

# Comprehensive Throughput Evaluation of LANs in Clusters of PCs with Switchbench — or How to Bring Your Switch to its Knees

Felix Rauch  
National ICT Australia (NICTA)\*  
Sydney, Australia  
felix.rauch@nicta.com.au

## Abstract

Understanding the performance of parallel applications for prevalent clusters of commodity PCs is still not an easy task: One must understand performance characteristics of all subsystems in the cluster machine, besides the inherently required knowledge about the applications' behaviour. While there are already many benchmarks that characterise a single node's subsystems like CPU, memory and I/O, as well as a few to evaluate its network interface with point-to-point data streams, there are to the best of our knowledge no benchmarks available that characterise a cluster network or LAN *as a whole*.

We present *Switchbench* [14], a set of three microbenchmarks that thoroughly evaluate the throughput characteristics of networks for clusters. A first microbenchmark tests the basic processing limitations of the switches, by sending and receiving data concurrently at maximum throughputs on all network interfaces. A second microbenchmark tests arbitrary communication patterns by pairwise connecting nodes for high-speed throughput tests. A third and slightly more realistic microbenchmark executes an all-to-all personalised communication (AAPC) algorithm to test many different patterns and critical bisections in the network. The microbenchmarks already proved to be extremely useful in a previous study to experimentally quantify performance limitations in different networks of clusters of PCs with up to 128 nodes. We also establish the suitability of our microbenchmarks by comparing their results with two application benchmarks.

The benchmarks consist of two C programs supported by shell scripts to start the programs on all nodes of the cluster with the correct execution parameters to automatically scale the workloads from a few nodes up to the full cluster size.

## 1 Introduction

Technological advances in the development of microprocessors established a new cost-effective platform for a

wide variety of parallel applications: Clusters of commodity and off-the-shelf PCs, which are often referred to as “Beowulf” clusters, named after an early prototype [2]. Many of these clusters are built by their users by piling PCs in a lab and interconnecting them with commodity network switches. More advanced clusters are professionally built by specialised companies, which mount multi-CPU PC blades in a rack and interconnect the nodes by high-performance data networks. A prerequisite of getting the best performance out of applications—whether on cheap self-constructed clusters or high-end supercomputer clusters—is to have a thorough knowledge of the underlying system's performance.

Fully understanding the performance of a parallel application is certainly not trivial, because the performance not only depends on the application's behaviour, but also on the performance of all the involved subsystems of the underlying architecture. As in any machine, this involves basic components like CPU, memory, system bus, I/O system and also system software. In a networked system, the network itself plays a major role for the achievable performance of an application. It is therefore imperative to know the performance characteristics of a node's network interface. However, a single link's performance is only one—important yet small—characteristic of the network. In order to better understand the performance of applications in clusters of PCs, we propose a set of microbenchmarks to thoroughly evaluate the network of a cluster *as a whole*.

Different network architectures have been proposed for clusters, ranging from mesh topologies over tori and fat trees to simple star topologies with a single central switch. Even when running a parallel application on a cluster with a single big Ethernet switch the simplistic assumption that all communication patterns through the switch show the same performance characteristic is clearly not true. Indeed, an earlier study [8] has shown that even apparently similar switches can have vastly different performance characteristics. The details of a cluster's network config-

---

\*National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through *Backing Australia's Ability* and the ICT Research Centre of Excellence programs.

uration may even be completely unknown to the user, e.g. when remotely accessing a big “cycle-crunching” cluster in a compute centre. Hence, we evaluate the performance of an interconnect for specific communication patterns, which can still be represented as microbenchmarks. The primary goal of looking at isolated communication primitives is to gain architectural insights into the bottlenecks of the network.

The rest of this paper is structured as follows: In Section 2 we briefly discuss some related work before we discuss our evaluation principles as well as the limitations of a cluster network interconnect in Section 3. Then we explain our set of microbenchmarks and how it is used to identify these limitations in Section 4. Section 5 lists the results of experimental network evaluations to show the different network performances as identified by our microbenchmarks and Section 6 compares the results with the performance of some real applications on the same networks. Finally in Section 7 we conclude.

## 2 Related Work

There is much work related to the performance evaluation of parallel machines or clusters of PCs, but most of it either analytically models the performance of the parallel machine or the network (of which LogP [5] and LogGP [1] are well-known representatives), evaluates the performance of the overall machine for a certain class of applications (like the popular High Performance Linpack HPL for parallel machines [6]) or quantifies the performance of a single network link. The latter category of work comes closest to our own set of microbenchmarks, however benchmarks of this category like Netperf [7], NetPIPE [15] or TTCF [10] measure the latency and throughput of just a single link in the network for different packet sizes. While these numbers are certainly important to evaluate the base network performance of the nodes, they ignore other important bottlenecks in the cluster’s network, like an under-performing switch or an insufficient topology of the network itself.

The ipbench [16] framework for distributed network benchmarking takes a slightly different approach. It allows many coordinated testers in a network to generate high loads with different protocols towards a designated device under test. Thus, it allows to test nodes and network links with higher loads than a single node can generate. While ipbench stresses the network more than the aforementioned point-to-point benchmarks, it does not address the network’s overall performance limitation.

## 3 Evaluation Principles

### 3.1 Bandwidth vs. Latency

There is a lot of emphasis on communication bandwidth in our benchmarks and the aspect of latency is not considered much. The commodity system architect can not do much about certain latency components in the system like the PCI bus arbitration latency. A common wisdom says that additional bandwidth can be purchased easily while latency is given by nature’s laws (or maybe better by the boundary conditions of systems engineering).

We understand the several order of magnitude difference of latency between a high performance interconnect, using a flit level worm-hole routing scheme in the switches and a communication co-processor at the endpoints vs. the commodity Ethernet that uses store-and-forward routing and a simple host interface using delayed interrupt processing due to coalescing of interrupts. Still many applications are affected by the granularity of communication instead of pure latency and can thus be reprogrammed accordingly to communicate in large blocks or use mechanisms of latency tolerance.

### 3.2 Network Limits

There are mainly three limitations in a cluster network: the end nodes’ performance, the basic processing limits of the switch(es) and the bisection bandwidths of the network. The first limit depends on the nodes’ hardware (network interface controller, CPU, memory and I/O bus) and software (communication protocol stack). We assume a balanced cluster architecture, where a node is able to saturate its network interface within the limits given by protocol overheads. If the nodes could not saturate their network interfaces, then the network installation would partially be wasted (and thus also money) and a thorough evaluation of the overall network performance seems needless, since the major bottleneck is clearly the nodes’ performance. Hence, we will not discuss the end nodes’ limitations in the remainder of this section.

#### 3.2.1 Processing Limits in Switches

The throughput limit of a switch is typically given by either one of two bottlenecks: The number of packets it can process in a given time or the bandwidth between two subparts of the switch. The later limit can be its central switching fabric that connects the external network ports of the switch, or it can be the connections between parts of the switch like switching ASICs or different line modules. These limitations are part of any switch in the network and do not depend on the topology of the network links external to the switches. For well designed switches, these limits cannot be reached, because they appear at higher

loads than the switch is able to receive over its network ports. When cheaper switches hit these limits, their aggregate performance can either remain constant at the maximal level, or worse, even collapse to a low minimum.

### 3.2.2 Full Bisection Bandwidth

Interconnect networks of most regular computing structures are characterised by their *bisection bandwidth*. In the discussion of bisection bandwidth, the worst-case performance-critical bisection of the network is determined (according to the topology) and addressed. For its quantification, the nodes are paired in such a way that all the communication must go across the links on the most critical bisection cut. If the network access provides simultaneous full-duplex links, the communication between the pair of nodes must also be full duplex, i.e. must go simultaneously in both directions.

A network is said to have a full or—in somewhat more precise terms—a *fully scalable bisection bandwidth* if it can sustain the full network access bandwidth of every node across the most critical bisection while all nodes communicate simultaneously. For a Fast Ethernet network with a bandwidth of 100 Mbit/s this means that every node must send and receive data at the same time with 100 Mbit/s over the critical bisection of the network. Network topologies with full or scalable bisection bandwidth include the full fat tree, the hypercube and the full crossbar central switches. The mesh, the torus and the plain/skimmed tree network configurations do not offer scalable bisection bandwidth in general, but for some cases some full bisection communication might be achievable for machines up to a certain fixed size.

### 3.2.3 Communication Patterns Requiring Full Bisection Bandwidth

Communication patterns requiring full-speed communication across the critical bisections are relatively rare and can be avoided in many cases by clever parallel programming or with probabilistic algorithms for large data sets. The most important parallel algorithm requiring full bisections are computations in a bitonic sorting network or an FFT butterfly network. The most common communication pattern limited by critical bisection is all-to-all personalised communication.

#### All-to-All Personalised Communication (AAPC)

The all-to-all personalised communication (AAPC) step is frequently encountered in parallel programs. In an AAPC step, each processor sends a block of distinct data to every other processor. The AAPC step occurs in multi-dimensional convolutions (e.g. FFTs) and in array transposes where only one dimension of the array is

distributed [13]. High-performance-computing applications with many such operations do not only require many GFlops/s, but in addition GByte/s communication performance.

A simple message-passing AAPC program looks like the following pseudo code:

```

parallel algorithm AAPC:
1 for  $i = 1$  to  $NumberOfProcessors - 1$  do
2   NBSendMsg( $Destination_i$ ,  $DataBlock_i$ )
3 for  $j = 1$  to  $NumberOfProcessors - 1$  do
4   NBReceiveMsg( $Source_j$ ,  $DataBlock_j$ )

```

We assume that NBSendMsg() and NBReceiveMsg() are buffered, non-blocking primitives offered by the message-passing library. Still this simple program will cause congestion, loss of packets and TCP retransmissions for any larger machine using simple Ethernet networks. For the microbenchmarks in Section 4, we modify the algorithm to proceed in phases carefully controlling the congestion in each phase.

## 4 Microbenchmark Description

In this section we describe our three microbenchmarks in more detail and also briefly describe their implementation.

### 4.1 Daisy Chain

Our first microbenchmark measures the basic processing limits of the switch by sending a TCP data stream through an increasing number of nodes in the cluster by using a daisy-chain topology. All nodes in the chain except the first and the last receive data and forward it concurrently to the next node in the chain at the highest possible throughput. This microbenchmark identifies the maximum attainable throughput which the switch’s subsystems can process. The benchmark does not address the bisection bandwidth, because the communication is only between neighbouring nodes.

The daisy-chain benchmark stems from our tool called “Dolly” [11, 12], which is used for high-performance hard-disk duplication (also known as “disk cloning”) over a network in clusters of PCs. In this benchmark, Dolly simply sends traffic through the daisy chain *without* accessing the nodes’ hard disks. Dolly scales very well as long as the switches are able to process full-speed duplex connections on all ports, and thus reveals processing limits of the switch(es) in the cluster.

### 4.2 Pairwise Streaming

The second benchmark in our set allows to test any bottlenecks that become visible in a network when special communication patterns are used. The benchmark sends data

in full-duplex mode between many different pairs of nodes in the cluster. The pairs are specified as parameters to the benchmark script and are chosen to test specific communication patterns in the network. The result of the benchmark is the minimal, maximal and average throughput of all pairs of streams as selected by the user. The benchmark is thus not fully automatic and will not give meaningful *general* numbers about the network. However, the benchmark is very useful to easily test specific bisections and communication patterns in the network and thereby identify bottlenecks that might limit application performance. The benchmark is included in this set as a useful tool for performance debugging of any cluster network. In fact, it already proved to be very useful to identify performance bottlenecks in big Ethernet switches for clusters of PCs.

### 4.3 Congestion-Controlled AAPC

As a benchmark which complements the two previously described benchmarks—and which uses a slightly more realistic communication pattern—we use a *congestion controlled AAPC*. A simple AAPC algorithm is described in Subsection 3.2.3. A more optimised *phased AAPC* algorithm can achieve optimal aggregate bandwidth once the different phases are carefully separated. Phase separation can be maintained by globally synchronising the entire cluster after each phase is completed. This strategy adds some overhead for synchronisations and might require additional communication resources, but it makes sure that no communication resources are wasted due to inefficient scheduling and due to unnecessary congestion.

A simple algorithm for phased AAPC proceeds as follows: In the first phase every node sends data to its next higher neighbour and receives data from its next lower neighbour. In the next phase, every node sends data to its next but one higher neighbour and receives data from its next but one lower neighbour etc. In the last phase every node sends data to its next lower neighbour and receives data from its next higher neighbour. A pseudo-code representation of our implementation looks as follows. Each node  $n_{self}$  runs the all-to-all algorithm shown here in pseudo code:

```
parallel algorithm all-to-all
1 for  $i = 1$  to  $n - 1$  do
2   concurrently send data to node  $n_{self+i \bmod n}$ 
   and receive data from node  $n_{self-i \bmod n}$ 
3   wait for barrier
```

For the evaluation of the AAPC performance we try to minimise the congestion in each phase. For every phase each node has a fixed communication partner to send to and to receive from. The patterns can be symmetric (same node to send to and receive from) or asymmetric (different nodes, as shown in the preceding pseudo code). Since

phases are synchronised across the entire cluster, the duration and the final throughput is determined by the slowest connection of each phase. The output of our benchmark program is the throughput over all phases or the corresponding execution time of the whole AAPC (which is more meaningful from an application’s point of view).

In the simple algorithm above the logical communication distance increases with each phase of the algorithm. The physical distance between the communicating nodes depends on the mapping of node numbers to communication ports and of the topology of the network. An AAPC contains many different communication patterns and a large number of network properties are exercised. In a more in-depth study [8] we looked in great detail at the performance of the individual phases of the AAPC, which give a more detailed insight into the network’s capabilities of dealing with increasingly distant communication partners.

### 4.4 Implementation

The core of the benchmarks is implemented in two C programs, one for the daisy-chain benchmark and one for the pairwise and AAPC benchmarks respectively. The programs need to run on all involved nodes and manage the connection setup between themselves as well as the actual measurements according to their input parameters. A few bash shell scripts support the user by taking care of correctly starting the programs on all nodes with the right parameters, collecting the results and terminating the programs when the measurements are done. The scripts allow to specify the nodes’ base name, alternative interface name, a numeric range of node numbers and an increment. These parameters will then automatically be translated into the list of nodes and interfaces to use for the benchmark runs. Thus, the invocation command is kept short, while still facilitating to run the benchmarks with different communication patterns on large clusters with automatic scaling of the participating nodes.

The benchmarks were written and tested on clusters running the GNU/Linux OS, but should be straightforward to port to other OSs, as they only use TCP, standard system calls and readily available OS tools. The complete source code of an updated version is available under the GNU general public license from [14].

## 5 Evaluation Examples

In this section we show the usefulness of the Switchbench benchmark set with evaluations of different networks in our clusters of PCs, which are described in detail in [11]. Each of the three evaluations with the benchmark reveals insights about the performance limits of the cluster network under test.

## 5.1 Daisy Chain

In a first set of experiments we measure the basic processing limits of three different Ethernet switches for an increasing number of nodes with the ‘‘Dolly’’ daisy-chain benchmark. The nodes are equipped with two 1-GHz PentiumIII processors, a Fast Ethernet and a Gigabit Ethernet network interface, and are all directly connected to a single switch. We characterise two Fast Ethernet and one Gigabit Ethernet switch. The results are depicted in Figure 1 (note the different y-axes for the different network technologies). The figure shows three different behaviours: the performance of the ATI switch scales perfectly up to the maximum number of available nodes, the Cisco switch’s performance collapses when the offered load is higher than only half of the maximum, and the SSR8600 switch reaches its maximum capacity with 12 nodes and then sustains that performance.

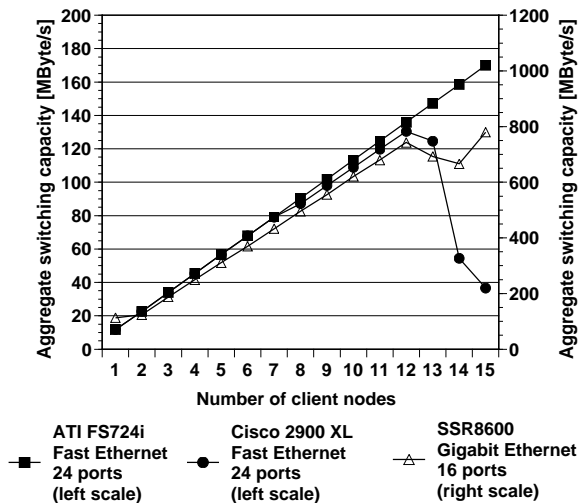


Figure 1: Daisy-chain benchmark with an increasing number of nodes with two Fast Ethernet and one Gigabit Ethernet switch (note the different axes).

## 5.2 Pairwise Streaming

In a second set of experiments we compare different communication patterns of a Fast Ethernet switch which was initially used in our 128-node cluster ‘‘Xibalba’’ [17] and which showed disappointing performance for some applications during early tests after its installation. The switch was an Enterasys Matrix E7 and comprised four 6H302-48 line cards each providing 48 Fast Ethernet ports. We run the pairwise streaming test and compare communication patterns of nodes that are connected to the same line cards (intra-module), as well as patterns with nodes that are connected to different line cards (inter-module) of the switch. The results are listed in Table 1 and reveal two bottlenecks

of the E7 switch: a single module’s switching capacity is not high enough to support full-duplex communication for all its ports with full throughput (top half of table) and the inter-module communication can not support more than eight pairs communicating with full throughput (bottom half of table). The latter problem was later admitted by the manufacturer<sup>1</sup>.

Performance Matrix E7 switch		
Communicating pairs	Nr of nodes	Transfer Rate [MByte/s]
Intra-module communication (next neighbour)	7+7	11.2
	8+8	10.5
	9+9	9.7
	12+12	7.8
Inter-module communication (line-module interconnect)	7+7	11.3
	8+8	10.2
	12+12	6.9
	48+48	<b>2.2</b>

Table 1: Pairwise benchmark with different communicating pairs of nodes, strikingly revealing the inter-module bottleneck of a non-performing Fast Ethernet switch.

## 5.3 Congestion-Controlled AAPC

In a third set of experiments we show the performance of the AAPC benchmark, in which each node exchanges data with every other node, thereby testing a wider variety of communication patterns in the network. We evaluate four different networks: The first one is a cheap maintenance network composed of eight small switches (one for 16 nodes each), of which each has a single Fast Ethernet uplink to a central switch. The uplinks are a major bottleneck when more than one pair of nodes from different switches communicate at the same time. The second network in this set of experiments is the same Matrix E7 switch from the previous subsection with all nodes of the cluster directly connected<sup>2</sup>. In the third network, all nodes are directly connected to a more expensive Enterasys X-pedition ER16 Fast Ethernet switch. The fourth network is the Myrinet [3] Gigabit network (in our configuration of the Myrinet network, two CPUs share a network link, hence we name it ‘‘shared Myrinet’’). The execution times and the respective AAPC throughputs are shown in Figure 2. The different networks and their respective limi-

<sup>1</sup>More details about the performance problems of the switch that we discovered with the pairwise streaming benchmark are described in [8].

<sup>2</sup>Depending on the number of populated network ports per line card of the switch, the execution time of the AAPC benchmark on the same ER16 switch varies between 711 and 468 seconds. For this study, we populated every line card with 32 out of 48 available ports to alleviate some of the aforementioned limitations of the switch.

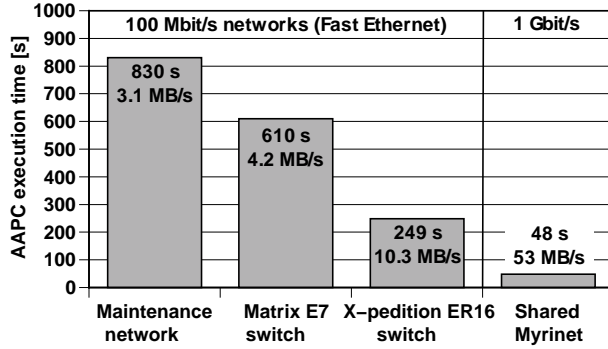


Figure 2: Execution times and throughputs for the all-to-all personalised communication (AAPC) benchmark with 60 CPUs on three different Fast Ethernet networks and on the Myrinet Gbit/s network.

tations lead to great variations in the performance of our AAPC microbenchmark, even in the case of the two identical network architectures with the apparently similar E7 and ER16 switches. More detailed results of the AAPC benchmark for the different Fast Ethernet networks are shown in Figure 3, which compares the networks' performance vs. communication distance. The logical communication distance increases with the phase number until the maximal distance is reached in the middle of the loop's execution, and then decreases towards the last phase (see phased AAPC algorithm in Subsection 4.3). The figure reveals that the cheap maintenance network with its strongly limited uplink bandwidth has severe problems with communications over distances larger than one. The Matrix E7 switch's limits for larger communication distances are not as drastic, but still clearly visible. The X-pedition ER16 switch offers nearly optimal communication performance, but still has some minor irregularities.

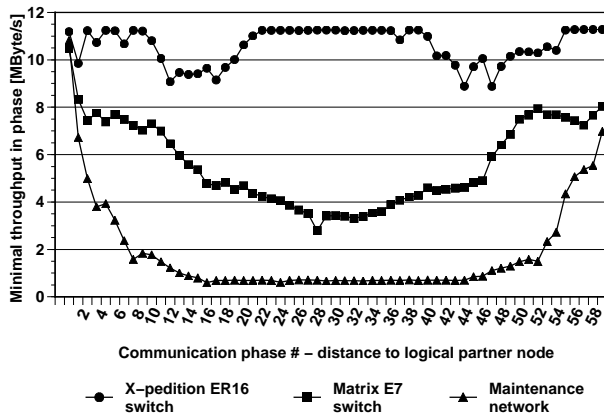


Figure 3: Minimal throughputs for each phase of the AAPC benchmark with 60 CPUs on three different Fast Ethernet networks.

## 6 Comparison with Application Benchmarks

In this section we present two application benchmarks in order to show the correlation of the performance of our microbenchmarks and real applications.

The first of the more comprehensive benchmarks in this section is HPL (High Performance Linpack [6]), which is a popular benchmark to evaluate the computational capabilities of supercomputers and clusters. The results of that benchmark are published semi-annually in the Top500 list of the worlds most powerful computers [9]. The benchmark involves solving a system of linear equations. The results of the benchmark depend only moderately on the performance of the underlying communication network and the tasks executing at the different nodes of the cluster are mostly compute bound. Despite the broadcasting of data in the computation, the communication is mostly between near neighbour nodes in any time-step and does not seem to require a high bisection bandwidth.

We examine the results of the HPL benchmark which was run on 16, 24, 32 and 64 processors on the same four networks previously described in Section 5. Three of them are based on Fast Ethernet: The cheap maintenance network with its very limited bisection bandwidth, the somewhat limited Matrix E7 switch and the much improved X-pedition ER16 switch. The fourth network is the Gbit/s Myrinet network with two processors sharing a network interface. The results of the benchmark are shown in Figure 4. The HPL benchmark was not tuned for maximal performance, as every node uses 50 MByte of memory during all the experiments. The results are fine to compare the different network architectures against each other, but should not be used to compare absolute performance with other clusters.

When more than 16 nodes are used, the limited bisection bandwidth of the maintenance and the E7 network—as measured with Switchbench in Section 5—become

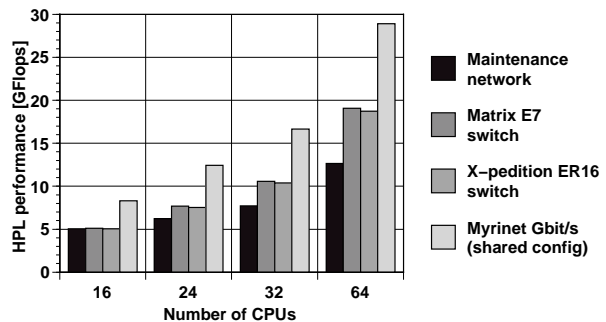


Figure 4: Performance comparison of the same networking architectures from Section 5 for the popular HPL benchmark in GFlops.

more important factors. The same holds for the additional latency due to the stacked switches in the maintenance network (which we do not analyse here). The low-cost maintenance network shows clearly worse performance than the fully switched Ethernet networks with single central switches. The two Ethernet networks E7 and ER16 respectively achieve about the same performance. Myrinet with its much higher bandwidth and lower latency surpassed all the Ethernet network architectures.

As a second application benchmark, we use the QTPlan parallel program to model queueing in traffic micro simulations [4]. In our experiment the application simulated 6 hours of real-time traffic in Switzerland. The input comprised 50'000 and 990'000 automobiles respectively, on their way through a two-lane tunnel of the single highway passage to the southern part of Switzerland. The road map is space partitioned in order to minimise the number of connections between the processor nodes. The crossing of the partitions around the actual traffic bottleneck translate into a network bottleneck between the two machines that hold these partitions. The 990'000 cars scenario is simulating a more balanced everyday scenario when most of the cars are on the way to and from work all over Switzerland. The QTPlan simulation has computing as well as communication intensive parts and its performance results are shown in Figure 5.

The execution times of the application tests are taken from a single test run with 64 CPUs. With a small working set (left side of Figure 5) there are less cars in each space partition (and therefore in each node) and the ratio of computation vs. communication drops. With a greater proportion of communication the factor network becomes more relevant resulting in runtime that doubled on the minimal cost maintenance network. With large working sets (right part of Figure 5), the amount of local computation increases in every space partition resulting in a higher computation vs. communication ratio. The contribution of the factor network to the total runtime decreases and results in a smaller difference in runtime between the networking architectures. Despite the lower influence of the network, the application benchmark shows similar performance differences as identified by Switchbench in Section 5.

## 7 Conclusions

In this short paper we identified the need for a thorough performance analysis of network interconnects in clusters of commodity PCs, in order to better understand and optimise parallel applications' performances and possibly improving applications by adapting them to the limits of the underlying network architecture. While other network benchmarks focus on the performance of single

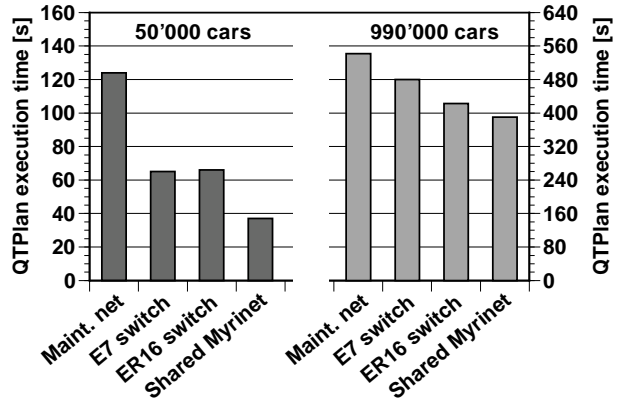


Figure 5: Runtime of QTPlan traffic simulations for 50'000 and 990'000 cars on 64 CPUs in seconds.

point-to-point links or the evaluation of a single server, our set of microbenchmarks called *Switchbench* evaluates the throughput performance of a network *as a whole*.

The daisy-chain benchmark is the first of our microbenchmarks and simply measures the general packet-processing and bandwidth limits of a switch or its subsystems respectively. By sending data in a virtual TCP daisy chain through the nodes, every participating node in the cluster sends and receives data at highest possible throughputs, thereby stressing every network port of the switch. The pairwise microbenchmark then allows to test specific communication patterns in order to identify critical bisections within to network. It is highly useful for debugging the performance of the network. Finally, the third microbenchmark is based on an all-to-all personalised communication, thereby stressing many different bisection bandwidths with a more realistic communication pattern.

Our microbenchmarks identify different critical bottlenecks in cluster networks, as we demonstrate with a series of experimental evaluations. The network differences identified by our microbenchmarks match the outcomes of two application benchmarks in the same networks within reasonable bounds. A perfect alignment is impossible to achieve, since the microbenchmarks evaluate just the network architecture's throughput, while real applications also exercise other parts of the system like most notably the nodes' CPUs and memory.

In conclusion, we believe our *Switchbench* microbenchmarks are a valuable contribution towards fulfilling the community's need for an improved performance characterisation of entire communication networks in LANs and clusters of PCs.

## Acknowledgements

The first version of Switchbench was written by C. Kurmann and used for much of the experimental work, which was conducted together with C. Kurmann under the supervision of T. M. Stricker. Further, we would like to thank N. Cetin and K. Nagel for their help with the QTPlan application and the opportunity to use their application code for our experiments. Finally, we would also like to thank M. Chakravarty, G. Keller and the University of New South Wales for access to their clusters of PCs, which enabled implementing and testing improvements to Switchbench.

## References

- [1] A. Alexandrov, M. F. Ionescu, K. E. Schauer, and C. Scheiman. LogGP: Incorporating long messages into the logp model—one step closer towards a realistic model for parallel computation. In *Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures*, July 1995.
- [2] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer. Beowulf: A Parallel Workstation for Scientific Computation. In *Proceedings of the 1995 International Conference on Parallel Processing*, Aug. 1995.
- [3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, Feb. 1995.
- [4] N. Cetin and K. Nagel. Parallel Queue Model Approach to Traffic Microsimulations. In *Proceedings of Swiss Transportation Research Conference*, Mar. 2002.
- [5] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauer, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. *SIGPLAN Notices*, 28(7):1–12, July 1993.
- [6] J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart. *Linpac users guide*. SIAM, Philadelphia, PA, 1979.
- [7] R. Jones. Netperf: A Network Performance Benchmark. <http://www.netperf.org/>.
- [8] C. Kurmann, F. Rauch, and T. M. Stricker. Cost/Performance Tradeoffs in Network Interconnects for Clusters of Commodity PCs. In *Workshop on Communication Architecture for Clusters, in conjunction with International Parallel and Distributed Processing Symposium (IPDPS '03)*, Nice, France, Apr. 2003. A slightly extended version is available as technical report 391, ETH Zurich, Department of Computer Science.
- [9] H. Meuer, E. Strohmaier, J. Dongarra, and H. D. Simon. TOP500 Supercomputer Sites. <http://www.top500.org/>.
- [10] PCAUSA. Test TCP (TTCP): Benchmarking Tool for Measuring TCP and UDP Performance. <http://www.pcausa.com/Utilities/pcattcp.htm>.
- [11] F. Rauch. *Distribution and Storage of Data on Local and Remote Disks in Multi-Use Clusters of PCs*. PhD thesis, Dept. of Computer Science, Swiss Federal Institute of Technology (ETH Zurich), Zurich, Switzerland, 2003. ISBN 3-89649-893-2.
- [12] F. Rauch, C. Kurmann, and T. M. Stricker. Optimizing the Distribution of Large Data Sets in Theory and Practice. *Concurrency and Computation: Practice and Experience*, 14(3):165–181, Apr. 2002.
- [13] T. Stricker and J. Hardwick. From AAPC Algorithms to High Performance Permutation Routing and Sorting. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 200–203, Padua, Italy, June 1996.
- [14] Download page with Switchbench sources. <http://ertos.nicta.com.au/Software/>, 2005.
- [15] D. Turner. Netpipe: A network protocol independent performance evaluator. <http://www.scl.ameslab.gov/netpipe/>.
- [16] I. Wienand and L. Macpherson. ipbench: A framework for distributed network benchmarking. In *AUUG Winter Conference*, Melbourne, Australia, Sept. 2004.
- [17] Xibalba — Computer Science Cluster of ETH Zürich. World Wide Web, <http://www.cs.inf.ethz.ch/xibalba/>, 2003.