



Kent Academic Repository

Marshall, Ian W. and Roadknight, Chris (2001) *Provision of quality of service for active services*. Computer Networks, 36 (1). pp. 75-87. ISSN 1389-1286.

Downloaded from

<https://kar.kent.ac.uk/13606/> The University of Kent's Academic Repository KAR

The version of record is available from

[https://doi.org/10.1016/S1389-1286\(01\)00156-6](https://doi.org/10.1016/S1389-1286(01)00156-6)

This document version

UNSPECIFIED

DOI for this version

Licence for this version

UNSPECIFIED

Additional information

Versions of research works

Versions of Record

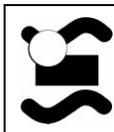
If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in *Title of Journal*, Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our [Take Down policy](https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies) (available from <https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies>).



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 μ s, 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).

53 In Section 2 we briefly describe our active ser- 84
 54 vices platform ALAN and its associated manage- 85
 55 ment system. We then justify our approach to QoS 86
 56 in an ALAN environment. We then describe a 87
 57 novel control algorithm, which can control QoS 88
 58 in the desired manner. Finally we show the results of 89
 59 some simulations using the novel algorithm. The 90
 60 results are very encouraging and illustrate for the 91
 61 first time that a distributed AI approach may be a 92
 62 productive QoS management tool in an active 93
 63 services network. However, further work is re- 94
 64 quired before we can justify the use of our ap- 95
 65 proach in a working active network. 96

66 2. ALAN

67 ALAN [1] is based on users supplying java 100
 68 based active code (proxylets) that runs on edge 101
 69 systems (dynamic proxy server – DPS) provided by 102
 70 network operators. Messaging uses HTML/XML 103
 71 and is normally carried over HTTP. There are 104
 72 likely to be many DPSs at a physical network node 105
 73 (at least one for each service provider using the 106
 74 node). It is not the intention that the DPS is able 107
 75 to act as an active router. ALAN is primarily an 108
 76 active service architecture, and the discussion in 109
 77 this paper refers to the management of active 110
 78 programming of intermediate servers. Fig. 1 shows 111
 79 a schematic of a possible DPS management archi- 112
 80 tecture. 113

81 The DPS has an autonomous control system 114
 82 that performs management functions delegated to 115
 83 it via policies (scripts and pointers embedded in

XML containers). Currently the control system 84
 supports a conventional management agent inter- 85
 face that can respond to high level instructions 86
 from system operators [3]. This interface is also 87
 open to use by users (who can use it to run pro- 88
 grams/active services) by adding a policy pointing 89
 to the location of their program and providing an 90
 invocation trigger. Typically the management 91
 policies for the program are included in an XML 92
 metafile associated with the code using an XML 93
 container [4,5], but users can also separately add 94
 management policies associated with their pro- 95
 grams using HTTP post commands. In addition 96
 the agent can accept policies from other agents and 97
 export policies to other agents. This autonomous 98
 control system is intended to be adaptive. 99

Not shown in the figure are some low level 100
 controls required to enforce sharing of resources 101
 between users, and minimise unwanted interac- 102
 tions between users. There is a set of kernel level 103
 routines [6] that enforce hard scheduling of the 104
 system resources used by a DPS and the associated 105
 virtual machine that supports user supplied code. 106
 In addition the DPS requires programs to offer 107
 payment tokens before they can run. In principle 108
 the tokens should be authenticated by a trusted 109
 third party. At present these low level management 110
 activities are carried out using a conventional hi- 111
 erarchical approach. We hope to address adaptive 112
 control of the o/s kernel supporting the DPS in 113
 future work. 114

115 3. Network level QoS

Currently there is great interest in enabling the 116
 Internet to handle low latency traffic more reliably 117
 than at present. Many approaches, such as intserv 118
 [7], rely on enabling the network to support some 119
 type of connection orientation. This matches the 120
 properties of older network applications, such as 121
 telephony, well. However it imposes an unaccept- 122
 able overhead on data applications that generate 123
 short packet sequences. Given that traffic forecasts 124
 indicate that by the end of the next decade tele- 125
 phony will be $\approx 5\%$ of total network traffic, and 126
 short data sequences will be around 50% of net- 127

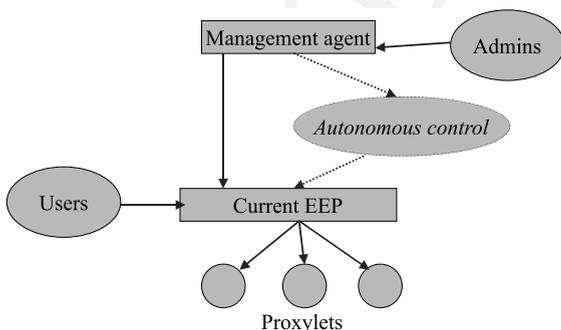


Fig. 1. Schematic of proposed ALAN design.

128 work traffic, it does not seem likely that connec-
129 tion orientation will deliver optimal results.

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

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

149 Despite the difficulties it is clear that differenti-
150 ated services is currently the best available alter-
151 native. It therefore seems advisable to base any
152 proposals for QoS management of active services
153 on the diffserv approach. However, it also seems
154 advisable to modify the approach and attempt to
155 avoid some of the difficulties identified.

156 4. Emergent approach to differentiated active ser- 157 vices

158 We propose a new approach to differentiating
159 active services, controlled by an emergent control
160 algorithm. Users can request low latency at the
161 cost of high loss, moderate latency and loss, or
162 high latency and low loss by adjusting the time to
163 live (ttl) of the packets they send, either by ma-
164 nipulating the IP header or using a user defined
165 header extension. Short ttl packets will experience
166 high loss when the network is congested and long
167 ttl packets will experience high delay when the
168 network is congested. Users cannot request low
169 loss and low delay together. This choice means
170 that all the classes of service we support have ap-
171 proximately the same resource cost, since the low
172 latency class does not rely on low utilization and

we can set the utilization to be the same for all the 173
service classes. As a result we do not have to 174
consider complex admission control to ensure a 175
favourable demand distribution, and we do not 176
have to allocate significant resources to support a 177
minority service. Two adaptations are possible if 178
the performance is reduced by congestion; either 179
the application sends less packets or the applica- 180
tion persists until an application specific latency 181
cut-off is reached and then terminates the session. 182
Services such as telephony would use a low laten- 183
cy/high loss transport regime. This would require 184
the application to be more loss tolerant than at 185
present, however as mobile telephones demon- 186
strate this is not hard to achieve. Interoperation 187
with legacy telephones could be achieved by run- 188
ning loss tolerant algorithms (e.g., FEC) in the 189
PSTN/IP gateway. We do not believe that users 190
want an expensive low loss, low latency service. 191
The current PSTN exemplifies this service and 192
users are moving to VoIP as fast as they are able, 193
despite lower quality, in order to benefit from re- 194
duced prices. 195

Near optimal end to end performance across the 196
network is obtained by enabling the servers to 197
retain options in their application layer routing 198
table for fast path, medium path and slow path 199
(i.e., high loss medium loss and low loss). Packets 200
are then quickly routed to a server whose perfor- 201
mance matches their ttl. This avoids any need to 202
perform flow control and force sequences of 203
packets to follow the same route. 204

For this approach to work well the properties of 205
the servers must adapt to local load conditions. 206
Fast servers have short queues and high drop 207
probabilities, slow servers have long queues and 208
low drop probabilities. If most of the traffic is low 209
latency the servers should all have short buffers 210
and if most of the demand is low loss the servers 211
should have long buffers. Adaptation of the buffer 212
length can be achieved using an adaptive control 213
mechanism [13], and penalising servers whenever a 214
packet in their queue expires. Use of adaptive 215
control has the additional advantage that it makes 216
no assumptions about traffic distributions, and 217
will work well in a situation where the traffic has 218
significant long range dependency (LRD). This 219

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

222 **5. Adaptive control**

223 Conventional control of dynamic systems is
224 based on monitoring state, deciding on the man-
225 agement actions required to optimise future state,
226 and enforcing the management actions. In classical
227 control the decision is based on a detailed knowl-
228 edge of how the current state will evolve, and a
229 detailed knowledge of what actions need to be
230 applied to move between any pair of states (the
231 equations of motion for the state space). Many
232 control schemes in the current Internet (SNMP,
233 OSPF) are based on this form of control. There is
234 also a less precise version known as stochastic
235 control, where the knowledge takes the form of
236 probability density functions, and statistical pre-
237 dictions. All existing forms of traffic management
238 are based on stochastic control, typically assuming
239 Poisson statistics.

240 Adaptive control [13] is based instead on
241 learning and adaptation. The idea is to compen-
242 sate for lack of knowledge by performing experi-
243 ments on the system and storing the results
244 (learning). Commonly the experimental strategy is
245 some form of iterative search, since this is known
246 to be an efficient exploration algorithm. Adapta-
247 tion is then based on selecting some actions that
248 the system has learnt are useful using some selec-
249 tion strategy (such as a Bayesian estimator) and
250 implementing the selected actions. Unlike in con-
251 ventional control, it is often not necessary to as-
252 sume the actions are reliably performed by all the
253 target entities. This style of control has been pro-
254 posed for a range of Internet applications includ-
255 ing routing [14], security [15,16], and fault
256 ticketing [17]. As far as we are aware the work
257 presented here is the first application of distributed
258 adaptive control to service configuration and
259 management.

260 Holland [18] has shown that genetic algorithms
261 (GAs) offer a robust approach to evolving effective
262 adaptive control solutions. More recently many
263 authors [19–21] have demonstrated the effective-
264 ness of distributed GAs using an unbounded gene

pool and based on local action (as would be re- 265
quired in a multi-owner internetwork). However, 266
many authors, starting with Ackley and Littman 267
[22], have demonstrated that to obtain optimal 268
solutions in an environment where significant 269
changes are likely within a generation or two, the 270
slow learning in GAs based on mutation and in- 271
heritance needs to be supplemented by an addi- 272
tional rapid learning mechanism. Harvey [23] 273
pointed out that gene interchange (as observed in 274
bacteria [24,25]) could provide the rapid learning 275
required. This was recently demonstrated by 276
Furuhashi [26] for a bounded, globally optimised 277
GA. In previous work [27] we have demonstrated 278
that a novel unbounded, distributed GA with 279
“bacterial learning” is an effective adaptive control 280
algorithm for the distribution of services in an 281
active service provision system derived from the 282
ALAN. In this paper we demonstrate for the first 283
time that our adaptive control algorithm can de- 284
liver differentiated QoS in response to user sup- 285
plied metrics. 286

287 **6. Algorithm details**

Our proposed solution makes each DPS within 288
the network responsible for its own behaviour. 289
The active service network is modelled as a com- 290
munity of cellular automata. Each automaton is a 291
single DPS that can run several programs 292
(proxylets) requested by users. Each proxylet is 293
considered to represent an instance of an active 294
service. Each member of the DPS community is 295
selfishly optimising its own (local) state, but this 296
‘selfishness’ has been proven as a stable model for 297
living organisms [28]. Partitioning a system into 298
selfishly adapting sub-systems has been shown to 299
be a viable approach for the solving of complex 300
and non-linear problems [29]. 301

In this paper we discuss results from an imple- 302
mentation that supports up to 10 active services. 303
The control parameters given below are examples 304
provided to illustrate our approach. Our current 305
implementation has 1000 vertices connected on a 306
rectangular grid (representing the network of 307
transport links between the DPSs). Each vertex 308
can support a single server (i.e., host) supporting a 309

310 single DPS, so the network can support up to 1000
 311 DPS nodes. In reality a network node (router)
 312 would be associated with many such hosts, possi-
 313 bly organised as a cluster. In this work we are
 314 assuming that the latency associated with a DPS is
 315 significantly greater than that associated with bit
 316 transport so we do not distinguish between DPS
 317 links that are local and DPS links that are remote.
 318 Each DPS has an amount of genetic material that
 319 codes for the rule set by which it lives. There is a
 320 set of rules that control the DPS behaviour. There
 321 is also a selection of genes representing active
 322 services. These define which services each node will
 323 handle and can be regarded as pointers to the ac-
 324 tual programs supplied by users. Each node can
 325 hold up to eight services (the limit is similar to that
 326 imposed by available RAM in commodity com-
 327 puters, such as could be used in future server
 328 clusters). The service genes also encode some
 329 simple conditionals that must be satisfied for the
 330 service to run. Currently each service gene takes
 331 the form $\{x, y, z\}$ where:

- 333 • x is a character representing the type of service
 requested (A–J).
- 335 • y is an integer between 0 and 200 which is inter-
 337 preted as the value in a statement of the form
 “Accept request for service [Val(x)] if queue
 length $<$ Val(y)”.
- 339 • z is an integer between 0 and 100 that is inter-
 preted as the value in a statement of the form
 “Accept request for service [Val(x)] if busyness
 $<$ Val(z)%”.

342 The system is initialised by populating a random
 343 selection of network vertices with DPSs (active
 344 nodes), and giving each DPS a random selection of
 345 the available service genes. Requests are then en-
 346 tered onto the system by injecting a random se-
 347 quence of characters (representing service
 348 requests), at a mean rate that varies stochastically,
 349 at each vertex in the array. If the vertex is popu-
 350 lated by a DPS, the items join a queue. If there is
 351 no DPS the requests are forwarded to a neigh-
 352 bouring vertex. The precise algorithm for this
 353 varies and is an active research area, however the
 354 results shown here are based on randomly select-
 355 ing a direction in the network and forwarding
 356 along that direction till a DPS is located. This is
 357 clearly sub-optimal but is easy to implement. The

358 traffic arriving at each DPS using this model shows
 359 some LRD, but significantly less than real WWW
 360 traffic. Increasing the degree of LRD would be
 361 straightforward. However, the necessary change
 362 involves additional memory operations that slows
 363 down the simulation and makes the results harder
 364 to interpret. In any case inclusion of significant
 365 LRD would not change the qualitative form of the
 366 main results since the algorithm is not predictive
 367 and makes no assumptions regarding the traffic
 368 pdf. Each DPS evaluates the items that arrive in its
 369 input queue on a FIFO principle. If the request at
 370 the front of the queue matches an available service
 371 gene, and the customer has included payment to-
 372 kens equal to (or greater than) the cost for that
 373 service in the DPS control rules, the service will
 374 run. In the simulation the request is deleted and
 375 deemed to have been served, and the node is re-
 376 warded by a value equal to the specified cost of the
 377 service. If there is no match the request is for-
 378 warded and no reward is given. In this case the
 379 forwarding is informed by a state table maintained
 380 by the DPS using a node state algorithm. Packets
 381 with a short ttl are forwarded to a DPS with a
 382 short queue and packets with a long ttl are for-
 383 warded to a DPS with a long queue. Each DPS is
 384 assumed to have the same processing power, and
 385 can handle the same request rate as all the others.
 386 In the simulation time is divided into epochs (to
 387 enable independent processing of several requests
 388 at each DPS before forwarding rejected requests).
 389 An epoch allows enough time for a DPS to execute
 390 3–4 service requests, or decide to forward 30–40
 391 (i.e., forwarding incurs a small time penalty). An
 392 epoch contains 100 time units and is estimated to
 393 represent $O(100)$ ms. The busyness of each DPS is
 394 calculated by combining the busyness at the pre-
 395 vious epoch with the busyness for the current ep-
 396 och in a 0.8–0.2 ratio, and is related to the revenue
 397 provided for processing a service request. For ex-
 398 ample, if the node has processed three requests this
 399 epoch (25 points each) it would have 75 points for
 400 this epoch, if its previous cumulative busyness
 401 value was 65 then the new cumulative busyness
 402 value will be 67. This method dampens any sudden
 403 changes in behaviour. A brief schematic of this is
 404 shown in Fig. 2.

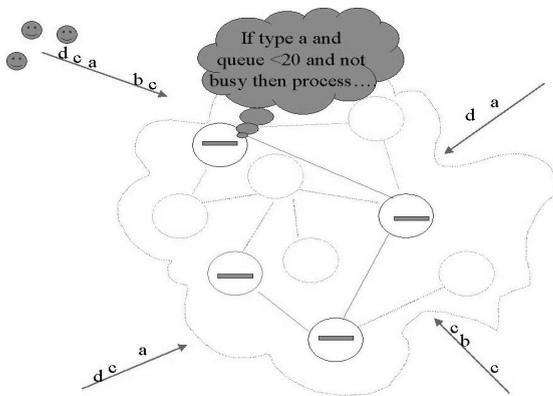


Fig. 2. Future network model.

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

415 Genetic diversity is created in at least two ways, mutation and plasmid migration. Mutation involves
 416 the random alteration of just one value in a single service gene, for example: "Accept request
 417 for service A if DPS <80% busy" could mutate to "Accept request for service C if DPS <80% busy"
 418 or alternatively could mutate to "Accept request for service A if DPS <60% busy".

423 Plasmid migration involves genes from healthy individuals being shed or replicated into the environment
 424 and subsequently being absorbed into the genetic material of less healthy individuals. If plasmid
 425 migration does not help weak strains increase their fitness they eventually die. If a DPS acquires
 426 more than 4-6 service genes through interchange the newest genes are repressed (registered as dormant).
 427 This provides a long term memory for genes that have been successful, and enables the community
 428 to successfully adapt to cyclic variations in demand. Currently, values for queue length and cumulative
 429 busyness are used as the basis for interchange actions, and evaluation is

437 performed every five epochs. Although the evaluation period is currently fixed there is no reason
 438 why it should not also be an adaptive variable.

439 If the queue length or busyness is above a threshold (both 50 in this example), a random section
 440 of the genome is copied into a 'rule pool' accessible to all DPSs. If a DPS continues to exceed
 441 the threshold for several evaluation periods, it replicates its entire genome into an adjacent network
 442 vertex where a DPS is not present. Healthy bacteria with a plentiful food supply thus reproduce
 443 by binary fission. Offspring produced in this way are exact clones of their parent.

444 If the busyness is below a different threshold (10), a service gene randomly selected from the rule
 445 pool is injected into the DPS's genome. If a DPS is 'idle' for several evaluation periods, its active
 446 genes are deleted, if dormant genes exist, these are brought into the active domain, if there are
 447 no dormant genes the node is switched off. This is analogous to death by nutrient deprivation.

448 So if a node with the genome {a, 40, 50/c, 10, 5} has a busyness of >50 when analysed, it will
 449 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}
 450 is later deemed to be idle it may import that rule and become {b, 2, 30/d, 30, 25/c, 10, 5}.

7. Experiments

464

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

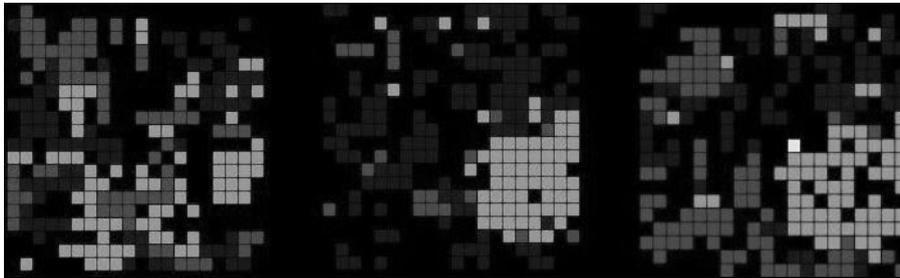


Fig. 3. Distribution of DPS nodes with short medium and long queues at three different times.

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

488 In Fig. 4 we show the average request drop rate
 489 across the network of bacteria illustrated in Fig. 3,
 490 and compare the performance with a number of
 491 alternative methods of distributing the active ser-
 492 vices. 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).

494
 495
 496
 497
 498 The tests were performed at loads of 10% (low),
 499 500 40% (medium) and 80% (high). At low loads all the
 501 algorithms offer similar performance levels. As
 502 might be expected, at medium and high load our
 503 algorithm is a significant improvement over ran-

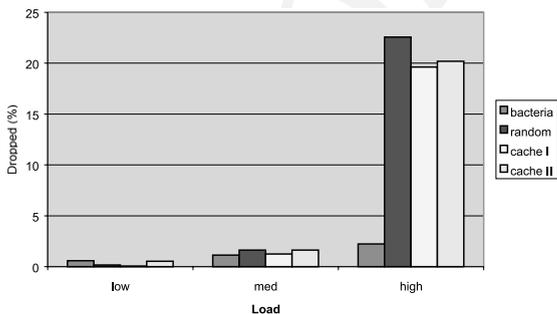


Fig. 4. Request drop rates for different distribution mechanisms.

504 dom placement. More surprisingly it also signifi-
 505 cantly outperforms caching. We believe this is due
 506 to the small size of the caches. Each cache holds up
 507 to eight services (i.e., the same as the bacteria).
 508 This is intended to represent the number of
 509 proxylets that can be held in the RAM of a low
 510 spec PC, such as might be used in a commodity
 511 based cluster at a network server farm. Since the
 512 load time for proxylets is currently long (~1 s) we
 513 do not model disk based caching.

514 Fig. 5 shows the average end to end latency
 515 experienced by service requests in our modelled
 516 network, and compares it with the latency experi-
 517 enced using the alternative active service distribu-
 518 tion mechanisms listed above. As before the
 519 adaptive bacterial approach is as good as the other
 520 alternatives at low loads, and is clearly an im-
 521 provement over the best alternative (standard
 522 LRU based caching – CacheII) at medium and
 523 high loads. We are therefore confident that our
 524 algorithm is delivering a useful level of perfor-
 525

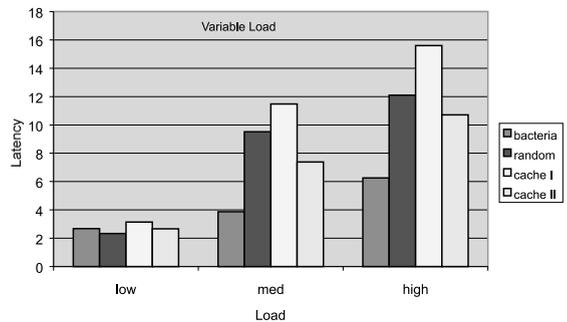


Fig. 5. Average latency of several approaches to distributing active services.

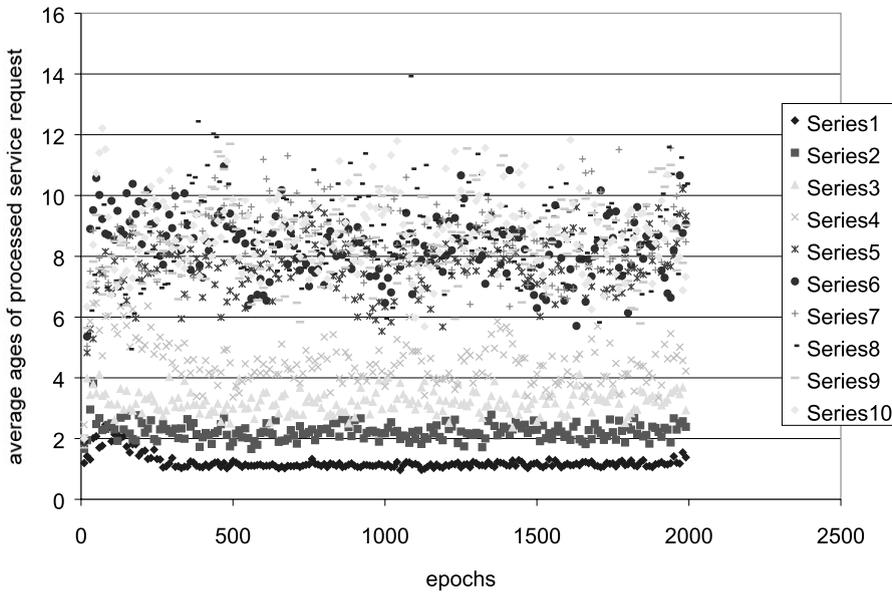


Fig. 6. Different latencies for requests with differing times to live.

526 Fig. 6 illustrates the differentiated QoS delivered
527 by the network of DPS nodes. The time taken to
528 process each request is shown on the y access and
529 the elapsed system time is shown on the x axis. It

530 can be seen that the service requests with shorter
531 times to live are being handled faster than those
532 with a longer time to live, as expected. Fig. 7 shows
533 the expected corollary. More service requests with

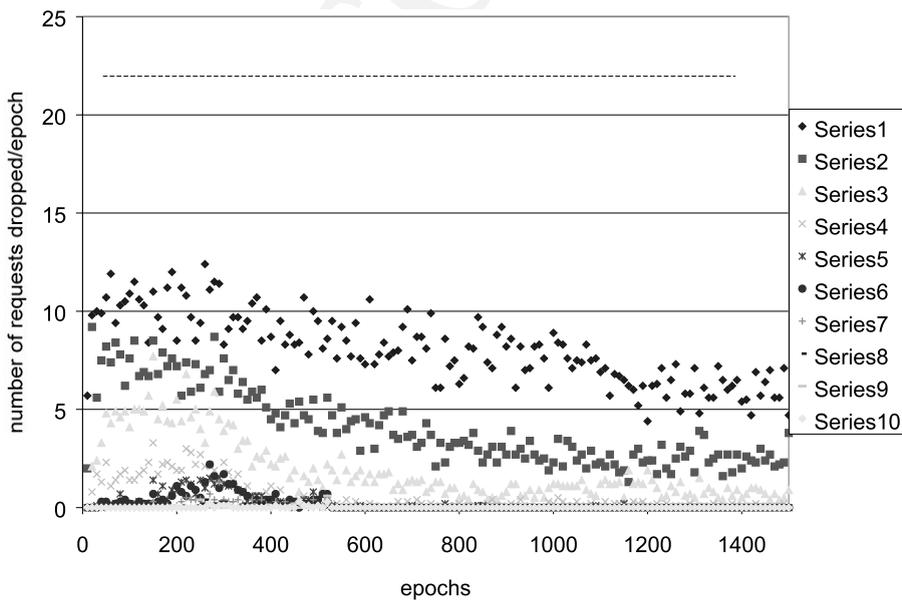


Fig. 7. Different dropping rates for requests with differing times to live.

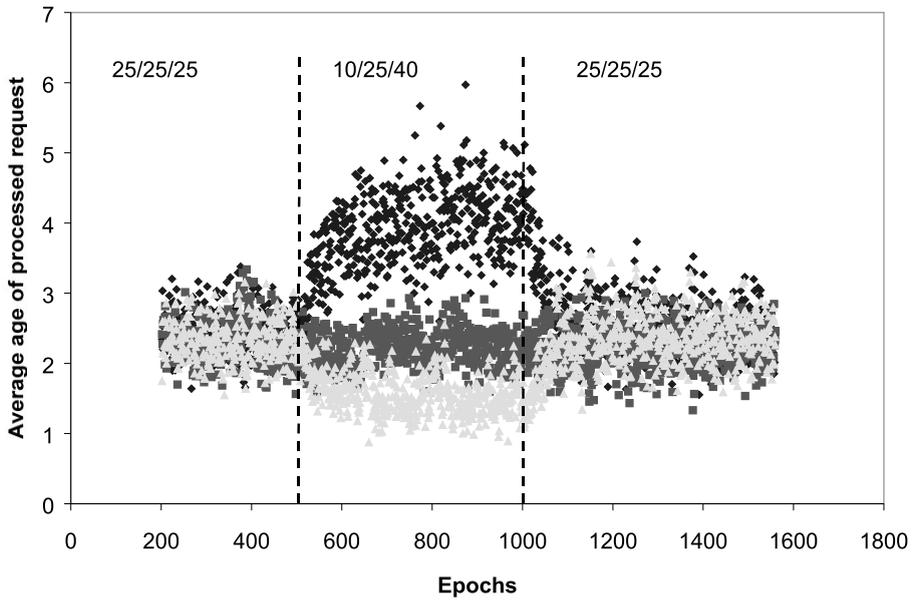


Fig. 8. Effects of different charging levels on age related QoS.

534 short ttls are being dropped. This is due to them
535 timing out, and is the essential down-side to
536 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 as-
sociated with service requests users may also wish

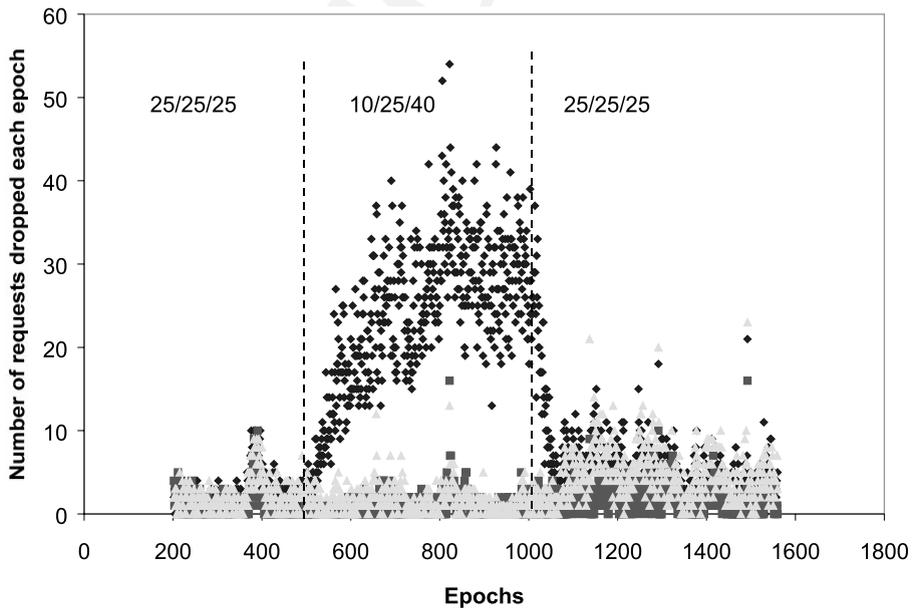


Fig. 9. Effects of different charging levels on dropping of requests.

541 to vary the price they are willing to pay. In the
 542 basic algorithm it was assumed that the network
 543 provider allocated a reward to each DPS for pro-
 544 cessing a service request. We investigated the im-
 545 pact of allowing the DPS to collect a greater
 546 reward. In the modified model the DPS is re-
 547 warded by the amount of tokens the user includes
 548 with the request. The traffic input was adjusted so
 549 that requests for different services carried different
 550 amounts of payment tokens. Initially the DPS
 551 nodes were rewarded equally (25 ‘tokens’) for each
 552 of three services A, B and C. After 500 epochs the
 553 rate of reward is changed so that each DPS is re-
 554 warded four times as much for processing service
 555 C (40 tokens) as it is for processing service A (10
 556 tokens), with B staying at 25. This is equivalent to
 557 offering users a choice of three prices for a single
 558 service. Fig. 8 shows the latency of service requests
 559 for the three different service types.

560 It is apparent that within 100 epochs the average
 561 latency for providing service C is reduced while the
 562 latency for A is increased. Fig. 9 shows that re-
 563 quests for service A are also dropped (due to
 564 timing out) more than requests for service B and
 565 C. Before the change in reward the numbers of
 566 DPSs handling each service were similar. After the
 567 reward rate change the plasmids for handling
 568 services C and B have spread much more widely
 569 around the network at the expense of the plasmid
 570 for the relatively unrewarding service A. After
 571 1000 epochs the rate of requests for all three ser-
 572 vices was returned to the original state. It can be
 573 seen, in both figures, that equality in quality of
 574 service, both in terms of loss rate and latency,
 575 quickly returned.

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

8. Conclusions

586

Our initial results show that the long-term self- 587
 stabilising, adaptive nature of bacterial communi- 588
 ties are well suited to the task of creating a stable 589
 community of autonomous active service nodes 590
 that can offer consistent end to end QoS across a 591
 network. The methods used for adaptation and 592
 evolution enable probabilistic guarantees for met- 593
 rics such as loss rate and latency similar to what 594
 can be achieved using more conventional ap- 595
 proaches to differentiated services. 596

References

597

- [1] M. Fry, A. Ghosh, Application layer active networking, *Computer Networks* 31 (7) (1999) 655–667. 598
599
- [2] R. Giaffreda, N. Walker, R.Y. Ruiz, Performance of the 600
DNS name resolution infrastructure, in: *Proceedings of the* 601
IEE colloquium on Control of Next Generation Networks, 602
London, October 1999. 603
- [3] I.W. Marshall, J. Hardwicke, H. Gharib, M. Fisher, P. 604
Mckee, Active management of multiservice networks, in: 605
Proceedings of the IEEE NOMS2000, pp. 981–983. 606
- [4] P. Mckee, I.W. Marshall, Behavioural specification using 607
XML, in: *Proceedings of the IEEE FTDCS '99,* Capetown, 608
pp. 53–59. 609
- [5] I.W. Marshall, M. Fry, L. Velasco, A. Ghosh, Active 610
information networks and XML, in: S. Covaci (Ed.), 611
Active Networks, LNCS 1653, Springer, Berlin, 1999, pp. 612
60–72. 613
- [6] D.G. Waddington, D. Hutchison, Resource partitioning in 614
general purpose operating systems, experimental results in 615
Windows NT, Operating Systems Review 33 (4) (1999) 52– 616
74. 617
- [7] IETF Intserv charter: [http://www.ietf.org/html.charters/](http://www.ietf.org/html.charters/intserv-charter.html) 618
[intserv-charter.html](http://www.ietf.org/html.charters/intserv-charter.html). 619
- [8] IETF Diffserv charter: [http://www.ietf.org/html.charters/](http://www.ietf.org/html.charters/diffserv-charter.html) 620
[diffserv-charter.html](http://www.ietf.org/html.charters/diffserv-charter.html). 621
- [9] V. Jacobson, Support for differentiated services in router 622
designs, in: *Proceedings of the Network Modelling in the* 623
21st Century, Royal Society, December 1999. 624
- [10] V. Paxson, S. Floyd, Wide area traffic: the failure of 625
poisson modelling, *IEEE/ACM Transactions on Network-* 626
ing 3 (3) (1995) 226–244. 627
- [11] M. Crovella, A. Bestavros, Self-similarity in world wide 628
web traffic: evidence and possible causes, *IEEE/ACM* 629
Transactions on Networking 5 (6) (1997) 835–846. 630
- [12] R. Gibbens, S.K. Sargood, F.P. Kelly, H. Azmoodeh, R. 631
MacFadyen, N. MacFadyen, An approach to service level 632
agreements for IP networks with differentiated services, 633
Philosophical Transactions of the Royal society A, 2000, 634

- 635 submitted, and available at [http://www.stat-](http://www.stat-slab.cam.ac.uk/~richard/research/papers/sla/)
 636 [slab.cam.ac.uk/~richard/research/papers/sla/](http://www.stat-slab.cam.ac.uk/~richard/research/papers/sla/).
- 637 [13] Y.Z. Tsyppin, Adaptation and learning in automatic
 638 systems, Mathematics in Science and Engineering, vol.
 639 73, Academic press, New York, 1971.
- 640 [14] G. DiCaro, M. Dorigo, AntNet: distributed stigmergic
 641 control for communications networks, Journal of Artificial
 642 Intelligence Research 9 (1998) 317–365.
- 643 [15] D.A. Fisher, H.F. Lipson, Emergent algorithms – a new
 644 method of enhancing survivability in unbounded systems,
 645 in: Proceedings of the 32nd Hawaii International Confer-
 646 ence on System Sciences, IEEE, 1999.
- 647 [16] M. Gregory, B. White, E.A. Fisch, U.W. Pooch, Cooper-
 648 ating security managers: a peer based intrusion detection
 649 system, IEEE Network 14 (4) (1996) 68–78.
- 650 [17] L. Lewis, A case based reasoning approach to the
 651 management of faults in telecommunications networks,
 652 in: Proceedings of the IEEE conference on Computer
 653 Communications, vol. 3, San Francisco, 1993, pp. 1422–
 654 1429.
- 655 [18] J.H. Holland, Adaptation in Natural and Artificial Sys-
 656 tems, MIT press, Cambridge, MA, 1992.
- 657 [19] S. Forrest, T. Jones, Modeling complex adaptive systems
 658 with echo, in: R.J. Stonier, X.H. Yu (Eds.), Complex
 659 Systems Mechanisms of Adaptation, IOS Press, 1994, pp.
 660 3–21.
- 661 [20] R. Burkhart, The swarm multi-agent simulation system,
 662 OOPSLA '94 Workshop on The Object Engine, September
 663 7, 1994.
- 664 [21] J.U. Kreft, G. Booth, J.W.T. Wimpenny, BacSim, a
 665 simulator for individual-based modelling of bacterial
 666 colony growth, Microbiology 144 (1997) 3275–3287.
- 667 [22] D.H. Ackley, M.L. Littman, Interactions between learning
 668 and evolution, in: C.G. Langton, C. Taylor, J.D. Farmer,
 669 S. Rasmussen (Eds.), Artificial Life II, Addison-Wesley,
 670 Reading, MA, 1993, pp. 487–507.
- 671 [23] I. Harvey, The Microbial Genetic Algorithm, unpublished
 672 work, 1996, available at [ftp://ftp.cogs.susx.ac.uk/pub/users/](ftp://ftp.cogs.susx.ac.uk/pub/users/inmanh/Microbe.ps.gz)
 673 [inmanh/Microbe.ps.gz](ftp://ftp.cogs.susx.ac.uk/pub/users/inmanh/Microbe.ps.gz).
- 674 [24] S. Sonea, M. Panisset, A new bacteriology, Jones and
 675 Bartlett, 1983.
- 676 [25] D.E. Caldwell, G.M. Wolfaardt, D.R. Korber, J.R. Law-
 677 rence, Do bacterial communities transcend darwinism?
 678 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 Interna-
 tional 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,
 Oxford, 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.