On the selection of management/monitoring nodes in highly dynamic networks

Richard G. Clegg, Stuart Clayman, George Pavlou, Lefteris Mamatas and Alex Galis
Department of Electronic Engineering, University College London, London, U.K.
Email: richard@richardclegg.org, sclayman@ee.ucl.ac.uk, gpavlou@ee.ucl.ac.uk, emamatas@ee.ucl.ac.uk, agalis@ee.ucl.ac.uk

***Abstract*—This paper addresses the problem of provisioning management/monitoring nodes within highly dynamic network environments, particularly virtual networks. In a network where nodes and links may be spontaneously created and destroyed (perhaps rapidly) there is a need for stable and responsive management and monitoring which does not create a large load (in terms of traffic or processing) for the system. A subset of nodes has to be chosen for management/monitoring, each of which will manage a subset of the nodes in the network.**

A new, simple and locally optimal greedy algorithm called *Pressure* is provided for choice of node position to minimise traffic. This algorithm is combined with a system for predicting the lifespan of nodes, and a tunable parameter is also given so that a system operator could express a preference for elected nodes to be chosen to reduce traffic, to be "stable", or some compromise between these positions. The combined algorithm called *PressureTime* is lightweight and could be run in a distributed manner. The resulting algorithms are tested both in simulation and in a testbed environment of virtual routers. They perform well, both at reducing traffic and at choosing long life-span nodes.

***Index Terms*—Network monitoring, Network management, Computer systems architecture**

I. Introduction

This paper addresses the problem of selecting a subset of management/monitoring nodes within dynamic networks. These networks are characterised by the fact that nodes and links may bootstrap or shut down, perhaps with little or no warning in order to adapt to local conditions. In this paper, we devised algorithms for management which: (i) select nodes for optimal placement within the network and (ii) select the nodes with the maximum estimated lifetime. Highly dynamic networks represent a number of scenarios where nodes and links could be temporary or may be short-lived, including virtual networks, mesh networks, mobile ad-hoc networks and the Internet of Things. The algorithms developed here are tested both in simulation and using monitoring software on a testbed of virtual machines. They are shown to be effective in realistic deployment scenarios.

The selected nodes will either generate or receive traffic and should be placed "close" (in network terms) to the nodes they would communicate with in order to reduce management traffic loads. Conversely, these nodes need to be "stable" (in the sense that they will remain connected to their neighbours). This creates potentially competing requirements, namely that nodes are selected for their placement within the network topology and that the nodes are selected to be stable. The contributions of this work are as follows: the design and validation of a locally optimal algorithm *Pressure* which places management/monitoring nodes in a dynamic network; the design and validation of an algorithm *PressureTime* which allows this placement to be tuned to account for estimates of node longevity; and finally, the construction of a virtual router testbed to test the above framework and algorithms in a real environment with real monitoring software.

Previous work by the authors in this area is described in [1]. That paper described a testbed implementation of thirty three virtual nodes in a static topology and a node placement algorithm known as HotSpot. This work uses the HotSpot algorithm from that paper (in specific the implementation called Greedy-B) as a point of comparison and develops the *Pressure* algorithm as an improved method for node placement. In comparison to the previous paper, this paper uses lightweight Java based virtual routers rather than having each virtual router running inside a (comparatively) heavier virtual machine controlled by a hypervisor and hence can deploy a larger testbed (220 virtual machines compared with 33) in addition to simulation results on even larger topologies. A companion paper [2] focuses on the monitoring architecture itself and the problem (in some sense the opposite problem) of a network where nodes are not removed but nodes may stop being management nodes if they are no longer needed.

The monitoring software used as a test application in this paper is known as the Information Management Overlay (IMO). The architecture, described in [1], is a distributed monitoring overlay where Information Collection Points (ICPs) send data to Information Aggregation Points (IAPs). The distributed IMO then collects and analyses filtered data from the IAPs.

The background and research context for the problem is given in section II. The node selection problem is formally described in section III. If node longevity is ignored, then the first step is to design an algorithm which places a new monitoring node in such a way as to maximally reduce the incurred network traffic. The *Pressure* algorithm is detailed in section III-A and is evaluated against both random placement and an algorithm from the literature known as HotSpot. A method to predict node lifespans based on the Kaplan–Meier estimator is described in section III-B. Finally, the two algorithms are combined in section III-C in a weighted way so that preference can be given to reducing traffic or increasing node longevity as desired. This complete algorithm is known as *PressureTime*.

II. Background

This research is within the field of highly dynamic networks: networks where nodes and links are regularly added or removed at short notice. This section provides a short summary of the dynamic networks context and management/monitoring context for the paper. Dynamic networks include virtual networks, logical networks, cloud computing networks, mobile ad-hoc networks, sensor networks and the Internet of Things. The paper is written within the context of future generations of management/monitoring infrastructures where nodes with special responsibility for management or monitoring tasks are embedded within the network under control. Such distributed management architectures need to be self-organized in order to match not only the requirements of users and network managers but also the constraints of the network infrastructures, including challenging networks with dynamic topologies (e.g., networks of mobile users, virtual networks with migratable virtual machines etc). An example of such a monitoring architecture would be the Information Management Overlay described in [3] and an example of such a management architecture would be the "cluster head" architecture in sensor networks [4].

A well-known dynamic network context is virtual networks [5], [6]. Virtual networks are a collection of virtual nodes connected together by a set of virtual links to form a virtual topology. In such networks, links and nodes may be reconfigured quickly and may be, for example, powered down to save energy or the node may be redeployed to a different logical area of the network. Both of these events are taken here to be equivalent to a node "death". On the other hand, virtual nodes may be brought online to deal with resources which are near their limits for bandwidth or CPU power. They are characterised in the literature either as a main means to test new Internet architectures or as a crucial component of future networks [7], [8].

Multiple logical networks can co-exist above the same physical substrate infrastructure. They can take the form of virtual private networks [9], programmable networks [10], overlay networks [5] or virtual networks [11]. The virtual nodes and links form a virtual topology over the underlying physical network.

Another area where management and monitoring of highly dynamic networks is important is that of cloud computing. The EU project RESERVOIR [12], [13] studied federated cloud computing and the interactions between a distributed system of computing clouds. The monitoring software used in this paper was developed within the RESERVOIR project for the task of collecting monitoring data in this type of dynamic network.

In mobile ad-hoc networks (MANETS), links and nodes may appear and reappear spontaneously with no prior notice [14], [15]. Sensor networks are another environment where network dynamism is extremely important [16], [17], [18]. In that case, the limited power budget gives increased importance to reducing the overall network traffic. A common management approach for such networks is that data collection will occur at many nodes but data is sent to one of a set of chosen nodes (sometimes termed "cluster-heads") for aggregation. In this context, the problem is one of choosing cluster heads which minimise power drain but do not put much traffic on the network [4]. A recent, related idea is that of the Internet of Things [19], [20] where many millions of objects are tracked, for example via RFID tags, forming a highly dynamic network where nodes and links may appear and disappear quickly but monitoring is crucial.

The context of the research is management and monitoring. Monitoring frameworks were analysed in [21], [22], [23]. Most proposed monitoring systems are not designed for dynamic networks. In fixed networks approaches to similar problems for traffic minimisation given a set of management or monitoring nodes are given by [24], [25]. Existing approaches over virtual networks primarily cover experimental infrastructure focused on monitoring or explore particular monitoring aspects [26]. A literature review on other monitoring approaches can be found in [2]. One approach detailed by the authors is in [27]. RESERVOIR addressed Federated Cloud Computing [12] [13] and it is the RESERVOIR monitoring software which formed the basis for the tests performed in this paper. The use of a real monitoring system is important in testing the devised placement strategy.

Other papers have addressed similar problems. For example [28] solves the problem of finding the smallest set of monitoring nodes to reduce the monitoring traffic to a given level in a static network. Also in the static network setting, [29] solves the problem of locating monitoring nodes to get the maximum coverage on a static network given assumptions about the cost of allocation and benefit of monitoring (and possible budget constraints). In the dynamic network setting, [30] uses a DHT to monitor the topology of a wireless network – in this context every node is part of the monitoring effort rather than a subset. In the MANETs context, [31] looks at a very specific monitoring problem, that of developing video sensors – in this context all nodes are collectors of data (like our ICPs) and the equivalent of IAPs are also fixed nodes in the network with higher capabilities for upload and download.

In a more generic setting, the problem of picking p sites to best serve n sites which generate "traffic" is known as the p-median problem [32]. In this problem, the p sites need not co-locate with the n sites and the weighted sum $\sum_{i=1}^{n} d_i r_i$ is minimised where d_i is the distance from site i to the nearest of the p sites and r_i is the rate of traffic generated by the i sites. The problem here is related but different in that the p sites must be co-located with the n sites and the distance d_i is over the network rather than a straightforward Euclidean distance. The typical solution to the p-median problem is relatively expensive computationally and does not deal with dynamic networks (although some exceptions exist [33]). However, the problem addressed here is in one sense simpler (only sites within the existing network can be chosen) and in another more complex (the network is changing quickly and site longevity is an issue).

Finally, the problem addressed here is tangentially related to "leader election problems" in distributed systems [34]. In this class of problems, a distributed system wishes to choose the "leader" in some unanimous way without a central control system. The algorithms here could all be run in a completely decentralised manner (the decentralised calculation of the *Pressure* score is described in section III-A) using such leader election algorithms.

The problem solved by this paper is a quite general one, the problem of selecting nodes which reduce management or monitoring traffic on a network. The addition of the node longevity aspect is a novel addition. However, it would help the reader's comprehension of the scheme by describing a few example use cases where the scheme described in this paper might be useful. In the scheme described in this paper, traffic is regularly sent from all nodes to a small subset of nodes chosen for their placement and longevity.

One context where such a set up is useful is service billing validation in virtual networks[35], [36]. In this scenario consider a user purchasing virtual nodes on an infrastructure provided by a third party. The user is being charged for traffic (and/or CPU) according to certain criteria provided by the host. The user wishes to check that the billing is correct but without generating more traffic on the network. In this scenario the IAPs collect netflow statistics and CPU usage statistics from the ICPs. Periodically, the IAPs summarise this data as an approximate bill (with summary data) which can be checked against the bill provided by the service provider and send this on to a management application. For scalability and to reduce traffic it would not be appropriate for every ICP to send all its netflow and CPU usage data directly to the management application. The IAPs would only need to send data rarely (as the billing period is likely to be, for example, monthly). An IAP which was shutting down would need to send on its summary data to the management application.

A second example context is resource discovery in the Internet of Things [20]. In this scenario a group of low-power devices communicate wirelessly over an ad-hoc network. The devices must minimise the traffic sent as they are low power. In this scenario the IAPs gather data about the connectivity of the network and the presence or absence of nodes joining and leaving the network in addition to the capabilities of the devices connecting to the network. The IAPs can then periodically and infrequently summarise and transmit this data to a management application (which may be centralised or distributed). The bulk of the traffic requirement is between ICPs and IAPs and the need for the IAPs to transmit to the management application is minimal hence the extra traffic required from IAPs to the management system is considered negligible.

III. Theoretical Model

This section describes the novel node selection algorithm *Pressure*, which is a locally optimal, greedy algorithm for placing a new management node in order to reduce network traffic. By locally optimal it is meant that each single node selected is the optimal node to reduce network traffic at that time but this does not account for the future evolution of the network or future nodes which may be selected. This section also describes a new *PressureTime* algorithm which combines the *Pressure* algorithm with a tunable life-time maximisation algorithm that attempts to select nodes based on their expected remaining lifetime. In the following section the nodes and links in a virtual network will be represented as nodes and edges within a graph, and the selected management/monitoring nodes as a subset of nodes within the graph.

A. Node selection algorithm

Consider a time-dependent graph $G(t) = (N(t), E(t))$ where $G(t)$ is the graph at time t, $N(t)$ is the nodes of the graph at time t and $E(t)$ is the set of undirected node edges. Assume that $G(t)$ remains connected for all t. At any given time, some subset of $N(t)$ are selected as leaders (namely management and monitoring nodes). Let $L(t)$ be the leaders (selected nodes) in the network at time t. Assume a node always connects to its nearest leader and will change leader if a nearer leader becomes available. In terms of the management application described in the introduction, the set $N(t)$ corresponds to the set of possible ICPs and the set $L(t)$ corresponds to the IAPs.

Informally, the desirable properties of selected nodes in $L(t)$ are:

- only a "small" subset of nodes are selected,
- nodes are not "too far" from their nearest leader and hence traffic over the network is minimised, and
- nodes which are selected will stay selected for a reasonable period of time before they either "die" (are deactivated or moved to a different part of the network) or are deselected. In this paper only "deaths" are considered.

Long-lived nodes are desirable because selecting nodes for management or for data collection will not be effective if the selected node disappears from the network soon afterwards. A selected node is only useful if its placement in the network is useful (that is, it is near nodes providing monitoring data or requiring management). The set of selected nodes $L(t)$ will determine how much monitoring and management traffic is placed on the network. In the work here nodes are never "deselected" as it is assumed that a long life time is an important quality in a selected node. In the case of a shrinking network this might lead temporarily to the proportion of management nodes being elevated.

The twin objectives of the algorithms proposed in this paper are (i) to select nodes which reduce management traffic on the network and (ii) to select nodes which exist for a long period of time. Instead of creating an objective function that is a weighted sum of these objectives, the approach taken here is to investigate tunable trade offs. Using this method, the network manager could choose a node selection policy which is efficient in terms of management traffic, or in terms of management node stability, or in terms of some combination of these aims, as appropriate.

For both the simulation results and in order to create an algorithm to minimise traffic on the network, an estimate of the amount of traffic generated is necessary. This can be simply done. Let $d_{i,j}$ be the distance from node i to node j in hops. Let l_i be the distance (in hops) from node i to its leader (in $L(t)$) in the network – assume that $l_i = \min_{j \in L(t)} d_{i,j}$ (that is a node's leader is the "nearest" node which is in the set of selected/leader nodes). Note that if $i \in L(t)$ then $l_i = 0$. Assuming that nodes are correctly sending traffic to the nearest leader at some rate r_i then, in each time unit, node i sends r_i units of traffic (this may be zero) over l_i hops. Therefore the total amount of traffic per unit time on the network at time t is given by

$$T(t) = \sum_{i \in N(t)} r_i l_i. \tag{1}$$

The best single node to add to the set $L(t)$ for the network at a given time is easily calculated (or locally optimal, greedy choice). It is necessary to calculate the amount of traffic which would be removed from the network were each node elected as leader. This can be thought of as traffic pressure and hence this algorithm is known as *Pressure*. For node i (which is not in $L(t)$) this *Pressure* score is

$$P(i) = \sum_{j \in N(t)} r_j \left[l_j - d_{i,j} \right]_+ \tag{2}$$

where $[x]_+$ means $\max(0, x)$.

Theorem 1: The pressure score $P(i)$ given by equation (2) is the amount of traffic which would be removed from the network if node i were added to the set of selected nodes. The node with the highest $P(i)$ is the optimal node to add to the set of leader nodes in order to reduce traffic on the network at time t.

Proof: Exactly those nodes with $j : d_{i,j} < l_j$ would send traffic to i were i added to $L(t)$. The current traffic from those nodes to their leader nodes is $r_j l_j$ and this would be removed. An amount of traffic $r_j d_{i,j}$ would be added. The total traffic removed from the network were i added to $L(t)$ would, therefore be $\sum_{j \in N(t)} r_j \left[l_j - d_{i,j} \right]_+$ which is exactly the expression for $P(i)$. (The $[x]_+$ ensures that only nodes which would send traffic to node i are counted.) The fact that the i with the largest $P(i)$ is the optimal node to add to $L(t)$ to remove traffic from the network at time t follows immediately. ■

In fact, this score can be trivially calculated in a distributed way. Each node knows its current "leader" and the distance to that leader. A node i sends out a score request with a hop count $h = 1$ to its neighbours. A neighbour j on receiving a score request from i (unless it has just received one) will either

1) If $h < l_j$ send back to i the value $r_j(l_j - h)$, set $h := h + 1$ and pass the request to its neighbours or
2) if $h \geq l_j$ ignore the request (since this node has a leader equally close to or closer than i).

The sum of the $r_j(l_j - h)$ values from the first step is $P(i)$ from equation (2). Once each node knows its score (it can tell which nodes are yet to reply from its routing table) then choosing the node with the highest score to be a "leader" is a solved problem using leader election algorithms. Knowing when to add a leader in a distributed way may be a more complex problem. If a given proportion of leader nodes is required then the current number of leaders and total number of nodes must be calculated either in a centralised or distributed way. The exact solution would depend on whether nodes (and leader nodes) left in a "graceful" way (notifying other nodes they would leave the network) or not.

As previously stated, choosing the node with the highest $P(i)$ score gives the optimal choice of a single node to select for the current network. This choice is only a locally optimal greedy choice as it does not account for the future evolution of the network (including future selections of management nodes).

B. Node longevity estimation algorithm

Determining node longevity requires an understanding of what will happen in the future. As such, node longevity must be estimated. Ideally we must be able to answer the questions *Given a node has been alive for length t how much longer is it likely to live?* and *How is this expected lifetime expressed as a percentile?* That is, given a current lifetime t, what proportion of nodes are likely to live longer than this one? This longevity estimation occurs in several steps. Assume the distribution of node lifetimes is given by some cumulative distribution function $F(t)$ where $F(t) = \mathbb{P}\left[\text{lifetime} < t\right]$. (This could be a simple distribution or could arise from dependent failures or from inhomogeneous subpopulations each of which has a different life-time distributions.) Firstly, $F(t)$ must be estimated from knowledge of observed lifetimes and node "deaths" in the system. Secondly, given $F(t)$ and some t what is the estimated "life remaining" for a given node and finally, given the estimate "life remaining" how does this compare with other nodes in the system?

In order for node longevity to be part of the estimation algorithm the node lifetime distribution must be estimated. There are two issues to overcome in solving this problem – firstly, many observations are "right censored", that is the "lifetime" of a node is not known until after it has died. Ignoring these will cause problems as it is likely that these are the longer lived nodes. The second problem is that there are lifetimes beyond which no (or little) data is available. If a node has been alive for a time t and no node has ever lived longer than this there still needs to be a way to estimate its remaining lifetime. For the first part of the problem (fitting an estimation of lifetime to right censored data), the Kaplan–Meier estimator is used to estimate an empirical distribution of lifetimes. For the second problem, a tail distribution is fitted to those longer lifetimes.

The Kaplan–Meier estimator is well-known in survival analysis and is used when estimation of lifetime distributions is needed in a situation where some of the objects under observation are still alive. Let t_i be the total life time for a node which has now died. That is, at least one node was in the system for exactly time t_i. Let d_i be the number of nodes which had a life time of exactly t_i. Finally, let n_i be the number of nodes which have lifetimes longer than t_i (these nodes may now be dead or may still be alive). The Kaplan–Meier estimator is an estimate of $F(t)$ the distribution function of the system and is given by

$$F_K(t) = \widehat{F(t)} = 1 - \prod_{t_i < t} \frac{n_i}{n_i + d_i}.$$

This is a general fitting procedure which produces a good estimate for any distribution of node lifetimes but it has a particular issue with estimation of remaining lifespan and in particular estimating lifespans for long-lived nodes: it cannot well estimate the tail of the distribution.

Let l be the longest lifetime observed for a node which has now died ($l = \max_i t_i$). If no nodes have ever lived longer than this then the Kaplan–Meier estimator gives $F_K(t) = 0$ for all $t > 0$ (since $n_i = 0$ for $i = \operatorname{argmax}_j t_j$) – in other words the estimate is that no nodes will ever live longer than l. On the other hand for a case where some existing nodes have lived longer than l then the estimator gives the same positive value for all $t > l$. The estimator cannot work for the largest node lifetimes (because it has no or little information with which to estimate). To compensate for this it is necessary to fit a tail distribution to $F(t)$ for large t. A reasonable assumption here is that the tail will be well-fitted by a log-normal distribution. This distribution includes a wide range of possibilities including that the node lifetime distribution is heavy tailed and is, in general, a good choice for a variety of survival situations. For a log-normal distribution the cumulative distribution function $F_L(x) = \mathbb{P}[X > x]$ has the form $F_L(x) = \frac{1}{2}\operatorname{erfc}\left[-\frac{\log(x)-\mu}{2\sigma^2}\right]$, where the parameters are μ, the "location" and $\sigma^2 > 0$ the "scale" (these are equivalent to the mean and variance in a normal distribution) and erfc is the complementary error function given by: $\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}dt$.

Let $F_K(x)$ be the estimated distribution from the K–M estimator and assume that this is reliable for all t less than some T – above T the log-normal tail is used. It is important that this tail "fits" onto the distribution estimated from the K–M estimator. $F_L(x)$ is modified in a linear manner to become $F'_L(x)$ so that $F_K(T) = F_L(T)$ – that is there is no discontinuity when the two distributions change. That is, for all $x \leq T$ the K–M estimator is used and for all $x > T$ the log normal tail estimator is used. The estimate for the complete distribution $F(x)$ will be

$$\widehat{F(x)} = \begin{cases} F_K(x) & x \leq T \\ F'_L(x) & x > T, \end{cases}$$

where $F'_L(x)$ is the log-normal tail with estimated values for μ and σ and scaled to fit the K–M estimate at T (that is $F_K(T) = F'_L(T)$).

Once the estimated $F(x)$ is known an estimate lifespan given current life time can be calculated. Let L be the random variable representing the lifetime of a node. Define $L(x)$ as the lifetime of a node which has already lived time x

$$L(x) = \mathrm{E}[L|L > x] = \int_x^\infty \frac{(1 - F(y))}{1 - F(x)}dy.$$

There are two cases for $x \leq T$ and $x > T$

$$\mathrm{E}[L(x)] = \begin{cases} \frac{1}{1-F_K(x)}\left[\sum_{i=l}^h \frac{t_i+t_{i+1}}{2}(1 - F_K(t_{i+1}))\right. \\ -\frac{t_l+x}{2}(1 - F_K(t_{l+1})) \\ -\frac{t_h+T}{2}(1 - F_K(t_{h+1})) \\ \left.+e^{\mu+\frac{1}{2}\sigma^2}\Phi\left(\frac{\mu+\sigma^2-\ln T}{\sigma}\right)\right] & x \leq T \\ \frac{1}{1-F'_L(x)}\left[e^{\mu+\frac{1}{2}\sigma^2}\Phi\left(\frac{\mu+\sigma^2-\ln x}{\sigma}\right)\right] & x > T, \end{cases}$$

where l is the largest number such that $t_l < x$ and h is the smallest number such that $t_h < T$ and $\Phi(x)$ is the CDF of a standard normal distribution given by

$$\Phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^x e^{-\frac{(y-\mu)^2}{2\sigma^2}}dy.$$

This section shows how to get an estimate for lifespan based upon the lifespan so far. Simulation results are shown in section VI-C. The next section shows how to combine the lifespan estimation with the placement algorithm to create a combined algorithm.

C. The combined algorithm

Having estimated the lifespan of a node, that estimate must be turned into a node selection policy. The placement policies given before, HotSpot, Pressure, and Random can be thought of as assigning a score to a node (with random assigning equal scores to each). Let S_i be the score assigned to node i by the placement policy. Now, a lifetime estimation system as described in the previous section can weight these scores to give preference to nodes which will be longer lived. The first step is to normalise the lifespan estimates into the range $[0, 1]$. Let L_i be the estimated remaining lifespan of node i. This could be immediately multiplied by the score to give a lifespan weighted score, however, this would allow no policy flexibility. Instead the score from the placement policy is multiplied by the normalised lifespan estimate raised to a positive power giving the lifespan adjusted score

$$S'_i = S_i L_i^\beta, \tag{3}$$

where $\beta \geq 0$ is the importance given to the lifespan estimate. The node with the highest lifespan weighted score is then selected. (Note that exactly the same ranking would be achieved by raising S_i to the power $1/\beta$.) If $\beta = 0$ then the lifespan estimate is ignored and the raw score is chosen. If $\beta = 1$ then the lifespan estimate is simply the product of the raw placement policy score and the proportion of the maximum estimated lifespan for this node. For $\beta > 1$ the lifespan estimate becomes more important and for $\beta < 1$ less important. Hence the β parameter can be used to tune the system designer's preference between selecting nodes with a long lifespan and selecting nodes with optimal positioning.

The node selection works as follows. Firstly, a minimum percentage of nodes to elect is set. Every time period the number of elected nodes is checked to see if it meets this minimum. If it is lower then a new node must be elected. This selection can be done either by a central controller or in a distributed manner using a node-election strategy. Following this, nodes connect to their closest (by hop count) elected node. Again, this can be done by a central controller or in a distributed manner by nodes flooding requests one hop, then two hops, then three hops, until they receive a response from an elected node. A maximum distance is set, beyond which, if a node receives no response from an elected node, then the node decides to become an elected node.

IV. Testbed

The proposed in-network management placement framework has evolved from the initial work in [1], and from the need to design an efficient monitoring framework for dynamic service elements in a cloud computing environment. Given the target application, a realistic testbed was paramount for evaluating the validity of the design assumptions and the overall monitoring system performance with respect to a number of set metrics. As such, a realistic network testbed was created for this (and future) virtual network research.

The Very Lightweight Service Platform (VLSP) testbed consists of a large number of software routers running as Java Virtual Machines (JVM) across a smaller number of physical machines. The routers are logically independent software entities which cannot communicate with each other except via network interfaces. The testbed set up has three components, as depicted in figure 1, which shows how the three components interact. The main component is the router itself, which runs in a JVM. The routers are complemented by a lightweight "local controller" which has the role of sending instructions to start up or shut down routers on the local machine and to routers to initiate or tear down connections with other machines. The experiment is supervised by a "global controller". The testbed system could be run in a completely decentralised manner without local controllers or global controllers, these are simply conveniences for management and experimental control.

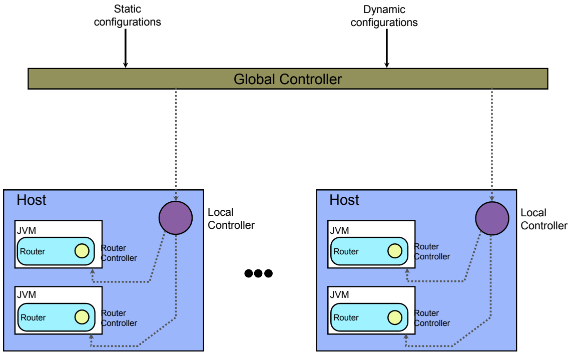

Fig. 1. The architecture of the VLSP testbed software

The software router is implemented in Java and is a relatively complex software entity. The routers hold network selections to the other virtual routers they are aware of, and exchange routing tables to determine the shortest path to each other router. Data packets are sent between routers and queued at input and output. A system of virtual ports (like the current transport layer ports) are exposed with an interface very similar to standard "sockets". Virtual applications can be run on the virtual routers and listen and send on their associated virtual sockets. Datagrams have headers with a source address, destination address, protocol, source port, destination port, length, checksum and time to live – many of the features of real IP packets are replicated.

A. Routing and packet transmission

Routing in these virtual routers is distance-vector based, incorporating the split horizon hack and poison reverse. To prevent routing storms, minimum times between table transmissions are set. In addition, because the experiment here demands a certain "churn" of virtual routers, addresses which disappear permanently must be dealt with. In distance vector routing it is well-known that dead addresses can leave routing loops. This is dealt with in the current system by implementing time to live (TTL) in packets (so that packets in a routing loop expire rather than fill the network) and also implementing a maximum routing distance beyond which a router is assumed unreachable and removed from routing tables (so that the routing loops do not persist forever).

Virtual applications can run on the routers. The networked applications tested include simple network test protocols such as ping, traceroute and ftp. The main virtual application used in the experiments described here is a monitoring application which sends monitoring information from collection points (ICPs) to aggregation points (IAPs). The virtual applications can listen on virtual ports and send datagrams to any virtual address and port.

Packets are queued both inbound and outbound. The outbound queue is blocking in order that transmitting applications can slow their sending rate. The inbound queue is tail-drop so that when too much traffic is sent drops will occur somewhere. TTL is decreased at each hop and, on expiry, a "TTL expired" packet is returned – this allows the virtual router system to implement traceroute as a virtual application. Virtual routers in the system send all traffic, including routing tables and other control messages, via network sockets. Control messages (routing tables, echo, echo reply, TTL expired and so on) are routed in the same way as data packets on a hop-by-hop basis using the routing tables. Datagram transmission is UDP-like, that is, delivery is not guaranteed and a failure to deliver will not be reported to the application (although if the router on which a virtual application runs has no route to the host this can be reported to the application).

B. Start up and tear down

Start up and tear down for routers is scheduled by the global controller and directly performed by the local controller, which resides on the same physical machine as the virtual router. Again, it should be stressed that the role of the global controller here should be seen as that of "experiment controller" rather than necessary management system. The virtual routers would operate perfectly well without such a controller, if for example, they were set up and connected manually or by some distributed control system. The local controller is necessary to spawn off new routers on a machine to avoid the overhead of, for example, making an ssh connection to start a new JVM on a remote physical machine. In addition, the local controller is used here to pass on global controller commands to shut down or connect virtual machines. It behaves in the same way a hypervisor does in other virtualised environments. However, it should again be stressed that this is to allow the experiments described here to be performed from a central global controller and not because the system designed requires co-ordination at this level. The global controller can be tuned to create different probability distributions for node lifetimes and for new node inter- arrival times.

V. Experimental Setup

The experiment control for the simulations and testbed has a common framework and uses the same control software for maximum comparability between simulation and testbed results. The simulation allows the experimental results to be tested on larger networks. The testbed allows realistic testing of monitoring software actually running in a genuine virtual router scenario.

In the testbed setting, the "global controller" was used to start, stop, and link routers and to collect data. The experimental set up used is based on probability distributions. Because virtual network research and cloud computing research is still in its early days the eventual topology and connection strategy of these networks remains a big unknown. The lifespan of the virtual components is another unknown. It may be that such networks will be highly dynamic with some nodes being brought into existence for a very short amount of time to serve a particular task and then shut down almost immediately. However, it may also be the cases that the networks evolve to become more stable with nodes and links becoming relatively permanent and long lasting.

The experiments are done on the basis of assuming a monitoring or management application. In the simulation this is assumed to be some generic monitoring or management application which sends one unit of traffic per unit time from every node to its nearest node in the set of selected nodes. For simplicity in these experiments every node is assumed to collect data (to be an ICP) and each node sends monitoring data once per second to its nearest aggregation point (IAP). In terms of equation (2) each node has an equal non zero r_i.

The simulation software allows node lifespans and new node interarrival times to be selected from a number of different probability distribution including "combined" distributions (for example 30% of lifespans selected from a Poisson distribution and 70% from a Weibull distribution). Tested distributions are Weibull, Gamma, log normal, normal, exponential, and uniform. The combination of node interarrival time distribution and node lifespan distribution will create a situation where the number of nodes in the situation will increase towards an equilibrium level at which nodes "die" as fast as they arrive – at this point the network size will fluctuate about a mean level. By changing the node inter-arrival time or the node lifespan distribution the mean number of nodes in the network can be controlled. The distributions used in the experiments are described in section VI.

Another important question is how routers link to each other. Again, because the simulation must be versatile, this is done using distributions. The number of links a router initially forms with other routers in the system is again picked from a distribution (in this case a discrete distribution with a minimum of one since each router must connect to at least one other). The routers connected to can be chosen entirely at random or by using the well-known Barabási–Albert (BA) preferential attachment model [37] which has been shown to explain some features of the real internet topology (see [38] for a review of this complex subject). Network disconnecting events were dealt with by ensuring that additional links were added to keep the network connected at all times.

A. Experiment parameters

For the results described in sections VI and VII, the node interarrival times were an exponential distribution. This was chosen as a simple to understand distribution which has been shown to well replicate a number of arrival processes for traffic on the internet (see [39] table 3). The rate of the exponential distribution was changed from an average of ten new routers a second to an average of one new router every fifty seconds. The simulation run period was one hour. Node lifetimes were 70% from a short lived exponential distribution (mean lifetime one minute) and 30% from a long lived log-normal distribution (mean lifetime a little under forty minutes – the log normal parameters were $\mu = 7.0$ and $\sigma^2 = 1.5$). The log-normal distribution was chosen as it is a common duration for the lifetime of many internet applications (again see [39] table 3).

Each router was initially connected to one router and then a random number of other routers chosen according to a Poisson distribution with mean 1.5. The routers were either chosen at random, in some experiments, or according to the preferential attachment model, in others [37]. If a node death caused a network disconnection an extra link was added to reconnect the network. Experiments lasted for one simulated hour (the simulation was not in real time). This was chosen so that simulation runs remained comparable to testbed runs. The timescale for testbed runs was chosen with a view to having each run take the maximum possible time without allowing the length of time to plot a graph becoming unreasonable. A graph with five points to a line and five repetitions for each point would take 25 hours per line. As can be seen, therefore, many of the testbed results graphs are the results of more than a week of real time.

It might be argued that such short timescales are not realistic for real life situations. However, it should be noted that in the simulation setting (non real-time), the time units are essentially arbitrary and nothing about the results would change if the time units were declared to be hours or days instead. As mentioned earlier, for the testbed the timescales were kept short so that experiment runs could be completed within a few days for each plotted graph (and that way a larger part of the parameter space could be explored). Note also that keeping the timescales short is, in fact, much more challenging for the testbed. When nodes may start up and shut down within minutes or seconds then network topologies change rapidly.

Three placement algorithms are used to calculate which nodes should be IAPs. Firstly the "Pressure" algorithm from section III-A is used. This algorithm is compared with the algorithm known as HotSpot from [1]. This algorithm gives a score $H(n)$ to node n according to the formula $H(n) = d_1^3 + d_2^2 + d_3$ where d_i is the number of neighbours exactly i hops from node n. (The paper [1] uses a static network topology but the HotSpot algorithm works in a dynamic environment just as Pressure does. In [1] this variant of HotSpot is described as Greedy-B.) In addition the "algorithm" Random is used as a worst-cases baseline – it simply selects any node with equal probability. All of these algorithms are computationally cheap to evaluate can be calculated either centrally or in a distributed manner as described.

In experiments IAPs were added according to one of the three algorithms Random, HotSpot, or Pressure and sometimes with the lifespan estimation algorithm in addition – to create RandomTime, HotSpotTime, and PressureTime algorithms. It was chosen that the number of IAPs would always be at least 10% of the number of ICPs and every network node would be an ICP. So, the first node to join a network will always be an IAP. A single new IAP is added (to make the percentage up to 10%) when either (a) an IAP died or (b) the network grew in size so that the proportion of IAPs fell below 10%. IAP addition was considered every 10 seconds of simulation. ICPs had their associated IAP changed to the nearest available on the same timescale.

VI. Simulation Results

This section describes work performed purely in simulation rather than using the VLSP testbed. Obviously, with pure simulation, runs are not constrained to be in "real-time" but where possible, for ease of comparison, results are obtained using the same parameters as for the testbed results in the next section.

While a wide set of parameters have been tested in simulation, obviously, only a limited number of results can be presented here. Again, for ease of comparison, results are presented with as many parameters the same as possible. The basic assumption behind these experiments is to abstract the management decisions which will cause the addition or removal of virtual routers and their linking as probability distributions. This allows the investigation of the monitoring software to proceed in isolation from specific worries about the management system which starts and stops virtual routers.

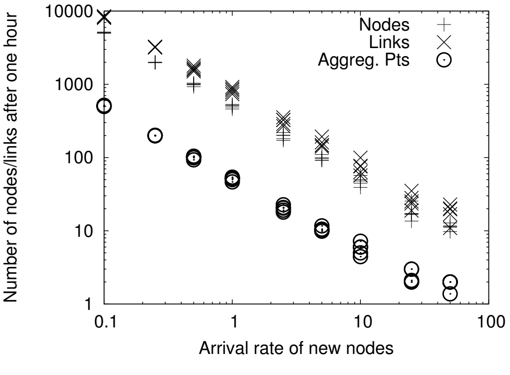

Fig. 2. Network size at the end of the experiment with different node arrival rates

Figure 2 shows on a logscale graph how the final network size varies with the arrival rate where arrival rates are between a node per 0.1 seconds (this is the mean of the exponential distribution) to a node per 50 seconds. Each setting is run five times to check repeatability. Note that at the smallest network size (arrival rate one node per 50 seconds) the network size was between 9 and 12 nodes with between 11 and 23 links. The number of aggregation points was either one or two. The fact that the number of aggregation points can double after the addition of a single run makes the results for these small network sizes highly variable. At the other end of the scale, the network size varied between 4998 and 5170 nodes and between 8058 and 8478 links with between 500 and 517 aggregation points. It should also be noted that the number of links per node could vary considerably between runs especially in the smallest networks.

A. Aggregation node placement for traffic minimisation

In this series of experiments, several policies are tested for their ability to minimise traffic load. The estimated traffic load on the network is that from (1), the number of links between an information source and its aggregation point (zero if the information source is an aggregation point) was added for every information source (which is every node). Five repetitions were made for each experiment. Preferential attachment and random links were used.

Figure 3 (bottom) shows the simulation results for the three placement algorithms with links selected at random. The graph shows the estimated traffic per node plotted against the number of nodes (on a logscale). The error bars show one standard deviation either side of the mean for the five runs with that policy and a given arrival rate. When the network is small, the three algorithms perform very similarly (the means are within a standard deviation) but for larger network sizes the Pressure algorithm is clearly the better algorithm for reducing traffic. The growth in traffic per node seems to be increasing with network size for the random algorithm and possibly for the HotSpot algorithm. For the pressure algorithm, however, it is hard to know if this increase continues at higher network sizes. Certainly the increase is at worst linear and this means that the algorithm is likely to scale well at higher network sizes.

Figure 3 (bottom) shows the same experiment with the preferential attachment model. In this case a small number of nodes get a very large number of links and it might be thought that this would advantage the HotSpot algorithm which will gravitate towards nodes which are highly connected. However, the performance was broadly similar to the no preferential attachment case. This was surprising considering how very different such networks are in terms of connectivity. The main differences seem to be in the smaller network sizes, however, such differences should be treated with great caution due to the large variance observed in the repeated runs with such networks (as evidenced by the large standard deviations).

The end conclusion of these experiments is that the Pressure algorithm is the best algorithm for reducing the estimated traffic. This should be no surprise as the choice made by this algorithm is locally optimal for the traffic estimation measure used – that is, every time a node is chosen it is the optimal node to reduce the traffic. The random placement caused on average 29% more traffic on the network than

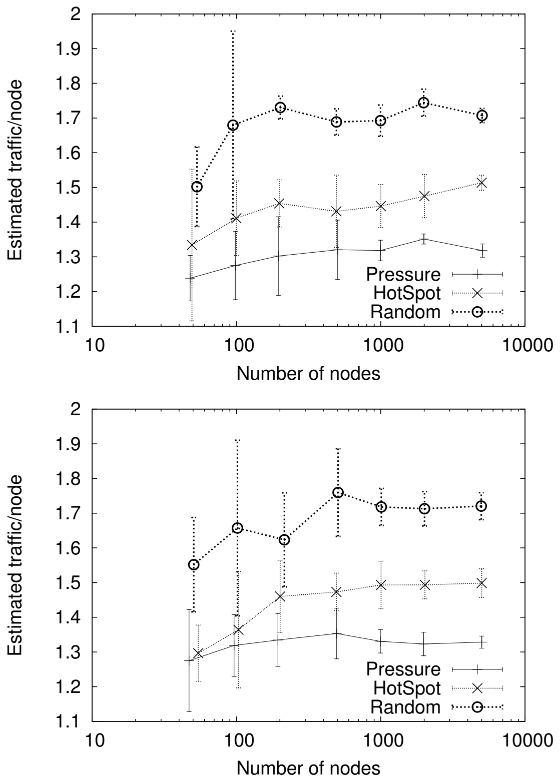

Fig. 3. Estimated traffic per node for various placement algorithms in simulation without (top) and with (bottom) preferential attachments

the Pressure algorithm in the largest network tested without preferential attachment and 28% more in the preferential attachment scenario.

B. Node lifetime estimation

In order to be able to estimate a nodes lifespan the estimation algorithm must first have a few observations of nodes to work with. The more node deaths observed the closer the fit to the real distribution is achieved. The first, and simplest test is to work with the lognormal distribution (this is the distribution assumed by the tail fitting algorithm).

Figure 4 shows the actual and estimated complimentary cumulative distribution function (CCDF) from the lifetime estimation algorithm after 20 observations, 100 observations and 1000 observations of lifetimes which have a lognormal distribution with $\mu = 7.0$ and $\sigma^2 = 1.5$. The line "Actual dist" is the correct distribution, "Estimated dist" is the raw Kaplan–Meier estimate and "Tail fit dist" is the estimate corrected with tail fitting. As mentioned in section III-B the Kaplan–Meier estimator is a non-parametric estimate which cannot make estimates about lifespans not yet observed so the distribution finishes with the longest lifespan observed in the system and cannot estimate further. The tail-fitting algorithm corrects this by fitting a lognormal tail to the final 20% of K–M points. When 20 observations have been made there is insufficient data for a tail fit and that line is omitted. The K–M algorithm does a good job at distribution estimation for early parts of the curve but is very poor at the tail. Unfortunately it is the tail of the distribution which is often most important for lifespan estimation (while only a small percentage of nodes have lifespans over 100,000 seconds they contribute a great deal to the expected lifespan of an average node). At 100 observations the fit to the distribution with the tail fitting is good and at 1,000 observations nearly perfect. In a running system the controller would quickly get 1,000 observations of nodes and could then run using only its most recent estimations. The estimation procedure is computationally inexpensive.

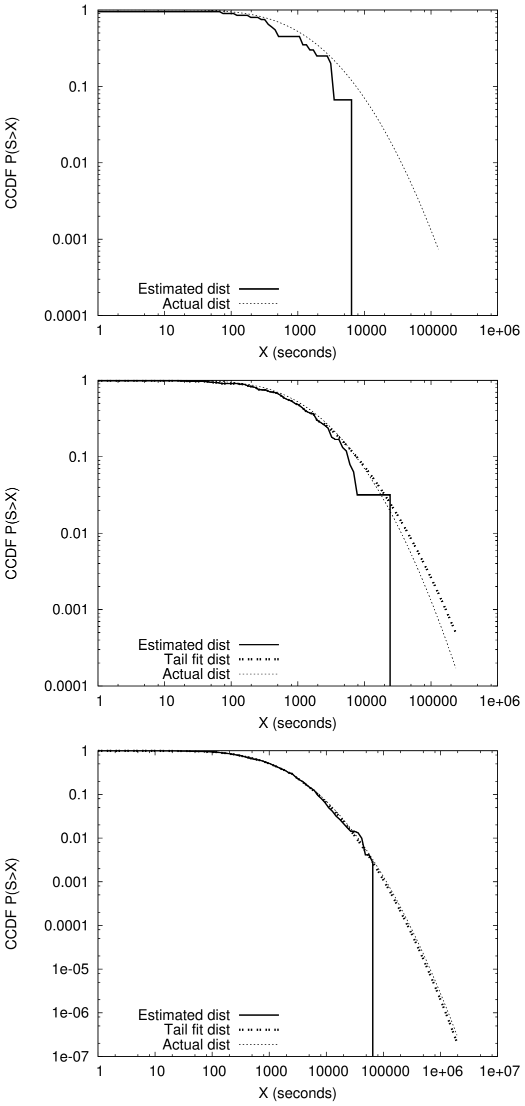

Fig. 4. Node lifespan distribution estimation for 20 lifespan observations (top), 100 (middle) and 1000 (bottom)

Figure 5 shows the lifespan estimation for the same experiment. As can be seen, after only 20 observations the estimation is quite wrong. At the tail the estimator cannot make an evaluation of lifespan from data and the program simply estimates that the node will live as long again as its current lifespan (the linear part of the estimate). By chance this

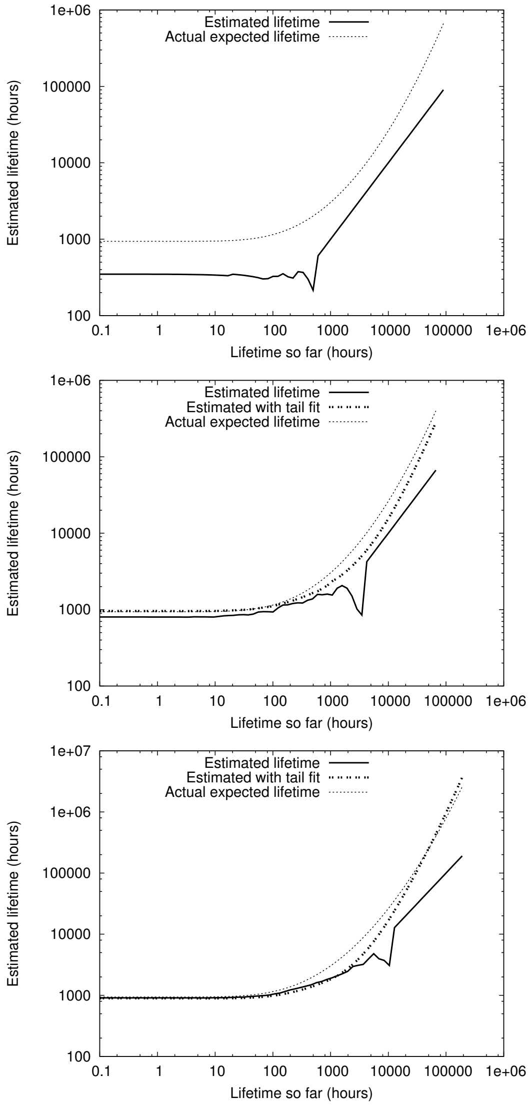

Fig. 5. Node lifespan estimation for 20 lifespan observations (top), 100 (middle) and 1000 (bottom)

follows the curve relatively well. The lifespan estimates here are all much too short (by a factor of about five). This is partly due to no observations of long lived nodes and partly due to the lack of tail fitting. By 100 observations the tail fitting algorithm can estimate the curve tail and makes a good job of estimating lifespan. By 1000 observations the lifespan estimate is very good indeed but with a slight "miss" in estimating the lifespan when it is around 1000 hours.

Of course, the lognormal distribution is the easiest one for the lifespan estimation algorithm. Figure 6 shows the lifespan estimation working with the distribution used in the experiments throughout this paper which is 30% lognormal with $\mu = 7.0$ and $\sigma^2 = 1.5$ and which is 70% exponential with $\mu = 60.0$. So 70% of nodes are very short lived with a mean lifespan of 60 seconds and 30% are relatively long lived with a mean lifespan of 2,300 seconds ($\exp \mu + \sigma^2/2$). The results of this are shown for 100 and 1000 samples. At 100 samples the estimation is already fairly good for the tail fit. The raw K–M estimator fails badly as would be expected (the peculiar shape of the raw estimator is where it gets near the point when the distribution estimate falls to zero and before the "expected life is equal to current life" approximation kicks in.

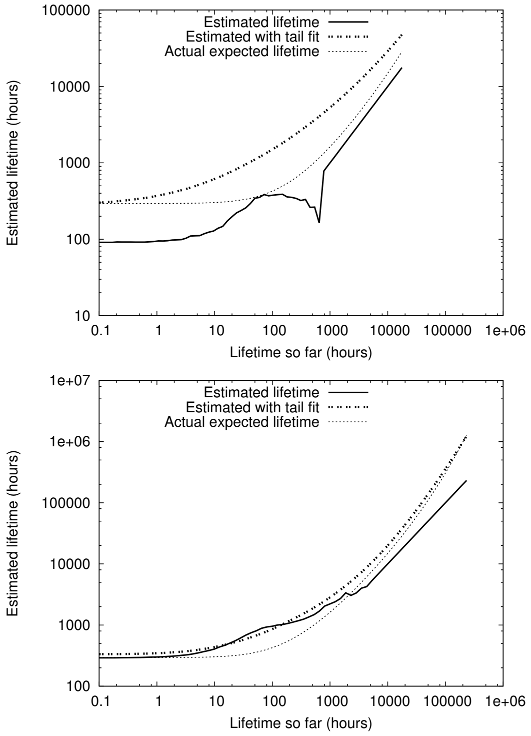

Fig. 6. Node lifespan estimation for 100 lifespan observations (top) and 1000 (bottom)

The conclusion of this section, therefore, is that the lifespan estimation algorithm does an extremely good job of estimating the expectation of node lifetime. Of course, in high variance distributions this may not translate well into improved node lifetime since a node with high expected lifetime may well have a high probability of dying very soon but a low probability of a very long life to follow.

C. Lifetime maximisation

In this section, simulation results are tried with various policies to maximise the lifespan of nodes selected as IAPs (or rather to select IAPs which will have long lifespans after selection). The lifespan estimation method is as described in section III-C. The β parameter trades off the strength of preference for nodes with higher estimated lifespans with $\beta = 0$ meaning ignore node lifespans totally and $\beta = 100$ indicating a very strong preference where the node's placement will have little or no bearing on whether it is chosen.

The experiment uses the same parameters as the previous section with the highest arrival rate tried (around 36,000 nodes by the end of a typical simulated hour). The β parameter is varied for each policy and the results collected. The mean lifespan is measured for IAPs which have exited the system (this creates a slight bias against the selection algorithms as obviously those nodes which are very long lived will not be counted in such a metric as they will not die before the experiment ends) and the figure given is the figure from the time they are selected as an IAP until the time they exit the system. Ten runs are performed for each β value for each placement policy with no preferential attachment.

For ease of plotting the time policy strength is given a single digit representing policy strength as shown below

Strength	β	meaning
0	0	Lifespan estimate ignored
1	0.1	Most importance on node placement
2	1	Compromise setting
3	10	Heavy emphasis on lifespan estimate
4	100	Lifespan almost sole measure

note that this is simply a convenience for graphing (there is no easy way to plot the β values since 0 does not show up on a logscale). The experiment is particularly interesting as there are many conflicting factors. For example, the HotSpot algorithm is likely to pick nodes with many neighbours and those nodes are likely to be older nodes in the system and hence longer lived. The Pressure algorithm may have this property too.

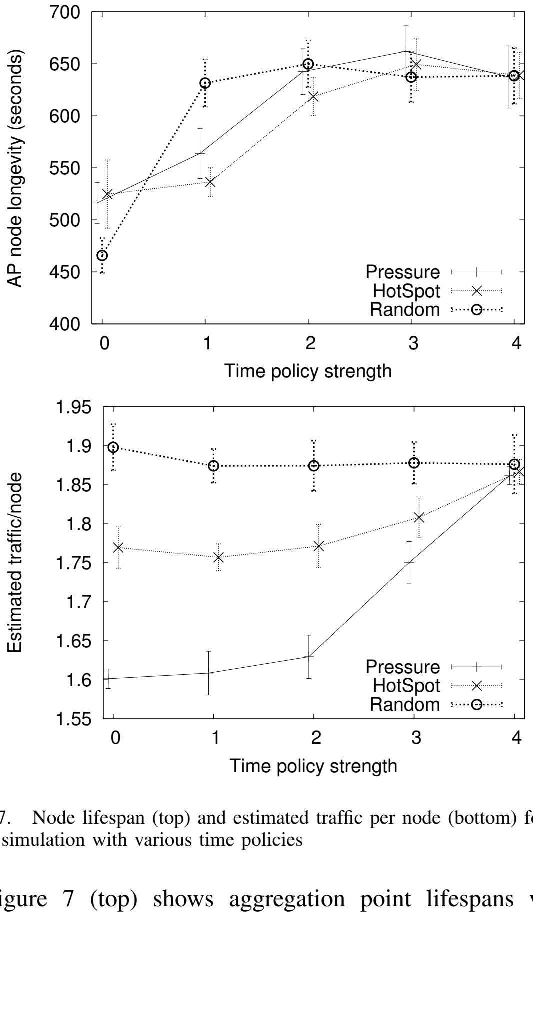

Fig. 7. Node lifespan (top) and estimated traffic per node (bottom) for one hour simulation with various time policies

Figure 7 (top) shows aggregation point lifespans when various levels of importance are placed on longevity. Note that in this graph the points are slightly displaced horizontally so the error bars do not lie over each other. The first thing to notice is that the Pressure and Hotspot algorithms have slightly longer access point lifetimes even when lifespan estimation is not considered. This is because those algorithms will favour making nodes with more connections aggregation points and those nodes are likely to be older (as nodes acquire links as they become older). It is also worth noting that the mean lifespan for ordinary nodes is only 180 seconds in all simulations (the difference is because nodes which live long are candidates for selection as an aggregation point more often, even in the Random algorithm).

The random algorithm with any $\beta > 0$ selects purely on estimated longevity, therefore the IAP node longevity should be the same for strengths 1, 2, 3 and 4 and indeed this appears to be the case. For the other two algorithms, the longevity appears to increase slightly with the weakest time policy strength (but this is within a standard deviation so this may be an illusion). For policy strength 2, both have increased the longevity greatly and little further increase in node longevity appears to be achieved by the stronger policies.

Figure 7 (bottom) shows the same results from the point of view of estimated traffic. The results with time policy strength zero should be the same as the rightmost points of 3 (top). One interesting feature of this graph is that the random policy and the HotSpot policy actually appear to perform slightly better when given a small preference for long lived nodes. In the case of the Random policy it could be because the nodes selected for lifespan acquire links and hence later become useful in the simulation. As expected, in Random the policy strengths 1, 2, 3 and 4 are identical (because there is no placement score to trade off and the highest node lifetime is always chosen). For Pressure and HotSpot it can be seen that the higher lifetime selection policies have a detrimental effect on those policies ability to minimise traffic and eventually at the highest time policy strength these policies are no more effective than random at minimising traffic.

Figure 8 show the same figures with the context of a network using preferential attachment for link connections. In the case of preferential attachment older links will tend to gather more links to them. This makes many of the effects in the non-preferential attachment case much more pronounced. In particular notice the sharp drop in traffic per node in the Random case when the lifespan policies are applied. Older nodes (and hence nodes likely to live longer) are also likely to have many links and hence are a good candidate for minimising traffic. Notice also that in this case for the time policy strengths one and two the Pressure algorithm does not suffer increased traffic. Because long-lived nodes will tend to gather more links, to some extent, picking long-lived nodes as aggregation points will be a good choice for position as well as new nodes in the system will tend to connect to them. It is somewhat surprising that the HotSpot algorithm does not, in fact, appear to be picking long-lived nodes initially in this algorithm. Therefore the HotSpot algorithm gains most in terms of aggregation point lifespan by selecting for life span.

Figures 7 and 8 show clearly the tradeoff allowed by

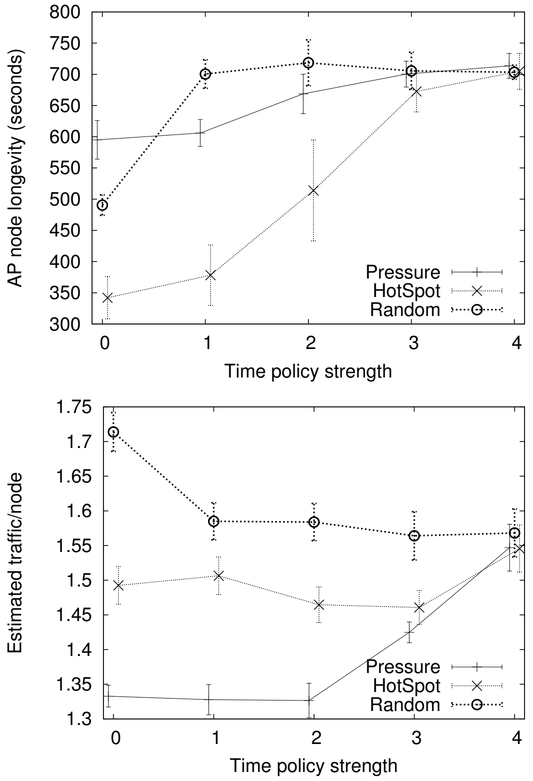

Fig. 8. Node lifespan (top) and estimated traffic per node (bottom) for one hour simulation with various time policies

equation 3 and the β parameter. A manager could use high values of β when stability is the primary concern and traffic due to node placement is less of an issue. However, the very highest values did not seem to make too much difference in lifespan seen. Therefore, some degree of optimal node placement can be made with $\beta = 10$ (policy strength 3) without a penalty to the achieved stability. On the other hand, if traffic reduction is the priority then it can be seen that $\beta = 0.1$ (policy strength 1) produces an increase in node lifetime with little damage to the traffic reducing effects.

VII. Testbed Results

The testbed results, using the VLSP platform, were run on six physical machines. For repeatability the VLSP experiments were run with the same parameters as the simulation experiments wherever possible. The VLSP can handle fewer nodes than pure simulation, however, it has the advantage that the traffic measured is real traffic, in our case, generated by the monitoring software. This traffic differs considerably from the estimated traffic algorithm as the routing may be non-optimal (since real systems take time for routing to converge after a node or link vanishes) and packets may be dropped. Information sources may continue to send traffic to dead aggregation points (until they are informed of a new aggregation point) and by the nature of distance vector routing, transient routing loops may occur.

The parameter settings were largely the same as for the simulation but the arrival rates were kept lower as, obviously, the VLSP could handle fewer routers than the simulation. The information sources send a small amount of monitoring information to the aggregation points every second. Aggregation points are recalculated (potentially added and potentially information sources are moved to new aggregation points) every ten seconds as with the simulation.

A. *Traffic minimisation*

Figure 9 shows the testbed results for the three node placement algorithms with and without preferential attachment. Because the network sizes are smaller than in simulation and the number of runs fewer, the error bars representing one standard deviation either side of the mean are relatively larger than in the simulation case. However, the pattern still remains clear with the Pressure algorithm out performing HotSpot which in turn out performs Random. For the smallest network size tested there are only approximately four aggregation points and the error bars are very large. This indicates that in those settings the difference between runs with the same algorithm is much greater than the difference made by the algorithms themselves. That is to say, the statistical variation in the network topologies themselves had a greater effect on traffic than the node placement algorithms. This is not surprising when such nodes would only have three or four aggregation points (and also the difference between a node with 39 nodes and hence 3 aggregation points and 40 notes and 4 aggregation points would be extremely large).

It should also be noted that the routing system responded well to the highly dynamic nature of the network. Even though in the most extreme scenario tested one virtual router would die on average every three seconds the number of lost packets in the network remained small (three runs averaged 0.75, 0.32 and 0.36 packets dropped per second for the last ten minutes of simulation – less than 0.5% of the traffic).

B. *Lifespan maximisation*

Figure 10 shows the results of the lifespan maximisation policies run on the testbed. The testbed runs have fewer nodes than simulation runs. The arrival rate is set to a mean value of one node every two seconds which means that in equilibrium there are around 250 nodes but the mean "churn" rate simulates around 1,500 nodes in the full hour of testbed time. These figures are for a non preferential attachment model with the same time policies as for the simulation results shown in figure 7. Ten runs are done for each policy and a data point is the mean and standard deviation of these points for a given placement policy and time policy as described in the table in section VI-C. Because of the smaller number of nodes these graphs are harder to interpret than the ones for the simulation and the error bars are larger

Figure 10 (bottom) shows that for the Pressure and HotSpot policies the trend appears to be what we would expect. The node maximising policies increase lifespan. However, with fewer nodes (and fewer node deaths from which to create estimates) the lifespan extension is not as great. The figure

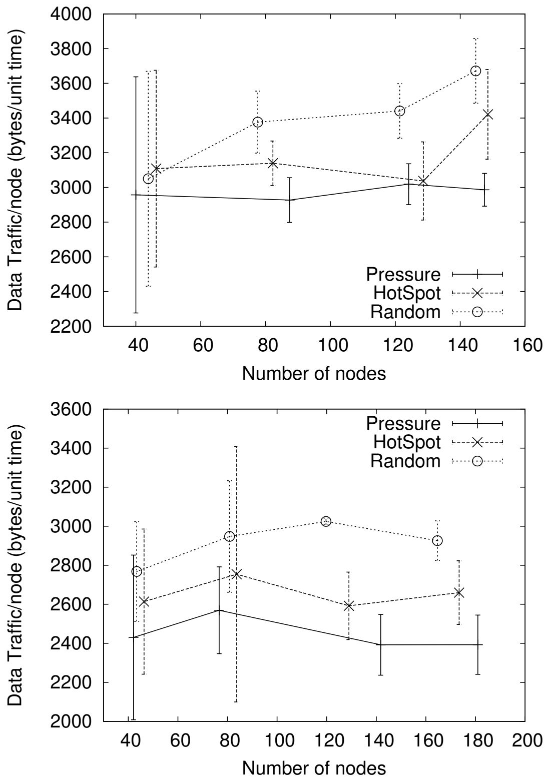

Fig. 9. Data traffic per node for various placement algorithms in the testbed without Preferential attachment (top) and with (bottom)

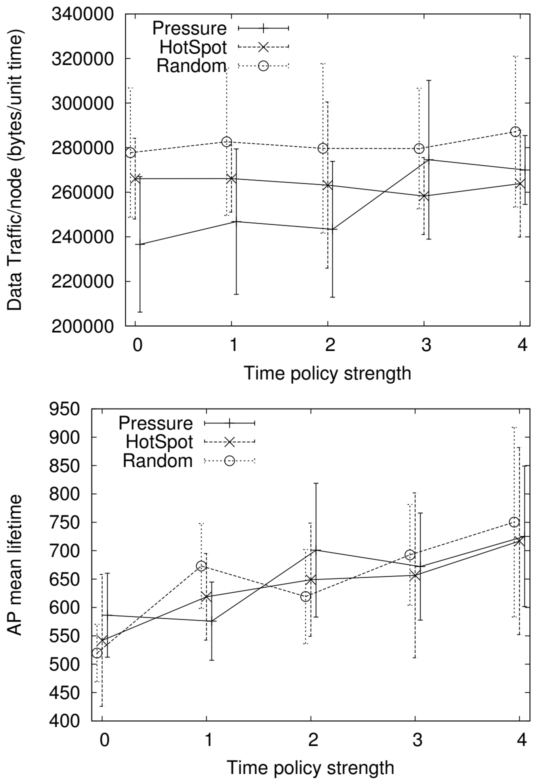

Fig. 10. Data traffic per node (top) and IAP lifespan (bottom) for one hour on the testbed with various time policies

also shows the effects of the various time policies. As expected the stronger (higher numbered) policies favouring longer lived nodes are producing a higher mean lifespan for nodes. Again, however, the size of the error bars makes interpreting too much from the graphs difficult. It seems loosely to show that the lifespan estimation does produce a benefit (particularly in the random case). The fact that the benefit is smaller is to be expected as the algorithm has fewer nodes to "learn" the distribution from in this system. Figure 10 (top) shows the same run from the point of view of network traffic. The results are what would be expected from simulation (see figure 7) for the Pressure and Random algorithms. The HotSpot algorithm does not seem to suffer increased traffic as the time penalty increases. However, this may be a product of the large error bars. While these figures are consistent with the simulation results (with that single exception) the large error bars make it hard to interpret further.

VIII. Summary and Conclusions

This paper has investigated the selection of management/monitoring nodes in a highly dynamic network environment. In a rapidly changing network environment, monitoring and management are both extremely critical and difficult to achieve tasks. The management algorithms presented are computationally "cheap" and can be implemented in a decentralised way. They can select nodes to reduce resulting traffic or to choose "stable" nodes (that is, those most likely to be long-lived). A new, locally optimal algorithm (*Pressure*) was proposed. It selects the best node to reduce current management and monitoring traffic on the network. Following the problem of selecting nodes by lifespan (that is the time remaining until the node exits the network) is investigated. A simple estimator for remaining lifespan is given, together with an improved estimator with tail fitting. The combined algorithm which tackles node placement and node-lifetime maximisation is known as *PressureTime*.

The algorithms were tested both in simulation and on a new testbed environment, the VLSP testbed. The simulation environment tests networks with up to 5,000 nodes and with a "churn" of 36,000 nodes (36,000 nodes in total created of which approximately 5,000 are in use simultaneously) over the simulation period. The testbed environment used virtual routers running on Java virtual machines and emulated around 220 nodes with a "churn" of around 6,000 nodes over the simulation period.

The *Pressure* algorithm proved successful in reducing the traffic generated by the monitoring nodes when compared with Random and HotSpot algorithms. In addition, the *PressureTime* algorithm combines node placement with lifespan maximisation in a tunable way. To some extent, selecting optimally placed nodes and selecting nodes with the longest lifespan are potentially "competing" problems (although in the case of the preferential attachment system it was shown that sometimes long-lived nodes were also well-placed). It was further demonstrated that the *PressureTime* algorithm is tunable, so a system manager could optimise their system to choose management and monitoring nodes either to reduce monitoring traffic, to increase monitoring node lifespan, or some combination of these two goals.

Future work includes developing methods for controlling virtual network stability. One problem is how to select management nodes with regard to future placement of links and nodes. Another problem is how to select links between virtual nodes in order to maximise the stability of a network, that is the problem of choosing those virtual links which reduce the risks of oscillations and disconnections within the network.

Acknowledgment

This work is partially supported by the European Union through the Autonomic Internet (AutoI) [3], [40], the RESERVOIR project [13] and the UniverSELF [41] project of the 7th Framework Program.

References

[1] L. Mamatas, S. Clayman, M. Charalambides, A. Galis, and G. Pavlou, "Towards an information management overlay for emerging networks," in *IEEE/IFIP NOMS*, 2010.

[2] S. Clayman, R. G. Clegg, L. Mamatas, G. Pavlou, and A. Galis, "Monitoring, aggregation and filtering for efficient management of virtual networks," in *International Conf. on Network and Service Management (CNSM)*, October 2011.

[3] A. Galis, S. Denazis, A. Bassi, P. Giacomin *et al.*, *Management Architecture and Systems for Future Internet Networks*. IOS Press, April 2009.

[4] K. Akkaya, F. Senel, and B. McLaughlan, "Clustering of wireless sensor and actor networks based on sensor distribution and connectivity," *J. Parallel Distrib. Comput.*, vol. 69, pp. 573–587, June 2009.

[5] N. M. K. Chowdhury and R. Boutaba, "Network virtualization: State of the art and research challenges," *IEEE Communications Magazine*, vol. 47, no. 7, 2009.

[6] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *PRESTO, CONEXT Workshop*, 2010.

[7] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, pp. 34–41, April 2005.

[8] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862 – 876, 2010.

[9] L. Andersson and T. Madsen, "Provider provisioned virtual private network (VPN) terminology," *Internet Engineering Task Force, RFC 4026*, March 2005.

[10] A. Galis, S. Denazis, C. Brou, and C. Klein, *Programmable Networks for IP Service Deployment*. Artech House Books, 2004.

[11] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, pp. 34–41, April 2005.

[12] B. Rochwerger, D. Breitgand, D. Hadas, I. Llorente, R. Montero, P. Massonet, E. Levy, A. Galis, M. Villari, Y. Wolfsthal, E. Elmroth, J. Caceres, C. Vazquez, and J. Tordsson, "An architecture for federated cloud computing," *Cloud Computing*, 2010.

[13] B. Rochwerger, D. Breitgand, E. Levy, A. Galis *et al.*, "The RESERVOIR model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, 2009.

[14] A. Malatras, G. Pavlou, and S. Sivavakeesar, "A programmable framework for the deployment of services and protocols in mobile ad hoc networks," *IEEE Transactions on Network and Service Management*, vol. 4, no. 3, pp. 12–24, Dec. 2007.

[15] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, "Reconfigurable context-sensitive middleware for pervasive computing," *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 33–40, 2002.

[16] C. Olston, B. T. Loo, and J. Widom, "Adaptive precision setting for cached approximate values," in *ACM SIGMOD*. New York, NY, USA: ACM, 2001, pp. 355–366.

[17] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi, "Holistic aggregates in a networked world: distributed tracking of approximate quantiles," in *ACM SIGMOD*, 2005, pp. 25–36.

[18] A. Sharaf, J. Beaver, A. Labrinidis, and K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," *The VLDB Journal*, vol. 13, no. 4, pp. 384–403, 2004.

[19] E. Welbourne, L. Battle, G. Cole, K. Gould, K. Rector, S. Raymer, M. Balazinska, and G. Borriello, "Building the internet of things using RFID: The RFID ecosystem experience," *Internet Computing, IEEE*, vol. 13, no. 3, pp. 48–55, 2009.

[20] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[21] A. G. Prieto and R.Stadler, "A-GAP: An adaptive protocol for continuous network monitoring with accuracy objectives," *IEEE Transactions on Network and Service Management*, vol. 4, no. 1, 2007.

[22] A. G. Prieto and R. Stadler, "Controlling performance trade-offs in adaptive network monitoring," *11th IFIP/IEEE International Symposium on Integrated Network Management*, 2009.

[23] A. Lahmadi, L. Andrey, and O. Festor, "Design and validation of an analytical model to evaluate monitoring frameworks limits," *The Eighth International Conference on Networks*, 2009.

[24] C. H. Crawford and A. Dan, "emodel: Addressing the need for a flexible modeling framework in autonomic computing," *International Symposium on Modeling, Analysis, and Simulation of Computer Systems*, 2002.

[25] X. Dong, S. Hariri, L. Xue, H. Chen *et al.*, "Autonomia: an autonomic computing environment," *Performance, Computing, and Communications Conference*, pp. 61–68, 2003.

[26] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," in *ACM SIGCOMM*, 2008, pp. 231–242.

[27] S. Clayman, A. Galis, and L. Mamatas, "Monitoring virtual networks with lattice," in *Management of Future Internet - ManFI 2010*, 2010.

[28] C. Chaudet, E. Fleury, I. G. Lassous, H. Rivano, and M.-E. Voge, "Optimal positioning of active and passive monitoring devices," in *Proc. of CoNEXT*, 2005, pp. 71–82.

[29] K. Suh, Y. Guo, J. Kurose, and D. Towsley, "Locating network monitors: complexity, heuristics, and coverage," in *Proc. of INFOCOM*, vol. 1, 2005, pp. 351–361.

[30] C. Popi and O. Festor, "A scheme for dynamic monitoring and logging of topology information in wireless mesh networks," in *Network Operations and Management Symposium*, 2008, pp. 759 –762.

[31] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P. Bellavista, and A. Corradi, "Mobeyes: smart mobs for urban monitoring with a vehicular sensor network," *Wireless Communications*, vol. 13, no. 5, pp. 52–57, 2006.

[32] S. Hakimi, "Optimum location of switching centers and the absolute centers and medians of a graph," *Operations Research*, vol. 12, pp. 450–459, 1964.

[33] Z. Drezner, "Dynamic facility location: The progressive p-median problem," *Location Science*, vol. 3, no. 1, pp. 1–7, 1995.

[34] E. Korach, S. Kutten, and S. Moran, "A modular technique for the design of efficient distributed leader finding algorithms," *ACM Trans. Program. Lang. Syst.*, vol. 12, no. 1, pp. 84–101, 1990.

[35] L. DaSilva, "Pricing for QoS-enabled networks: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 3, no. 2, pp. 2–8, 2000.

[36] M. Lindner, F. G. Marquez, C. Chapman, S. Clayman, and D. Hendriksson, "Cloud supply chain – a comprehensive framework," in *CloudComp 2010, 2nd International ICST Conference on Cloud Computing*, October 2010.

[37] A. L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, (1999).

[38] R. Clegg, C. Di Cairano-Gilfedder, and S. Zhou, "A critical look at power law modelling of the internet," *Computer Communications*, vol. 33, no. 3, pp. 259–268, 2010.

[39] C. Di Cairano-Gilfedder and R. Clegg, "A decade of internet research: Advances in models and practices," *BT Technology Journal*, vol. 23, no. 4, pp. 115–128, 2005.

[40] D. Macedo, Z. Movahedi, J. Rubio-Loyola, A. Astorga, G. Koumoutsos, and G. Pujolle, "The AutoI approach for the orchestration of autonomic networks," *Annals of Telecommunications*, vol. 66, pp. 243–255, 2011.

[41] UniverSELF consortium, "UniverSELF project," http://www.univerself-project.eu/.

Authors' Biographies

Richard G. Clegg is a Senior Research Fellow at University College London. His PhD in mathematics and statistics from the University of York was gained in 2005. His research interests include network topologies, network traffic statistics and overlay network.

Stuart Clayman received his PhD in Computer Science from University College London in 1994. He has worked as a Research Lecturer at Kingston University and is now a Senior Research Fellow at UCL. He co-authored 30 conference and journal papers. His research interests and expertise lie in the areas of software engineering and programming paradigms; distributed systems; cloud systems, systems management; networked media; and knowledge-based systems. He also has previous extensive experience in the commercial arena undertaking architecture and development for software engineering and networking systems.

George Pavlou holds the Chair of Communication Networks at the Dept. of Electronic & Electrical Engineering, University College London. Over the last 25 years he has undertaken and directed research in networking and network/service management, having extensively published in these areas. He has contributed to ISO, ITU-T and IETF standardization activities and has been instrumental in a number of key European and UK projects that produced significant results. His current research interests include traffic engineering, information-centric networking, autonomic networking, communications middleware and infrastructure-less wireless networks.

Lefteris Mamatas is a postdoctoral researcher at the University College London. His research interests lie in the areas of autonomic network and services management, delay-tolerant networks and energy efficient communication. He participated in several international research projects, such as UniverSELF (FP7), Autonomic Internet (FP7), Ambient Networks (FP7) and Extending Internet into Space (European Space Agency). He published more than 30 papers in international journals and conferences. He served as a TPC chair in the WWIC 2012 and E-DTN 2009 conferences.

Alex Galis is a Visiting Professor at University College London. He has co-authored 7 research books and more than 190 publications in journals and conferences in the Future Internet areas: networks, services and management. He acted as PTC chair of 14 IEEE conferences and reviewer in more than 100 IEEE conferences http://www.ee.ucl.ac.uk/~agalis. He worked as a Vice-Chairman for the ITU-T Focus Group on Future Networks, which is defining design goals and requirements in the Future Network http://www.itu.int/ITU-T/focusgroups/fn/index.html.