Programmable Resource Discovery Using Peer-to-Peer Networks

Paul Smith, Steven Simpson, and David Hutchison

Computing Department
Lancaster University
Lancaster, LA1 4YR, UK
{p.smith, ss, dh}@comp.lancs.ac.uk

Abstract. Some forms of programmable networks such as funnelWeb allow service components to be deployed discretely (i.e. out-of-band) on a suitable configuration of elements, but do not define mechanisms to determine such configurations.
We present a mechanism to resolve arbitrary service-specific deployment constraints into a suitable node configuration. To focus constraint resolution, we arrange programmable elements into an overlay, and use this to interpolate/extrapolate more favourable locations. Programmable service components are used to evaluate suitability of individual nodes.

1 Introduction

Critical to the successful operation of a network service is the deployment of service components on appropriate network elements. The services supported by a conventional network element are intrinsic to that element and static. However, services on a programmable element are dynamic and transient. The challenge in a programmable networking environment is to select from a set of *general purpose* programmable elements a subset that is suitable for deploying a set of service components.

When considering programmable service component deployment, two main approaches have been adopted to date: the capsule (or in-band) approach, where computational components are included with or referenced from within network traffic that is to be augmented [WGT99, Wet99]; or the discrete (or out-of-band) approach, where computational elements are loaded prior to the traffic's arrival by a third-party process [FG98, SBSH01]. The discrete approach requires a process to select the appropriate set of programmable elements to deploy components on, unlike the capsule approach where component deployment is integral to the manner in which the service is executed and so there is no such requirement.

In this paper, we are concerned with the discrete approach to service deployment. We present a critical part of a service deployment architecture that enables the identification of programmable entities based upon a set of service-specific constraints. Using our `GROUPNET` algorithm, programmable elements are organised into a peer-to-peer overlay network. Programmable service components are used to resolve service-specific constraints into suitable node configurations.

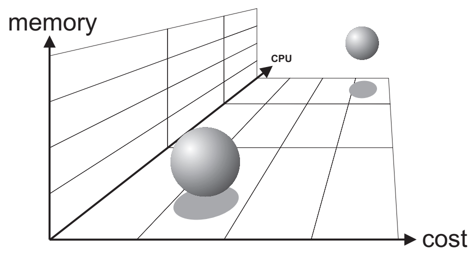


Fig. 1. An example set of dimensions and two different services with distinct desired values within those dimensions

We continue this section with a discussion of issues related to service deployment constraints. Following in Sect. 2 an overview of our `GROUPNET` overlay construction algorithm is given, which enables us to infer locations to search when resolving deployment constraints. Section 3 presents a search algorithm using `GROUPNET` and shows simulation results of a number of searches. In Sect. 4 we present an example service deployment scenario and show how our approach can be used. In Sect. 5 related work are discussed. Finally, in Sect. 6 we present some conclusions and outline further work.

1.1 Service Deployment Constraints

Peer-to-peer networking services such as Freenet [CSWH01] perform single-dimensional constraint resolution. For example, they transform the name of a shared file into a set of peers from which the file can be obtained. In programmable networks there are potentially multiple service deployment constraint dimensions that can be distinct on a per-service-instantiation basis, for example monetary cost and topological location. This is exemplified in Fig. 1, where one service has relatively low computational and cost constraints and another with relatively higher constraints.

In some cases, values within constraint dimensions can be traded against others in different dimensions – for example, a customer may be willing to pay a price in proportion to the service quality. Values within some dimensions can change in relation to network topology. For example, a telephone call may cost less when the end-points are topologically nearer. Programmable services that demonstrate these properties can be seen at [GF97, SMCH01].

A programmable element that is intended to support a number of potentially transient services cannot natively understand the nature of their operational constraints. It is for this reason we advocate the use of programmable service components to evaluate the suitability of individual elements. In this situation, elements expose interfaces that enable appropriately privileged programmable resource discovery components to interrogate local state and perform measurements with interesting nodes. Constraint resolution intelligence resides in programmable service components that interrogate *dumb* programmable elements.

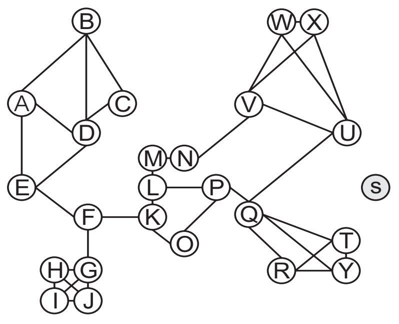
2 GROUPNET: A GROUPing Meshed Overlay NETwork

Programmable elements are organised into an overlay network. This overlay network is constructed using the GROUPNET algorithm. A property of the overlay enables us to interpolate/extrapolate improved locations to search. A description of the GROUPNET overlay construction algorithm will now be given.

The procedure that a new peer wishing to join a GROUPNET mesh must follow can be described as follows. A peer s locally decides its desired degree d, which will influence the number of immediate peers it has. It will maintain $N(s)$, its set of immediate neighbours. (Initially, using a bootstrap mechanism, s is assigned a set of peers $N(s)$. This initial $N(s)$ can either be randomly selected, or to improve algorithm performance some heuristics can be used for selection.) s now undergoes a sequence of optimisations.

In each optimisation, for each peer p in $N(s)$, s obtains measurements to p and $N(p)$, and orders them to produce a list $L(p)$ (which includes p) of those peers in order. The new set of neighbours of s, $N'(s)$ will at least consist of the best of each $L(p)$ – if this falls short of d, the best of the union of the remaining members of all the $L(p)$'s can make up the shortfall.

The result is that for each $p \in N(s)$, $(N'(p) \cup p) \cap N'(s)$ is not empty, i.e. there is a path of no more than two hops between s and each of its original neighbours, ensuring that the mesh does not become disconnected.



$d = 3$

1. $N(s) = \{K\}$

$N(K) = \{F, O, L\}$, $L(K) = < O, L, K, F >$

2. $N'(s) = \{O, L, K\}$

$N(O) = \{K, P\}$, $L(O) = < P, O, K >$

$N(K) = \{F, L, O\}$, $L(K) = < O, L, K, F >$

$N(L) = \{K, P, M\}$, $L(L) = < P, L, M, K >$

3. $N''(s) = \{P, O, L\}$

$N(P) = \{L, O, Q\}$, $L(P) = < Q, P, O, L >$

$N(O) = \{K, P\}$, $L(O) = < P, O, K >$

$N(L) = \{K, P, M\}$, $L(L) = < P, L, M, K >$

4. $N'''(s) = \{Q, P, O\}$

Fig. 2. An example peer membership scenario with some of the stages of evaluation shown

2.1 Peer Membership Example

Figure 2 shows an example of a peer wishing to join a mesh and some of the steps toward achieving this. The node s wishes to join the mesh and has selected a degree d of 3. Using a bootstrap mechanism s is assigned $\{K\}$ as its initial $N(s)$ as shown in step 1 of Fig. 2. The neighbours of K are evaluated along with K itself and the ordered list $L(K)$ is produced. In this example, peers closer to s score higher than those more distant and therefore appear toward the start of the list. From $L(K)$, the best d are taken to produce $N'(s)$ as shown in step 2. This completes a single iteration of the GROUPNET membership algorithm.

We anticipate a number of iterations would be performed until $N'(s)$ does not improve on $N(s)$. In step 2 of Fig. 2, $N'(s) = \{O, L, K\}$ which improves upon the immediate neighbours of s. Another iteration of the membership algorithm involves: determining $N(p)$ where p is a peer in $N'(s)$ and $N(p)$ are all the neighbours of p (for example $N(O) = \{K, P\}$), evaluating them to create $L(p)$ ($L(O) =< P, O, K >$) and then selecting the best from each $L(p)$ to generate $N''(s)$. If we find that we have already selected the best from a list, we simply take the next best as demonstrated in the example. It can be seen that the immediate peers of s draw gradually closer to it. Ultimately, the algorithm will terminate with $N(s) = \{U, T, Y\}$.

2.2 Peer Evaluation Metrics

The metric used to evaluate a peer will depend on the application of the overlay mesh. For example, for programmable resource discovery we wish to construct a topologically aware overlay network. To this end, we could use round-trip delay between peers to evaluate suitability. Peers with smaller round trip delays score higher in this situation that those with longer delays.

If one considers a resource sharing application, it may be desirable to group peers sharing similar resources. By grouping peers which share similar resources (for example by music from a similar genre) searching could be more effective as peers probable to have a desirable resource will be fewer overlay hops away. If a user searches for resources that are distinct from those available from its immediate peers, inferences can be made regarding optimal locations to search.

2.3 Evaluation of GROUPNET

We have simulated the GROUPNET overlay construction algorithm. Peers are randomly distributed in a two-dimensional coordinate space and the group membership algorithm run at each peer. Distance between peers is used as the configuration metric. Peers with a smaller distance score better than those with larger distances. Figure 3 presents a graphical representation of the meshes produced. Node degrees of 3, 4 and 5 were assigned shown in Fig. 3a–c respectively. The top line of Fig. 3 (*random*) shows meshes produced after peers are assigned random neighbours. The bottom line of Fig. 3 (*groupnet*) show the meshes produced

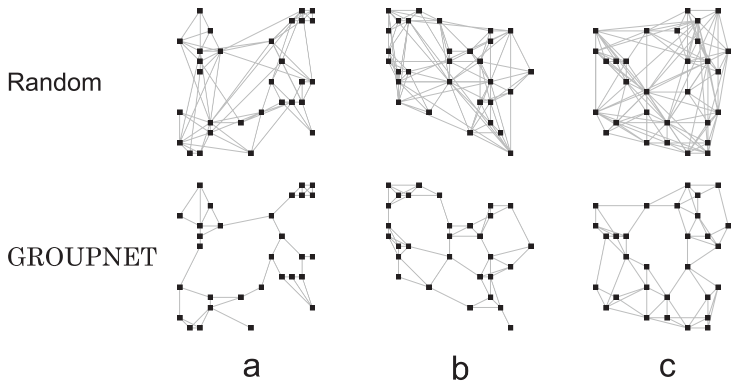


Fig. 3. Meshes created both randomly and using the `GROUPNET` algorithm with different node degrees

after running the `GROUPNET` overlay membership algorithm presented earlier. To optimise the mesh produced, we ran the algorithm 30 times at each node[1].

The aim of these simulations is to demonstrate that the membership algorithm produces overlays with desirable properties. The random meshes shown in Fig. 3 show undesirable properties as peers have links that are connected to relatively distant peers. The searching mechanism described in section 3 cannot be conducted on overlay configurations such as these, since there is no correlation between distances on the overlay and the space. The meshes constructed using the `GROUPNET` algorithm have the desired properties for enabling interpolation/extrapolation as there is a smooth transition over the space as one traverses the overlay.

3 Searching over `GROUPNET`

To evaluate the effectiveness and scalability of our approach, we have simulated searching over a `GROUPNET` mesh. To do this we construct meshes as in the simulations presented in Sect. 2. We define a target set of peers with three parameters: a point p in the coordinate space, a threshold T from p, that peers should fall within, and a timeout. Figure 4 shows the simulation scenario, with the target peers falling within the threshold shown.

A peer is capable of performing two operations on other peers. A node can either *evaluate* or *spawn* on another peer. The evaluate operation simulates remotely determining the suitability of another peer, for example by loading evaluation components. In our simulation, evaluation takes the form of a peer calculating its distance from p and returning that value. The spawn operation

[1] Normally, we expect that optimisation would happen periodically, increasing in frequency the more excess links a peer has.

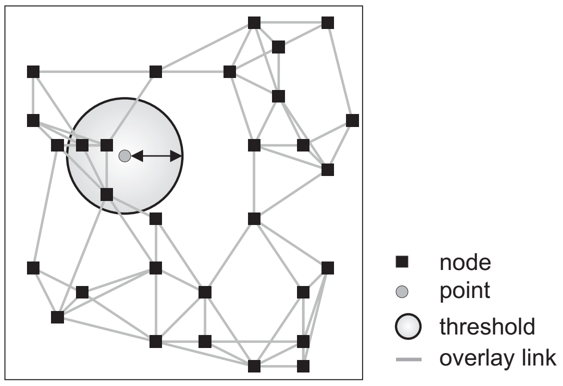


Fig. 4. Scenario used for searching simulations

simulates instantiating programmable service components onto a peer. These are two operations we envisage as being necessary to conduct searches.

3.1 Search Algorithm

To conduct the search, a peer is randomly selected from the mesh. Initially, the timeout is checked and if it has expired the search algorithm terminates on that peer. The peer then randomly selects a number of other peers n hops away and evaluates them. If the peers just evaluated have larger distances s to p than a peer's own s, a subsequent set of peers are selected $n/2$ hops away and evaluated. This process is iterated until $n = 0$, when the search is terminated at that peer. However, if a set of evaluated peers have smaller values of s than a peer's own, the search algorithm is spawned on those peers. Peers with $s <= T$ are immediately reported after evaluation. The search should migrate or spawn toward peers with improved values of s and terminate when better values of s cannot be found.

3.2 Simulation Results

To assess the scalability of our search algorithm, we observed the average number of evaluations and spawns. We ran the search algorithm with the following parameters. We exponentially increased the number of peers in the mesh and adjusted the size of the coordinate space to maintain a constant density of peers. The value of T was maintained to statistically yield a constant number of target peers. The initial hop count n was kept at 20% of the mesh size. The algorithm was performed 50 times using each mesh size, with a new p and initial peer to start the search on being selected at each iteration.

Spawns and evaluations are relatively expensive operations to perform. For example, programmable service component deployment using funnelWeb involves invoking a *load* command on a programmable element, fetching the component(s) over HTTP to the element and then instantiating the components. Figures 5 and 6 present the average number of spawns and the percentage of the total mesh spawned upon. The average number of spawns remains fairly constant in relation to increased topology size. The ability of the search algorithm to scale is best demonstrated in Fig. 6, where the total percent of the mesh spawned upon falls dramatically with the increase in size of the mesh.

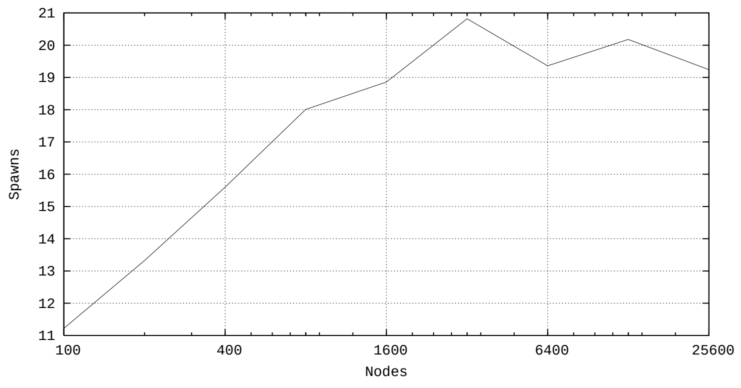


Fig. 5. Average number of spawns conducted on different size meshes

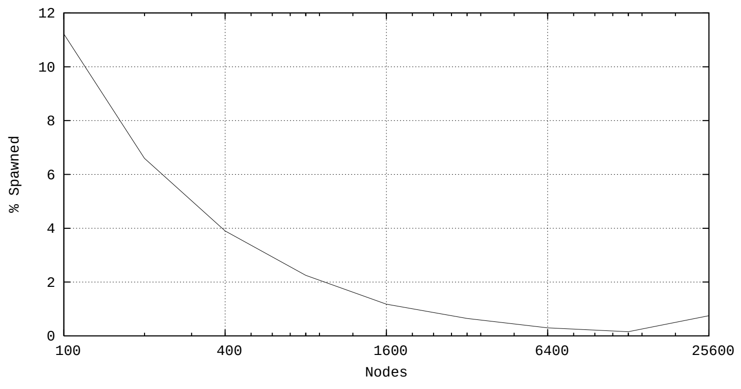


Fig. 6. Percentage of total mesh size spawned on when searching

Figure 7 presents the average number of evaluations conducted when executing the search algorithm. Again, as with the average number of spawns the number of evaluations remains relatively constant in relation to increased mesh size. The percentage of the mesh evaluated is presented in Fig. 8. The percentage of the total mesh evaluated falls significantly with increased mesh size.

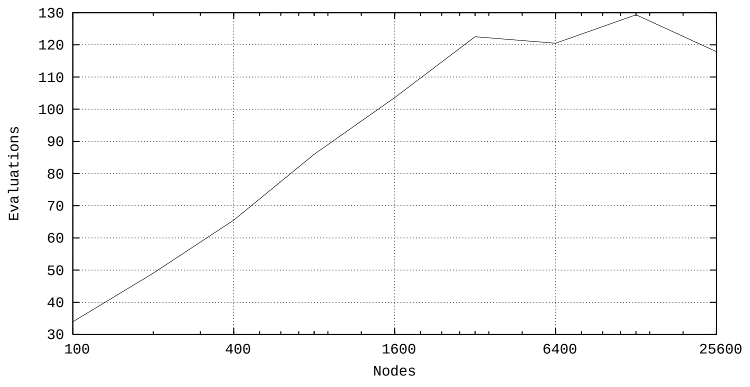


Fig. 7. Average number of evaluations conducted on different size meshes

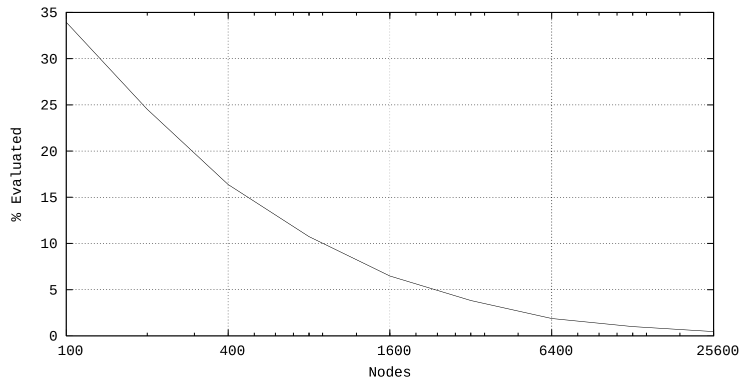


Fig. 8. Percentage of total mesh size spawned on when searching

It can be seen from the results presented that the target peers can be rapidly converged upon in a manner that scales with large mesh sizes. This is achieved by using a search algorithm that uses the properties of the GROUPNET overlay to infer improved locations to search. To demonstrate the effectiveness of the algorithm, we observed that out of the total number of peers to be found the algorithm located on average 89%.

4 Service Deployment Scenario

To demonstrate the applicability of our approach we will present an example service deployment scenario. The service we present in this example aims to reduce the cost of making long distance calls by using the Internet and Voice over IP (VoIP) to make the long distance connection [GFC00]. At the remote end of a connection, a network element with a modem attached to the PSTN can be used to complete a call to a local receiver, therefore making the total cost of a long distance call at most two local connections: one to connect to the Internet and the other to a local receiver. In our example scenario, the network element with modem functionality is programmable and the aim is to discover from the set of elements available a subset most suited to serve our application demands.

By using the Internet to form the long distance part of a connection, one exacerbates the problem of round-trip call delay. However the greater part the Internet constitutes the connection, the cheaper the cost of the call is likely to be. A balance must be struck between the total cost of the call and the delay experienced. Both parts of a connection contribute to a monetary cost and the total round-trip delay. We can use the function $c = c_n + c_p$ to calculate the total cost of making the call, where c_n is the fixed cost of connecting to the Internet and c_p is cost of the call on the PSTN. We use $t = t_n + t_p$ to determine the total round-trip delay, where t_n is the delay across the Internet and t_p is the delay across the PSTN. The values of t and c have acceptable upper bounds represented by t_{max} and c_{max} respectively. Elements with values greater than t_{max} and c_{max} can be immediately dismissed as unacceptable. The product of t and c can be used to select the most appropriate element from the available set. Table 1 shows example values of five candidate elements.

Element	t_n	t_p	c_p	c_n	t	c	tc
1	20	17	17	3	37	19	703
2	40	14	12	3	54	15	810
3	60	11	6	3	71	9	639
4	80	8	3	3	88	6	528
5	100	5	3	3	105	6	630

Table 1. Example costs for a set of candidate elements for deploying a Voice over IP service

If $t_{max} = 100$ms and $c_{max} = 10$p/min we can immediately consider elements 1, 2 and 5 from Table 1 as unsuitable. Elements 3 and 4 meet our constraints and we consider the value of tc to select the most a appropriate element, in this case element 4 best meets the service constraints.

By constructing the overlay of programmable elements using `GROUPNET`, there is a correlation between relative location in the overlay and network topology.

One would anticipate the cost of the call c_p to change in relation to network topology. Therefore traversing the overlay network should result in a consistent variation in the cost c_p. We can use this fact to interpolate/extrapolate better locations for searching. For example, if a direction along the overlay yields elements with a better c_p than those already known, searching can continue in that direction in the hope that c_p will continue to improve. Other inferences such as these can be made in order to find the set of programmable elements that have acceptable values of c. If such elements are found, t_n can be determined and ultimately tc.

5 Related Work

There are peer-to-peer networking based solutions that are able to take multi-dimensional resource constraints and resolve/route them to the nearest set of peers that meet the constraints. Noteworthy approaches include CANs [RFH⁺01] and Pastry [RD01].

The design of Content-Addressable Networks (CANs) is focused on the use of a virtual d-dimensional Cartesian coordinate space, which is partitioned into zones that are controlled by peers in the CAN. A peer wishing to join the CAN selects a point in the coordinate space at which they wish to reside. They send a join message to an arbitrary point in the CAN with their desired coordinates. This message is routed over the coordinate space until it reaches a peer who controls the zone that occupies that point. The peer who owns the zone partitions it with the new peer and shares neighbour information. The new peer is now reachable across the CAN and is adjacent to peers with similar coordinates.

Peers in the Pastry network possess a unique node identifier that is obtained using a secure hash of its public key. As with CANs, a peer's identifier dictates its position within the Pastry network. The group membership algorithm used by Pastry enables peers to be co-located with others that have similar identifiers. Peers in Pastry maintain routing tables containing node identifiers of adjacent peers in the network.

Points in a CAN coordinate space and public keys in Pastry can be used to identify peers that satisfy multi-dimensional resource constraints. However, the values within a dimension are absolute – they do not vary in relation to different view points. It is also unclear how such approaches manage when there are highly dynamic values within dimensions as in a programmable networking environment.

For the purpose of programmable resource discovery, `GROUPNET` is used to construct a topologically-aware overlay network. Other approaches to constructing such overlays exist, for example Distributed Binning [RHKS02] and Hierarchical Clustering [MCSH02].

Distributed Binning involves associating nodes with logical *bins* based upon relative or absolute values that are derived from measuring a node's distance from a number of well-known landmarks. We believe that scalability issues could arise if a significant number of nodes measure themselves against a much smaller number of landmarks. More acutely, this problem could lead to a node being inaccurately associated with a bin due to significant link stress at a landmark.

Scalable Adaptive Hierarchical Clustering is a method of building a hierarchy of nodes, based on the notion of proximity, in a distributed and scalable way. The hierarchy is built through a series of "local" decisions involving only a small subset of the hierarchy's population for each decision. By using only a series local decisions and an innovative approach to adaptive cluster size distribution, the authors present a scalable means of constructing application-level overlay networks. Primarily this approach is intended for tree construction, however the algorithm can be trivially extended to generate overlay meshes.

6 Conclusions and Further Work

In this paper we have presented a novel approach to discovering a set appropriate programmable elements based upon service-specific deployment constraints.

We have suggested that deployment constraints differ in many dimensions on a per-service basis and programmable elements cannot natively understand the nature of these constraints. To solve this problem we propose that programmable service components are used to resolve deployment constraints and programmable elements make available to these components their local state for interrogation. We construct an overlay mesh consisting of programmable elements, which has the property of an element's immediate peers being topologically close. Using this overlay, we can perform searches on constraint dimensions whose ranges differ relative to network topology.

We have demonstrated some of the dimensions of scalability and effectiveness of searching over a `GROUPNET` mesh. Further work is required to evaluate the scalability and timeliness of the overlay construction algorithm. However, we feel this is secondary to searching performance, as joining the overlay would only occur at boot time. The use of programmable components for resolution may present unacceptable performance issues, we need to investigate these and suggest how performance could be improved. Additional consideration needs to be given to searching in multiple dimensions and how to converge on a configuration of resources with acceptable values within each. Further challenges lie in searching dimensions that have less correlation to the target and are not considered a part of the mesh dimensions.

7 Acknowledgements

This work was carried out under the Alpine project funded by BTexact Technologies. Paul Smith has a CASE studentship with the EPSRC and BTexact technologies.

References

[CSWH01] I. Clarke, Oskar. S., B. Wiley, and T.W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, New York, 2001. Springer.

[FG98] M. Fry and A. Ghosh. Application Level Active Networking. In *Computer Networks and ISDN Systems*, 1998.

[GF97] A. Ghosh and M. Fry. Javaradio: an application level active network. In *Third International Workshop on High Performance Protocols (HIPPARCH '97)*, London, June 1997.

[GFC00] A. Ghosh, M. Fry, and J. Crowcroft. An Architecture for Application Layer Routing. In *IWAN 2000*, Tokyo, Japan, October 2000.

[MCSH02] L. Mathy, R. Canonic, S. Simpson, and D. Hutchison. Scalable Adaptive Hierarchical Clustering. *IEEE Communications Letters*, 6(3):117–119, March 2002.

[RD01] A. Rowstron and P. Drushcel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001.

[RFH$^+$01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S Shenker. A Scalable Content-Addressable Network. In *ACM Sigcomm 2001*, August 2001.

[RHKS02] S. Ratnasmay, M. Handley, R. Karp, and S. Shenker. Topologically-Aware Overlay Construction and Server Selection. In *IEEE INFOCOM 2002*, June 2002.

[SBSH01] S. Simpson, M. Banfield, P. Smith, and D. Hutchison. Component Selection for Heterogeneous Active Networking. volume 2207 of *LNCS*, pages 84–100. Springer, September/October 2001.

[SMCH01] P. Smith, L. Mathy, R. Canonico, and D. Hutchison. ALM and ProgNets for v4-to-v6 Multicast Transition. In *IEEE OpenArch 2001 - Short Paper Session "Ghosts of the Net!"*, Anchorage, Alaska, April 2001.

[Wet99] D. Wetherall. Active network vision and reality: lessons from a capsule-based system. In *Operating Systems Review*, volume 34(5), pages 64–79, December 1999.

[WGT99] D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS: Network Services Without the Red Tape. *IEEE Computer*, 32(4):42–49, April 1999.