Adaptive management of an Active Service Network

Ian W. Marshall and Chris Roadknight

BT Adastral Park, Martlesham Heath, Ipswich, Suffolk, UK IP5 3RE

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

Abstract.

The benefits of active services and networks cannot be realised unless the associated increase in system complexity can be efficiently managed. An adaptive management solution is required. Simulation results show that a distributed genetic algorithm, inspired by observations of bacterial communities, can offer many key management functions. The algorithm is fast and efficient, even when the demand for network services is rapidly varying.

Keywords. Active Network, Network Management, Genetic Algorithms, ALAN

Introduction.

New Internet services and features are currently being introduced more slowly than users require, since existing human-intensive processes cannot cope with the rate of change. The problem can only get worse as customers gain experience of the Internet, and their expectations rise. Clearly the management solutions for the future Internet must require significantly less manual intervention than at present. Active services [1, 2] are based on programs supplied by the users of the services. The programs run on devices owned by network operators or network service providers (such as caches, mirrors conference controllers and firewalls), and add value through manipulations such as transcoding, filtering and state maintenance. 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 an active service network, inspired by observations of bacterial communities. We also present some initial results that demonstrate our proposal can address some of the key issues, and is worthy of considerable further study.

Control and Management

A future network providing active services will be unbounded in both scale and function. The network will be constantly expanded, upgraded, and re-dimensioned.

Even the hardware will probably become programmable [3]. An enormous range of services will develop and evolve at an unprecedented rate. It is impossible to predict accurately what the user defined services will look or behave like but the number of entities (independent software objects) could ultimately fill the IPv6 address space. A 'global state' of the system will be impossible to ascertain due to its massive scale. The equations of motion for the state space will be equally indeterminate due to the ever changing complexity. Even stochastic prediction will be infeasible due to the fractal properties of the traffic [4,5]. In cases, such as this, where the system state and/or the equation(s) of motion are not known, conventional methods of control and management do not apply and adaptive methods of control must be used [6].

Conventional control of dynamic systems is based on monitoring state, deciding on the management actions required to optimise future state, and enforcing the management actions. 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). 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 (pdf), and statistical predictions. All existing forms of traffic management are based on stochastic control, typically assuming Poisson statistics.

Adaptive control [6] is based instead on learning and adaptation. The idea is to compensate for lack of knowledge by performing experiments on the system and storing the results (learning). Commonly the experimental strategy is some form of iterative

search, since this is known to be an efficient exploration algorithm. Adaptation is then based on choosing some actions that the system has learnt are useful, using a selection strategy (such as a Bayesian estimator), and implementing the selected actions. Unlike in conventional control, it is often not necessary to assume the actions are reliably performed by all the target entities. This style of control has been proposed for a range of Internet applications including routing [7], security [8,9], and fault ticketing [10]. As far as we are aware the work presented here is the first application of distributed adaptive control to service configuration and management.

Holland [11] has shown that Genetic Algorithms (GAs) offer a robust approach to evolving effective adaptive control solutions. More recently many authors [12, 13, 14] 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 [15], 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 [17,18]) could provide the rapid learning required. This was recently demonstrated by Furuhashi [19] for a bounded, globally optimised GA. In this paper we demonstrate for the first time 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) [1, 20].

ALAN

ALAN [1] is based on users supplying java based active code (proxylets) that runs on edge systems (dynamic proxy servers - DPS) provided by network operators. It is assumed that many proxylets will be multiuser, and most requests will be to "run" a proxylet that already exists in the network. Messaging uses HTML/XML and is normally carried over HTTP. There are likely to be many DPSs at a physical network node. It is not the intention that the DPS is able to act as an active router. ALAN is primarily an active service architecture, and the discussion in this paper refers to the management of active programming of intermediate servers. Figure 1 shows a schematic of a possible DPS management architecture.

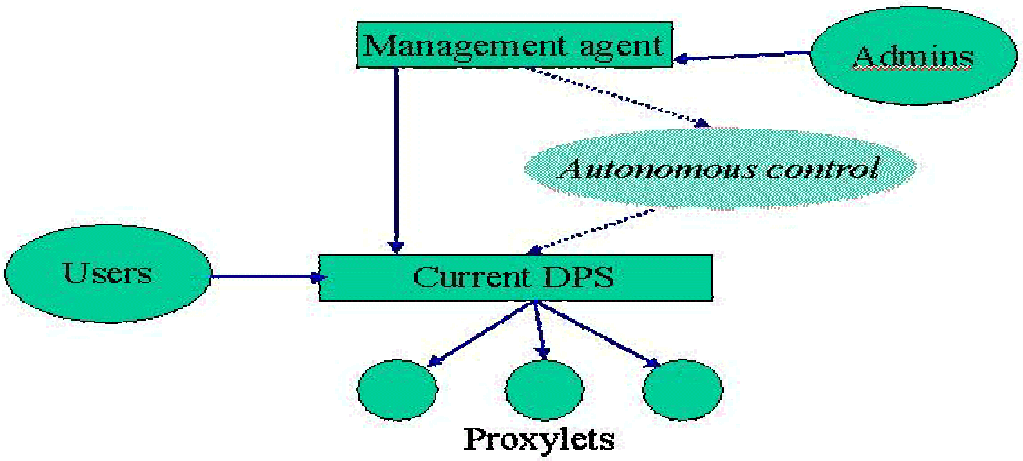

Figure 1. Schematic of proposed ALAN design

The DPS has an autonomous control system that performs management functions delegated to it via policies (scripts and pointers embedded in XML containers). Currently the control system supports a conventional management agent interface that can respond to high level instructions from system operators [21]. This interface is also open to use

by users (who can use it to run programs/active services) by adding a policy pointing to the location of their program and providing an invocation trigger. Typically the management policies for the program are included in an XML metafile associated with the code using an XML container, but users can also separately add management policies associated with their programs using HTTP post commands. In addition the agent can accept policies from other agents and export policies to other agents. This autonomous control system is intended to be adaptive.

Not shown in the figure are some low level controls required to enforce sharing of resources between users, and minimise unwanted interactions between users. There is a set of kernel level routines [22] that enforce hard scheduling of the system resources used by a DPS and the associated virtual machine that supports user supplied code. In addition the DPS requires programs to offer payment tokens before they can run. In principle the tokens should be authenticated by a trusted third party. At present these low level management activities are carried out using a conventional hierarchical approach. We hope to address adaptive control of the O/S kernel supporting the DPS in future work.

Genetic algorithms and bacteria

Genetic Algorithms involve the recombination and replication of a genotype that carries out a function or task. Their success at carrying out the chosen task is then assessed and related to the reproductive potency of the 'organism[1]'. A common approach (eg. [23]) begins by creating a set of non-interacting 'organisms' whose genomes have a fixed size. The fitness of each 'organism' is judged against a fixed metric (fitness

[1] In the context of a GA organism should not be interpreted as a biological term. We use quote marks to distinguish from more biological usage elsewhere in the paper

criterion), and the most fit members are allowed to combine pairwise and produce offspring, based on some mixture of their genotypes, which then replace the least fit members. The process is repeated until the summed fitness of the community stops increasing, at which point the most fit member is chosen as an optimum solution. Distributed GAs offer a less batch-style process by allowing 'organisms' to interact with an environment and/or each other in order to accumulate fitness. Reproduction in this approach is based on 'organisms' that have accumulated sufficient fitness to breed with the nearest neighbour who has also crossed the fitness threshold for breeding [24]. Both methods involve genetic fitness being passed onto the next generation, but the second is much more appropriate for Internet-based applications where global state is not known and evolution is continuous.

Bacteria are a set of metabolically diverse, simple, single-cell organisms [17]. Their internal structure is simpler than many other types of living cells, with no membrane-bound nucleus or organelles, and short circular DNA. This is because the genetic structure of bacteria a relatively simple. It has been demonstrated that only around 250 genes are required to code for an independent self-sustaining bacterium [25] and highly competent bacteria for whom the entire genome is known have only 2-3000 genes [26]. As a result bacteria can reproduce within 30 mins of 'birth'. In addition Bacterial evolution 'transcends Darwinism' [18]. While asexual reproduction ensures survival of the fittest, a more Lamarkian[2] mechanism of evolution occurs in parallel, with individuals capable of exchanging elements of their genetic material (plasmids) during their lifetime, in response to environmental stress, and passing the acquired characteristics to their

[2] Lamark was an 18th century French scientist who argued that evolution occurs because organisms can inherit traits, which have been acquired by their ancestors during their lifetime.

descendants. This exchange is known as plasmid migration. Plasmid migration allows much quicker reaction to sudden changes in influential environmental factors and can be modelled as a learning mechanism. Sustaining fitness in a complex changing fitness landscape has been shown to require evolution together with a fast learning mechanism [14]. It is vital that the learning is continuous, and does not require off-line training as with many neural net based approaches [27]. When a population of *E.Coli* (a common bacterium) is introduced to a new environment adaptation begins immediately, with significant results apparent in a few days (i.e. O(1000)[3] generations) [28]. Despite the rapid adaptation bacterial communities are also remarkably stable. Fossils of communities (stromatolites) that lived 3.5 billion years ago in Australia appear identical to present day communities living in Shark Bay on Australia's West coast. Bacteria thus have many of the properties (simplicity, resilience, rapid response to change) that are desirable for network entities. It seems it would be perfectly possible to simulate a single bacterial genome (2000 or so rules and an interpreter) on every single embedded processor embedded in possible future Internet devices

In similar biological systems such as protocists[4], where the probability of mutation occuring during the copying of a gene is around 1 in a billion adaptive evolution can occur within 1m generations (O(1000)yrs). GAs typically evolve much faster since the generation time is O(100)ms and the mutation rate is raised to 1 in a million but adaptation can still only deal with changes on a timescale of O(10)s. Using bacterial learning (plasmid interchange) in a GA will improve this to O(10)ms, and substantially improve its performance when faced with rapid change. Important properties of a

[3] O(10) means of the order of 10
[4] Unicellular organisms with nuclei

community of bacteria have been modeled very accurately using a distributed GA [29]. Qualitatively it seems clear that basing adaptive control of the Internet on a distributed GA, and using observations of bacterial behaviour to motivate improvements to the GA is likely to be a constructive research method. The objective of this paper is to provide some quantitative results that support this argument. Quantitative proof is the topic of ongoing work.

Algorithm Details.

Our proposed solution makes each DPS within the network responsible for its own behaviour. The active service network is modelled as a community of cellular automata. Each automaton is a single DPS that can run several programs (proxylets) requested by users. Each proxylet is considered to represent an instance of an active service. Each member of the DPS community is selfishly optimising its own (local) state, but 'selfishness' of this kind has been widely used as a basis for modeling biological systems [30], where commonly cooperation is an emergent property of coevolution (or competition) [31]. Partitioning into selfishly adapting sub-systems has also been shown to be a viable approach for the solving of other complex and non-linear problems [32].

In this paper we discuss results from an implementation that supports up to 10 active services. The control parameters given below are examples provided to illustrate our approach. Our current implementation has up to 1000 vertices connected on a rectangular grid (representing the network of transport links between the dynamic proxy servers). Each DPS has an amount of genetic material that codes for the rule set by which it lives. There is a set of rules that control the DPS behaviour. There is also a selection of

genes representing active services. These define which services each node will handle and can be regarded as pointers to the actual programs supplied by users. The service genes also encode some simple conditionals that must be satisfied for the service to run. Currently each service gene takes the form {x,y,z} where:

x. is a character representing the type of service requested (A-J)

y. is an integer between 0 and 200 which is interpreted as the value in a statement of the form "Accept request for service [Val(x)] if queue length < Val(y)".

z. is an integer between 0 and 100 that is interpreted as the value in a statement of the form "Accept request for service [Val(x)] if busyness < Val(z)% "

If either condition is not satisfied the service does not run

The system is initialised by populating a random selection of network vertices with DPSs (active nodes), and giving each DPS a random selection of the available service genes. Requests are then entered onto the system by injecting a random sequence of characters (representing service requests), at a mean rate that varies stochastically, at each vertex in the array. If the vertex is populated by a DPS, the items join a queue. If there is no DPS the requests are forwarded to a neighbouring vertex. The precise algorithm for this varies and is an active research area, however the results shown in this paper are based on randomly selecting a direction in the network and forwarding along that direction till a DPS is located. This is clearly sub-optimal but is easy to implement. In any case a different routing algorithm would not change the qualitative form of the main results since the algorithm is not predictive and makes no assumptions regarding the traffic pdf. In fact the traffic arriving at each DPS using this simple model shows some Long Range Dependency (LRD), but significantly less than real WWW traffic [5]. LRD

at the servers can be easily increased by adding a history dependency to the traffic sources, but this slows down the simulation without changing the results significantly, and has not been used in the results reported here. Each DPS evaluates the items that arrive in its input queue on a FIFO principle. If the request at the front of the queue matches an available service gene, and the network provider has specified a value for that service in the DPS control rules, the service will run. In the simulation the request is deleted and deemed to have been served, and the node is rewarded by the value of one execution of the service. This value is set by the operator to a number between 1 and 100 and should equal or exceed the temporal cost of the service (also a number between 1 and 100 in the simulation). In the simulations presented here cost and value were equal for all services. If there is no match the request is forwarded and no reward is given. Each DPS is assumed to have the same processing power, and can handle the same request rate as all the others. In the simulation time is divided into epochs (to enable independent processing of several requests at each DPS before forwarding rejected requests). An epoch allows enough time for a DPS to execute 3-4 service requests, or decide to forward 30-40 (i.e. forwarding incurs a small time penalty). An epoch is estimated to represent O(100)ms. The busyness of each DPS is calculated by combining the busyness at the previous epoch with the busyness for the current epoch in a 0.8 to 0.2 ratio, and is related to the revenue provided for processing a service request. For example, if the node has processed three requests this epoch (assume 25 points each) it would have 75 points for this epoch, if its previous cumulative busyness value was 65 then the new cumulative busyness value will be 67. This method dampens any sudden changes in behaviour. A brief schematic of this is shown in figure 2.

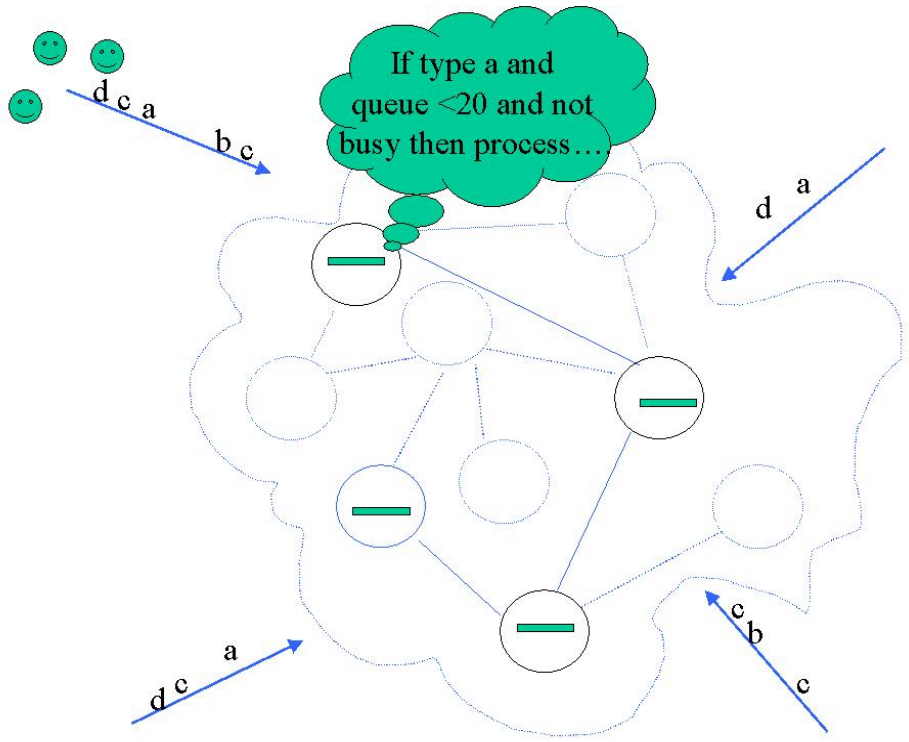

Figure 2 Future ALAN model. Live DPS nodes have a green bar, empty vertices are empty. Incoming requests (a,b,c,d) are routed to the nearest live DPS by an underlying transport network (not shown)

The DPS also has rules for reproduction, evolution, death and plasmid migration. It is possible to envisage each DPS as a bacterium and each request for a service as food. The revenue earned when a request is handled is then analagous with the energy released when food is digested. This analogy is consistent with the metabolic diversity of bacteria, capable of using various energy sources as food and metabolising these in an aerobic or anaerobic manner.

Genetic diversity is created in at least 2 ways, mutation and plasmid migration. Mutation involves the random alteration of just one value in a single service gene, for example:

"Accept request for service A if DPS < 80% busy" could mutate to to "Accept request for service C if DPS < 80% busy" or alternatively could mutate to "Accept request for service A if DPS < 60% busy".

Plasmid migration involves genes from healthy individuals being shed or replicated into the environment and subsequently being absorbed into the genetic material of less healthy individuals. If plasmid migration doesn't help weak strains increase their fitness they eventually die. If a DPS acquires more than 4-6 service genes through interchange the newest genes are repressed (registered as dormant). This provides a long term memory for genes that have been successful, and enables the community to successfully adapt to cyclic variations in demand. Currently, values for queue length and cumulative busyness are used as the basis for interchange actions, and evaluation is performed every five epochs. Although the evaluation period is currently fixed there is no reason why it should not also be an adaptive variable.

If the queue length or busyness is above a threshold (both 50 in this example), a random section of the active genome is copied into a 'rule pool' accessible to all DPSs. If a DPS continues to exceed the threshold for several evaluation periods, and there are vacant vertices, it replicates its entire genome into the nearest network vertex where a DPS is not present. Healthy nodes with a plentiful revenue supply thus reproduce by binary fission. Offspring produced in this way are exact clones of their parent.

If the busyness is below a different threshold (10), a service gene randomly selected from the rule pool is injected into the DPS's genome. If a DPS is 'idle' for several evaluation periods, its active genes are deleted, if dormant genes exist, these are

brought into the active domain, if there are no dormant genes the node is switched off. This is analogous to death by nutrient deprivation.

So if a node with the genome {a,40,50/c,10,5} has a busyness of >50 when analysed, it will put a random rule (e.g. c,10,5) into the rule pool. If a node with the genome {b,2,30/d,30,25} is later deemed to be idle it may import that rule and become {b,2,30/d,30,25/c,10,5}.

The algorithm described above is an example of a distributed GA and uses earned revenue as the fitness function.

Experiments

One of the key questions for ALAN management is proxylet placement. Given that the number of DPSs will be large, and the number of proxylets unbounded, the correct algorithm will be one that needs as little human/manual intervention as possible, as the manual optimisation of proxylet placement soon becomes untenable. Any automated algorithm must also deal with the fact that demand for services will vary in a chaotic manner as observed for the WWW [5]. Proxylets must thus replicate and distribute very rapidly, and also persist for long time periods. To demonstrate that our proposal can meet these requirements a simple experiment involving repeated increases in load on the simulated network of DPSs was carried out. In this experiment there was a fixed set of active services.

The average age (expressed in epochs) of all requests on the network was measured, this was used as our quality of service (QoS) measurement. The QoS was measured over time as the average load on the network was increased in a series of

significant steps, 1, 4, 12, 30, 45, 60 times the initial load. Figure 3 shows how the QoS reacts to these increases.

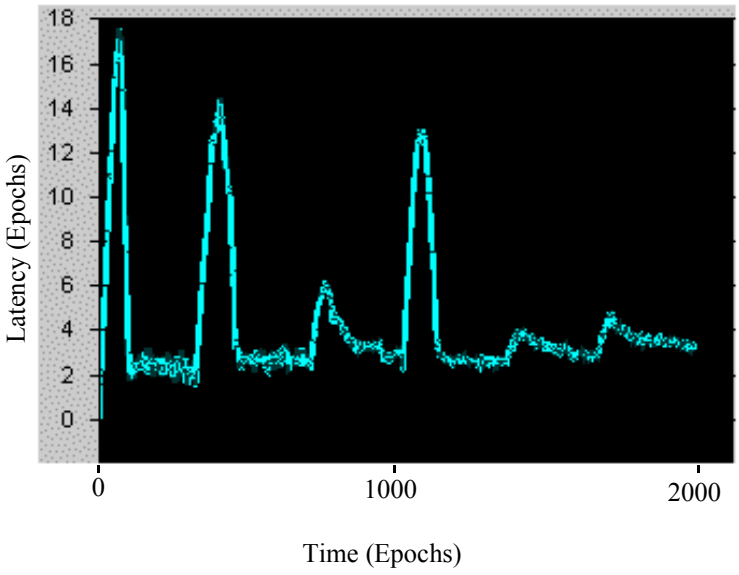

Figure 3. QoS reaction to sequential increases in load over time.

At 60 times the initial rate, the network was 95% loaded, yet performance had degraded very little. This shows the algorithm evolves efficient solutions. A short period of worsened service was observed, followed by a return to more acceptable QoS as the network adapted (by spawning more nodes and spreading genes for busy services to active nodes that were less busy). The system is reactive, so there will always be some delay in responding to increases in load. The noise in the plot primarily represents the response of the network to the noise introduced in the traffic model. The return to a stable, sufficient level of performance took ~50 epochs each time the load was increased.

This initial result was extremely encouraging so a more realistic experiment was performed.

An active services network must respond rapidly to the introduction of new services, enabling them to spread to wherever there is demand, whilst providing a stable quality of service for existing services. When a user develops a new proxylet, or an improved version of an existing proxylet, he should not be required to identify all the locations where it should be stored and/or run. Typically the user lacks both the time and the knowledge to make such a decision for himself and in any case cannot predict demand from other users of his program. At the same time if a user introduces a new service he should not be able to access his service until he has paid the appropriate fee.

Our second experiment illustrates the handling of new active services by the community of DPSs. The network was allowed to self-optimise to handle a set of 6 active services. A new service gene was introduced after 550 epochs, along with some demand for the service provided by the corresponding proxylet. Initially the network provider gave no value to the network for this new service request as the user had not yet paid. It can be seen that for 2 epochs some requests are handled (hence the dropping rate is momentarily >0) but the network soon fails to execute any of the requests for the new service. This is because any proxylets for this service are not earning their DPS any revenue, and their genes are therefore being replaced by more lucrative genes. This illustrates how the autonomous controls deal with the introduction of malicious code intended to defraud the operator. A reward is granted (the operator updates the DPS rule base from which all active DPSs inherit when payment is received from the users' bank) for the service after 1000 epochs. The request dropping rate soon decreases as the

proxylet spreads around the network. Within 200 epochs the service is being provisioned as successfully as any other (fig 4). Note the adaptation is slower than for existing services in the first experiment as the active service network has no memory of the new service and cannot adapt by reviving dormant genes.

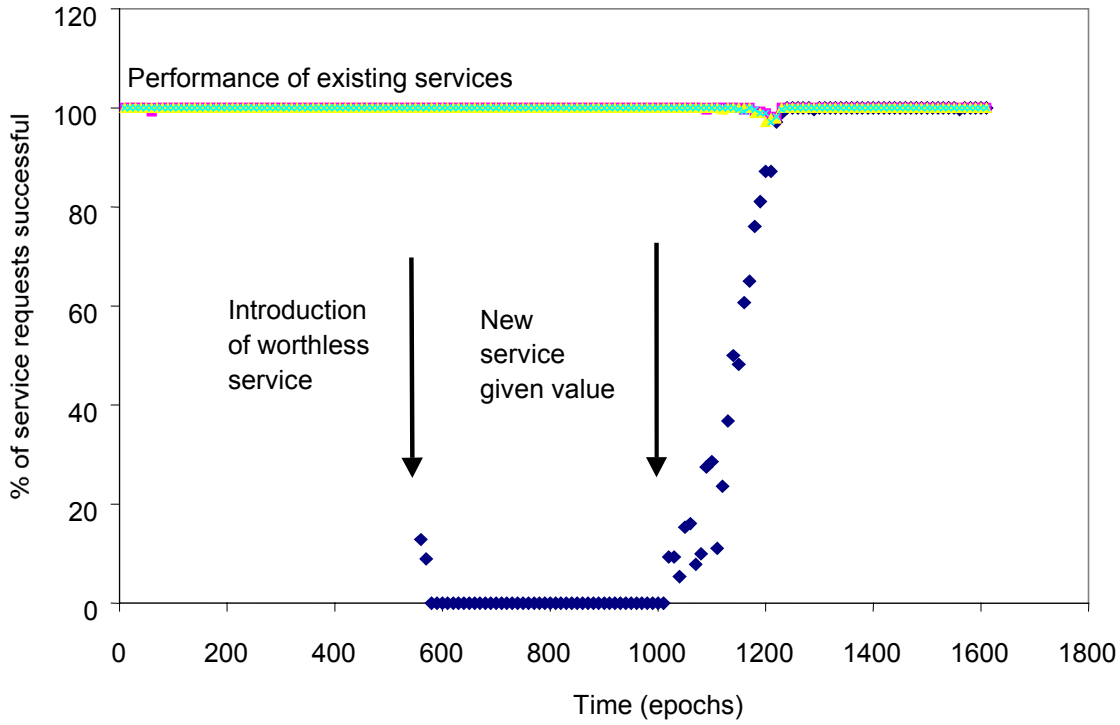

Figure 4. Service provisioning consequences of introducing a new service.

Figure 5 shows the results of the same experiment expressed in terms of handling latency. Initially no latency is given for the new service as no nodes are processing it (the latency is actually infinite, but this would not show on the graph). When the new service gene is introduced (this time the DPS rules indicating the revenue earned from handling are changed simultaneously) there is a period of high latency as the plasmid distributes around the network but soon its latency is similar to that for the other services. In an implemented network the revenue would ideally be a property of the service request.

This would eliminate operator intervention entirely. We have chosen to illustrate the case where the operator retains control to demonstrate that our proposal could easily be integrated with existing management schemes.

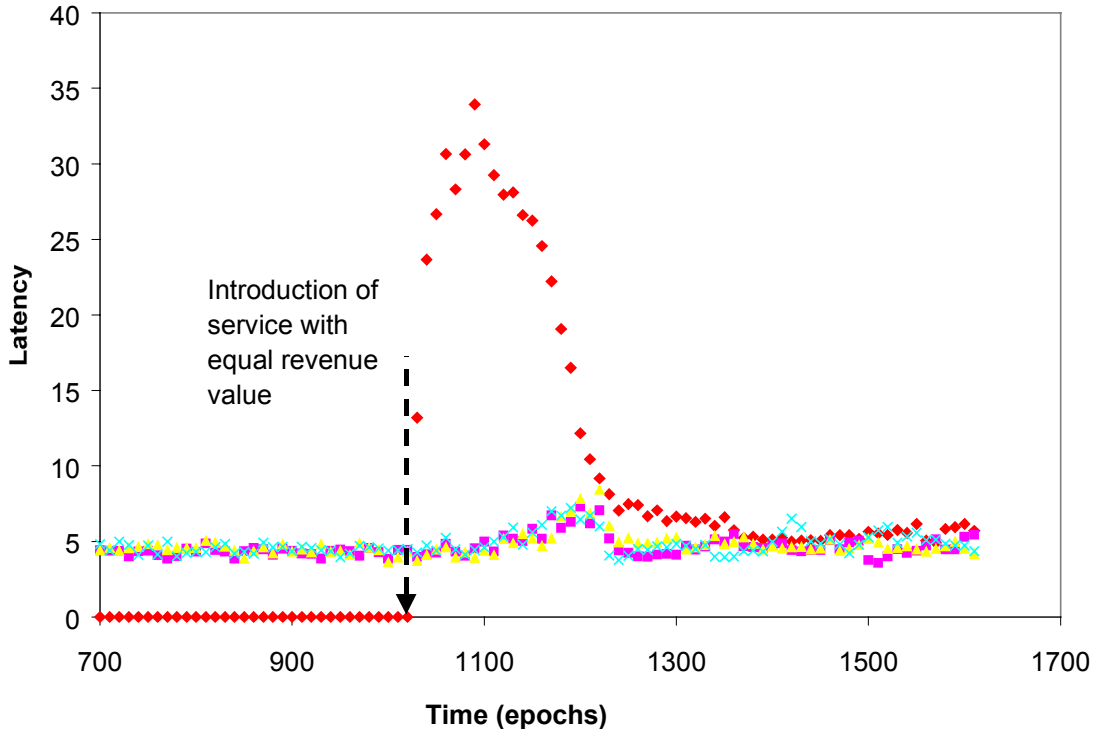

Figure 5. Latency Consequences of introducing a new service

The simple experiments we have shown illustrate that our algorithm can automate key aspects of performance, configuration, account and security management of the services in an active service network. It is also clear that the network will adaptively work around faults until they either die, or are manually repaired. Our approach has also shown it can deal with rapidly varying noisy traffic sources, without manual intervention, and can make effective use of the available resources, whilst adapting rapidly to major

changes. We feel confident that further work on the proposals we are making will prove both productive and rewarding

Conclusions.

Active networks will require extensive use of adaptive control techniques, particularly at the point where user supplied code has to be managed. The most obvious control point is the programmable device, which is the DPS in our ALAN approach. An autonomous adaptive control agent for Dynamic proxy servers in an active network has been proposed, based on a novel genetic algorithm inspired by observation of real bacterial communities. Simulations have shown that the proposed agent can handle key aspects of fault, configuration, account, performance and security management successfully in a large scale, dynamic environment. The agent is thus a promising approach that deserves further study and detailed comparison with other approaches.

References.

[1] M. Fry and A. Ghosh "Application Layer Active Networking" Computer Networks, 31, 7, pp. 655-667, 1999.
[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] 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.
[4] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poisson Modelling", IEEE/ACM Transactions on Networking, 3, 3, pp 226-244, 1995.
[5] 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.
[6] Y.Z. Tsypkin. "Adaptation and learning in automatic systems", Mathematics in Science and Engineering Vol 73, Academic press, 1971.
[7] G. DiCaro and M. Dorigo, "AntNet: Distributed stigmergic control for communications networks", J. Artificial Intelligence Research, 9, pp. 317-365, 1998.
[8] 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
[9] 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.
[10] L. Lewis, "A case based reasoning approach to the management of faults in telecommunications networks", Proc. IEEE conf. on Computer Communications (Vol 3), pp. 1422-29, San Francisco, 1993.

[11] J.H. Holland, "Adaptation in Natural and Artificial Systems" MIT press, 1992.
[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] R. Burkhart, "The Swarm Multi-Agent Simulation System", OOPSLA '94 Workshop on "The Object Engine", 7 September 1994.
[14] G. Booth, "Gecko: A Continuous 2D World for Ecological Modeling", Artificial Life, 3, pp. 147-163, Summer 1997.
[15] 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), Adison Wesley, 1993.
[16] I. Harvey, "The Microbial Genetic Algorithm", unbublished work, 1996, available at ftp://ftp.cogs.susx.ac.uk/pub/users/inmanh/Microbe.ps.gz
[17] S. Sonea and M. Panisset, "A new bacteriology" Jones and Bartlett, 1983.
[18] 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.
[19] 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.
[20] I.W. Marshall et. al., "Active management of multiservice networks", Proc. IEEE NOMS2000 pp981-3
[21] I.W. Marshall et. al. "Application Layer Programmable Internetwork Environment", British Telecom. Technol. J., 17, 2, pp 82-94, April 1999.
[22] D.G. Waddington and D. Hutchison, "Resource Partitioning in General Purpose Operating Systems, Experimental Results in Windows NT", Operating Systems Review, 33, 4, 52-74, Oct 1999.
[23] D. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1983.
[24] D. Whitely, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best", In Proc. Third International Conference on Genetic Algorithms (ed. J.D. Schaffer), pp.116-121, Morgan Kaufman, 1989.
[25] Hutchison CA, Peterson SN, Gill SR, Cline RT, White O, Fraser CM, Smith HO, Venter JC, " Global transposon mutagenesis and a minimal Mycoplasma genome", Science, 286, pp2165-9, Dec 10 1999
[26] H.Ochman, J.G.Lawrence and E.A.Groisman, " Lateral gene transfer and the nature of bacterial innovation", Nature, 405, pp299-305, May 18 2000
[27] C.M. Roadknight, G.R. Balls, D. Palmer-Brown and G.E. Mills, "Modelling of complex environmental data", IEEE Transactions on Neural Networks, 8, 4, pp 852-862, 1997.
[28] R.E. Lenski and M. Travisano, "Dynamics of Adaptation and Diversification", Proc. Nat Acad. Sci. 91, pp 6808-6814, 1994.
[29] J.U. Kreft, G. Booth, and J.W.T. Wimpenny, "BacSim, a simulator for individual-based modelling of bacterial colony growth", Microbiology, 144, pp. 3275-3287, 1997.
[30] R. Dawkins, "The Selfish Gene", Oxford University Press, 1976.
[31] L.Margulis and L.Olendzenski, "Environmental evolution", MIT Press 1992
[32] S. Kauffman, "The origins of order", Oxford University Press 1993