Risky Business: Motivations for Markets in Programmable Networks

Ian Wakeman, David Ellis, Tim Owen, Julian Rathke, and Des Watson

University of Sussex

Abstract. We believe that the problems of safety, security and resource usage combine to make it unlikely that programmable networks will ever be viable without mechanisms to transfer risk from the platform provider to the user and the programmer. However, we have well established mechanisms for managing risk - markets. In this paper we argue for the establishment of markets to manage the risk in running a piece of software and to ensure that the risk is reflected on all the stakeholders.
We describe a strawman architecture for third party computation in the programmable network. Within this architecture, we identify two major novel features:- Dynamic price setting, and a reputation service. We investigate the feasibility of these features and provide evidence that a practical system can indeed be built.
Our contributions are in the argument for markets providing a risk management mechanism for programmable networks, the development of an economic model showing incentives for developing better software, and in the first analysis of a real transaction graph for reputation systems from an Internet commerce site.

1 Introduction

The core problem being tackled in active networking is the same as for any mobile code approach - how can we allow computation to take place on behalf of some third party, yet be sure with high probability that the outcome of the computation will not be harmful to the local machine or environment? There have been attempts to use technologies from the programming language design community, such as safe typing [1,2] and namespace protection [3,4], to design safe programming languages for active networks and service. Systems have been built to control the execution environment of third party programs using and extending the techniques from operating systems for controlling scheduling, memory usage and general access control [5,6,7]. Yet another approach for component based programs is to check a priori that the program composition graph is acceptable [8,9].

Yet despite this panoply of technical solutions, there is no widespread deployment of platforms supporting third party computation for networks or anything else[1]. We would argue that this situation exists because there is no benefit to the

[1] One could argue that PlanetLab is a distributed programming platform, yet the users have to pay by providing computers to use the system. This is therefore evidence that the market has to be involved.

N. Wakamiya et al. (Eds.): IWAN 2003, LNCS 2982, pp. 266–279, 2004.

manager of any platform in supporting third party computation, and further, that there is no way for the manager to reflect the risk undertaken by hosting the platform.

The risks to the platform from running software on behalf of someone else are that the software, whether through malice or accident, will deny resources to other users, potentially leading to crashes of their programs and the platform. This will result in the platform being perceived as *unsafe* by the affected users, and they will become reluctant to run software on the platform again. In the worst case, the software may open up the platform owner to liability for loss and damage caused to other users. Despite the increasing reliability and safety of operating systems, there remains a probability that software can deny resources to other users.

Instead we propose that the manager of the computing platform has to be appropriately reimbursed for the risk they are taking in hosting the third party computation, and that the amount of reimbursement should be decided through market mechanisms. The remainder of this paper provides an overview of market based scheduling of programs, and outline a strawman architecture for using markets to calculate the riskiness of running a given program. We then demonstrate the feasibility of two novel aspects of this strawman architecture; the spreading of risk through market mechanisms and the use of a distributed reputation system. We conclude with a discussion of the possibilities for future work.

2 Market Based Computational Systems

Market mechanisms for scheduling computation are not novel. There have been systems designed for the grid and other systems, such as Condor [10], Spawn [11], the Java Market [12] and Nimrod-G [13]. Surveys and taxonomies of the various approaches can be found in [14] and [15]. Market mechanisms appear in many other places, most notably for controlling congestion [16,17,18].

Negotiating which buyers buy from which sellers is a fundamental choice in the design of any market-based system. Given that our motivation for a market based system is for sellers to set prices to guard against the various risks of the different buyers, we consider mechanisms which allow sellers to set prices. A more commonly adopted approach is to use a tender/contract approach, where buyers issue a tender for contract, to which sellers respond with a bid based upon their estimate of their costs and the value of their service to the seller. Buyers can then set a buyer-specific price, resulting in different buyers receiving different prices for the same product.

It is a well-established economic principle that differential pricing with large fixed costs can be very efficient [19]. By selling each unit at the highest price each individual is willing to pay, the producer maximises profit and increases the number of consumers who are able to buy the product. Telecommunications and airlines are good examples of real markets where this happens. There are two main arguments against differential pricing in reality:

1. How can sellers get enough information to set prices appropriately?
2. What happens if buyers resell the product?

If differential pricing is to be effective, it behooves the infrastructure designers to ensure that there is sufficient information for sellers to match prices to customers' perceived value. We propose the use of a distributed database containing the outcomes of each transaction, which can then be used to determine the reputation of each seller, customer and program. The reputation of the seller and program can be used to adjust prices to account not only for the expected resource usage, but also for the lost opportunity cost if the program misbehaves and affects other customers.

We would argue that reselling is not necessarily a bad thing. In our current design, when the contract between the seller and buyer is made, the right to process uses a cryptographic ticket tied to the identities of the seller, buyer and program. To transfer a ticket, the buyer has to chain the identity of the new ticket owner onto the ticket by re-signing the ticket to confirm the ticket has been transferred. The outcomes of running the program are then inserted into the reputation system under both the original buyer and the rebuyer. Any future price setting will use these outcomes to set price, so that if the reseller wishes to continue trading, they have to ensure that their customers don't adversely affect their reputation. By allowing trade in tickets bearing a right to process, the market can be made more efficient, since the brokers can accept the responsibility of matching prices to buyers, and the processor owners need only worry about the reputations of the resellers.

3 The Risk Compensation Architecture for Third Party Computation

To explore the design space for building an infrastructure to support risk compensated third party compensation, we produced a strawman architecture, based on rapid prototyping of the various system components. We assume that there are a small number of platforms at each locality which can have the necessary capabilities to run software on behalf of a set of customers. The software can run for a short duration, e.g. in providing rerouting of critical services during a Denial of Service attack, or of long duration, such as offering a game server. We assume that the service location problem is solved elsewhere. The customer either directly interacts with the system, or a software agent [20] acts on the customer's behalf.

The sequence of interactions is illustrated in Figure 1.

1. The customer makes a request for a price to run an identified piece of software. The customer and software are named through a self-certifying naming system, e.g. by the software using a tree of hashes [21] and identifying the customer through a hash of their public key.
2. The processors look up the customer's entries in the reputation service database. All relevant records are returned.

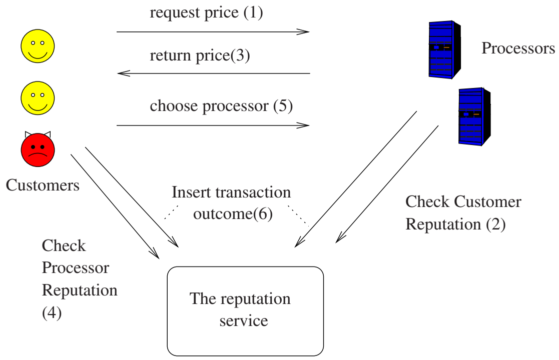

Fig. 1. The Strawman Risk Compensation Architecture for Third Party Computation

3. The processor calculates and returns a price to the customer, using the entries from the reputation service.
4. The customers check the processors' reputations.
5. The customers make a judgment on which is the best processor for their job and request execution.
6. After execution, the processor and customer sign an xml certificate describing the memory and processor cycles used, and whether the jobs generated uncaught signals or exceptions. They insert the certificate into the reputation service.

Many of the pieces of this architecture have been built and used together before, e.g. in the systems surveyed in [14,15]. Cryptographic requests and tickets are easily built from standard security protocols and digital signatures [22]. Mobile code systems have been shown to work in many projects, such as in our previous work on SafetyNet [23].

The novel features of the strawman design are in calculating an offer price based upon the reputation of the customer, and in the use of a reputation service. In our initial prototype, the setting of an offer price would often lead to wild fluctuations in the offered price. To understand this behaviour further and to develop more stable pricing systems, we built further simulations described in Section 4.1. Our initial implementation of the reputation service was based upon a single mysql database. If the reputation service is to be practical, then it must scale in the number of entries and the number of users, be resistant to manipulation by malevolent users, and provide appropriate information to the users of the service. We undertook studies of existing reputation systems, and show how the results of these studies should guide the design of future reputation services.

4 A Simple Market Model

We base our model firmly in the microeconomic theory of oligopolies. We assume that each location has a small number of processors available, and that there are a set of customers who wish to use these machines. For each processor, there are zero marginal costs for supplying customers - there is purely a fixed cost for maintaining and running the processor. Each processor can set a customer specific price for the customer to run their software. Each processor is aware of the funds available to each customer, and has knowledge of the utility function of each customer - i.e. how the customer perceives *value*. Each customer is aware of the other customers' utility functions and can calculate the expected *load* upon each processor.

We model each processor as a simple M/M/1 queuing system. Thus for each customer i there is a load λ_{ij} placed upon the processor j. Each processor has a capacity μ_j. The total waiting and service time, T_j, from the processor is therefore:

$$T_j = \frac{1}{\mu_j - \sum^i \lambda_{ij}}$$

The customer cares about the response time from the processor (larger is worse), their own load placed upon the processor (larger is better) and the cost of using the processor (less is better). We follow a conventional economic approach and model the customer's utility function as:

$$u_i = \sum^j T_j^\alpha \lambda_{ij}^\beta (p_{ij}\lambda_{ij})^\gamma$$

If we set $\alpha = -1$,$\beta = 2$ and $\gamma = -1$, then we obtain[2]:

$$u_i = \sum^j \frac{\lambda_{ij}(\mu_j - \sum^k \lambda_{kj})}{p_{ij}}$$

where p_{ij} is the price offered by processor j to customer i per unit of service. We model the load setting game as follows: For each customer, the customer searches for the load vector to load on each processor that would maximise the customer's utility, subject to the constraint that their expenditure mustn't exceed their income I_i:

$$p_i.\lambda_i \leq I_i$$

This set of loads is their *best response* to the loads set by the other processors. This game is repeated until no processor can increase its income by adjusting its price vector. This is the *Nash Equilibrium* point of the of the customer utilities for a given price matrix by definition. It should be noted that the Nash equilibrium is guaranteed to exist since the utility function is continuous and concave in the the Euclidean space defined by the feasible set of loads [24]. We describe this theory more in the accompanying technical report [25].

[2] These values are chosen arbitrarily

We assume that the processors set prices, as in the oligopoly theory of Bertrand [26]. Processors will set prices so as to maximise their income. The income received by each processor π_j is simply the sum of the price offered to each customer multiplied by the load offered by that processor over all customers.

$$\pi_j = \sum^{i} p_{ij}\lambda_{ij}$$

We model risk by assuming that there is a failure rate associated with the software run by each customer, f_i. Our simulation runs through a set of discrete rounds, where the prices and associated loads are calculated for each round. The perceived failure rate of a processor is then the weighted sum of the failure rates of the software run upon the processors, weighted by how much load each customer places upon the processor. We modify the utility function of the customers to use the last seen failure rate in their utility calculation.

$$u_i = \sum^{j} \frac{\lambda_{ij}(\mu_j - \sum^{k} \lambda_{kj})}{p_{ij} f_j^2}$$

The analytic solution of the load setting game for the customers is derived in the Appendix.

In our model, the processors take turns setting prices that maximise their revenues from the loads which the customers have maximised. This game is repeated until no processor would increase its revenues by changing its prices. This is again a Nash equilibrium. In calculating the expected revenue, the processors not only maximise the revenue for the current round, but the expected future revenue based upon how their perceived failure rate will be modified.

We built a Matlab simulation of the above model and ran it through many different sets of experiments. As we discuss in the appendix, the discounting of future income will generally result in a complex equation to predict future income, and the price setting game is not guaranteed to converge on a single optimum point. Instead, we have found that the price setting may develop a *limit cycle* in which there are a cycle of best responses to each of the other processors' prices. We are attempting to develop the theory further to show that the game will either converge to a single point in the pricing space, or will develop a limit cycle and will never diverge. We have not discovered any combinations of settings where the prices diverge. To deal with this eventuality, our simulation terminates a round either when the price setting converges to a stable set of prices, or a limit cycle is detected over a number of iterations.

4.1 Case Studies and Simulations

We illustrate two of the more interesting cases below. In each of the scenarios, the processors discount 2 rounds into the future, calculating their expected income as a weighted sum of income, assuming that perceived failure rates change based on the expected customer load, and using the same price vector in the future. The customer and processor tournaments carry on until the change per round is less than a given small amount.

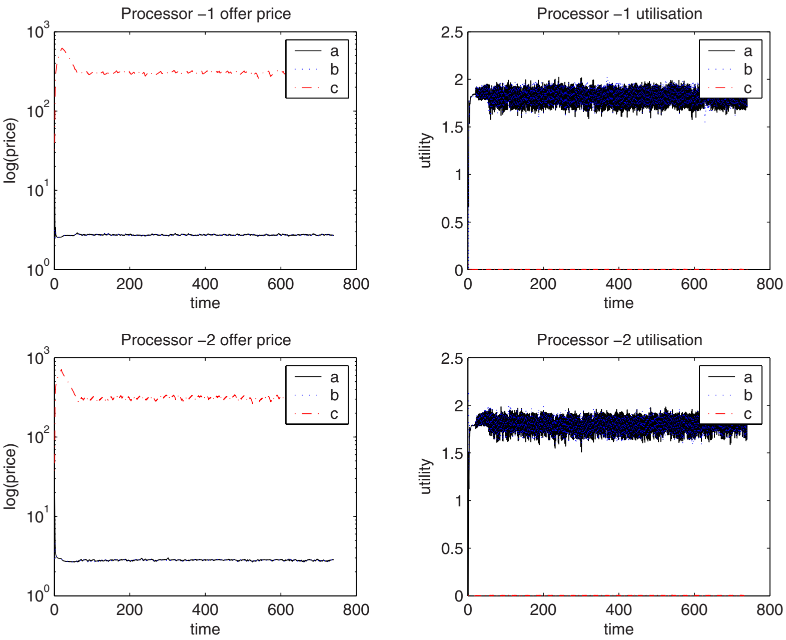

Fig. 2. Offered price for bad software

Bad *Software.* In this scenario, there are 3 customers wishing to run software on either of 2 equal processors. Customers 1 and 2 are *good* customers whose software doesn't crash the processor with income of 10 units each. Customer 3 has *bad* software which crashes the processor, disrupting the other jobs and a relatively low income of 1 unit. In Figure 2, notice how the price offered to customer 3 by either of the processors becomes so high that customer 3 cannot afford to offer very much load. Even with a relatively simplistic optimisation game, there are emergent behaviours which penalise bad software due to their detrimental effects on the reputation of the processors.

Bad *Software and Risk-Taking Processors.* In this scenario, there are 3 customers wishing to run software on 3 processors. Customers *a* and *b* run good software with a failure rate of 0.1 units, whilst customer *c* has bad software with a failure rate of 1 unit. Each of the customers has 10 units of income. Processors 1 and 2 have capacity of 10 units, whilst processor 3 has a capacity of 1 unit. In Figure 3, the processors serve two different market segments. Processors 1 and 2 become low failure rate processors mainly servicing the good customers, whilst the other processor becomes a relatively failure prone machine thus getting a large part of customer c's income. This illustrates that differential pricing can create niche markets.

We believe that a major advantage of using software reputation in setting the price for running a program will be in creating incentives for better software.

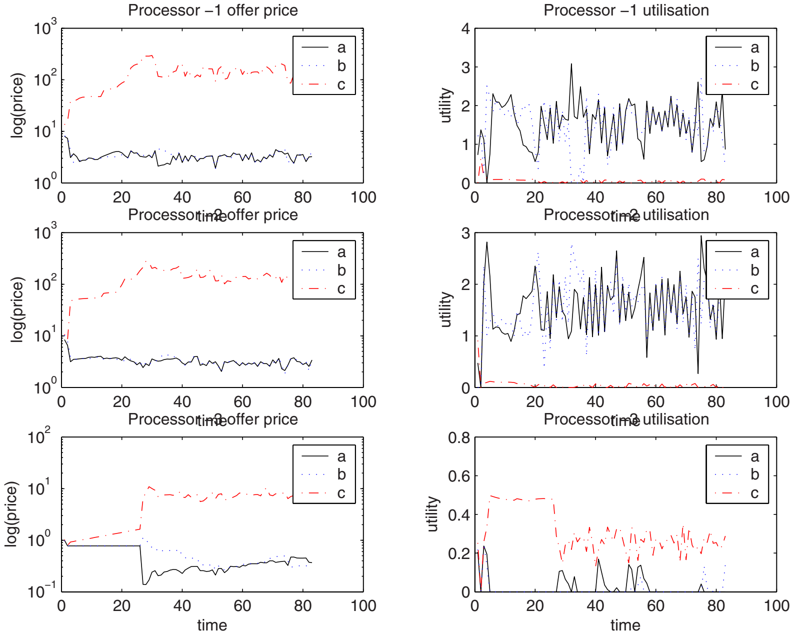

Fig. 3. The emergence of risk taking processors

If software has a lot of bugs, then its reputation will decrease and the price to run it will rise. Customers will then have an incentive to pressure the software developers for better software, either through economic pressure by reducing sales, or through exerting social pressure to reduce the number of bugs.

As program developers will feel more direct pressure to produce better software, then developers will have incentives to adopt technology which increases the safety of their programs. In this way software technology, such as type safe languages have better chance to compete in the market.

Segmentation of the market will provide separate resources for production code and for testing code. Free and best effort platforms may emerge, whose reputation and load will provide indicators as to whether they should be trusted.

Although our simulation modes indicate that differential pricing provides many good features, more work needs to be done yet. Our model only uses one approach to negotiating a price. There are many other possible mechanisms for negotiating prices, and the emerging area of mechanism design for the Internet and computational systems may provide better technology [27].

5 The Feasibility of a Distributed Reputation System

In our initial implementation of a reputation service, we had no data upon which to base our design. To ensure that our future work was tested upon real data, we analysed an existing reputation system in which products and services were bought by real people with real money. The Amazon.co.uk website provides a variety of commerce activities through its "Marketplace" and "Z-Shops". Each user of these services chooses a unique user id and receives an associated tag from Amazon. After each transaction, feedback is entered where the recommender gives a rating of between 0 and 5 to each recommendee. These recommendations can be inspected for each user.

The data was collected by using the web services available from Amazon.co.uk. We set up an initial set of seeds by collecting names through searching for a set of disparate items. The graph of recommendations was then followed to collect the set of links and entities. In all we collected 405,661 different nicknames, or *nodes.*

Each node has one or more *recommendations* going from it to other nodes, or one or more recommendations pointing at itself. Each recommendation consists of a numeric rating of the transaction, from 0 to 5, the date of the rating and a comment. We did not collect the comments.

From this collected data, we determined the proportion of nodes which have k neighbours, $P(k)$. We excluded multiple recommendations between the same nodes, and self-recommendations. These are plotted in the log/log graphs of Figures 4 and 5. We show the residuals from the fitted curves, which indicate that both the outbound and inbound $P(k)$ functions are better fitted by polynomials of degree 2 in the log/log curves, ie that the relationship between $P(k)$ and k can be modeled by:

$$P(k) \propto k^{f(k)}$$

If we examine the undirected graph formed by the recommendations, we find the graph to be connected, and the diameter of the graph i.e. the largest shortest path between two nodes, is measured to be 21 hops. The average shortest path between two nodes is 8.1.

5.1 Analysis of Reputation Services

As expected, the transaction graph exhibits the properties of a power law graph. Following Barabasi [28] the transaction graph will grow in the number of nodes and each node will exhibit preferential attachment. If we classify nodes as either predominantly buyers or sellers, then a buyer will tend to buy from existing sellers with established reputations, whilst incoming sellers will be used predominantly by existing buyers who are already well-connected. The low connectivity of the directed graph can be explained by the clustering of buyers and sellers around particular types of item, such as videos or books.

In the following discussion, we assume that the characteristics of the transaction graph described above can be generalised to other transactional based systems, such as the execution of a third party piece of software. In the absence of any other data, it is valid to criticise design approaches to reputation systems assuming that their transaction graph will show similar characteristics.

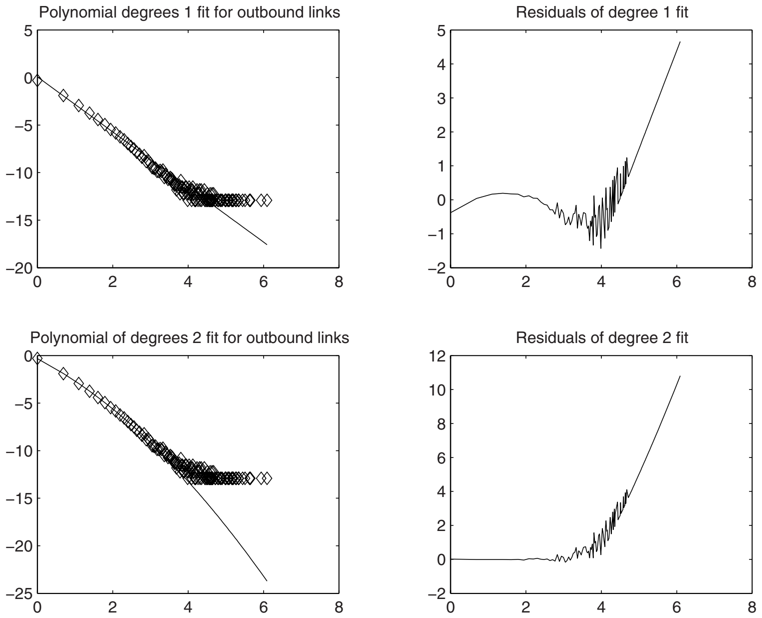

Fig. 4. Log/log plot of relative frequency of outbound links per node of the undirected graph

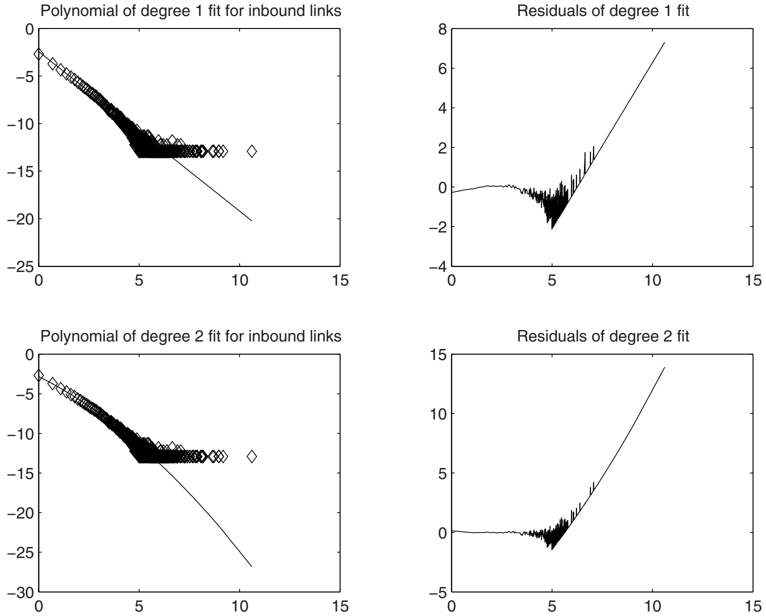

Fig. 5. Log/log plot of relative frequency of inbound links per node of the undirected graph

There are a number of peer to peer reputation systems which rely on polling trusted neighbours to find the reputation of the target system, hoping that they will have had some direct interaction with the target system such as [29,30]. The P2PRep system [30] is designed to operate in Gnutella file sharing systems and is typical of such an approach. To check the reputation of a server, the requesting node polls other local servents to get certificates detailing their interactions with the servers. If this technique is reused where the transaction graph is a power law graph, then there are many nodes which will have had few interactions with other nodes, and thus will be unlikely to receive any certificates. However, the localised nature of the cliques will mean that this technique is effective for highly connected nodes.

The Eigentrust system [31] is based on the distributed computation of the principal left eigenvector over a normalised version of the transaction graph, using a similar approach to the pagerank algorithm of Google. In the secure version discussed by the paper, a node's "trust" value is computed by a set of score managers. In each iteration, the score manager's collect the current trust value from each of the neighbours of the node, and return the results after calculating using the local column of the transaction graph matrix. If the transaction graph is a power law graph, this may result in local hot spots, and a high number of messages for some nodes, slowing down the computation. Since each change in the reputation graph may trigger a recomputation of reputation, the load may become unacceptable.

In making an assessment of a node in the transaction graph, it is necessary to both have the directly relevant information, and ways of assessing the reliability of the direct information. If this is not provided, then the direct information is susceptible to *sybil* based attacks, where the attacker inserts multiple entries into the database. We propose a multi-metric, multi-type approach, in which each transaction entry is accorded a type, so that users can check entries of relevant type, and the use of metrics to assess the reliability of each transaction entry.

Since the graph is a power law graph, it becomes feasible to search through the graph in a distributed fashion, since power law graphs are amenable to efficient distributed searching [32,33]. We are currently investigating and comparing distributed path finding techniques starting from trusted nodes as alternatives to metrics such as EigenTrust.

Any system should have high availability. The most effective solution may be to have a dedicated set of high availability servers, running either a DBMS, or a distributed storage system such as a distributed hash table e.g. Pastry [34]. The tendency to hot spots in the graph may make the DHT approach vulnerable to hot spots.

6 Future Work

We are currently building a distributed reputation system, providing a multimetric view of the transaction graph, using a distributed hash table as the underlying storage. We will be using the collected data as the basis for analysing performance.

We are working towards incorporating information about resource usage in the next generation of the SafetyNet language, and its associated runtime. We are hoping to incorporate both endogenous information from static analysis of the program and exogenous information from the transaction certificates. We will be looking to port the runtime to the XenoServer environment[35].

7 Conclusion

In this paper we have argued that the difficulty of managing risk is one of the major reasons that active networks and other third party computational infrastructures have not achieved widespread deployment. If risk is to be managed, programs must be priced according to the potential lost revenue from running a program which deters other customers from using the platform. We have described simulations which indicate that pricing according to risk produces a stable environment which provides market niches for different platforms, and incentives for good software. Finally, we have presented the first analysis of the transaction graphs of an ecommerce site, and used this to motivate the design principles required for a distributed reputation service. The results reported here indicate that it is feasible to develop a third party computational infrastructure.

References

1. Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, Wenyi Zhou, R. Dennis Rockwell, and Craig Partbridge. Smart packets: applying active networks to network management. *ACM Transactions on Computer Systems*, 18(1):67–88, 2000.
2. Dennis Volpano and Geoffrey Smith. Language issues in mobile program security. *Lecture Notes in Computer Science*, 1419, 1998.
3. D. Alexander. *Alien: A Generalized Computing Model of Active Networks*. PhD thesis, University of Pennsylvania, 1998.
4. Pankaj Kakkar, Michael Hicks, Jonathan T. Moore, and Carl A. Gunter. Specifying the PLAN networking programming language. In *Higher Order Operational Techniques in Semantics*, volume 26 of *Electronic Notes in Theoretical Computer Science*. Elsevier, September 1999.
5. Paul Menage. RCANE: A Resource Controlled Framework for Active Network Services. In *Proceedings of the First International Working Conference on Active Networks (IWAN '99)*, volume 1653, pages 25–36. Springer-Verlag, 1999.
6. Grzegorz Czajkowski and Thorsten von Eicken. JRes: A resource accounting interface for Java. In *Proceedings of OOPSLA'98*, pages 21–35.
7. Ian M. Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul T. Barham, David Evers, Robin Fairbairns, and Eoin Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal of Selected Areas in Communications*, 14(7):1280–1297, 1996.
8. Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The click modular router. In *ACM Symposium on Operating Systems Principles*, pages 217 – 231, Charleston, South Carolina, 1999. ACM Press.

9. Stefan Schmid, Tim Chart, Manolis Sifalakis, and Andrew Scott. Flexible, dynamic and scalable service composition for active routers. In *Proceedings of the 4th International Working Conference on Active Networks (IWAN'02)*, December 2002.
10. M. Litzkow, M. Livny, and M. Mutka. Condor — a hunter of idle workstations. In *Proceedings of the Eighth Conference on Distributed Computing Systems*, San Jose, California, June 1988.
11. Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *Software Engineering*, 18(2):103–117, 1992.
12. Yair Amir, Baruch Awerbuch, and Ryan S. Borgstrom. The java market: Transforming the internet into a metacomputer. Technical Report Technical Report CNDS-98-1, Johns Hopkins University, 1998.
13. R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *Proceedings of the 4th International Conference on High Performance Computing in the Asia Pacific Region*, May 2000.
14. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, May 2002.
15. D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. *Market-Based Control: A Paradigm for Distributed Resource Allocation*, chapter Economic models for allocating resources in computer systems. World Scientific, 1996.
16. R. J. Gibbens and F. P. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, 35:1969–1985, 1999.
17. Ron Cocchi, Scott Shenker, Deborah Estrin, and Lixia Zhang. Pricing in computer networks: Motivation, formulation, and example. *IEEE/ACM Transactions on Networking*, 1, December 1993.
18. Xi-Ren Cao, Hongxia Shen, Rodolfo Milito, and Patrica Wirth. Internet pricing with a game theoretical approach: Concepts and examples. *IEEE/ACM Transactions on Networking*, 10:208–216, 2002.
19. Hal Varian. Differential pricing and efficiency. *First Monday*, 1(2), August 1996.
20. Michael Wooldridge. *Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.
21. Kevin Fu, M. Frans Kaashoek, and David Mazières. Fast and secure distributed read-only file system. *Computer Systems*, 20(1):1–24, 2002.
22. Bruce Schneier. *Applied Cryptography: Protocols, Algorithms*. John Wiley & Sons, second edition, 1995.
23. Ian Wakeman, Alan Jeffrey, Tim Owen, and Damyan Pepper. Safetynet: A language-based approach to programmable networks. *Computer Networks and ISDN Systems*, 36,1, June 2001.
24. Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, 1991.
25. Ian Wakeman, David Ellis, Tim Owen, Julian Rathke, and Des Watson. Risky business: Motivations for markets in programmable networks. Computer science technical report, University of Sussex, 2003.
26. H. Gravelle and R Rees. *Microeconomics*. Longman, 1992.
27. Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, New York, 2002.

28. Barabasi, Albert, and Jeong. Mean-field theory for scale-free random networks. *Physica A*, 2272:173–187, 199.
29. Giorgos Zacharia, Alexandros Moukas, and Pattie Maes. Collaborative reputation systems in electronic marketplaces. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, 1999.
30. F. Cornelli, E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servents in a p2p network. In *Proc. of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, May 2002.
31. Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.
32. L. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in power-law networks. *Phys. Rev. E*, 64, 2001.
33. Beom Jun Kim, Chang No Yoon, Seung Kee Han, and Hawoong Jeong. Path finding strategies in scale-free networks. *Physical Review E*, 65(027103), 2002.
34. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.
35. Dickon Reed, Ian Pratt, Paul Menage, Stephen Early, and Neil Stratford. Xenoservers; accounted execution of untrusted code. In *IEEE Hot Topics in Operating Systems (HotOS) VII*, March 1999.