Management of Future Data Networks: An Approach Based on Bacterial Colony Behavior.

Chris Roadknight and Ian W. Marshall

BTexact Technologies.

B54/124, Adastral Park, Martlesham Heath, Ipswich, Suffolk, UK. IP5 7RE

christopher.roadknight@bt.com, ian.w.marshall@bt.com

Abstract.

Network complexity will increase dramatically over the next 5 years as will the amount of devices inhabiting these networks. Ad-hoc and active paradigms will make the already onerous task of network management increasingly problematic. An approach to managing such networks based on bacterial colony behaviour is discussed, offering innate abilities for essential tasks such as software proliferation, load balancing and differing but distinct qualities of service. Robustness to realistic request streams is also demonstrated using real world requests as a source of simulated network load. The 'hands off' element of the adaptive algorithm is a major asset for any configuration and optimisation task. This biologically inspired adaptive management solution could be the ideal approach to managing the behaviour of complex data networks of the future.

Index Terms.

Networks, Genetic Algorithms, Cellular Automata

I. Introduction

Services on future networks will become increasingly sophisticated and dynamic with escalating user demands and competencies. The networks will also become more flexible with the adoption of ad-hoc [19] and active networks [2, 13]. The users of the constantly evolving collection of hardware devices inhabiting these networks will also require much more support in configuring and managing these devices [22]. These trends are driven by the continued pervasion of computer hardware into all walks of life [10]. This progress will accelerate as the decreasing cost of networkable chipsets enables them to be inserted in cars, household devices, clothes and virtually any other product of value. Clearly the management solutions for future networks, both traditional core networks and networks of embedded hardware, must require significantly less manual intervention than at present. The premise that someday individuals or organisations will need assistance in managing these networks of devices is the basis of this research.

Any future network providing active services will be unbounded in both scale and function. The network hardware and software will be constantly expanded, upgraded, and re-dimensioned. Even the hardware itself could become programmable [26]. Given this, the fractal properties of the traffic providing a mixture of stochastic and deterministic load [25, 7] and the massive scale of future networks it is clear that conventional methods of control and management cannot apply and that adaptive methods of control are essential [30]. Artificial Life solutions provide the richest set of robust adaptive approaches to problem solving.

Any network of this kind will have much more scope for failure states. With many more options at many more points in the network the combinatorial possibilities for network state become intractable. In addition, the more dynamic nature of future networks makes any available state information temporally unstable and divergent.

The network over which most data is currently sent is both fixed and (in many ways) dumb. Fixed because the hardware devices are hardwired to other devices using electrical or optical data lines. The software is also mostly static, installed on one device and then used from that device. Dumb because much of the software used by the end user often resides on the end user's terminal or on the server from which he is requesting a service or data, thereby making the network itself little more than a bit routing environment.

Mobile code is already being run on many devices connected to the internet and code portability and re-use is one obvious facilitator of a future ‘devices everywhere’ environment. An active service is a program that is run on devices owned by network operators or network service providers (such as caches, mirrors conference controllers and firewalls), migrating software from that currently residing on edge devices to more suitable places around the network. Active services are based on programs supplied by the users of the services. The aim is to enable users to have access to the services they require (custom services), whilst avoiding any requirement for operators and providers to manage large numbers of services. Active services should prevent the current problems being exacerbated by increased diversity of demand, but will not in themselves solve the current difficulties. In order to fully realise the intended flexibility it will be necessary to combine active services with a highly automated management and control system.

In this paper we motivate and describe a novel proposal for automating the management of future networks, the devices that inhabit them and the software that will run on them. We show how key network performance goals (such as stability under bursty load and quality of service) can be accommodated. This adaptive approach is inspired by observations of bacterial communities. These approaches will particularly suit ad-hoc networks of mobile devices where hardware constraints are more of an issue and the ability to move code on and off devices becomes essential. We also present results that clearly demonstrate that our proposal can offer 'hands off' management for networks of hardware devices using load scenarios based on mathematical models and real world inspired request streams.

II. Proposed Management Approaches

Future services and applications will be diverse, customisable and have data dependent behaviour. They will also be dynamic due to issues such as time varying resource demands and availability, and their increasingly distributed and heterogeneous nature. It is proposed that any management and provisioning system must be adaptive. The reasons for this, with examples, are outlined in this section.

Adaptive control is based on learning and adaptation [30]. The idea is to compensate for lack of detailed knowledge by performing experiments on the system and storing the results. This means monitoring resource demands, selecting policies based on behaviour and then implementing policy changes on a local or global level. This style of control has been proposed for a range of Internet applications including routing [9], security [11, 15]. At present there seems no application of adaptive control service configuration and management.

At first sight adaptation can seem an unsophisticated option in comparison to classical control methods, which are based on monitoring state, deciding on the management actions required to optimise future state, and enforcing the management actions. Adaptive control is much more human inspired, as it attempt to automate the behaviour of an expert, monitoring the network and making changes based on their interpretation of this monitoring. In classical control the decision is based on a detailed knowledge of how the current state will evolve, and a detailed knowledge of what actions need to be applied to move between any pair of states (the equations of motion for the state space). It is this need for complete and detailed knowledge of all cause/effect relationships that make such method untenable for many real world, dynamic, non-linear problems. Many control schemes in the current Internet (SNMP, OSPF) are based on this form of control. There is also a less precise version known as stochastic control, where the knowledge takes the form of probability density functions, and statistical predictions. All existing forms of traffic management are based on stochastic control, typically assuming Poisson statistics.

Genetic Algorithms (GAs) have been shown to offer a robust approach to evolving effective adaptive control solutions [17]. Recently many authors [12, 4, 3] have demonstrated the effectiveness of distributed GAs using an unbounded gene pool and based on local action (as would be required in a multi-owner internetwork). However, many authors, starting with Ackley and Littman [1], have demonstrated that to obtain optimal solutions in an environment where significant changes are likely within a generation or two, the slow learning in GAs based on mutation and inheritance needs to be supplemented by an additional rapid learning mechanism. Harvey [16] pointed out that gene interchange (as observed in bacteria [29,5]) could provide the rapid learning required. This was also demonstrated for a bounded, globally optimised GA [24]. In this paper we demonstrate that an unbounded, distributed GA with “bacterial learning” is an effective adaptive control algorithm for many aspects of an active service provision system derived from the application layer active network (ALAN) [13, 20, 21].

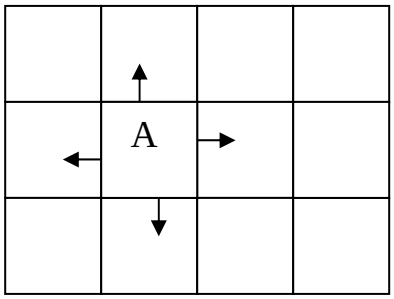
There seems to be no clear example of adaptive management algorithms for future mobile services on an active or ad-hoc network. There is research on schemes for load balancing and resource management on servers that offers some comparisons. One such work looks at adaptive algorithms for load balancing over a cluster of web servers [7]. Here it is proposed that the TTL's of DNS address mappings are calculated adaptively, taking in to account the distribution of client requests and the heterogeneity of the web servers.

Another method of service allocation control involves setting thresholds for acceptance/rejection based on hardware requirements and a priority class reward system [6]. Server capacity is divided up virtually with priority threshold values for each virtual partition. While the setting of dynamic thresholds is reward based and hence has some adaptive features, it does not deal with mobile software and the policies acting upon this software.

III. Simulation Design

Here the algorithm is described from a biological standpoint, followed by a description of how this relates to a network simulation and finally a set of finer granularity settings and implementation rules.

The simulation loosely takes the form of a cellular automata [28] where each cell behaves as a simple organism fighting for survival against other similar organisms in an environment of changing 'resources'. All interactions take place on a local basis, with each cell only communicating with a few other 'nearby' cells and there being a large number of cells. In the following diagram the state of cell A would effect the state of the 4 neighbouring cells:



Each cell has several values that are changed with time like fitness, activity and queue length, this and other details covered are fully discussed in the appendix. These values can effect the behaviour and state of the neighbouring cells. Each cell also sends and receives objects, each object being analogous to a nutrient, but in the network the object may be a request for a service or application. Each cell also has a genetic component that specifies certain behaviours, what requests it handles, when it handles them. This genetic component, the location of the cell and the nature of the request stream decides the fitness of the organism.

There is an important difference in comparison to standard genetic algorithms in that in this case a **fit colony** is the ultimate goal, not one superfit individual. The organisms that we chose to base our automata on were bacteria, there are several reasons for this choice, bacteria have novel reproduction methods, including plasmid migration, that mean they evolve very quickly but with a robustness that has kept many varieties present for several billions of years [29, 14]. Plasmid migration means sections of genome can be transferred between living organisms effecting their phenotype in a much more direct way than Darwinian methods. They are also very simple organisms whose genetic structure is completely known.

The model presented here randomly places automata, with random initial genomes, at some of the vertices of a 2-dimensional grid that forms their environment. The grid structure aids the understanding of the model, but a simple grid like structure is not a pre-requisite. The genome of each bacteria-like automata contain plasmids that enable the host to metabolise different kinds of available nutrients, and earn a reward (energy) for each molecule metabolised. For the purposes of this research a plasmid is the specific software that requires a certain nutrient plus a set of subjective rules as to whether the nutrient should be accepted (eg. Software A will be run if the unit is less than X% busy and has a queue smaller than Y. X and Y being initially randomly generated but are subject to subsequent evolutionary selective pressure). There are up to 100 different nutrient molecules. A random subset of these molecules is injected at each vertex along one side of the grid, at the beginning of every epoch (an epoch is 100 timesteps or 1 hop). The size of the injected subset is a variable of the model, and can frequently be close to zero. Once injected, nutrient molecules are forwarded (away from or parallel to the side they entered) automatically from vertex to vertex until a vertex is reached with a resident ‘bacterium’. If a suitable gene is present, and the bacterium has not already got enough nutrients, the molecule is metabolized. Otherwise it is forwarded after a small, specified delay. Each movement from one vertex to another is referred to as a hop. The bacteria can maintain a ’queue’ of up to 100 nutrient molecules before they are forced to forward nutrients. If a lot of molecules are decaying in the queue the bacterium can move one place up the grid if there is a vacant vertex. If no molecules are decaying the bacterium can move one place down the grid if there is a vacant vertex. They communicate with neighbours immediately above and below, so swaps are also possible. The time it takes to process 4 molecules is referred to as an epoch. The colony evolves through the exchange and mutation of plasmids. Bacteria that accumulate sufficient energy replicate. Bacteria that

do not accumulate energy die. It is possible for some individuals to survive without ever replicating, but they must do some metabolic processing, since energy is deducted each cycle (to simulate respiration).

Nutrient molecules are forwarded with some randomised lateral movement, but when forwarding would take the nutrient beyond the bottom of a sub-colony of microbes the nutrient is forwarded to the top of another sub-colony. Molecules are forwarded to the next OR the next but one layer. These changes enable us to spread the model across several physical computers and increase the size of colony that can be modeled real time, since links between the concentrations are less rich and slightly slower than within concentrations. Our test topology had 6 semi-independent sub-colonies. The 6 sub-colonies are connected in a star formation (1 forwards to 2 and 4, 2 forwards to 1 and 5, 3 forwards to 4 and 6, 4 forwards to 3 and 1, 5 forwards to 6 and 2 and 6 forwards to 5 and 3). Each sub-colony runs on identical hardware and has the same available resources at it's disposal.

To allow the automata to be truly autonomous, as they would be in a real colony, randomly chosen plasmids attach to molecules that are being forwarded, in line with the following rules.

1. If an individual FAILS to process a molecule (and so must forward it) AND a random number between 0-100 is less than the no. of nutrient molecules it has already stored for later processing (max 100) THEN a 'plasmid' from it's genome is attached to the request. In simpler terms; if do not process this item but I AM busy then attach a plasmid from my genome to the item.

2. If an individual succeeds OR fails to process a food item AND its busyness (plus a constant, currently 5%) is less than a random number between 0-100 THEN take a 'plasmid' from the 'plasmid attachment' space, if there is one. In simpler terms; regardless of if I process the item or not, if I am NOT BUSY then take a 'plasmid' from the request if there is one. Further implementation details of the model given in appendix I.

This explains the fundamentals of this biologically inspired algorithm but the analogy to networks needs more explanation. Each organism representing a node (or virtual node) on a future multi-service communication network. Each node having the capability (software) to process one or more types of service request in accordance to policies attached to the node or software component. The activity and performance of each node is monitored and used both for transfer of software and as a threshold for differential execution of services. The processing priority of a service request can be determined by a service request function, either 'time to live' or the value derived for carrying out the service request. Each nodal policy comprises a service request identifier and a service request criteria that decides if a service is processed.

If the activity indicator at a node reaches a probabilistic threshold it may add software (or a pointer to software) and the set of subjective policies to a request that it is forwarding. If it exceeds a second threshold for a sufficient period of time the whole nodal components can be replicated to a nearby available node. If a node's activity indicator fails to reach a probabilistic threshold then it can import software (and it's subjective policies) from a request, should these be available. If it's activity indicator stays below a certain threshold for an extended period of time all the active software can be deleted making the node available for new software and policies to be installed.

IV. Results

The simulation was tested with a variety of loading scenarios, it's performance when faced with these scenarios would tell us how suitable the approach would be for future network management. The focus of each test is outlined and it's performance presented and discussed in turn.

A. Reaction to increase in total network load.

The cluster was loaded with a poisson distributed traffic stream at all 6 nodes. Initially the software distribution and the associated subjective variables are randomly distributed, so there will be a period of

software re-distribution and variable optimisation. Then the load was varied over all nodes to the same extent. After the load is increased there is a period of adaptation as new organisms are generated and plasmids are exchanged. As the load gets excessive at greater than 750 requests per that the system begins to break down and drop rates and latency go up significantly. This is equivalent to a load of about 50% of saturation.

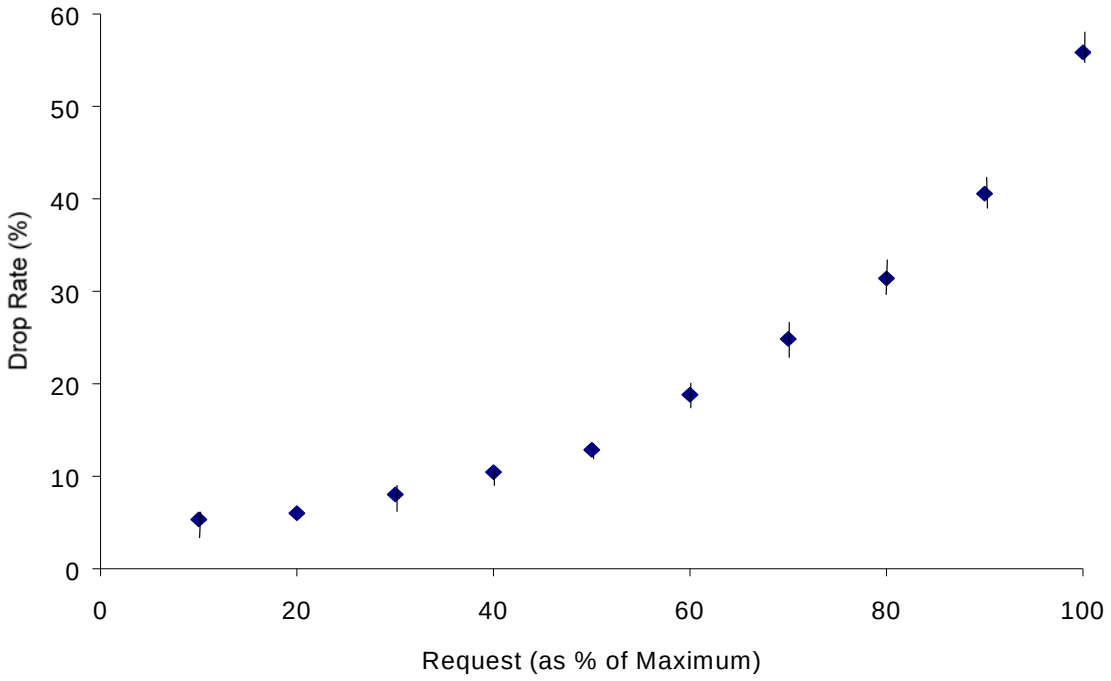


Fig. 1. Drop rate increasing exponentially as load is increased in steps.

The response in the latency of the system is somewhat different (Figure 2.), it increases proportionately at all increases in load. This again is due to the increased queue lengths and time taken to find a suitable node. Lines indicating standard deviations are given, values are generally less than 0.5.

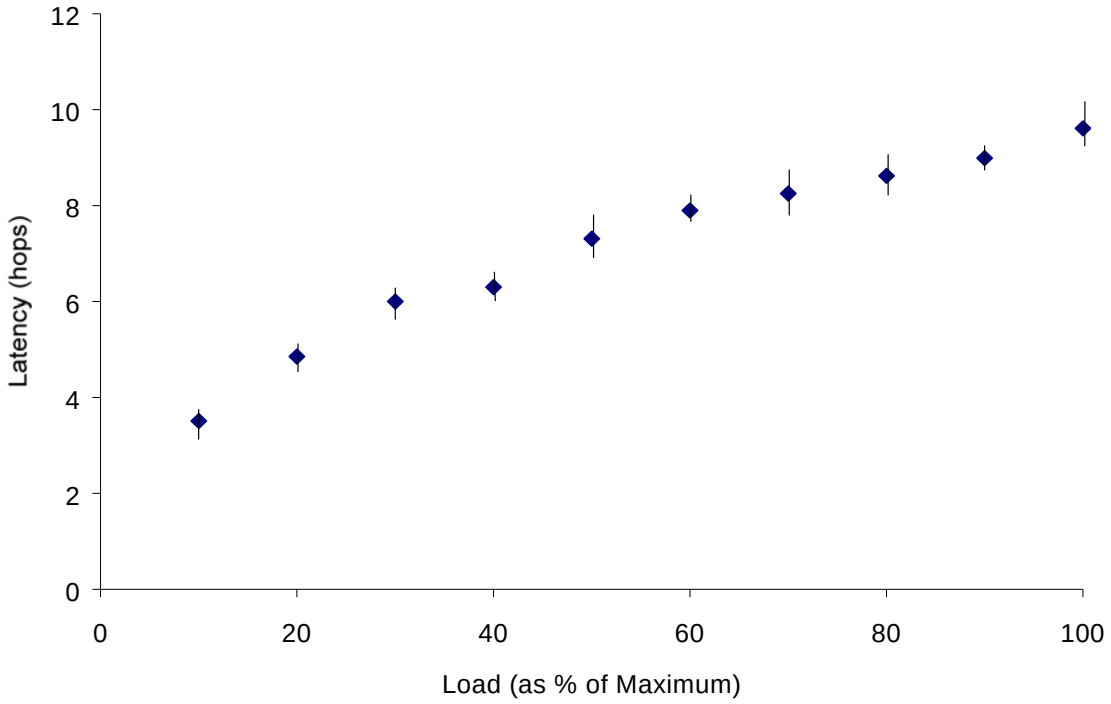


Fig 2. Change in service request latency as load is increased in steps

It is also important to see how the cluster of nodes reacts to load being redistributed asymmetrically, loading one of the nodes excessively while lightening the load on the rest of the network. Figure 3 shows the degradation in performance of two networks as the heterogeneity of the load is increased, a high '*% of total requests at most loaded node*' value indicates one node is heavily loaded with load taken from the rest of the network. It is apparent that an adaptive approach outperforms a randomly distributed approach at both low and high levels of heterogeneity.

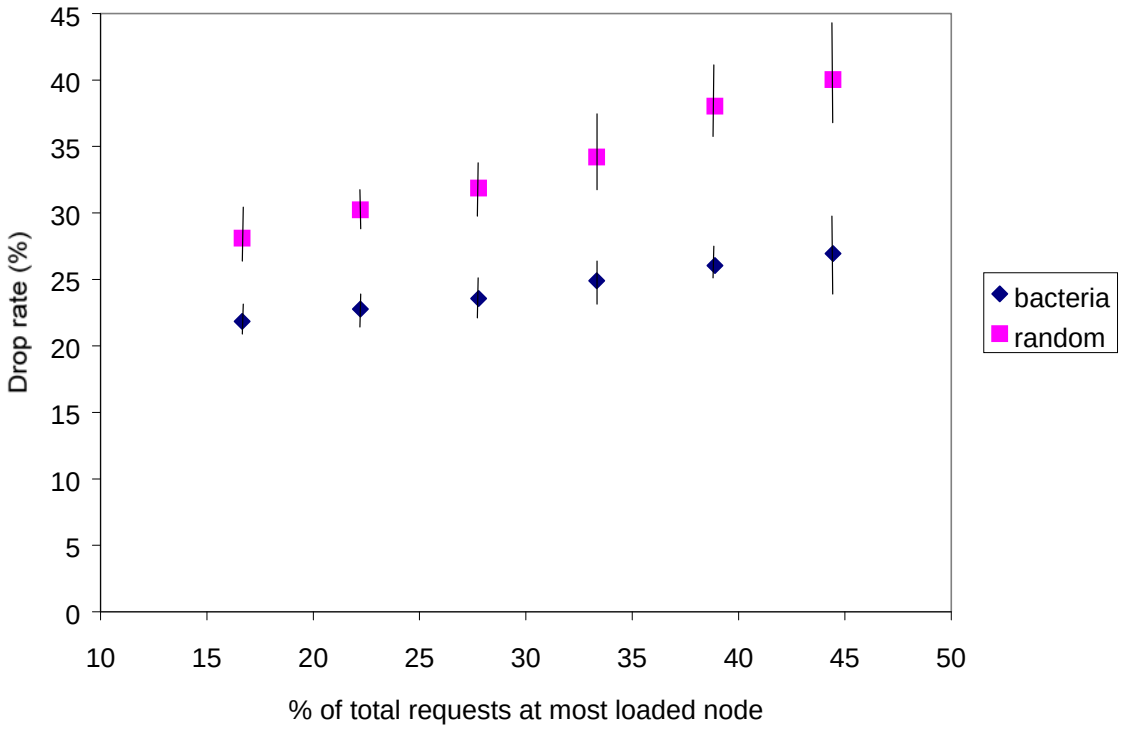


Fig 3. Comparing the efficiency of a bacteria based algorithm with a random placement of software.

B. Payment related service quality.

Users may wish to pay more for a service in the belief that they might receive a better quality of service in return for their investment, this may be especially true of time sensitive or high importance requests. A fitness function supplies an implicit mechanism for payment based reward. Fitness can be based on some function of the payment received. To test this, requests were supplied to the simulation with the same reward value for the organisms handling them, once stable service provision was achieved the rates of reward for the services were changed with some offering a higher reward and some a lower reward. The changes in the quality of service supplied are shown in Figure 4.

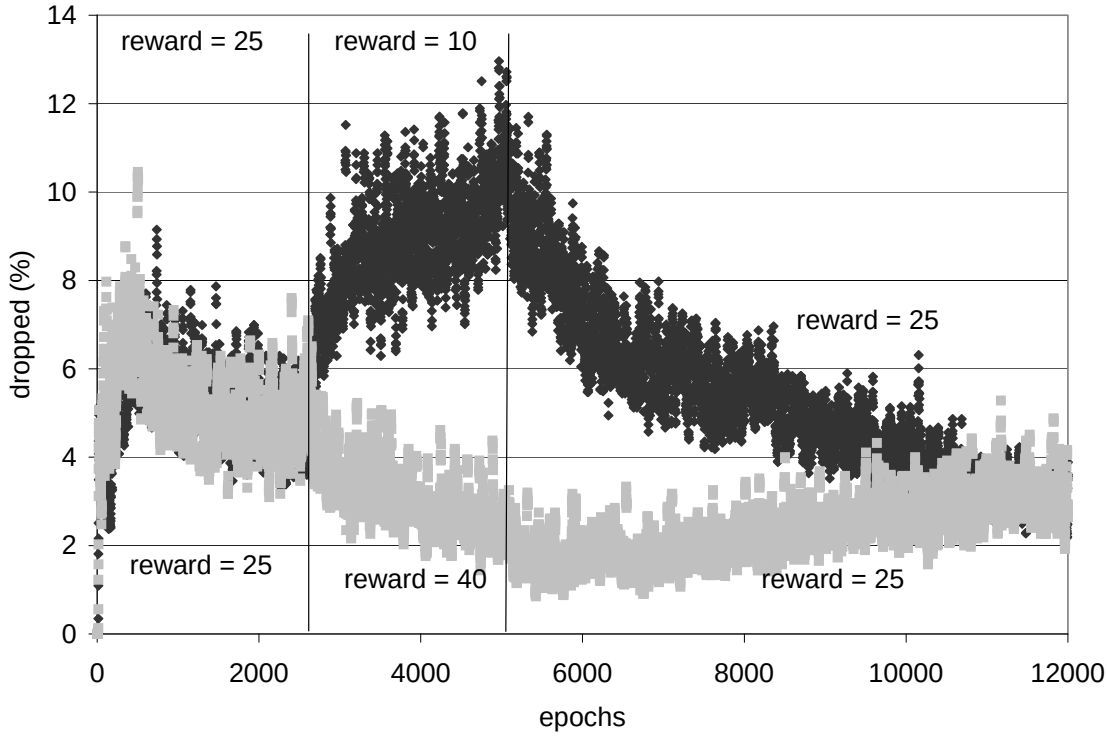


Fig 4. Changes in drop rate based on reward, drop rate being inversely proportional to reward.

C. Adoption of new services.

One very desirable asset of any collection of networked devices will be an infrastructure where the network operator does not have responsibility for installing, managing and distributing active software around the network. It would be ideal if software was placed on the network by users and adopted, replicated or rejected by the network management algorithms. This was tested by introducing a new software onto the network and a demand for the service it provided. This service should then proliferate around the network to an extent governed by the reward it offered to the network provider. This was demonstrated first for a simple instance where one service is introduced along with a demand for that service and once it is established a new service is introduced alongside the original one. Fig 5 shows how the software for both services is adopted quickly, though quicker for the second service than the first[1]. Figure 6 demonstrates a more complex instance of this behaviour. A set of services are established, after some time a new service is added, this service has no value so offers no reward to cells serving it, it therefore fails to be adopted around the network. The third service to be introduced does have a value, it can be seen that over time the new service does get itself sufficiently embedded into the network to allow comparable performance to the first service. In Both cases the second service takes longer to reach as a low a drop rate than the first. One reason for this could be that the genetic content of each node is already more optimised when the new service is introduced, making it more difficult for genes that handle this service to take hold. It is easier to outcompete a random gene than it is to outcompete a gene that is present for evolutionary reasons.

[1] Figure 5 shows one instance of the experiment but this test was subsequently repeated numerous times to get more precise statistics on time taken for service components to be successfully embedded in the network. This showed that on average it took 662 ±140 epochs to achieve a 99% success rate (10 epoch moving average) for the first service, and 1957 ±110 for the second service.

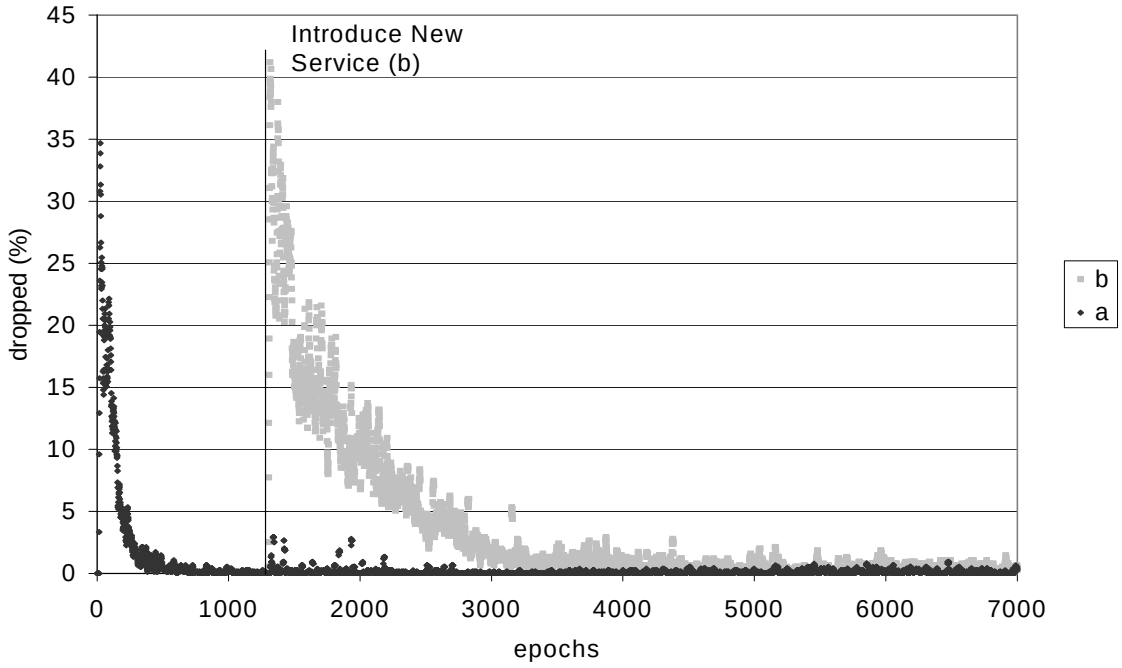


Fig. 5. Drop rate on introduction of 2 new services.

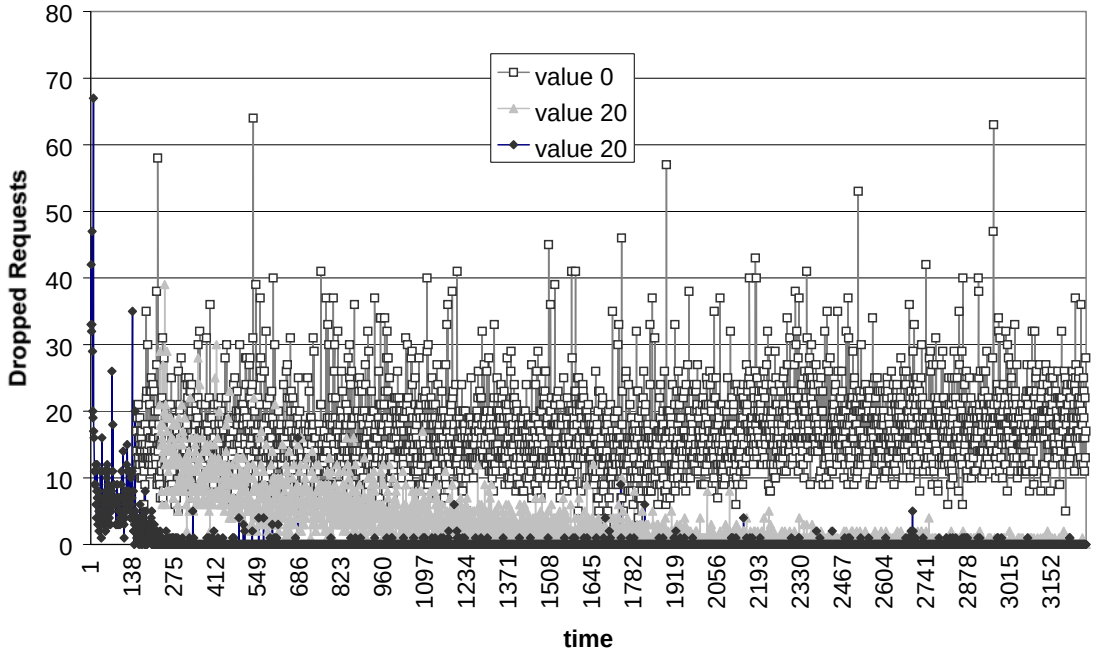


Fig. 6. Drop rate on introduction of new services, valued falling to similar levels to original service, unvalued failing to be adopted

To emphasise the innate hands off adaptability of the bacteria inspired algorithm, the same test was given a network that attempted to provide a better service by caching programs in a straightforward way. When new services are introduced the software to run them proliferates around the network regardless of the reward structure, this is shown by the rapid adoption of all three services regardless of the reward they offer (Figure 7). The cache based algorithm could be modified to be more discerning but this would be the first step to turning it into a fitness based algorithm.

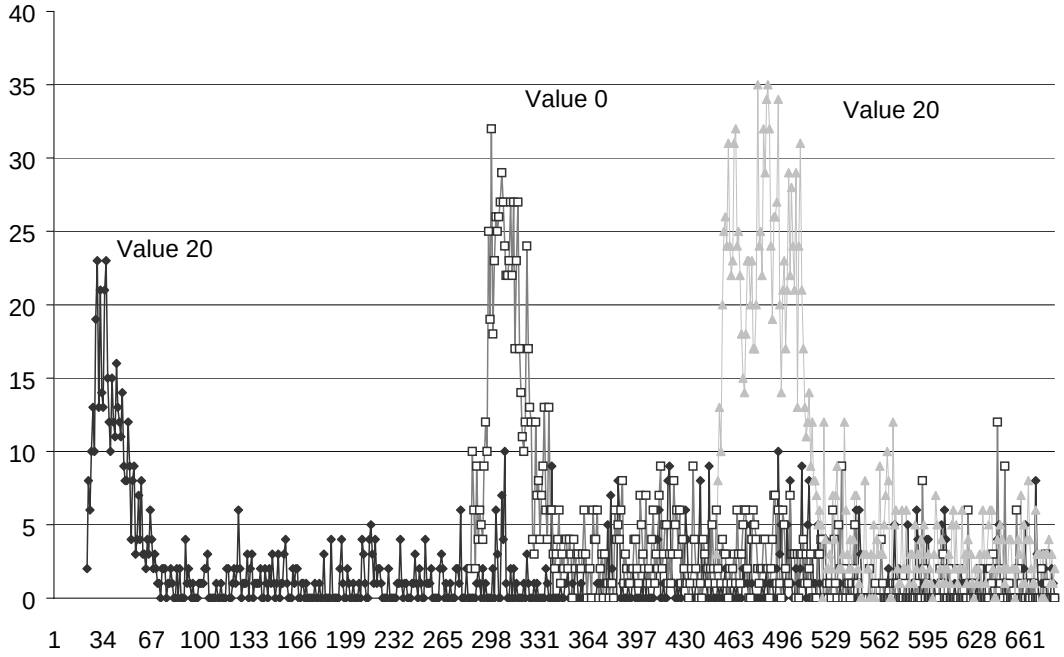


Fig. 7. Drop rate on introduction of new services to caching algorithm service network, both valued and unvalued services being adopted equally well.

D. Software Layering

For the experiments, groups of 25% of the different nutrient molecules were given lifetimes of 2, 5, 10 and 20 hops respectively, and the injection rate was around 25% of the maximum allowed by the model. This is enough to allow around 50% of the vertices to be inhabited. The processing efficiency is compared to a simulation in which evolution was blocked by stopping plasmid interchange, so the initial random gene distribution persists. Figure 8 gives visual idea of how differently configured automata distribute themselves around the grid of vertices, here 4 different service types with differing subjective rules (not shown). Different types of organism occupy different regions based on their functional constraints, particularly the time to live of the service requests. [23]

This layering of different strains resembles the way different types of simple organisms exist at different strata of a water based environment. With aerobic phototrophs like phytoplanktons floating in the photic zone while anoxygenic phototrophic bacteria like *Chlorobium* live much deeper in the same water course.

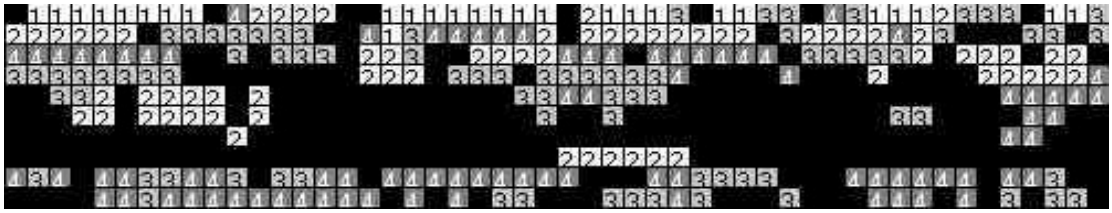


Fig 8. Colony of software agents showing layering based on role.

E. Stability under real network derived loads.

Internet traffic has very irregular characteristics, showing determinism and long-range dependence at both the packet [18] and application request [8] level. This puts different kinds of stresses on any data network. All the previous tests used a randomly generated request stream displaying roughly poisson characteristics. It is important to see how well the algorithm copes with a long term test based on real internet requests. To do this log files from 6 high level web caches were taken for 7 days. The caches all had requests from a different subset of users, had very different total loads (daily log files varied from Each request was grouped into one of ten possible groups based on the host domain name (.com or other) and the file type (.jpg/.gif/ .html/ .txt/ other). These were then divided randomly into one of 10 subgroups, making a total of 100 different request types. If we look at the distribution of the 10 initial groups for a randomly slected one hour period it is apparent that they provide a very asymmetrical test (Figure 9), with .gif files from .com sites being requested over 25 000 times per hour to .txt files from non .com sites being requested 37 times an hour.

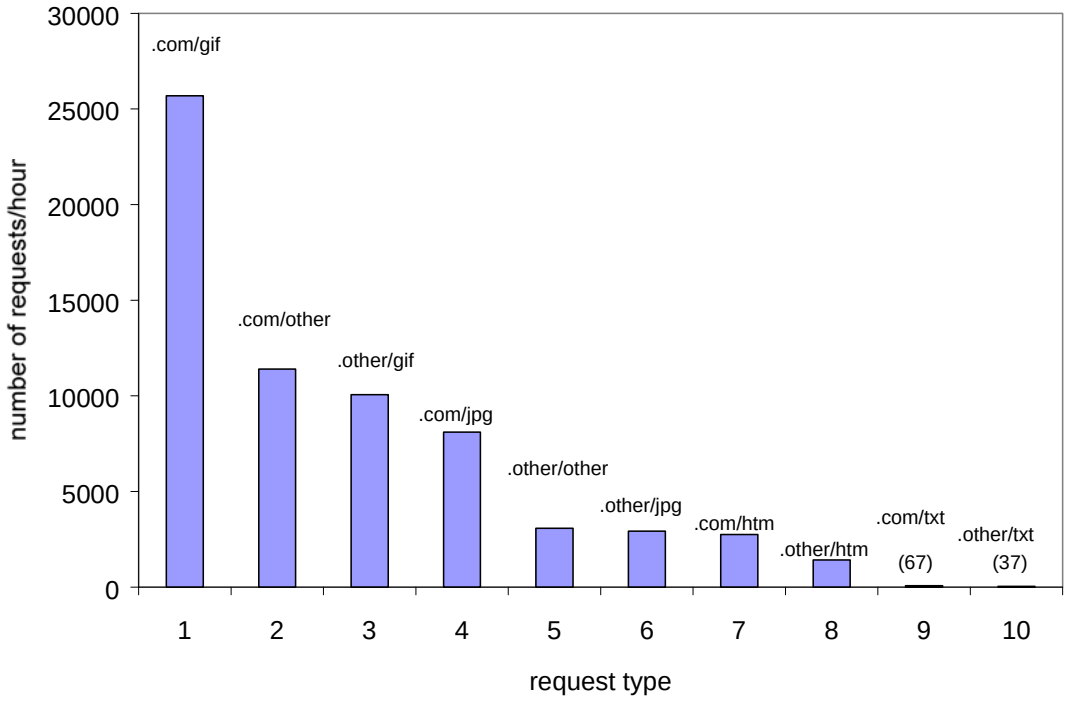


Fig. 9. Distribution of request sub-types, based on domain and file types.

If we look at the Hurst parameter for occurrences of requests we can see that at 0.911 it is well above 0.5 suggesting a significant amount of self-similarity is present in the request stream (figure 10)

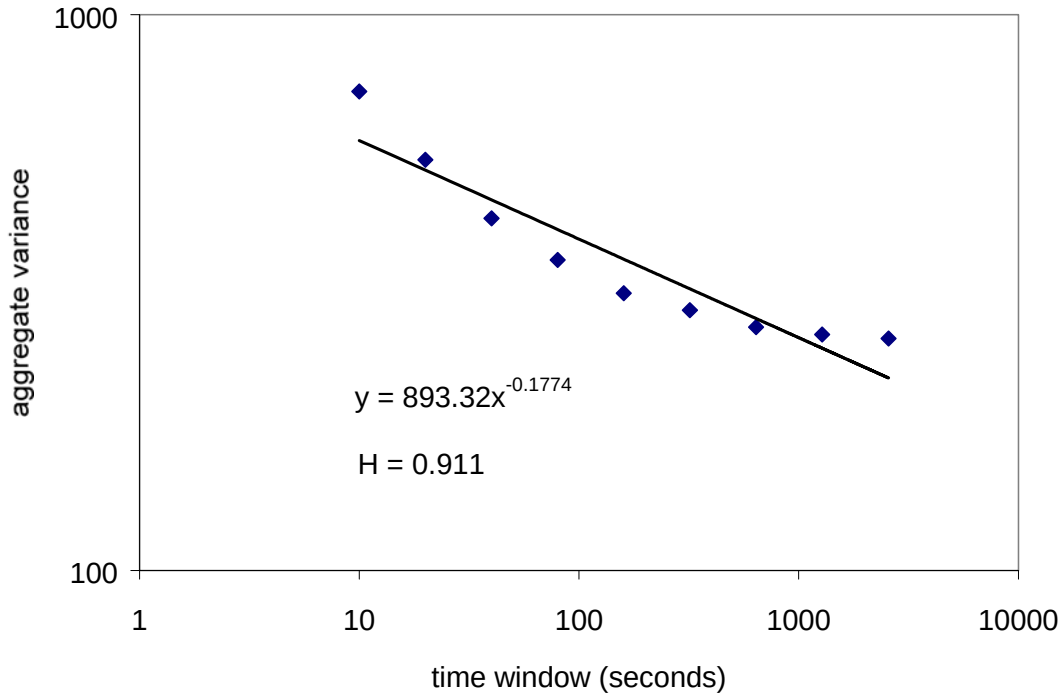


Fig. 10. High Hurst parameter showing long range dependence in request stream.

The two performance metrics of latency and drop rate were examined for the 10 request sets, it is apparent from Figure 11 and Figure 12 that there is a relationship between performance and popularity of file type averaged over 6 sets of results. More popular request types have lower latency and lower drop rates. This is a result of the increased replication and plasmid transfer that organisms handling this type of request, thereby proliferating the network much more than the less requested file types.

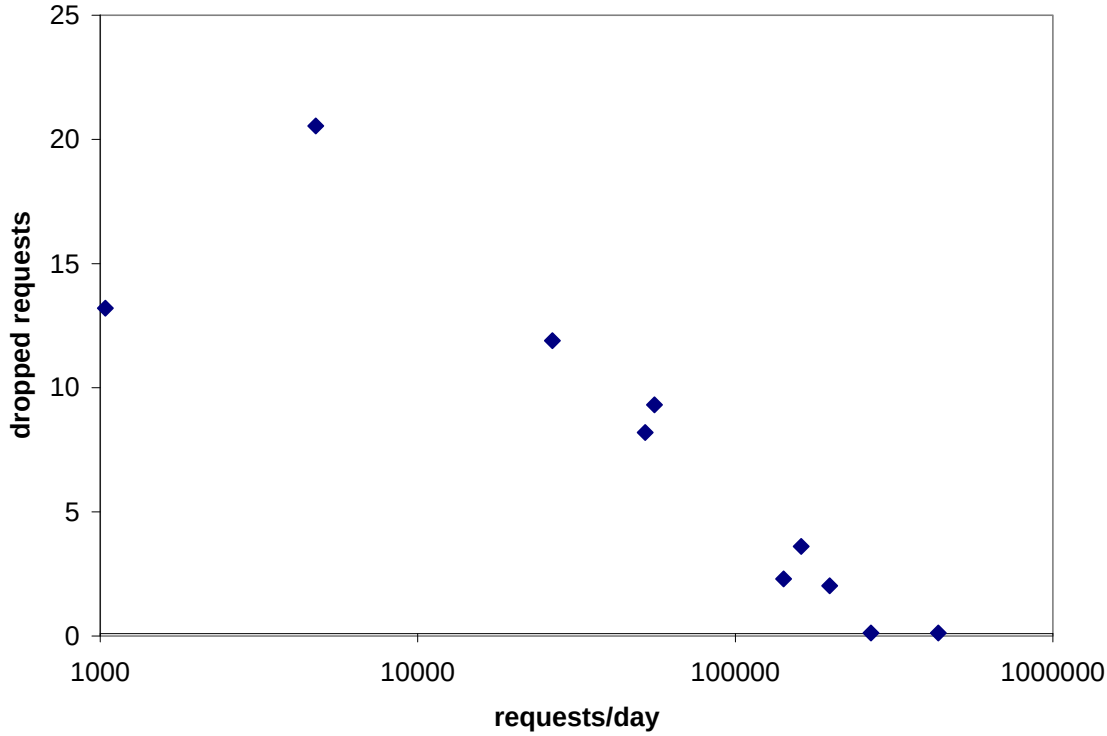


Fig. 11. Drop rate of 10 service groups with differing popularity improves with more requests.

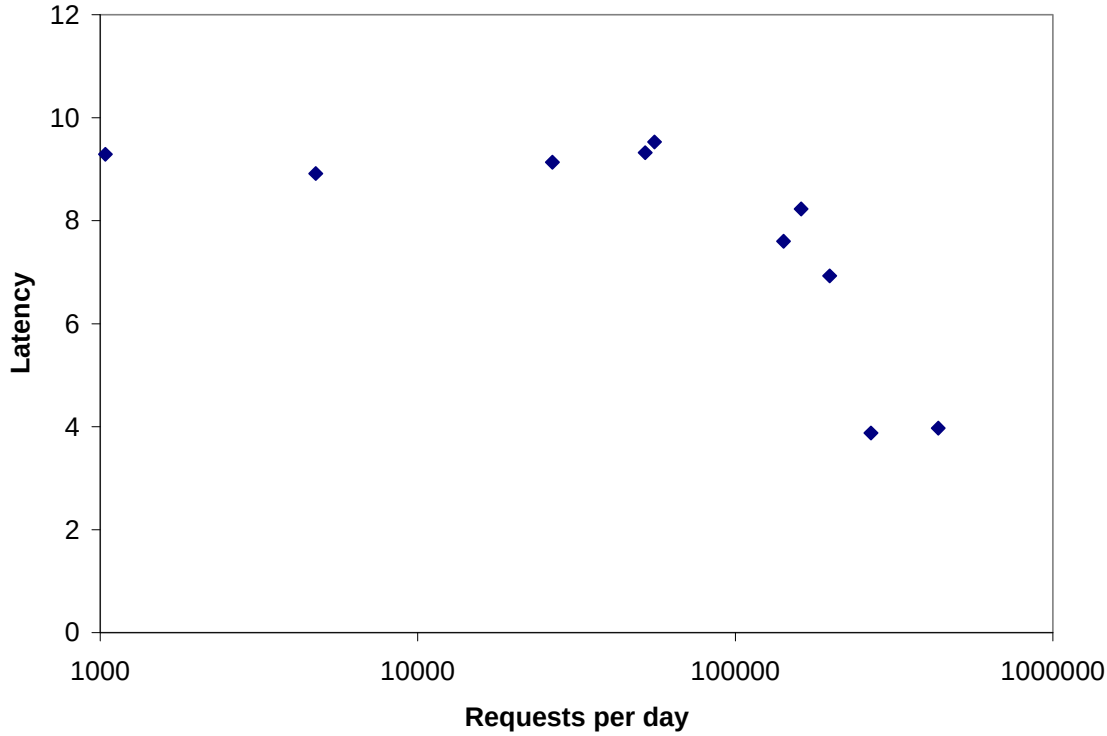


Fig 12. Latency for 10 request groups in relation to their popularity improving with increased requests.

F. Comparison with other algorithms.

While earlier experiments have compared the bacterial algorithm with itself at different conditions it is also important to compare it to other possible information dissemination algorithms. Therefore the same problem was tackled by an algorithm that randomly places software and subjective rules around the network (random algorithm), and a algorithm that simulates caching by replicating a rule, in the direction of it's request, each time it is used (caching algorithm).

With relatively simple loading (ie. similar levels of all request types, all with the same time to live) there seems very little difference between the two non-random algorithms. Figure 13 shows the drop rate for the three algorithms at identical loads, requests for this load vary in both a stochastic way, varying around a mean but also taking a random walk (see section III). It is only when the network of nodes is loaded in a more disproportionate way that the benefits of a truly adaptive approach is shown. The load for the next trial was made up of different services who's requests expire after a number of hops, each request type having a different time to live (Figure 14). The overall load also varies in a centered, random walk way. The third load scenario (figure 15) is made up of real world inspired requests as discussed in section D.

The values given are averages from up to 10 repeat experiments with the lines on the top of each column representing the standard deviations (ie. In Figure 13 the SD for the 6 means are 0.8, 0.16, 0.49, 1.48, 0.21, 0.27).

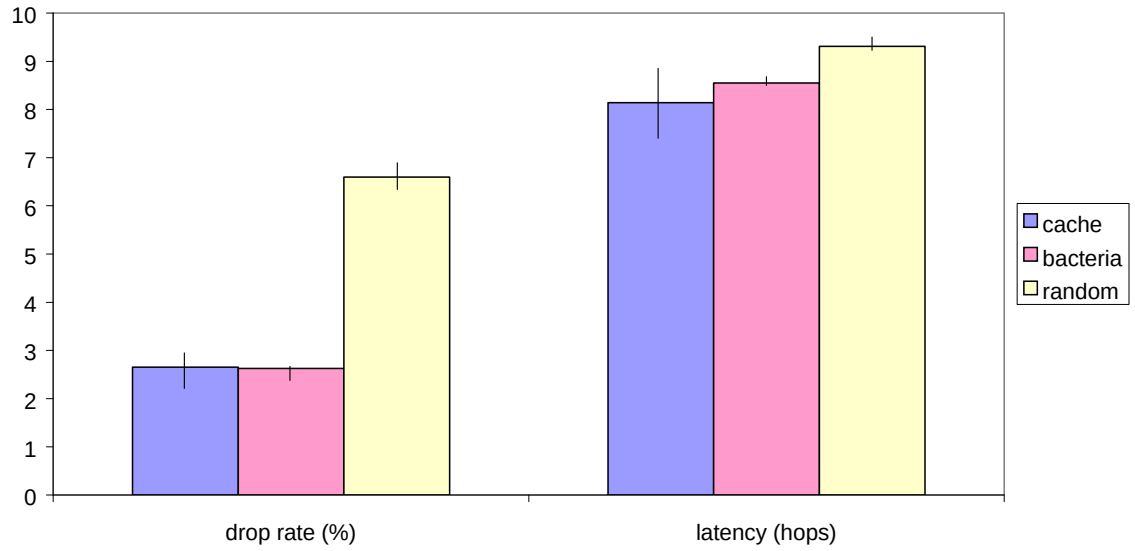


Figure 13. Differences in performance under artificially generated load

.

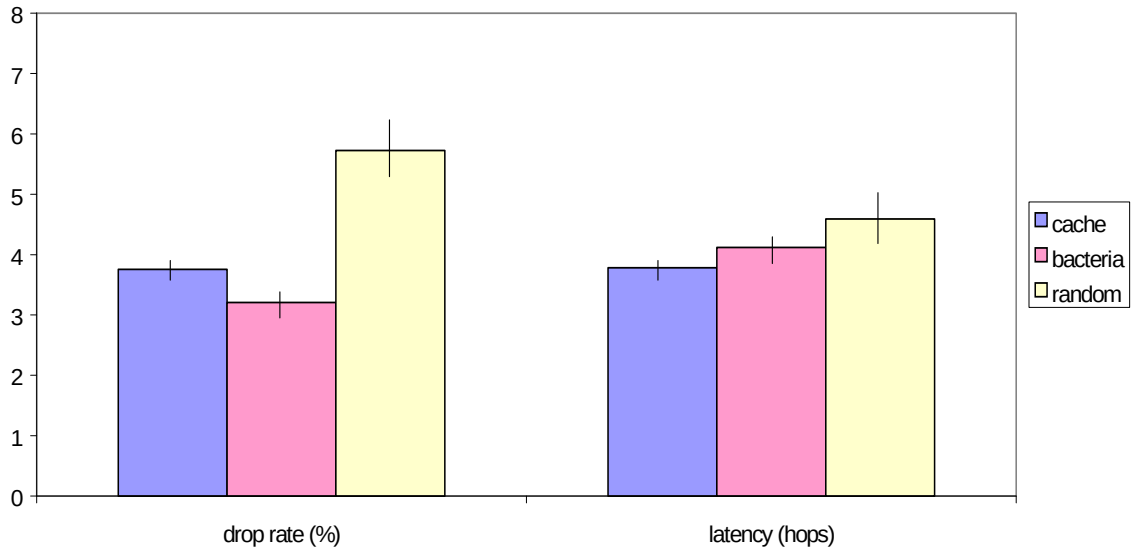


Figure 14. Differences in performance under artificially generated load with differing request times to live

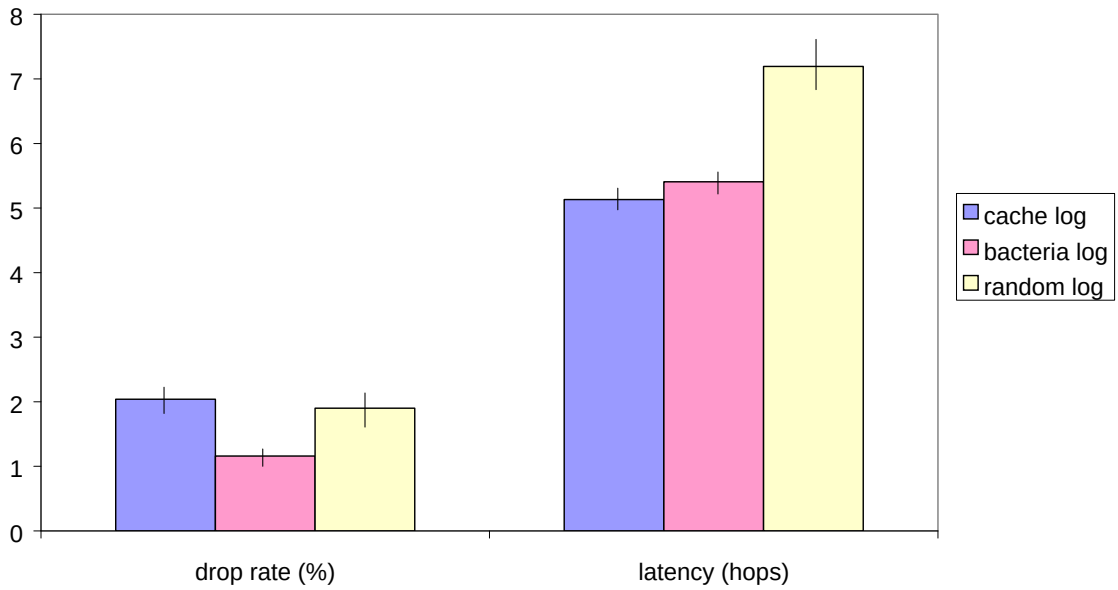


Figure 15. Differences in performance under web request derived load.

In the last 2 tests it is clear that bacteria based algorithm outperforms the other 2 algorithms in terms of minimising the number of requests dropped, though the latency is higher for the bacterial algorithm when compared to the cache based approach. There are 2 reasons for the difference in latency. Firstly the bacterial approach processes more requests per unit time, hence the lower drop rate, it is fair to accept that to do this it might take longer to process these requests. More significantly, the adaptive element in this approach means that some hardware nodes in the cluster have many unpopulated vertices, the cache approach uses all vertices. So the bacterial approach is dropping fewer requests while also using less of the

hardware resources. It should also be remembered that the strong performance of a bacteria analogy is achieved with the added benefits of increased flexibility and a completely 'hands off' nature.

G. Global Fitness.

While drop rate and latency are both important metrics and good indicators of performance, the fitness of individuals within the network is the most accurate measure of learning. Cells reproduce when their fitness is sufficiently high, so any measure of fitness must be a global one for the whole network not an average for all the cells, fitness is described by both the fitness of an individual and the number of individuals that can survive in the environment. A measure of global fitness was calculated by simply adding together the fitness of all individuals, if this showed asymptotic properties then we can be happy that the learning algorithm is reaching (at least) a stable local maxima. Figure 16 shows precisely this behaviour with respect to the global fitness. The optimisation of the drop rate is also shown highlighting the direct relationship between performance and global fitness.

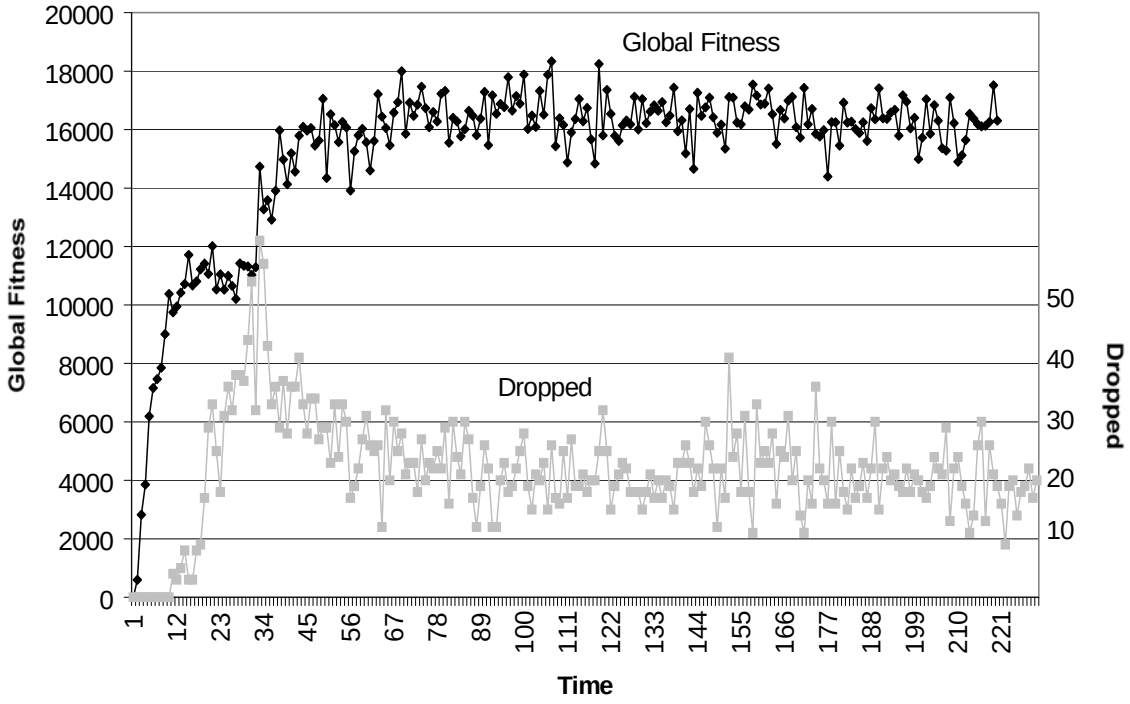


Figure 16. Asymptotic convergence of global fitness and drop rate over time.

V. Conclusions

The basic tasks of a network management algorithm are shown to be carried out robustly and efficiently. Load balancing and the distribution of new software are implicit parts of the reproductive and evolutionary nature of the algorithm, so it is no surprise that these functions are performed so readily. More complex functionality like allowing some varied quality of service, some payment based security and ability to handle realistic traffic streams are all present and essential in any distributed service management system.

It is apparent that the bacteria inspired algorithm is a suitable choice for providing service completion in a simulated, mobile software enabled network. It has also been shown, in terms of latency, that various discreet qualities of service can be achieved.

When compared to conventional resource allocation algorithms (ie. Caching) it is apparent that a bacterially inspired approach offers many qualitative benefits when a network is loaded in a realistic way.

It can be argued that the management algorithm is not designed to optimally handle small numbers of requests for one specific service. Though if a service is unpopular it can be assumed that the user will expect to have to supply code for this service with his requests. The fact that the system offers some facility for these unpopular requests, albeit at a lower standard, is encouraging. Improved performance for the less requested services or at the less loaded nodes could be achieved by using various flavours of quorum sensing [27], for example, an organism could sense the number of organisms with similar phenotypes and be less procreative if there are high numbers of such organisms nearby. A simpler approach could just detect how many neighbours of any kind were present and modifying reproductive thresholds accordingly. But any improvement in the performance of least requested services might be accompanied by a decrease in performance for the popular services (on the 'no free lunch' principle), so the impact of this must be studied.

VI. References

[1] D.H. Ackley and M.L. Littman, "Interactions between learning and evolution". pp. 487-507 in Artificial Life II (ed C.G. Langton, C. Taylor, J.D.Farmer and S. Rasmussen), Addison Wesley, 1993.

[2] E. Amir, S. McCanne, R. Katz, “An active service framework and its application to real time multimedia transcoding” Computer Communications review 28, 4, pp178-189, Oct 1998.

[3] G. Booth, "GECCO: A Continuous 2D World for Ecological Modeling", Artificial Life, 3, pp. 147-163, Summer 1997.

[4] R. Burkhart, "The Swarm Multi-Agent Simulation System", OOPSLA '94 Workshop on "The Object Engine", 7 September 1994.

[5] D.E. Caldwell, G.M. Wolfaardt, D.R. Korber and J.R. Lawrence, "Do Bacterial Communities Transcend Darwinism?" Advances in Microbial Ecology, Vol 15, p.105-191, 1997.

[6] I. Chen and C. Chen, "Threshold Based Admission Control Policies for Multimedia Servers", Computer Journal, Oxford University Press, Surrey, GB. Vol 39, no. 9, 1996, pages 757-766.

[7] M. Colajanni and P. Yu, “Adaptive TTL schemes for Load Balancing of Distributed Web Servers” Performance Evaluation Review -- ACM SIGMETRICS, 25(2):36--42, September 1997.

[8] M. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes", IEEE/ACM Transactions on Networking, 5, 6, pp 835-846, Dec 1997.

[9] G. DiCaro and M. Dorigo, "AntNet: Distributed stigmergic control for communications networks", J. Artificial Intelligence Research, 9, pp. 317-365, 1998.

[10] D. Estrin, R. Govindan and J. Heidemann, "Embedding the internet", Communications of the ACM 43, 5, pp 39-41

[11] D.A. Fisher and H.F. Lipson, "Emergent algorithms - a new method of enhancing survivability in unbounded systems", Proc 32nd Hawaii international conference on system sciences, IEEE, 1999

[12] S. Forrest and T. Jones, "Modeling Complex Adaptive Systems with Echo", In Complex Systems: Mechanisms of Adaptation, (Ed. R.J. Stonier and X.H. Yu), IOS press pp. 3-21, 1994.

[13] M. Fry and A. Ghosh “Application Layer Active Networking” Computer Networks, 31, 7, pp. 655-667, 1999.

[14] S.Golubic “Stromatolites of Shark Bay” pp103-149 in Environmental evolution ed. Margulis and Olendzenski, MIT press 1999

[15] M. Gregory, B. White, E.A. Fisch and U.W. Pooch, "Cooperating security managers: A peer based intrusion detection system", IEEE Network, 14, 4, pp.68-78, 1996.

[16] I. Harvey, "The Microbial Genetic Algorithm", unpublished work, 1996, available at ftp://ftp.cogs.susx.ac.uk/pub/users/inmanh/Microbe.ps.gz

[17] J.H. Holland, "Adaptation in Natural and Artificial Systems" MIT press, 1992.

[18] W.E. Leland, W. Willinger, M.S. Taqqu and D.V. Wilson, "On the self similar nature of ethernet traffic". Proceedings of ACM SIGComm, September 1993.

[19] J.P.Macker, M.S.Corson, "Mobile Ad Hoc Networking and the IETF", ACM Mobile Computing and Communications Review, Vol. 2, Jan 1998.

[20] I.W. Marshall *et. al*., "Active management of multiservice networks", Proc. IEEE NOMS2000 pp981-3

[21] I.W.Marshall and C.Roadknight, "Adaptive management of an active services network", British Telecom. Technol. J., 18, 4, pp78-84 Oct 2000

[22] I.W.Marshall and C.Roadknight “Provision of quality of service for active services” Computer Networks, April 2001

[23] I.W.Marshall and C.M.Roadknight, "Emergent organisation in colonies of simple automata." 6th European Conference on artificial life. 2001.

[24] N.E. Nawa, T. Furuhashi, T. Hashiyama and Y. Uchikawa, "A Study on the Relevant Fuzzy Rules Using Pseudo-Bacterial Genetic Algorithm" Proc IEEE Int Conf. on evolutionary computation 1997.

[25] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modelling", IEEE/ACM Transactions on Networking, 3, 3, pp 226-244, 1995.

[26] E. Sanchez, M. Sipper, J. O. Haenni, J. L. Beuchat, A. Stauffer, and A. Pérez-Uribe, "Static and dynamic configurable systems", IEEE Transaction on Computers, 48, 6, pp. 556-564, June 1999.

[27] J. A. Shapiro. (1998) Thinking of bacteria as multicellular organisms. Annu. Rev. Microbiol. 52, 81-104.

[28] M. Sipper. "Studying artificial life using a simple, general cellular model." Artificial Life Journal, 2(1), 1995. The MIT Press, Cambridge, MA.

[29] S. Sonea and M. Panisset, "A new bacteriology" Jones and Bartlett, 1983.

[30] Y.Z. Tsypkin. "Adaptation and learning in automatic systems", Mathematics in Science and Engineering Vol 73, Academic press, 1971.

Appendix I. Implementation Details.

Rule set generation.
Vertices are chosen in the initial start up and populated with rule sets. Each rule is given a queue length threshold (rand *60), an activity threshold (10+random*80) and a randomly selected request type. Each node is given a maximum queue length (random *200) and a trophism value (5+random*90). So an example rule set may be:
34,12,d (max queue length = 34, max activity = 12% and request type = B)
An example node rule set may be:
56,12 (max queue tlength for any request, regardless of request max queue length = 56, and trophism value = 12 (node more likely to send rules down)

Trophism.
This is the tendency for an organism to move or reproduce in a specific direction, this is genetically programmed.

Request Rate.
A mean number of requests can be specified per epoch. The actual number of requests will be equal to a random number between 0 and 1 multiplied by the request rate divided by the number of columns.

This in turn can be varied every epoch by applying the following algorithm.
If random (1) < 0.5 then new rate = (5*initial rate + (random (1)*(initial rate/10)))+ initial rate/6
If random (1) > 0.5 then new rate = (5*initial rate - (random (1)*(initial rate/10)))+ initial rate/6

Ratios.
10 groups of services can be requested at differing ratios that must add up to 100%.

These ratios can be made to vary in a random walk way, whereby two of the non-zero values are randomly chosen every epoch, one is increased by 1%, the other decreased by 1% (as long as neither of the values is 0 or 100).

Each request has a value, time to run and an age when dropped.

Mutation
At rate chosen, mutate in one of the following ways:
Change queue length variable of a rule
Change activity variable of a rule
Change request type of a rule
Change max queue length of an organism
Change trophism of an organism

Also 1 mutation in 5 kills the organism, removing its genotype from the vertex

Time taken to check for a rule
This is subtracted from the available time every time an available rule is checked for suitability.

Time to pass on a request
This is the time taken to reject an unsuitable request and transfer it to a neighboring queue.

Number of rules used/ max number of rules
Each organism contains rules. There is a maximum useable number of rules which are the rules checked when looking for compatibility match. And there is a maximum total number of rules, which include rules that are dormant (not checked or selected for)

Initial Nodes
The total number of initial nodes can be set, up to the maximum number.

Penalty for a request timing out in a queue.
Once a request reaches the point of comparison, it is checked to make sure it's expiry time has not been exceeded., if it has, it is deleted from the network and the appropriate penalty applied to the fitness of the organism.

Penalty for too long queue
If too many requests are placed on a queue a penalty can be applied for every request that has to be transfered. This takes the form of a deduction of some of the available time.

Penalty associated with TTL of passed on request
Rather than having a fixed penalty for passing on a request, this penalty can be linked to the ttl of the request. Requests with a low TTL incur a higher penalty for forwarding.

Reproduction threshold
The activity of a node needs to be above the reproduction threshold for several consecutive evaluation periods for reproduction to occur. If it has a short maximum queue length then this will be 3 periods, a medium maximum queue length will mean 4 periods are needed and a long maximum queue length will mean the reproduction threshold must exceed the reproduction threshold for 5 consecutive evaluation periods. When reproduction is merited it will be carried out to a neighboring node using a trophism type effect. If the trophism value for a node is high then the reproduction will favour moving towards the influx of requests, if the trophism value is low then the reproduction will favour a direction away from the entry of requests. A node will search for an empty neighboring vertex in a random fashion (subject to the trophism modifier), but if there is not one available it will not reproduce.

Death threshold
If a node is repeatedly idle it 's active rules can be deleted, if this means it has no active rules then it is considered dead and it's vertex can be used by another set of rules. If it has a long maximum queue length then this will be 2 evaluation periods, medium queue length gives 3 evaluation periods and short max queue length gives 4 evaluation periods.

Performance
The fitness of a node is the sum of many penalties and rewards. Penalties have been discussed in the previous entries. Rewards are given for processing a request and are set by the user.

Forwarding
If a request arrives at a vertex and there is no node on that vertex, the request is forwarded downwards to a neighboring vertex using the following route randomising scheme.

5%	X	5%
15%	15%	15%
15%	15%	15%

Each location is checked for an active node, if one is not found then another of the vertices is checked until all possible neighboring vertices have been checked. If none are found then the search continues from the last neighboring vertex.

If forwarding places the request outside the boundry of the current network of nodes then it is sent to one of two nearby members of the cluster.

The same forwarding mechanism applies to requests that cannot be served by the set of rules available at the node, except to discover this the request must wait in the queue until it can be processed.