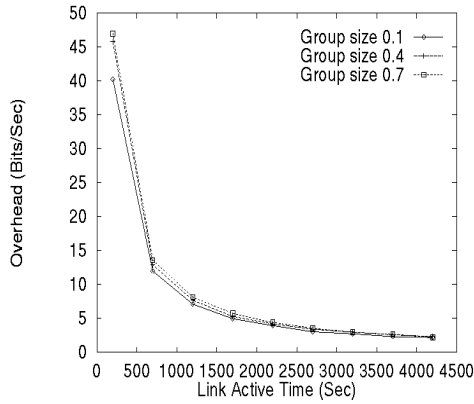


Figure 7 – AMP Overhead.

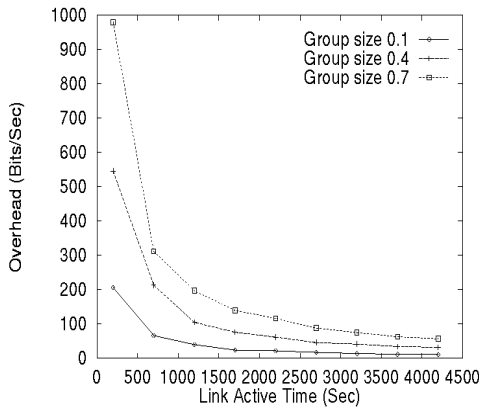


Figure 8 – RBM Overhead.

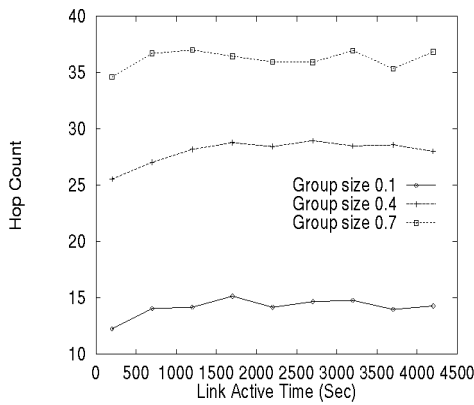


Figure 9 – AMP Hop Count

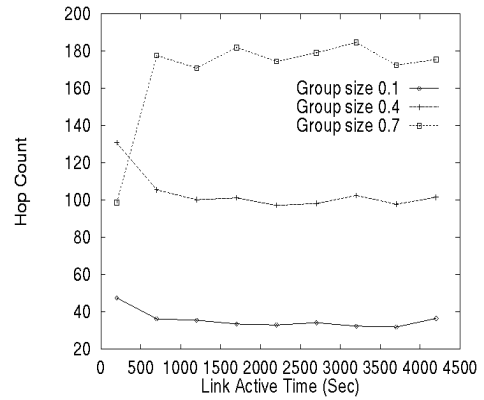


Figure 10 – RBM Hop Count.

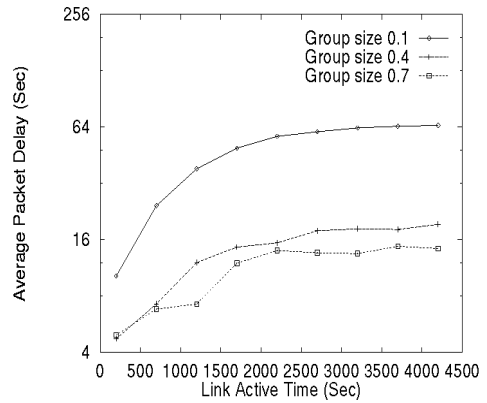


Figure 11 – AMP Packet Delay

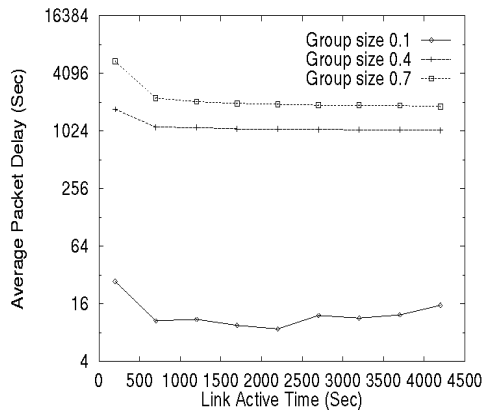


Figure 12 – RBM Packet Delay

8 Acknowledgements

The authors would like to thank Mr James Aarons and Mr Johnny Darous, who performed this work as part of their third year Computer Science project at Monash University, and Russell Kennett who was employed on a summer scholarship. We also acknowledge support from the Monash University Research Fund for providing the

computational resources, and Active Tools for providing the Cluster package. Nimrod was developed as part of a research program supported by the Co-operative Research Centre for Distributed Systems Technology. Bin Qiu, Mahbub Hasan and Jim Breen contributed to the design of the AMP protocol.

9 References

- [1] A. J. Ballardie, P. F. Francis and J. Crowcroft. Core-based trees (CBT). *Proc. SIGCOMM '93*, Pages 85-95.
- [2] C. Huitema. *Routing in the Internet*. Prentice Hall PTR, 1995.
- [3] D. Ridge, D. Becker, P. Merkey, T. Sterling, "Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs", Proceedings, IEEE Aerospace, 1997.
- [4] D. Abramson, R. Susic, J. Giddy, M. Cope, "The Laboratory Bench: Distributed Computing for Parametised Simulations", 1994 Parallel Computing and Transputers Conference, Wollongong, Nov 94, pp 17 - 27.
- [5] D. Abramson and J. Giddy, "Scheduling Large Parametric Modelling Experiments on a Distributed Meta-computer", PCW '97, September 25 and 26, 1997, Australian National University, Canberra, pp P2-H-1 - P2-H-8.
- [6] D. Abramson, I. Foster, J. Giddy, A. Lewis, R. Susic, R. Sutherst, N. White, The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing, Australian Computer Science Conference (ACSC 97), Macquarie University, Sydney, Feb 1997, pp 17 - 26.
- [7] D. Abramson, R. Susic, J. Giddy and B. Hall, "Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations", The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
- [8] D. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawak, C. Packer, "BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION", Proceedings, International Conference on Parallel Processing, 95.
- [9] D. Waitzman, C. Partridge and S. Deering. Distance vector multicast routing protocol. *RFC 1075*, November 1988.
- [10] E. W. Dijkstra and C. S. Scholten. Termination detection for diffusing computations, *Inform. Process. Lett.*, 11: 1-4, August 1980.
- [11] <http://beowulf.gsfc.nasa.gov/>
- [12] <http://www.activetools.com>
- [13] <http://www.mil3.com>
- [14] <http://www.myri.com/>
- [15] I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit. ", *Intl J. Supercomputer Applications*, 11(2):115-128, 1997.
- [16] J. Moy. Multicast extension to OSPF. *Internet Draft 1584*, 1994
- [17] K. Power, B. Qiu, J. Breen and M. Hassan. A Multicast routing protocol for ad hoc networks, *APCC/ICCS '98*, November 1998.
- [18] K. Power, B. Qui and J. Breen. AMP: A multicast routing protocol for ad hoc networks, *ISPACS '98*, November 1998.
- [19] Lewis, D. Abramson., R. Susic, J. Giddy, "Tool-based Parameterisation : An Application Perspective", Computational Techniques and Applications Conference, Melbourne, July 1995.
- [20] M. Behzad. *Graphs and Digraphs*. Prindle, Weber and Schmidt, 1979.
- [21] M. Parsa and J. J. Garcia-Luna-Aceves. A protocol for scalable loop-free multicast routing. *IEEE JSAC*, 15:316-331, April 1997.
- [22] M. S. Corson and S. G. Batsell. A reservation-based multicast routing protocol for mobile networks: Overview of initial route construction. *Wireless Networks*, 1:427-450, November 1995.
- [23] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, W. Su "MYRINET: A GIGABIT PER SECOND LOCAL AREA NETWORK", *IEEE-Micro*, Vol.15, No.1, February 1995, pp.29-36.
- [24] R. Bhattacharya and A. Ephremides. A distributed multicast routing protocol for ad hoc (flat) mobile wireless networks. *8th Int. Sym. Personal, Indoor and Mobile Wireless Communications*. Pages 877-881.
- [25] S. E. Deering et al. The PIM architecture for wide-area multicast routing. *IEEE/ACM Trans. Networking*, 4:153-162, April 1996.