

# Performance Simulation of a Dependable Distributed System

Yinong Chen

*Computer Science & Engineering Department  
Arizona State University  
Tempe, AZ 85287-5406, USA  
yinong@asu.edu*

Roger Mateer

*School of Computer Science  
University of the Witwatersrand  
Johannesburg, South Africa  
roger@cs.wits.ac.za*

## Abstract

*The aim of our research is to develop a distributed system platform that supports a variety of tasks. Currently, we are implementing Internet applications on the system, including a service redirector, firewall, and web applications. These applications have different levels of dependability requirements. Depending on their criticality, a single task may execute on one or more computer nodes. Software-implemented fault-tolerant protocols are used to detect the disagreement among replicas. A reconfiguration protocol is used to identify faulty nodes according to the fault reports from other fault-tolerant protocols and to reallocate their tasks to other working nodes. As a part of the project, this work focuses on the implementation and simulation of the system and the firewall application. Data transfer throughput of the system are measured and analyzed. The results are being used in supporting our development of the overall system.*

**Keywords:** *prototyping, performance, modeling, fault-tolerant computing, networking.*

## 1. Introduction

Dependability has been defined as the property of a computer system such that reliance can justifiably be placed on the service it delivers [1]. Different kinds of software and hardware dependable techniques have been developed to produce various kinds of highly dependable systems with different dependability attributes, including

- Reliability: The property of continuity regarding service delivery. Reliability is denoted by  $R(t)$ , which is the probability that no failure occurs in the time period  $[0, t]$ .
- Availability: The property of readiness regarding service delivery.
- Fail-safe: The property of non-occurrence of unacceptable external failure mode due to occurrence of internal faults.

Highly dependable systems are traditionally used in safety-critical control and monitoring systems like nuclear reactors, flight control and traffic scheduling, etc. The recent development of using commercial off-the-shelf components to build dependable systems has greatly encouraged the use of dependable computing techniques in cost-sensitive commercial systems [2]. Internet and

e-commerce are good examples of such systems. Services provided via Internet are not safety-critical, at least at this stage. It has, however, become business-critical. For example, all major banks worldwide have offered Internet-based services to their clients. The banks heavily rely on the correct, secure and continuous operation of their servers. Another example is the traffic cache server in Internet Service Providers (ISPs). A large portion of South African Internet accesses is to repeatedly retrieve information from the same sources overseas. A cost-effective approach for ISPs is to cache information in local traffic servers. Highly dependable cache servers are extremely important for ISPs.

Supported by South African National Research Foundation and a local ISP, we have been working on developing a dependable distributed system for Internet applications since 1996. The distributed system uses software and hardware replication to support dependable Internet applications. Performance, reliability, availability and fail-safe properties of the system are main attributes to be studied. The design of the distributed system was given in [3, 4]. In a number of related projects in our research group, the verification and modeling of core parts of this system were outlined in [5]. Representation of firewall rule-base and its BDD (binary decision diagram) implementation were investigated in [6]. Availability, reliability and risk analysis were studied under Markov models in [7]. This work describes the prototype we recently implemented, presents and analyzes the throughput of data transfer measured on the system.

In the next section, we outline the design of the system and discuss the fault-tolerant protocols aimed at improving performance, availability and reliability. Section 3 then presents the experiment system and scripts that define experimental trials. Section 4 reports and analyzes the results from these experimental trials. Finally, section 5 concludes the paper.

## **2. The System Design**

The overall system design we developed over the past years is shown in Figure 1. A number of computer nodes form a distributed system that runs a variety of applications. Each application may have different numbers of replicas, depending on their criticality. In this hypothetical example, the telnet application has three copies, the firewall has two copies and the mail and web have only one copy each. These nodes are connected to three Ethernet networks. One is used for communicating with the Internet, one is used for communicating with the Intranet, and one is used for administrative communications.

The fault tolerance is implemented by fault-tolerant protocols running in the distributed system. Take the firewall as an example. A request sent from the Internet will be doubled to two firewall copies. The comparison protocol will compare the decision of the two firewalls. The packets accepted by the comparison protocol will be sent to the Intranet. The voting protocol works in the similar way except that acceptance is based on the majority-voting scheme.

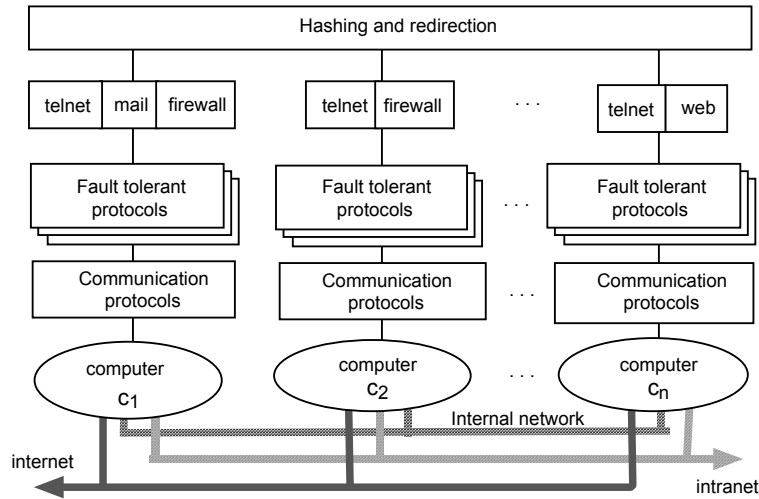


Figure 1. The system Overview

### 2.1. Task Allocation

A dynamic allocation algorithm is used in the system to distribute the workload among the nodes. The allocation algorithm takes current node's availability and load balance into account. Task allocation is represented by a bipartite graph with one set of vertices representing computer nodes and the other set of vertices representing tasks. An edge between a task vertex and a node vertex indicates an allocation of the task to the node. The graph representation of the hypothetical task allocation is given in Figure 2.

The allocation is dynamic. When a node becomes faulty, its tasks will be reallocated to other nodes by the reconfiguration protocol. For example, if  $c_1$  is faulty, the firewall and mail applications on the node can be transferred to, say,  $c_n$ , and the telnet application can be transferred to another working node, say,  $c_3$ .

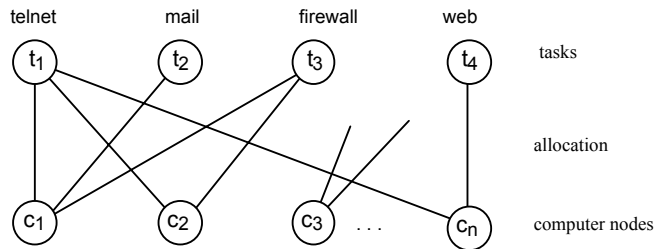


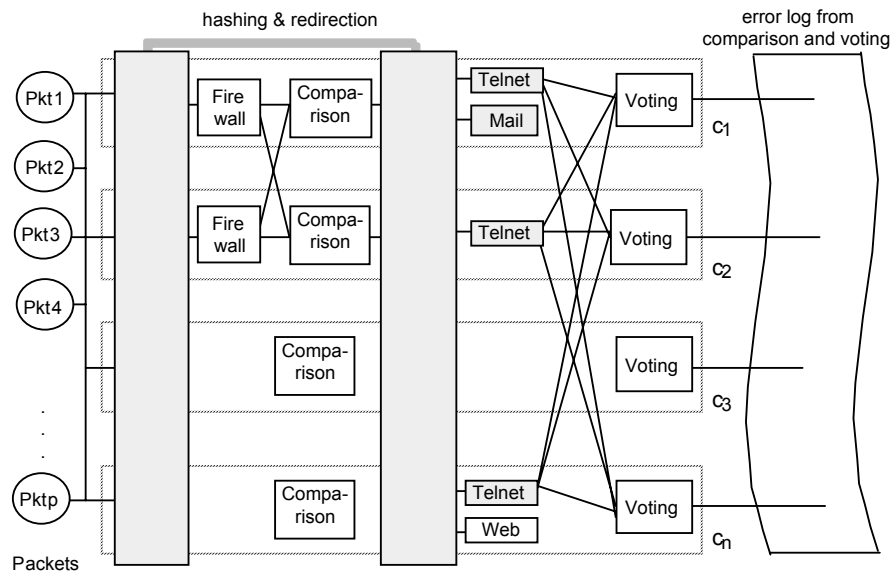
Figure 2. Task allocation

### 2.2. The Hashing, Voting and Comparison Protocols

A request coming from the Internet is first received by the hashing and redirection protocol, which duplicates the requests and redirects them to the firewalls. The firewalls either reject the requests or accept them by sending back to the redirector. The redirector then multiplies the requests,

if necessary, and forwards them to the relevant nodes where the requesting applications are running. In the case of telnet, the requests will be forwarded to three nodes where requests will then be handled. The results from three nodes will be voted and the majority be used as the final result.

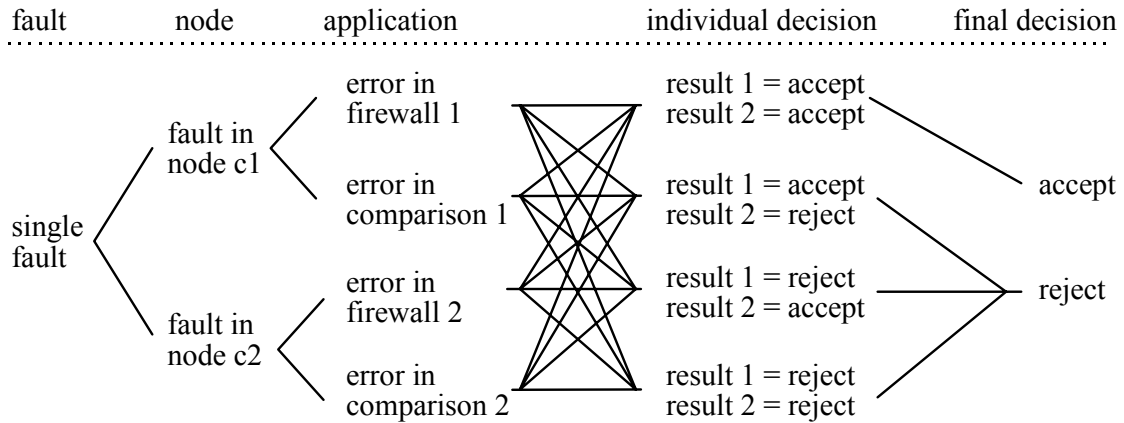
A typical single-fault-tolerant technique is the Triple Modular Redundancy (TMR) where a task is running on three nodes and the results are voted. Applications running under triple modular redundancy with voting can tolerate any single node fault without being immediately interrupted. That is, the voting protocol supports highly reliable applications. In order to tolerate possible faults during voting, a voting protocol with three copies of voting entities running on three nodes is used. For the duplicate tasks, general single-fault tolerance is not possible. It can only ensure fail-safe requirement [1, 8, 9, 10]. Figure 3 illustrates a scenario where packets are first checked by the duplicate firewalls, and then redirected to the relevant applications. In this example, the comparison protocol is only used by the firewall, and the voting protocol is only used by the telnet. Generally, they are running on all nodes and may be used by other applications where replicas are running.



**Figure 3. Packets are redirected to relevant applications**

Let's consider the duplicate firewalls. If both firewall copies agree with each other, the decision (accept or reject) is clear. If they disagree, i.e., one copy accepts while the other rejects a request, an error must have occurred in one of the firewall copies or comparison copies. A safe decision (without jeopardizing the security) is to reject the request. Figure 4 illustrates the impact of a single fault and the safe decision from the comparison protocol.

The advantage of using duplicate firewalls over triple redundancy is the lower cost. The drawback of duplication is that a request that should be accepted may be rejected due to a single fault. However, no request that should be rejected would be accepted even if a single fault occurred in any node at any time.



**Figure 4. Impact of fault and fail-safe decision**

### 2.3. Reconfiguration Protocol

The objective of the reconfiguration protocol is to transfer critical tasks on a faulty node to another working node. A critical task is a task that has at least two copies running on different nodes. The reconfiguration is implemented by a mobile agent system, which migrates the task from a working copy to another node to replace the copy on the faulty node.

Fault diagnosis is based on the syndrome table obtained from the error logs maintained by the comparison and voting protocols. In the comparison protocol, if a *disagreement* is detected by a comparison entity, a fault must have occurred in one of the two nodes. One error-point will be added to both nodes. In the voting protocol, if a node's decision is different from the majority, two error-points will be added to the node. The number of error-points related to a node in a given period of time is its error-frequency.

A fault can be considered as transient or permanent, depending on how frequently the fault occurs. Obviously, the higher the frequency is, the more likely is that the computer node has a permanent fault. The threshold to consider whether a computer is permanently faulty can be calculated based on the acceptable level of risk and the failure probability of computer nodes.

If a permanent fault is detected, reconfiguration will take place immediately. As the result, the tasks on a faulty node will be recreated other working nodes.

## 3. The Experiment System

In this section we present the prototype of the distributed system we recently developed. This prototype doesn't implement the entire design described in section 2. It focuses on the redundant firewall subsystem and its performance. The goals are to prove the concepts of our dependable distributed system design and to measure the performance of the system under different number of nodes and different node performance.

The experiment system we implemented is shown in Figure 5. The node "*internet*" (*client*) and *net1* simulate the Internet that sends requests to the Intranet. The node "*intranet*" (*server*) and *net2*

simulate the Intranet. The *node1* to *noden* simulate the core part of the dependable distributed system. Currently, only a redirector and a firewall system are implemented. The administrative network (*net3*) and an administrative node "*admin*" are used to configure, start and control all nodes in the system remotely and record their behaviors and performance.

The hardware used for each node (except the admin node) of the prototype is a diskless Pentium 133 machine, with 48 MB of RAM, and three 10Mb/s Ethernet network cards. The operating system used is a slightly customized floppy disk distribution of Linux 2.0.37. In the following subsections we describe the components of the system.

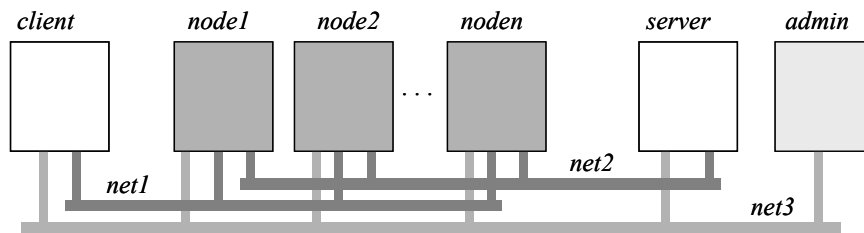


Figure 5. Experiment system

### 3.1. Filtering and Redirection Mechanisms

The Linux kernel has a built-in IP filtering mechanism, which allows an ordinary Linux host to route IP packets between two or more networks, selectively accepting, denying or rejecting them based on the configurable sequence of rules which comprises the access control list and the decision-making part of the firewall. An accept action routes the packet to its outgoing network interface, a deny action drops the packet silently, and a reject action drops the packet and sends an ICMP Destination Unreachable message to the sender as notification. We make use of this mechanism to implement our firewall. The redirection mechanism attempts to spread the load of packet filtering evenly over all the firewall nodes, by rapidly switching the route from client to server through each of the firewall nodes in turn in a round-robin fashion, while sending packets along this cycling route. A similar but independent mechanism cycles the route from server to client while packets are sent back. Each packet is currently only sent to one firewall node (simplex mode), but could potentially be extended to sending each packet to two firewall nodes for comparison purposes (fail-safe mode).

### 3.2. Traffic Generation

The client node, simulated by internal generators, and server node, simulated by an external generator, in the experimental system together for the traffic generator which is a distributed mechanism for recreating the traffic recorded in a *tcpdump* packet trace. It is assumed that if all instances of the generator on their separate nodes are supplied with the same packet trace and the same configuration file, then they will be able to cooperate to produce an approximate reproduction of the traffic recorded in the packet trace. In other words, the traffic generator creates an environment in which the *tcpdump* packet trace is converted into a collection of approximate

descriptions of the connections which originally produced it, and in which the client and/or server sides of these connections are approximately recreated.

Every complete connection in the trace is assigned one generator node to host its client side and one generator node to host its server side. The two nodes may, but need not, be the same. This assignment is done according to a map contained in a map-file, which deterministically maps any ip address in the trace output into the collection of all ip addresses of traffic generator nodes, attempting to spread the workload evenly across its range. The recorded server port addresses are also mapped into existing ports our system, which is the collection of ports being listened on.

### **3.3. Scaling the Firewall**

To evaluate the performance (throughput) of the distributed firewall system under different configurations, we implemented a scaling mechanism that can scale the number of nodes and the performance of each node.

Scaling the number of nodes across experiment trials was effected simply by using some fixed subset of the actually available nodes for each trial. In order to effect a desired node performance scale, a handicap factor was specified for each node, which is simply an integral factor by which the node's ability to function is reduced.

The actual consequence of specifying a handicap factor  $H$  for a given node in our implementation was simply to cause each rule to be replicated  $H$  times in its place in the node's access control list. Since every rule tested for a given packet (except the last) is repeated  $H$  times, the amount of work done (look-up time) per packet is approximately  $H$  times the amount of work that would be done for an unhandicapped node, and the node appears to function at a performance scale  $1 = H$  times the normal. In effect, scaling up of node performance is faked by scaling it down, and the trials in fact simulate the situation where the generators and the networks are as much faster than the firewall nodes as the handicap factor being used.

Obviously, setting up the system in this way limits both the maximum number of nodes and the maximum performance of any node across any set of experiment trials.

## **4. Results and Discussion**

We now present the collected data. To review definitions we used in our analysis, the availability score is the proportion of connection attempts which were successful; the reliability score is the proportion of successfully opened connections which also successfully closed; the throughput score is the average rate at which data was transferred between communication endpoints; and the duration (or total latency) score is the average time required for a packet in a given set of packets to be transferred from one communication endpoint to the other. The experimental system is currently unable to measure availability and reliability appropriately: the major effect that is measured is a property of the traffic generator, not the firewall system, as desired. So these data have been omitted. Due to the space limit, the data and analysis for duration have also been omitted. Thus we will focus on presenting and attempting to explain the results collected for throughput in this paper.

## 4.1. Experiment Configuration

The following configuration of the Experiment was used to produce the data that are presented in the next section.

The trace was configured to use *tcpdump* output recorded on one of the local subnets in the school of computer science at the University of the Witwatersrand. An extract recording 5000 packets belonging to non-local connections (connections where one endpoint was not on the local subnet) were extracted from this output.

The generator list was configured for all trials to hold precisely two host addresses, one corresponding to the internal generator, and one to the external generator.

All addresses on our local subnet were mapped to the internal generator, and all others were mapped to the external generator. Hence the reason for removing local-to-local addresses from the original packet trace: they would have mapped to extraneous loop back connections on the internal generator, and wasted space in the packet trace without contributing towards the filtering load on the firewall. Such space was at a premium because the packet trace happened to be downloaded into a small RAM disk on each diskless generator node. With hindsight, using NFS from the admin node over the admin network would have been a better choice, and would have allowed a much larger trace to be used.

The port list was configured to list eight server ports, which both generators listened on for all incoming connections, and to which the server ports of all connections listed in the trace were mapped.

The running of trials was split into two parts according to the number of router nodes involved: trials for 1 to 2 nodes were run on one system, and trials for 3 to 8 nodes were run on another. Every effort was made to ensure that the hardware on the two systems was alike, but such anomalies in experiment execution must be mentioned.

The handicap factors 768, 640, 512, 384, 256, and 128 were used. This choice was based on trial and error, aiming to trade the time required per trial to set up the access control list, for a noticeable effect on the throughput rate. These factors correspond to nodes of performance levels 1.0, 1.2, 1.5, 2.0, 3.0, and 6.0 times some base performance level -- a reasonable representation of the performance levels of computers found in a series made at one time by a typical manufacturer.

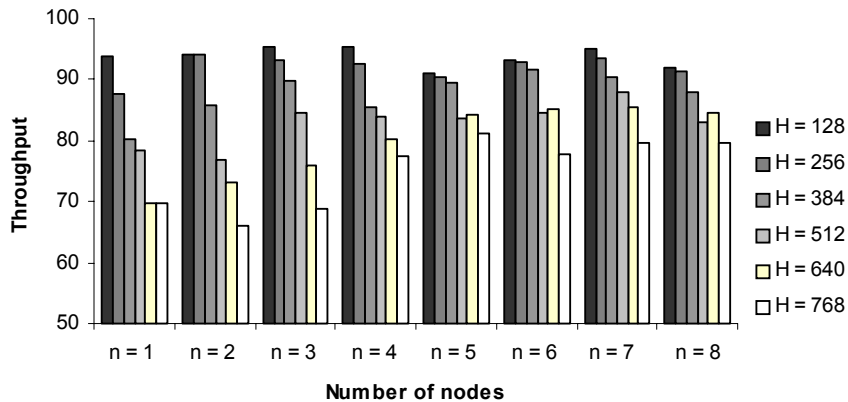
Five trials were performed for each of these combinations of handicap factor and number of router nodes.

## 4.2. Throughput Measured

The overall throughput mean for each configuration is presented from two different perspectives: Figure 6 emphasizes the effect of varying the handicap factor, while Figure 7 emphasizes the effect of varying the number of nodes.

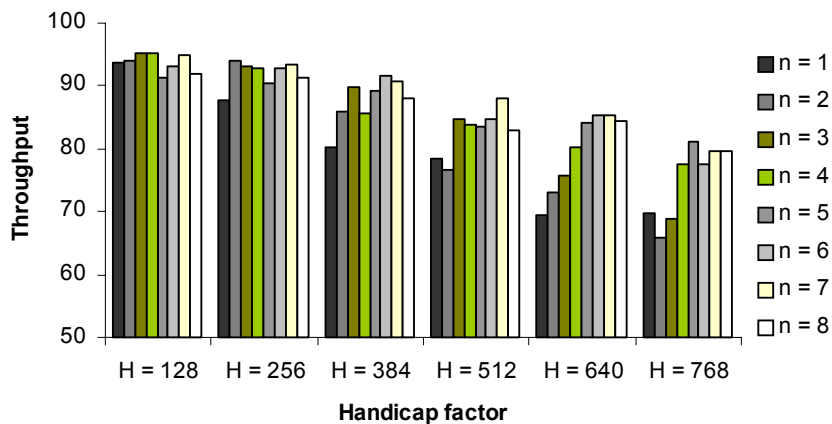
Let's take a detailed look at Figure 6. There are eight sets of data, corresponding to the number of nodes  $n = 1, 2, 3, 4, 5, 6, 7,$  and  $8$ , respectively. There are 6 elements in each set of data, corresponding to the handicap factors  $H = 128, 256, 384, 512, 640,$  and  $768$ , respectively. For  $n = 1$ , the throughput mean decreases clearly with the increasing handicap factor, which means that the

amount of data transferred between the two endpoint machines drops when node performance decreases. This trend can be observed throughout all eight sets of data. For increasing number of nodes  $n = 2, 3, \dots, 8$ , we can still see the amount of data transferred drops when node performance decreases, though it is less and less obvious. This observation means that data transfer rate is less sensitive to the node performance in multi-node distributed systems.



**Figure 6. Effect of varying handicap factor on throughput for each fixed number of nodes**

For the data collected in Figure 7, handicap factor = 128 and 256 show no clear indication that throughput increases or decreases with increasing number of nodes. However, for handicap factor = 384, 512, 640, and 768, we can see that throughput tends to increase with increasing number of nodes.



**Figure 7. Effect of varying number of nodes on throughput for each fixed handicap factor**

## 5 Result Analysis

In this section we demonstrate how the data we collected can be used in supporting the development of our dependable distributed system and present methods that can be used to extend the results beyond the scaling ranges we applied in our experiment.

### 5.1 Fraction of Enhancement

Through a careful observation of Figures 6 and 7 we can see that the throughput decreases, but not proportionally, with the increasing handicap factors. This is because the handicap factor is not applied to the entire system, but applied to the part responsible for the rule-base look-up. We believed that the rule-base look-up time was the major part of the firewall delay. With the experimental data collected, we are now in the position to check whether our assumption is correct. The theory we use is the Amdahl's law [11], which provides an accurate way to calculate the impact of components to the overall system. Assume we have improved a part of the system. The proportion of the improved part is denoted by  $Fraction_{enhanced}$  and the speedup obtained for the improved part is denoted by  $Speedup_{enhanced}$ , then the overall speedup is [11]:

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

As discussed in the previous sections, we simulated the varying performance of nodes by adding a handicap factor in the rule-base look-up part of a firewall node. For example, by halving the handicap factor, we actually double the speed of the fraction of the system that performs the look-up operation (*speedup of enhanced part*). This *speedup* corresponds to  $Speedup_{enhanced}$  in Amdahl's Law. Furthermore, we can obtain the  $Speedup_{overall}$  from the data collected. Then, according to Amdahl's Law, we can find the  $Fraction_{enhanced}$ , the portion in the system that is affected by the handicap factor. In order to obtain the overall speedup, we present in Table 1 the throughputs that plot part of Figures 6 and 7, and the overall speedup.

<i>mean</i>	<i>n = 1</i>	<i>n = 2</i>	<i>n = 3</i>	<i>n = 4</i>	<i>n = 5</i>	<i>n = 6</i>	<i>n = 7</i>	<i>n = 8</i>
<i>H = 512</i>	78.36	76.75	84.62	83.83	83.65	84.68	88.07	82.90
<i>H = 256</i>	87.79	94.11	93.25	92.67	90.56	92.86	93.51	91.40
<i>H = 128</i>	93.68	94.13	95.25	95.33	91.17	93.20	95.04	92.02
<i>speedup</i>	<i>1.094</i>	<i>1.113</i>	<i>1.062</i>	<i>1.067</i>	<i>1.045</i>	<i>1.05</i>	<i>1.039</i>	<i>1.055</i>

**Table 1** Throughput mean and speedup for  $n$  node configuration at handicap factor  $H$

According to this table, we can obtain:

$Speedup_{enhanced} = 2$ , since the handicap factors halved from 512 to 256, and from 256 to 128;

$Speedup_{overall} = 1.066$  by averaging the speedups in the table.

Then we can calculate the fraction of enhancement  $Fraction_{enhanced}$  as follows.

$1.066 = 1 / ((1 - Fraction_{enhanced}) + Fraction_{enhanced} / 2)$ , or

$$Fraction_{enhanced} = (1 - 1/1.066) * 2 = 0.124 = 12.4\%$$

That is, the fraction of the system that is affected by the handicap factor corresponds to 12.4% of the entire system that contributed to the throughput measurement.

These experiments clearly indicate that rule-base look-up time, while significant, is not the dominant cause of delay that impacts the measured firewall performance.

The most likely dominant cause of delay is incidental computation overhead at the communication endpoints. Specifically, the logging mechanism (used to record, amongst other things, the sending or receipt of every packet), was implemented using synchronous I/O which blocked the generator component wishing to record a log message until that message was actually logged.

Thus, we seem to have a case of the act of measuring overwhelming the actual effect we wished to measure. However, the situation is not irredeemable, since an alternative logging implementation (such as queuing pending log messages) could substantially reduce this incidental overhead.

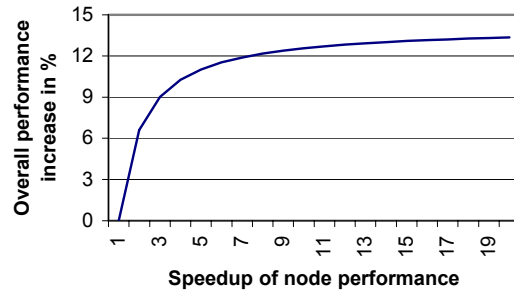
## **5.2 Impact of the Node Performance and Node Number**

The task of developing the experimental system from scratch to its current (still immature) state involved dealing with significant technical difficulties and blind alleys. The data collected so far do not provide a clear picture of the degree to which the system's performance can be scaled by increasing the performance or number of firewall nodes. In this section we discuss how we extend our results beyond the data ranges applied in our experiment system.

First, we discuss how node performance impacts the system overall performance when we scale node performance up beyond the range we studied in the experiment.

We assume that the computing overhead of a node is mainly from the rule base looking-up. According to the analysis in section 5.1, we know that the look-up time constitutes 12.4% of the system's overall time. Increasing the node performance by a factor  $p$  only decreases the time taken by look-up, but does not decrease the communication or endpoint processing times. Again, the overall performance enhancement can be calculated according to Amdahl's Law. Figure 8 shows the overall performance increase in percentage when the node performance is scaled from 1 to 20 times.

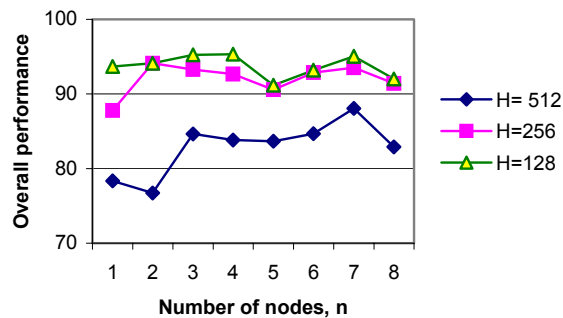
Now, we discuss how the number of nodes impacts the system overall performance when we scale node number up beyond the range we studied in the experiment.



**Figure 8 Scaling node performances**

Increasing the number of nodes has a much more complicated impact on the system, because the speedup depends on: (1) the level of workload imposed; (2) the degree of uniformity with which the redirector spreads this workload over the available firewall nodes; and (3) the level of communication overhead required for the firewall nodes to cooperate. Currently there is no such communication overhead, but some will be introduced by implementing reconfiguration, comparison, or voting protocols, and some may be introduced by alternative redirection mechanisms. However, this impact is well represented in the data we present in figure 6 and table 1. Figure 9 plots the overall performance when the number of nodes  $n$  increases from 1 to 8 regarding three different handicap factors, respectively.

It can be seen that increasing number of nodes doesn't clearly indicate an increase of the overall performance. Before we can model the impact of each of the abovementioned factors on the overall performance as the number of nodes increases, we need to do further experimentation (1) to remove anomalous influences like endpoint processing overhead, and (2) to implement more of the theoretical system described in section 2.



**Figure 9 Impact of the number of nodes on overall performance**

## 6. Conclusions and Future Work

We outlined the design of a dependable distributed system and the prototype we recently implemented. Emphasis was put on the experiment design and simulation of throughput. We have presented a summary of the data obtained from this simulation, and attempted to explain it. Using a part of the experimental data we showed that the firewall rule-base look-up part, which we believed to be the dominant part to the system performance, counted only 12.4% to the overall system.

We used Amdahl's Law to estimate the trend of overall performance when the performance of nodes is further scaled up. We didn't have sufficient data to scale the impact of the increasing number of nodes on the overall performance.

Although the size of our prototype and the simulation is relatively small, it does give us important experimental results that show the strength and weakness of our system. Our immediate future work is to develop the experimental system to simulate more of the theoretical system presented, and to measure its behavior more accurately. Developing models of the impacts of workload level, degree of load balance and firewall node communication overhead is obviously an important step towards this effort. Including other computational overheads, e.g., operating system and protocol entity overhead, may also give us more accurate simulation results.

The experiment system is a milestone in the development of our dependable distributed system. It will facilitate us in experimenting on various techniques and ideas we proposed in our research. It will be used in other related projects conducted in our research group too, including firewall rule-base design, model checking, system verification and Markov modeling projects as reported in [5, 6, 7].

## Acknowledgement

This project was funded by the South African National Research Foundation (NRF), grant no. 2047353 and by the University of the Witwatersrand's Research Programme for Highly Dependable Systems (PHDS).

## References

- [1] J. -C. Laprie, "Dependable computing and fault tolerance: Concept and terminology", *IEEE 15th Annual int'l symposium on fault-tolerant computing (FTCS-15)*, Michigan, June 1985, pp. 1 - 11.
- [2] N. Bowen, D. Sturman, T. Liu, "Towards continuous availability of Internet services through availability domains", *the International conference on dependable systems and networks*, New York, June 2000, pp. 559 - 566.
- [3] Y. Chen, "On development of a dependable distributed system", *Proc. of the 1998 IFIP International Workshop on Dependable Computing and its Applications*, Johannesburg, January 1998, pp. 83 - 96.
- [4] R.I. Mateer; and Y. Chen, "Highly-Available Firewall Service using Virtual Redirectors", *Technical Report TR-Wits-CS-1999-11*, August 1999. [www.cs.wits.ac.za/research/pubs.html](http://www.cs.wits.ac.za/research/pubs.html)

- [5] Y. Chen, V. Galpin, S. Hazelhurst, R. Mateer, and C. Mueller, "Modelling software development of a decentralised virtual service redirector for Internet applications", *The 7th IEEE Workshop on Future Trends of Distributed Computing Systems*, Cape Town, December 1999, pp.235 - 241.
- [6] S. Hazelhurst, A. Attar, R. Sinnappan. "Algorithms for Improving the Dependability of Firewall and Filter Rule Lists", Workshop on Dependability of IP Applications, Platforms and Networks. in Proc. Int Conf on Dependable Systems and Network, IEEE Computer Society Press, New York, June 2000.
- [7] Y. Chen, Z. He, "Dependability Modeling of the Service Redirector in the Internet Applications", in Proc. *the 5th International Symposium on Autonomous Decentralized Systems*, Dallas, March 2001, pp. 176-183.
- [8] D. P. Siewiorek, R. S. Swarz, "Reliable computer systems", Burlington, Mass. Digital Press, 1992
- [9] Y. Chen, T. Chen: "Implementing fault tolerance via modular redundancy with comparison", *IEEE Trans. on Reliability*, Vol. 39, No. 2, 1990, pp. 217-225.
- [10] K. Echtle, "Fault-masking with reduced redundant communication", *IEEE 16th Annual International Symposium on Fault-Tolerant Computing (FTCS-16)*, June 1986, pp. 178 - 183.
- [11] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities", *Proceedings AFIPS Spring Joint Computer Conference*, Atlantic City, April 1967, pp. 483 - 485.

## Biographies

(Photo)

Yinong CHEN received his doctoral degree from the University of Karlsruhe, Germany, in 1993. He did postdoctoral research in Germany in 1994 and in France in 1995 and 1996. From 1994 to 2000, he was a faculty member in the School of Computer Science and was the leader of the Research Programme for Highly Dependable Systems (PHDS) at the University of the Witwatersrand, Johannesburg. He joined the Computer Science and Engineering department at Arizona State University in 2001. Dr. Chen's research interests are in the areas of dependable computing, performance and reliability evaluation of distributed systems and computer networks. He has published a number of books and over 50 technical papers in these areas.

(Photo)

Roger Mateer received the B.Sc. (Hons.) degree in 1996 from the School of Computer Science at the University of the Witwatersrand, Johannesburg, South Africa. He is currently working towards the M.Sc. degree in their Programme for Highly Dependable Systems (PHDS) research group.