ELSEVIER SCIENCE B.V. [DTD 4.1.0]
JOURNAL COMPNW ARTICLE No. 2466
PAGES 01-11 DISPATCH 7 February 2001
COMPNW 2466 A
PROD. TYPE: FROM Disk

ELSEVIER
Computer Networks 000 (2001) 000–000
COMPUTER NETWORKS
www.elsevier.com/locate/comnet

Provision of quality of service for active services

Ian W Marshall, Chris Roadknight *

BT Adastral Park, BT Research Laboratories, Room 148 B54, Martlesham Heath, Ipswich, Suffolk IP5 3RE, UK

Abstract

A novel approach to quality of service control in an active service network (application layer active network) is described. The approach makes use of a distributed genetic algorithm based on the unique methods that bacteria use to transfer and share genetic material. We have used this algorithm in the design of a robust adaptive control system for the active nodes in an active service network. The system has been simulated and results show that it can offer clear differentiation of active services. The algorithm places the right software, at the right place, in the right proportions; allows different time dependencies to be satisfied and simple payment related increases in performance. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Network management; Genetic algorithms; ALAN

1. Introduction

To be popular with customers an active service platform must provide some clear service quality assurances. Users of an active service network supply the programs and policies required for their custom services in transport packets alongside their data. Clearly it should be possible for these users to specify the Quality of Service (QoS) using any metric that is important to them. The rate of loss of packets carrying service requests or policies, and the service response time (latency) are two obvious examples. In this paper we discuss the management of QoS in an application layer active network (ALAN) [1] that enables users to place software (application layer services) on servers embedded in the network. Despite the obvious virtual networking overheads, the resulting end to end service performance will often be significantly better than if the services executed in the user's end systems (as at present). For example, a network based conference gateway can be located so as to minimise the latency of the paths used in the conference, whereas an end system based gateway will usually be in a sub-optimal location.

For the purposes of this work we have assumed that the latency and loss associated with the network based servers is significantly greater than the latency and loss associated with the underlying network. In the case of latency this is clear – packet handling times in broadband routers are around 10 μs, whilst the time taken to move a packet into the user space for application layer processing is a few milliseconds. In the case of loss the situation is less clear since currently servers do not drop requests, they simply time-out. However, measurement of DNS lookup [2] suggest DNS time-outs due to server overloads occur significantly more frequently than DNS packet losses, so we feel our assumption is reasonable.

* Corresponding author.
E-mail addresses: ian.w.marshall@bt.com (I. W Marshall), christopher.roadknight@bt.com (C. Roadknight).

1389-1286/00/$ - see front matter © 2001 Elsevier Science B.V. All rights reserved.
PII: S1389-1286(01)00156-6

UNCORRECTED PROOF

In Section 2 we briefly describe our active services platform ALAN and its associated management system. We then justify our approach to QoS in an ALAN environment. We then describe a novel control algorithm, which can control QoS in the desired manner. Finally we show the results of some simulations using the novel algorithm. The results are very encouraging and illustrate for the first time that a distributed AI approach may be a productive QoS management tool in an active services network. However, further work is required before we can justify the use of our approach in a working active network.

2. ALAN

ALAN [1] is based on users supplying java based active code (proxylets) that runs on edge systems (dynamic proxy server – DPS) provided by network operators. Messaging uses HTML/XML and is normally carried over HTTP. There are likely to be many DPSs at a physical network node (at least one for each service provider using the 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. Fig. 1 shows a schematic of a possible DPS management architecture.

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 [3]. 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 [4,5], 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.

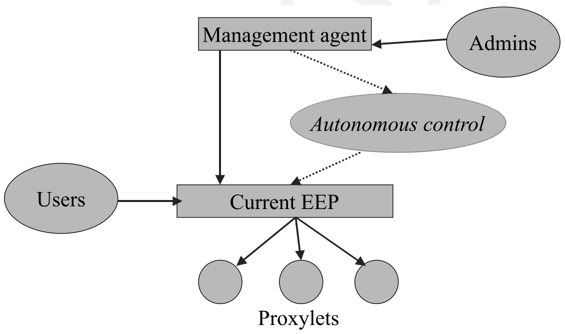


Fig. 1. Schematic of proposed ALAN design.

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 [6] 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.

3. Network level QoS

Currently there is great interest in enabling the Internet to handle low latency traffic more reliably than at present. Many approaches, such as intserv [7], rely on enabling the network to support some type of connection orientation. This matches the properties of older network applications, such as telephony, well. However it imposes an unacceptable overhead on data applications that generate short packet sequences. Given that traffic forecasts indicate that by the end of the next decade telephony will be ≈5% of total network traffic, and short data sequences will be around 50% of network traffic, it does not seem likely that connection orientation will deliver optimal results.

A recent alternative has been to propose differentiated services [8], an approach that is based on using different forwarding rules for different classes of packet, and maintaining the properties of the best class by admission control at the ingress to the network. There are difficulties however.

- Admission control does not work well with short packet sequences [9].
- The proposed algorithms assume Poisson burst intervals when real traffic is in fact fractional Gaussian [10,11] and much harder to predict.
- The performance benefits can only be obtained if the distribution of demand is such that only a small proportion of the traffic wishes to use the better classes [12].
- The proposed classes typically propose a low loss, low latency class that uses a disproportionate proportion of the available network resources.

Despite the difficulties it is clear that differentiated services is currently the best available alternative. It therefore seems advisable to base any proposals for QoS management of active services on the diffserv approach. However, it also seems advisable to modify the approach and attempt to avoid some of the difficulties identified.

4. Emergent approach to differentiated active services

We propose a new approach to differentiating active services, controlled by an emergent control algorithm. Users can request low latency at the cost of high loss, moderate latency and loss, or high latency and low loss by adjusting the time to live (ttl) of the packets they send, either by manipulating the IP header or using a user defined header extension. Short ttl packets will experience high loss when the network is congested and long ttl packets will experience high delay when the network is congested. Users cannot request low loss and low delay together. This choice means that all the classes of service we support have approximately the same resource cost, since the low latency class does not rely on low utilization and we can set the utilization to be the same for all the service classes. As a result we do not have to consider complex admission control to ensure a favourable demand distribution, and we do not have to allocate significant resources to support a minority service. Two adaptations are possible if the performance is reduced by congestion; either the application sends less packets or the application persists until an application specific latency cut-off is reached and then terminates the session. Services such as telephony would use a low latency/high loss transport regime. This would require the application to be more loss tolerant than at present, however as mobile telephones demonstrate this is not hard to achieve. Interoperation with legacy telephones could be achieved by running loss tolerant algorithms (e.g., FEC) in the PSTN/IP gateway. We do not believe that users want an expensive low loss, low latency service. The current PSTN exemplifies this service and users are moving to VoIP as fast as they are able, despite lower quality, in order to benefit from reduced prices.

Near optimal end to end performance across the network is obtained by enabling the servers to retain options in their application layer routing table for fast path, medium path and slow path (i.e., high loss medium loss and low loss). Packets are then quickly routed to a server whose performance matches their ttl. This avoids any need to perform flow control and force sequences of packets to follow the same route.

For this approach to work well the properties of the servers must adapt to local load conditions. Fast servers have short queues and high drop probabilities, slow servers have long queues and low drop probabilities. If most of the traffic is low latency the servers should all have short buffers and if most of the demand is low loss the servers should have long buffers. Adaptation of the buffer length can be achieved using an adaptive control mechanism [13], and penalising servers whenever a packet in their queue expires. Use of adaptive control has the additional advantage that it makes no assumptions about traffic distributions, and will work well in a situation where the traffic has significant long range dependency (LRD). This

UNCORRECTED PROOF

then resolves the final difficulty we noted with the current network level diffserv proposals.

5. Adaptive control

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, and statistical predictions. All existing forms of traffic management are based on stochastic control, typically assuming Poisson statistics.

Adaptive control [13] 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 selecting some actions that the system has learnt are useful using some 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 [14], security [15,16], and fault ticketing [17]. As far as we are aware the work presented here is the first application of distributed adaptive control to service configuration and management.

Holland [18] has shown that genetic algorithms (GAs) offer a robust approach to evolving effective adaptive control solutions. More recently many authors [19–21] 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 [22], 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 [23] pointed out that gene interchange (as observed in bacteria [24,25]) could provide the rapid learning required. This was recently demonstrated by Furuhashi [26] for a bounded, globally optimised GA. In previous work [27] we have demonstrated that a novel unbounded, distributed GA with "bacterial learning" is an effective adaptive control algorithm for the distribution of services in an active service provision system derived from the ALAN. In this paper we demonstrate for the first time that our adaptive control algorithm can deliver differentiated QoS in response to user supplied metrics.

6. 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 this 'selfishness' has been proven as a stable model for living organisms [28]. Partitioning a system into selfishly adapting sub-systems has been shown to be a viable approach for the solving of complex and non-linear problems [29].

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 1000 vertices connected on a rectangular grid (representing the network of transport links between the DPSs). Each vertex can support a single server (i.e., host) supporting a

UNCORRECTED PROOF

single DPS, so the network can support up to 1000 DPS nodes. In reality a network node (router) would be associated with many such hosts, possibly organised as a cluster. In this work we are assuming that the latency associated with a DPS is significantly greater than that associated with bit transport so we do not distinguish between DPS links that are local and DPS links that are remote. 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. Each node can hold up to eight services (the limit is similar to that imposed by available RAM in commodity computers, such as could be used in future server clusters). 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)% ”.

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 here 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. The traffic arriving at each DPS using this model shows some LRD, but significantly less than real WWW traffic. Increasing the degree of LRD would be straightforward. However, the necessary change involves additional memory operations that slows down the simulation and makes the results harder to interpret. In any case inclusion of significant LRD would not change the qualitative form of the main results since the algorithm is not predictive and makes no assumptions regarding the traffic pdf. 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 customer has included payment tokens equal to (or greater than) the cost 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 a value equal to the specified cost of the service. If there is no match the request is forwarded and no reward is given. In this case the forwarding is informed by a state table maintained by the DPS using a node state algorithm. Packets with a short ttl are forwarded to a DPS with a short queue and packets with a long ttl are forwarded to a DPS with a long queue. 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 contains 100 time units and 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 0.8–0.2 ratio, and is related to the revenue provided for processing a service request. For example, if the node has processed three requests this epoch (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 Fig. 2.

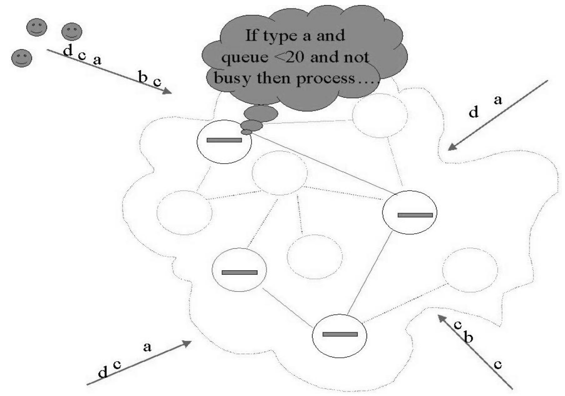


Fig. 2. Future network model.

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 two 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 "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 does not 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 genome is copied into a 'rule pool' accessible to all DPSs. If a DPS continues to exceed the threshold for several evaluation periods, it replicates its entire genome into an adjacent network vertex where a DPS is not present. Healthy bacteria with a plentiful food 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}.

7. Experiments

The basic traffic model outlined above was adjusted to enable a range of ttls to be specified. The ttls used were 4, 7, 10, 15, 20, 25, 30, 40, 50, 100 (expressed in epochs). Approximately the same number of requests were injected at each ttl. The DPS nodes were also given an extra gene coding for queue length, and penalised by four time units whenever packets in the queue were found to have timed out. A DPS with a short queue will handle packets with a short ttl more efficiently since the ttl will not be exceeded in the queue and the DPS will not be penalised for dropping packets. Thus if local demand is predominantly for short ttl DPS nodes with short queues will replicate faster, and a colony of short queue nodes will develop. The converse is true if long ttl requests predominate. If traffic is mixed a mixed community will develop. In

UNCORRECTED PROOF

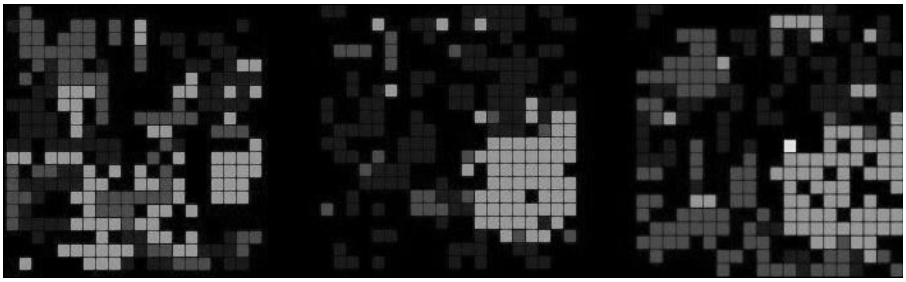


Fig. 3. Distribution of DPS nodes with short medium and long queues at three different times.

Fig. 3 the red dots represent DPS nodes with long queues, the blue dots represent intermediate queues and the green dots represent short queues. It is clear that the distribution of capability changes over time to reflect the distribution of demand, in the manner described above.

In Fig. 4 we show the average request drop rate across the network of bacteria illustrated in Fig. 3, and compare the performance with a number of alternative methods of distributing the active services. The alternatives are:

- Random static placement of services at network nodes.
- Caching of requested services with a random replacement algorithm (Cache I).
- Caching of requested services using a least recently used replacement algorithm (Cache II).

The tests were performed at loads of 10% (low), 40% (medium) and 80% (high). At low loads all the algorithms offer similar performance levels. As might be expected, at medium and high load our algorithm is a significant improvement over random placement. More surprisingly it also significantly outperforms caching. We believe this is due to the small size of the caches. Each cache holds up to eight services (i.e., the same as the bacteria). This is intended to represent the number of proxylets that can be held in the RAM of a low spec PC, such as might be used in a commodity based cluster at a network server farm. Since the load time for proxylets is currently long (~1 s) we do not model disk based caching.

Fig. 5 shows the average end to end latency experienced by service requests in our modelled network, and compares it with the latency experienced using the alternative active service distribution mechanisms listed above. As before the adaptive bacterial approach is as good as the other alternatives at low loads, and is clearly an improvement over the best alternative (standard LRU based caching – CacheII) at medium and high loads. We are therefore confident that our algorithm is delivering a useful level of performance.

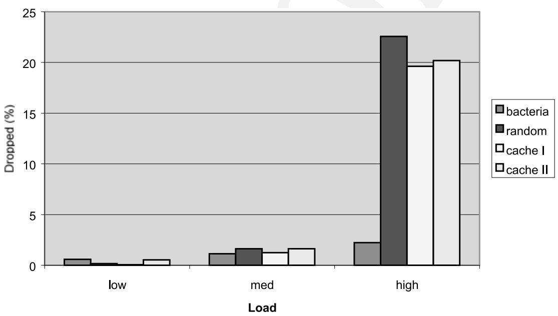


Fig. 4. Request drop rates for different distribution mechanisms.

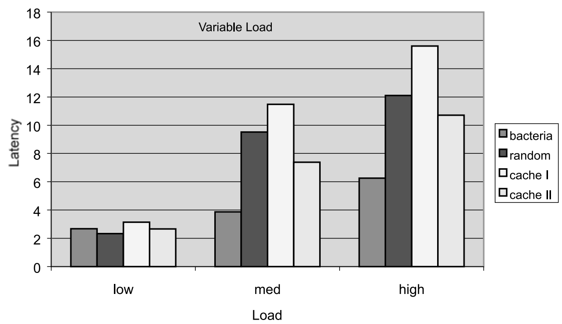


Fig. 5. Average latency of several approaches to distributing active services.

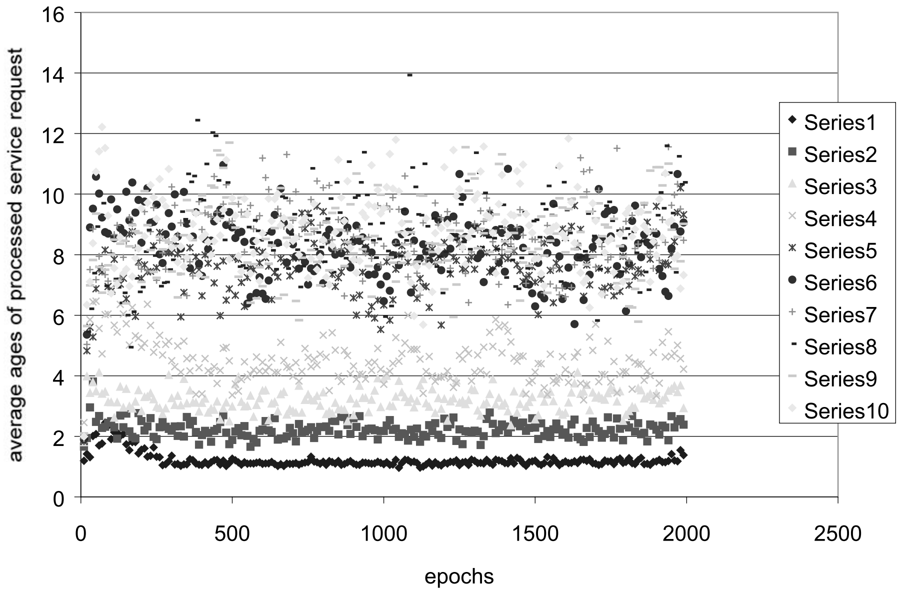


Fig. 6. Different latencies for requests with differing times to live.

Fig. 6 illustrates the differentiated QoS delivered by the network of DPS nodes. The time taken to process each request is shown on the y access and the elapsed system time is shown on the x axis. It can be seen that the service requests with shorter times to live are being handled faster than those with a longer time to live, as expected. Fig. 7 shows the expected corrollary. More service requests with

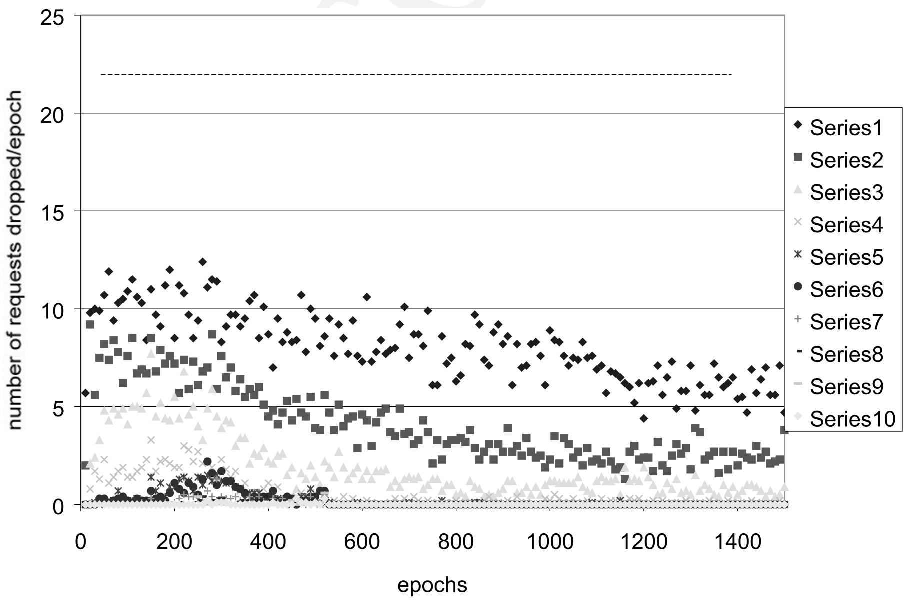


Fig. 7. Different dropping rates for requests with differing times to live.

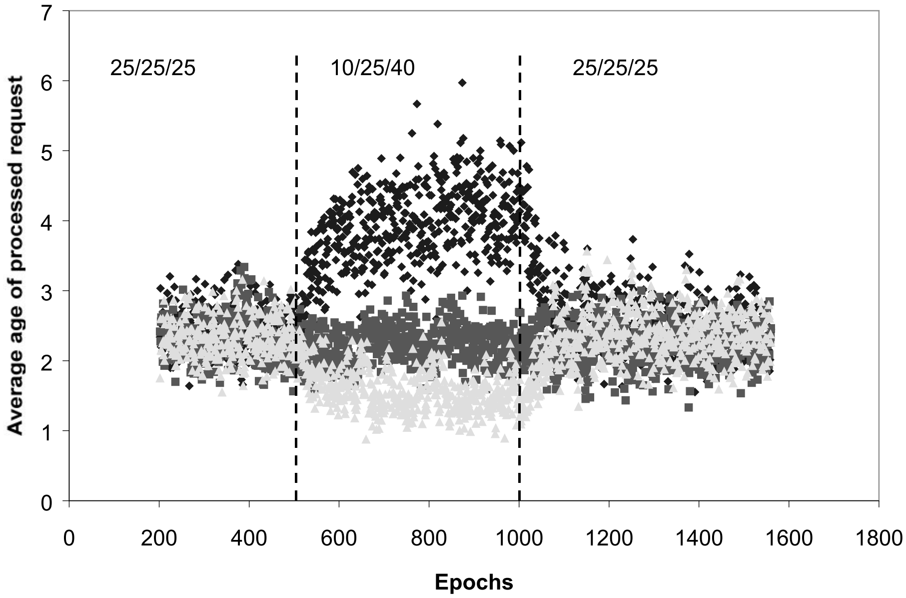


Fig. 8. Effects of different charging levels on age related QoS.

short ttls are being dropped. This is due to them timing out, and is the essential down-side to specifying a short ttl. Although the numbers of requests at each ttl value are roughly equal, fewer short ttl requests are handled.

In addition to varying the latency and loss associated with service requests users may also wish

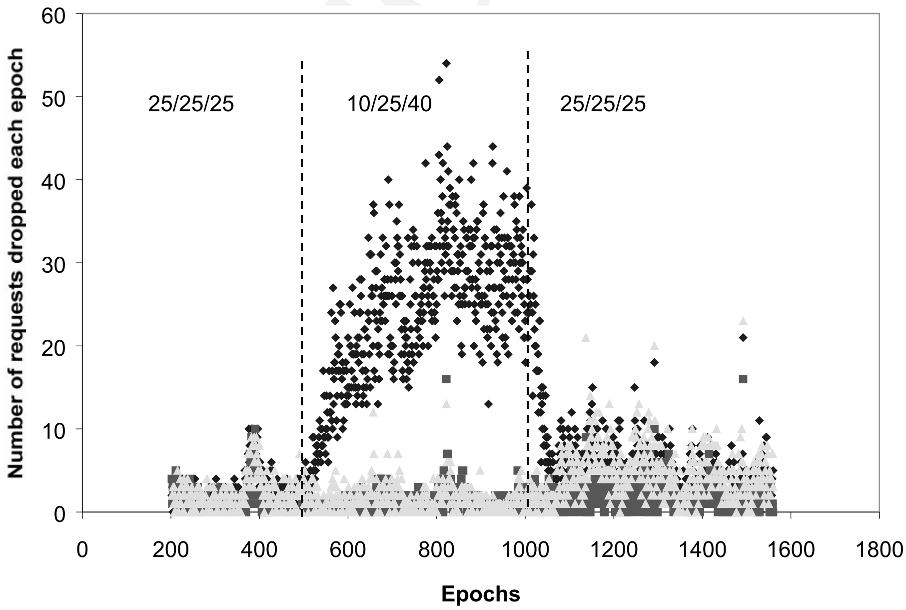


Fig. 9. Effects of different charging levels on dropping of requests.

to vary the price they are willing to pay. In the basic algorithm it was assumed that the network provider allocated a reward to each DPS for processing a service request. We investigated the impact of allowing the DPS to collect a greater reward. In the modified model the DPS is rewarded by the amount of tokens the user includes with the request. The traffic input was adjusted so that requests for different services carried different amounts of payment tokens. Initially the DPS nodes were rewarded equally (25 'tokens') for each of three services A, B and C. After 500 epochs the rate of reward is changed so that each DPS is rewarded four times as much for processing service C (40 tokens) as it is for processing service A (10 tokens), with B staying at 25. This is equivalent to offering users a choice of three prices for a single service. Fig. 8 shows the latency of service requests for the three different service types.

It is apparent that within 100 epochs the average latency for providing service C is reduced while the latency for A is increased. Fig. 9 shows that requests for service A are also dropped (due to timing out) more than requests for service B and C. Before the change in reward the numbers of DPSs handling each service were similar. After the reward rate change the plasmids for handling services C and B have spread much more widely around the network at the expense of the plasmid for the relatively unrewarding service A. After 1000 epochs the rate of requests for all three services was returned to the original state. It can be seen, in both figures, that equality in quality of service, both in terms of loss rate and latency, quickly returned.

These last results indicate that the control method could potentially be used for a range of user specified parameters. We see no reason why other parameters of interest could not be added to the model, and are very encouraged by the initial results. In particular we note that the latencies and loss rates are comparable to those obtained in many conventional approaches to differentiated services, but many of the difficulties concerning admission control have been avoided.

8. Conclusions

Our initial results show that the long-term self-stabilising, adaptive nature of bacterial communities are well suited to the task of creating a stable community of autonomous active service nodes that can offer consistent end to end QoS across a network. The methods used for adaptation and evolution enable probabilistic guarantees for metrics such as loss rate and latency similar to what can be achieved using more conventional approaches to differentiated services.

References

[1] M. Fry, A. Ghosh, Application layer active networking, Computer Networks 31 (7) (1999) 655–667.

[2] R. Giaffreda, N. Walker, R.Y. Ruiz, Performance of the DNS name resolution infrastructure, in: Proceedings of the IEE colloquium on Control of Next Generation Networks, London, October 1999.

[3] I.W. Marshall, J. Hardwicke, H. Gharib, M. Fisher, P. Mckee, Active management of multiservice networks, in: Proceedings of the IEEE NOMS2000, pp. 981–983.

[4] P. Mckee, I.W. Marshall, Behavioural specification using XML, in: Proceedings of the IEEE FTDCS '99, Capetown, pp. 53–59.

[5] I.W. Marshall, M. Fry, L. Velasco, A. Ghosh, Active information networks and XML, in: S. Covaci (Ed.), Active Networks, LNCS 1653, Springer, Berlin, 1999, pp. 60–72.

[6] D.G. Waddington, D. Hutchison, Resource partitioning in general purpose operating systems, experimental results in Windows NT, Operating Systems Review 33 (4) (1999) 52–74.

[7] IETF Intserv charter: http://www.ietf.org/html.charters/intserv-charter.html.

[8] IETF Diffserv charter: http://www.ietf.org/html.charters/diffserv-charter.html.

[9] V. Jacobson, Support for differentiated services in router designs, in: Proceedings of the Network Modelling in the 21st Century, Royal Society, December 1999.

[10] V. Paxson, S. Floyd, Wide area traffic: the failure of poisson modelling, IEEE/ACM Transactions on Networking 3 (3) (1995) 226–244.

[11] M. Crovella, A. Bestavros, Self-similarity in world wide web traffic: evidence and possible causes, IEEE/ACM Transactions on Networking 5 (6) (1997) 835–846.

[12] R. Gibbens, S.K. Sargood, F.P. Kelly, H. Azmoodeh, R. MacFadyen, N. MacFadyen, An approach to service level agreements for IP networks with differentiated services, Philosophical Transactions of the Royal society A, 2000,

UNCORRECTED PROOF

submitted, and available at http://www.stat-slab.cam.ac.uk~richard/research/papers/sla/.

[13] Y.Z. Tsypkin, Adaptation and learning in automatic systems, Mathematics in Science and Engineering, vol. 73, Academic press, New York, 1971.

[14] G. DiCaro, M. Dorigo, AntNet: distributed stigmergic control for communications networks, Journal of Artificial Intelligence Research 9 (1998) 317–365.

[15] D.A. Fisher, H.F. Lipson, Emergent algorithms – a new method of enhancing survivability in unbounded systems, in: Proceedings of the 32nd Hawaii International Conference on System Sciences, IEEE, 1999.

[16] M. Gregory, B. White, E.A. Fisch, U.W. Pooch, Cooperating security managers: a peer based intrusion detection system, IEEE Network 14 (4) (1996) 68–78.

[17] L. Lewis, A case based reasoning approach to the management of faults in telecommunications networks, in: Proceedings of the IEEE conference on Computer Communications, vol. 3, San Francisco, 1993, pp. 1422–1429.

[18] J.H. Holland, Adaptation in Natural and Artificial Systems, MIT press, Cambridge, MA, 1992.

[19] S. Forrest, T. Jones, Modeling complex adaptive systems with echo, in: R.J. Stonier, X.H. Yu (Eds.), Complex Systems Mechanisms of Adaptation, IOS Press, 1994, pp. 3–21.

[20] R. Burkhart, The swarm multi-agent simulation system, OOPSLA '94 Workshop on The Object Engine, September 7, 1994.

[21] J.U. Kreft, G. Booth, J.W.T. Wimpenny, BacSim, a simulator for individual-based modelling of bacterial colony growth, Microbiology 144 (1997) 3275–3287.

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

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

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

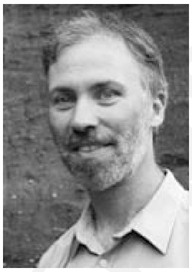
[25] D.E. Caldwell, G.M. Wolfaardt, D.R. Korber, J.R. Lawrence, Do bacterial communities transcend darwinism? Advances in Microbial Ecology 15 (1997) 105–191.

[26] N.E. Nawa, T. Furuhashi, T. Hashiyama, Y. Uchikawa, A study on the relevant fuzzy rules using pseudo-bacterial genetic algorithm, in: Proceedings of the IEEE International Conference on Evolutionary Computation, 1997.

[27] I.W. Marshall, C. Roadknight, Adaptive management of an active services network, British Telecommunication Technology Journal 18 (4) (2000) 78–84.

[28] R. Dawkins, The Selfish Gene, Oxford University Press, Oxfoxd, 1976.

[29] S. Kauffman, The Origins of Order, Oxford University Press, Oxford, 1993.



Ian Marshall is a senior research advisor at BT and visiting Professor of Telecommunications at South Bank University. Since 1997 he has been leading research on Ad-Hoc Systems, including active networks. This work currently includes the Alpine Research Initiative funded by BT, the fifth framework project Android and the Eurescom project Caspian, together with some smaller university research contracts. Previously he worked in optical networks, broadband networks, network strategy and distributed systems. He is a chartered engineer, a fellow of the Institute of Physics and the British Computer Society, a senior member of IEEE and a member of the ACM. He serves on several institute committees, on EPSRC and European research panels, and on numerous programme committees.

Chris Roadknight holds a B.Sc. in Biological Sciences and a M.Sc. in Computer Sciences from Manchester University and was recently awarded a Doctorate for his thesis on 'Transparent Neural Network Data Modelling' by Nottingham Trent University. He joined BT in 1997 and initially worked on characterising WWW requests and user behaviour, with applications to proxy cache performance modelling. More recently his work within the programmable networks lab has focused on artificial-life based solutions for active network management.

UNCORRECTED PROOF