Traffic Reduction in Randomly Generated Information Networks

16th International Symposium on Mathematical Theory of Networks and Systems, MTNS 2004

Natasha Kenny[*], Mark Muldoon[†]
UMIST, Manchester, England
Problem suggested by Keith Briggs[‡], BT Exact, Ipswich, England

May 2004

Abstract

A random Boolean network (RBN) is used to model a computer network. When thinking of a RBN in this way; a node becomes a server, edges become network connections and state requests become packets. The synchronous updating, found in classic RBNs, is compared to the fixed periodic refreshing used in, for example, the domain name system (DNS).

The proposed method for reducing traffic in the network is to give each node the power to regulate the accuracy of the information it holds. That is, the nodes attempt to control the probability that their information is up to date (correct). Each node is fitted with a learning algorithm that requires only local information and determines how frequently the node should refresh in order to achieve its desired accuracy. This approach permits substantial reductions in the volume of traffic without altering network topology or consideration of particular routing/queueing protocols.

1 Background

The setting for the model is Boolean networks. Random Boolean networks have been chosen due to the variety of dynamics they can produce. These dynamics will be used to represent a range of traffic flows. The algorithm to be described uses local information and so does not depend on the topology of the network provided any feedback effects are negligible.

The Random Boolean Network (RBN)
A random Boolean network is a random, directed graph. Each node has a state, $x_i \in \{0,1\} = \mathbf{B}$, and a Boolean function, $f_i : \mathbf{B}^{k_i} \to \mathbf{B}$, where k_i is the number of predecessors of the ith node. The states of the nodes in a classic RBN are updated synchronously, thus the dynamics are deterministic. Such systems and their dynamics have been studied at length. They were introduced by Kauffman to model random genetic nets with gene activity as a binary on-off switch [1]. Using the

[*]n.kenny@postgrad.umist.ac.uk
[†]M.Muldoon@umist.ac.uk
[‡]keith.briggs@bt.com

dynamics he predicted the number of cell types within an organism and cell reproduction times as functions of the number of genes within the organism and cell respectively. Since then RBN's have been used to model a variety of systems from economics [2], to music [3], to brain activity [4].

RBN dynamics have also been studied in a more abstract setting by such authors as Derrida and Pomeau [5]. They randomly assigned predecessors and Boolean functions to each node, at every time step. This enabled them to make an annealed approximation confirming Kauffmans observations [1]; that the complexity of the dynamics increases dramatically when the number of predecessors per node is greater than two. Solé and Luque analyzed generalized RBNs and showed that the same phenomenon exists when the mean number of predecessors per node is greater than two [6]. Luque and Solé went on to define a discrete Lyapunov exponent using Boolean calculus [7].

There have been arguments against the assumption of synchronous updating within RBNs when modelling gene interactions [8],[9]. As a result, a number of adjustments to the classical Kauffman RBN have been developed and analyzed. Gershenson summarizes a few of these modifications and classes them in terms of updating strategies [10]. This paper does not cover all RBN types; for example, Shmulevich introduces RBNs that update synchronously, with Boolean function uncertainty [9]. A comprehensive overview of the developments of RBNs within the genetic setting is given by Shmulevich, Dougherty and Zhang [11].

The RBN studied here updates probabilistically. Each node has two update probabilities $\tau_{i,0}$ and $\tau_{i,1}$: $\tau_{i,0}$ is used when $x_i = 0$ and $\tau_{i,1}$ is used when $x_i = 1$, x_i is the state of node i. The motivation for this will be discussed in the next section.

A possible application of this work is as a model of the domain name system (DNS), particularly DNS replication. The DNS is a distributed database of mappings used to map machine names to IP addresses. The database has a tree structure, with thirteen root zones. In all there are around 20 million zones (nodes) within the graph [12].

To prevent the loss of any data and to improve performance, each DNS server is replicated at least twice [13]. The replicated servers contact each other periodically, once or twice a day, to check their files are up to date [13]. There were estimated to be over 2 million DNS servers last year [14]. The traffic generated by replication maintenance may seam insignificant when we consider that some of the one hundred physical root servers have reported 100 million queries per day [15]. This makes replication traffic less than 0.05% of DNS traffic. Wessels and Fomenkov observed the query traffic of the four machines that make the F root server [12] . Only 2.15% of all traffic was legitimate, the rest was attributed to misuse and neglect of DNS software. Extending this to DNS traffic across all root servers, replication traffic is closer to 2% of legitimate traffic.

In the context of the model, the fixed periodic refreshing of DNS servers, to maintain replication, can be compared to the synchronous updating of the generalized RBN. When the state of a node within the RBN changes, this would represent updated data within the DNS server.

2 Accuracy algorithm

The accuracy algorithm is a simple learning algorithm [16], enabling each node to determine its own re-evaluation probabilities, $\tau_i = \{\tau_{i,0}, \tau_{i,1}\}$, using only local information. A node estimates the probability distribution of its state by observation, then uses prior knowledge, in the form of equations, to make the adjustments to τ_i.

Here a comparison of two forms of the same algorithm will be made. In one case, the nodes will estimate the distribution of their state using a vector memory and update τ_i periodically (periodic updating). In the other case, each node will have a scalar memory and update τ_i after each time step. Periodic updating may be preferred to reduce work done by a node, while continuous updating may be preferred for more rapid response.

As nodes within the network are updating probabilistically there may be occasions where node i does not update at time t but some of its predecessor nodes do. Thus, x_i may become incorrect with respect to its predecessor states.

Accuracy theory

Let ϵ_i be the probability of node i being incorrect. Then $1-\epsilon_i$ is the probability that node i is correct. Assume node i updates continuously, ($\epsilon_i = 0$), generating the bit string, $X_i = (x_{i,0}, x_{i,1}, \ldots, x_{i,t}, \ldots)$. Throughout this paper it will be assumed that members of X_i are independent and identically distributed with distribution $G_i(0) = 1-p_i$, $G_i(1) = p_i$. This is a reasonable assumption even though RBNs may contain many feedback loops; as the number of nodes within the random network increase the effects of feedback become negligible.

Consider a simple $k+1$ node network where node 0 receives inputs from nodes 1 to k. Nodes 1 to k generate bit strings $X_1, \ldots, X_k$ with distributions $G_1, \ldots, G_k$. Initially node 0 updates continuously generating a bit string, X_0, with distribution G_0. If node 0 were to update probabilistically, using τ_0, the probability of its state being accurate would be:

$$[\tau_{0,0} + (1-\tau_{0,0})G_0(0)]\, G_0(0) + [\tau_{0,1} + (1-\tau_{0,1})G_0(1)]\, G_0(1) \quad \equiv \quad 1-\epsilon_0 \qquad (1)$$

Where, by definition, $1-\epsilon_0$ is the probabilistic accuracy of node 0 given τ_0. However, the aim of the algorithm is to choose the $\tau_{0,b}$ so as to achieve a prescribed ϵ_0. During a simulation, if node 0 were in state 0 then, for design purposes, equation (1) could be reduced to:

$$1-\epsilon_0 \quad = \quad \tau_{0,0} + (1-\tau_{0,0})G_0(0)$$

Thus determining $\tau_{0,0}$:

$$\tau_{0,0} \quad = \quad 1 + \frac{\epsilon_0}{G_0(0)-1} \qquad (2)$$

A similar argument gives $\tau_{0,1}$ in terms of ϵ_0 and $G_0(1)$.

Once node 0 begins to update probabilistically the distribution of its state alters. For clarity, we shall call the bit string generated by node 0 when the node updates probabilistically Y_0, write H_0 for the distribution of Y_0. The relationship between G_0 and H_0 can be found using Markov analysis as Y_0 is a Markov chain. This generates the matrix equation, equation (3), which reduces to the eigenvalue problem (4).

$$\begin{pmatrix} p(y_{t+1} \text{ is } 0) \\ p(y_{t+1} \text{ is } 1) \end{pmatrix} = \begin{pmatrix} 1-\tau_{0,0}G_0(1) & \tau_{0,1}G_0(0) \\ \tau_{0,0}G_0(1) & 1-\tau_{0,1}G_0(0) \end{pmatrix} \begin{pmatrix} p(y_t \text{ is } 0) \\ p(y_t \text{ is } 1) \end{pmatrix} \qquad (3)$$

$$= \lambda \begin{pmatrix} p(y_t \text{ is } 0) \\ p(y_t \text{ is } 1) \end{pmatrix} \qquad (4)$$

Solving equation (4) gives two eigenvalues, 1 and $2\epsilon_0$. As $\epsilon_0 < 0.5$, $(2\epsilon_0)^t \to 0$ with increasing t, so we need only consider the eigenvector equation with eigenvalue 1, written below:

$$\begin{pmatrix} p(y_{t+1} \text{ is } 0) \\ p(y_{t+1} \text{ is } 1) \end{pmatrix} = \frac{1}{1-2\epsilon_0} \begin{pmatrix} G_0(0)-\epsilon_0 \\ G_0(1)-\epsilon_0 \end{pmatrix}$$

The above equation is the relationship between $H_0(b)$ and $G_0(b)$ because $p(y_{t+1} \text{ is } 0) = H_0(0)$.

$$H_0(b) \quad = \quad \frac{G_0(b)-\epsilon_0}{1-2\epsilon_0} \qquad \text{or}$$

$$G_0(b) \quad = \quad \epsilon_0(1-2H_0(b)) + H_0(b) \qquad (5)$$

Algorithm

Accuracy estimation
The aim of the algorithm is to use equations (2) and (5) to equip each node with the knowledge to adjust its update probabilities. We can apply these equations to a node regardless of the updating strategies of its predecessors, because the equations only depend on the behavior of the node itself. The update probabilities will be adjusted such that x_i is inaccurate with probability ϵ_i. It is important here to distinguish between target inaccuracy of ϵ_i, and the actual inaccuracy of the node. Let ψ_i denote the actual inaccuracy of node i.

Now consider general nodes within a RBN. Each node is aware of its state, x_i, at all times. Hence, nodes can estimate the distribution of the bit string they generate. Nodes will know if their re-evaluation probabilities are less than one so will know if they are estimating G_i or H_i. Nodes will be estimating H_i most of the time as the aim of the model is to reduce information traffic. For simplicity of explanation we shall discuss H_i from now on.

Node i estimates $H_i(b)$. In order to adjust τ_i, equation (5) must be used to find $G_i(b)$. There is no way for a node to know the current probability of its state being incorrect, ψ_i, without re-evaluating on every time step. This means that the calculation of $G_i(b)$ is subject to error. Let $\delta_i = \psi_i - \epsilon_i$ be the difference between the true accuracy ψ_i and the target ϵ_i, Equivalently, $\epsilon_i = \psi_i + \delta_i$ and equation (5) becomes:

$$G_i(b) \quad = \quad (\psi_i + \delta_i)(1 - 2H_i(b)) + H_i(b) \tag{6}$$

Substituting into equation (2) gives:

$$\tau_{i,b} \quad = \quad 1 + \frac{\epsilon_i}{(\psi_i + \delta_i)(1 - 2H_i(b)) + H_i(b) - 1} \tag{7}$$

Applying the Taylor series expansion to equation (7) around ψ_i leads to:

$$\tau_{i,b} \quad = \quad \left(1 + \frac{\epsilon_i}{\psi_i(1 - 2H_i(b)) + H_i(b) - 1}\right) + O(\delta_i)$$

The term in brackets is the exact equation for $\tau_{i,b}$. The error will be further discussed in section 3.

Distribution estimation
One of the key features of the algorithm is the probability distribution estimation of the bit strings generated by each node. To make the estimate the node has to observe its state and store the information. Two methods were compared, the vector and scalar memory.

In the vector memory method, a memory block, size T, is filled with the bit string. On every time step the state of the node is put into the memory. Once the memory is full, the node counts the number of ones within and uses this as its probability of being in state one. It then calculates the probability of being in state zero, adjusts its update probabilities, empties its memory and starts again.

The variance of the estimate of $H_i(1)$ produced by the vector memory is σ_i^2, where:

$$\sigma_i^2 \quad = \quad \frac{H_i(1)H_i(0)}{T} \tag{8}$$

The scalar memory is the exponentially decaying average [16]

$$y_i(t) = \sum_{l=0}^{\infty} w^{l+1} b_{i,(t-l)}$$

where $y_i(t)$ is the value of the scalar memory at time t and $w \in (0,1)$ is the weight parameter. It is straightforward to show that the distribution of Y_i, the bit string output of node i, at time t, is,

$$\begin{aligned} H_i(1) &= \frac{1-w}{w} y_i(t) \\ H_i(0) &= 1 - H_i(1) \end{aligned}$$

The variance of $H_i(1)$ when estimated using the scalar memory is $\bar{\sigma}_i^2$, where:

$$\bar{\sigma}_i^2 = H_i(1)H_i(0)\frac{1-w}{1+w} \tag{9}$$

We will compare accuracy convergence using the two methods for estimating $H_i(b)$. Equating (8) and (9) gives a relationship between the size of the vector memory, T, and the weight, w, that yields a kind of "comparable" scalar memory.

$$\begin{aligned} T &= \frac{1+w}{1-w} \\ w &= \frac{T-1}{T+1} \end{aligned}$$

Using this relationship we can say that the effective scalar memory, defined by w, is comparable to a vector memory of length T.

3 Simulation

In this section the results of a simulation of a generalized RBN with nodes that run the accuracy algorithm. A generalized RBN has a variable number of predecessors per node, chosen uniformly from $\{1, 2, \ldots, k_{max}\}$. As the RBN will have probabilistic updating its dynamics should not exhibit order [10],[17], but to ensure 'interesting' dynamics the mean number of inputs per node, $< k >$, will be taken to be greater than two [6]. Initially the nodes will have re-evaluation probabilities $\tau_i = \{1, 1\}$. The results of this simulation are typical to every simulation run so far.

The network has 100 nodes, $k_{max} = 7$, $< k >= 4$, $\epsilon_i = 0.1\ \forall i$, $T = 1000$, $w = 0.998$. The simulations were left to run for 20000 time steps.

Figure 1 shows the probabilistic accuracy of every node within the network, during both simulations. The graph on the right records the scalar memory results while the other graph shows the vector memory results. The accuracies reported in these figures are absolute in the sense that they were monitored at every time step, independent of any probabilistic updating.

The graph on the left of figure 2 shows the probabilistic accuracy of node 63. We see, especially clearly with the scalar memory, that node 63 is performing very close to its target accuracy. However, node 96, to the right of figure 2, has accuracy probability just over 0.92. In fact a large number of nodes within the network achieve an accuracy higher than that required, figure 1. This can be attributed to the error described in section 2.

The error within the algorithm depends on $H_i(b)$, equation (6). The closer $H_i(b)$ is to 0.5 the smaller the error. The effect of this can be seen in figure 2, $H_{63}(1) \approx 0.54$, $H_{96}(1) \approx 0.78$. On each node, $H_i(b) > 0.5$ has a more prominent effect on the accuracy of the node as the node will mostly be in state b. For $H_i(b) > 0.5$, $(1 - 2H_i(b)) < 0$. As the initial re-evaluation probabilities are 1, $\psi_i < \epsilon_i$, thus $\delta_i > 0$. Looking again at equation (6) we see that, under these conditions $G_i(b)$ is underestimated by the algorithm, hence $\tau_{i,b}$ is over estimated and the accuracy is greater than required.

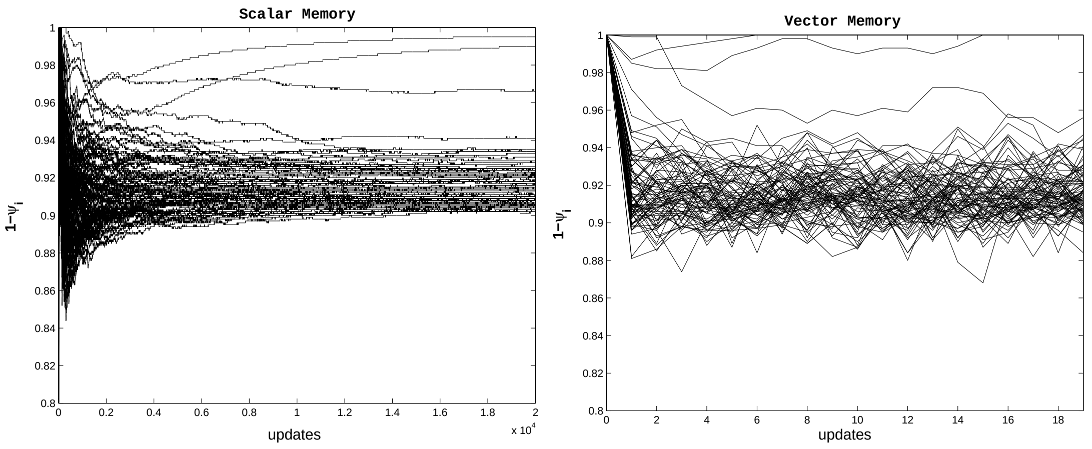


Figure 1: The probability of node i being accurate

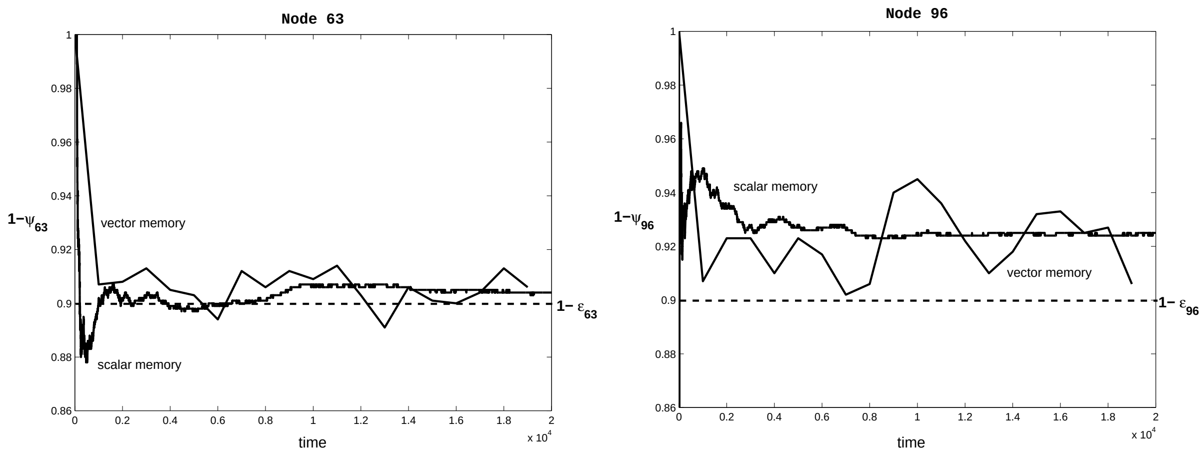


Figure 2: The probability of nodes 63 and 96 being accurate

4 Traffic reduction

Single node

When thinking of the Boolean network as an information network, we can say, a node updating generates traffic in the form of a request. An assumption of the model is that when a node requests predecessor states, it then receives replies. Let a request and reply be a unit of traffic. Hence, if node i were part of a continuously updating, generalized RBN it would generate Tk_i units of traffic over the time period T.

Assume node i has been implementing the accuracy algorithm which has converged on some τ_i. When the node is in state y_i, it generates k_i units of traffic with probability τ_{i,y_i}. This is equivalent to the node sending $k_i\tau_{i,y_i}$ units of traffic on every time step while it is in sate y_i. Let the network run for T time steps, the amount of time node i spends in state y_i is $TH_i(y_i)$. Thus, node i, on average, generates $Tk_i\left(\tau_{i,0}H_i(0) + \tau_{i,1}H_i(1)\right)$ units of traffic within period T.

Call $(\tau_{i,0}H_i(0) + \tau_{i,1}H_i(1))$ the traffic reduction factor, Ω_i. Express Ω_i in terms of G_i and ϵ_i, (equations (2) and (5)):

$$\Omega_i = H_i(0)\tau_{i,0} + H_i(1)\tau_{i,1} = \frac{G_i(b)(G_i(b)-1) - \epsilon_i(\epsilon_i - 1)}{(1-2\epsilon_i)G_i(b)(G_i(b)-1)}$$

In each simulation to date each node has converged to some $H_i \Rightarrow G_i$. For fixed $G_i(b)$, Ω_i can be estimated by the linear function, $\hat{\Omega}_i(\epsilon_i)$:

$$\hat{\Omega}_i(\epsilon_i) = \frac{1}{G_i(b)-1}\epsilon_i + 1$$

Thus, traffic reduction in roughly linear w.r.t. ϵ_i. Looking at the graph of Ω_i for fixed ϵ_i, figure 3, the curve $\Omega_i(G_i(b))$ shows the greater the distance between $G_i(b)$ and 0.5, the greater the possible reduction in traffic, as expected.

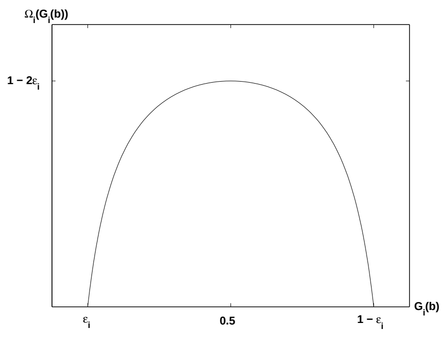
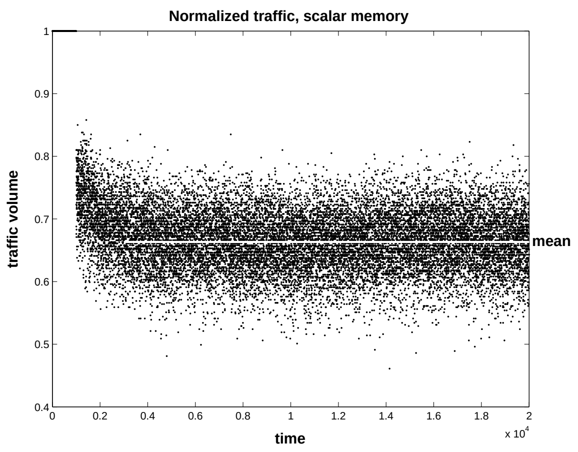


Figure 3: Traffic generated by accuracy node for all $G_i(x_i)$

Over the network
Analysis of traffic reduction over the whole network for $\epsilon_i = \epsilon \; \forall i$ is currently in progress so there are no general results to report as yet. To give an idea of what may be expected the traffic generated by the simulated network in section 3 was recorded.



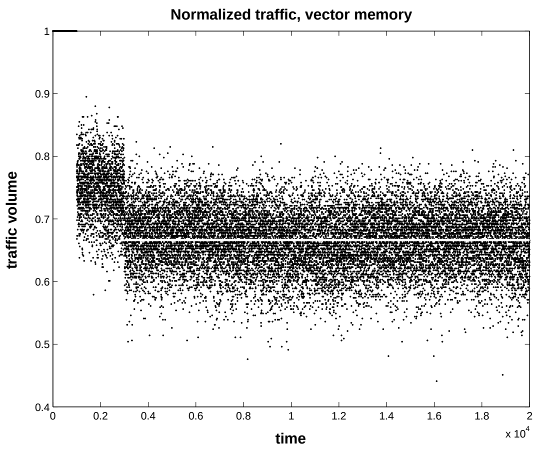


Figure 4: Normalized traffic volume, $\epsilon = 0.1$

Figure 4 shows the normalized traffic volume across the whole network on every clock tick, $\epsilon = 0.1$. Normalized traffic is the fraction of traffic remaining once accuracy algorithm is in use. The graph on the right shows the results for the vector memory simulation, the graph on the left for the scalar memory. As one would expect there is little difference between the performance of the two memory types. We see that for $\epsilon = 0.1$ the network experiences approximately 35% less traffic.

Figure 5 shows simulated results of normalized traffic plus standard deviation on the same network for varying ϵ.

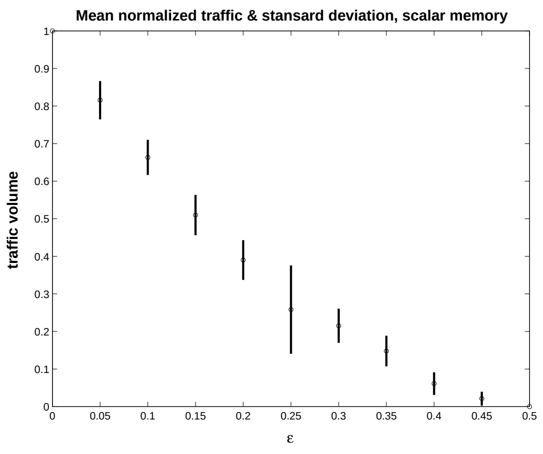


Figure 5: Mean normalized traffic with standard deviation

5 Conclusion

An algorithm has been proposed to reduce information traffic within a computer network. The method by which the algorithm achieves a decline in traffic volume is a controlled reduction in the accuracy of the information. The algorithm has been developed to fulfill its task using local information only, moving closer to creating an algorithm that may be of use in real information networks.

Regarding the DNS application, DNS servers could replace periodic refreshing with an algorithm similar to the accuracy algorithm. It is conceivable that zones are updated at different rates. Thus, a refreshing algorithm taking this into account may reduce DNS replication traffic. However, there is a slight chance that 'intelligent' refreshing could increase DNS replication traffic as currently DNS protocols do not consider staleness of address in any detail [13].

Work in progress; a general result on reduction in traffic over the whole network. Further work on this model which is not discussed in this paper include; network dynamics, different network topologies, cases where the independent and identically distributed assumption may not hold, for example, small world networks and designated nodes feeding bit strings into the network that are generated using an intermittency map [18] [19].

References

[1] S. Kauffman (1969). Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets. *J. Theoret. Biol. 22*. pp. 437-467.

[2] J. Alexander (2002). Random Boolean Networks and Evolutionary Game Theory. `http://philsci-archive.pitt.edu/archive/00001049/index.html` .

[3] A. Dorin (2000). Boolean Networks for the Generation of Rhythmic Structure, *Proceedings ACMC 2000, Australian Computer Music Conference*. pp38-45.

[4] R. Cotterill (1988). Computer Simulation in Brain Science. *Cambridge Univercity Press*. GB.

[5] B. Derrida, Y. Pomeau (1986). Random Networks of Automata: A Simple Annealed Approximation. *Europhys. Lett. 1 (2)*. pp. 45-49.

[6] R. Solé, B. Luque (1995). Phase Transitions and Antichaos in Generalised Kauffman Networks. *Physics Letters A. 196*. pp. 331-334.

[7] B. Luque, R. Solé (2000). Lyapunov Exponents in Random Boolean Networks. *Physica A: Statistical Mechanics and its Applications. Vol 284. 1-4*. pp. 33-45.

[8] I. Harvey, T. Bossomaier (1997). Time out of Joint: Attractors in Asynchronous Random Boolean Networks.*Proceedings of the Fourth European Conference on Artificial Life* MIT Press, pp. 67-75

[9] I. Shmulevich, E. Dougherty, S. Kim, W. Zhang (2002). *Bioinformatics. Vol 18. 2.* pp. 261-274.

[10] C. Gershenson (2002). Classification of Random Boolean Networks. *Artificial life VIII: Proceedings of the Eighth International Conference of Artificial Life* . MIT press, pp. 1-8.

[11] I. Shmulevich, E. Dougherty, W. Zhang (2002). From Boolean to Probabilistic Boolean Networks as Models of Genetic Regulatory Networks. *Proceedings of the IEEE. Vol 90. 11.* pp. 1778-1792.

[12] D. Wessels, M. Fomenkov (2003). Wow, That's a Lot of Packets. *Proceedings of the Passive and Active Measurement Workshop.*

[13] G. Coulouris, J. Dollimore, T. Kindberg (2001). Distributed Systems, concepts and design. *Pearson Education Ltd*. USA.

[14] P. Mockapetris (2003). DNS growth has just begun: the challenge is to find new ways to use the almost limitless resource - Guest Column - domain name system. `http://articles.findarticles.com/p/articles/mi_m0CMN/is_10_40/ai_109082402`

[15] D. Wessels, M. Fomenkov, N. Browlee, K Claffy (2004). Measurements and Laboratory Simulations of the Upper DNS Hieraachy. `www.caida.org/outreach/papers/2004/dnspam`

[16] S. Haykin (1999). Neural Networks, a comprehensive foundation. *Prentice-Hall Inc.*. USA.

[17] E. Di Paolo. Searching for Rhythms in Asynchronous Random Boolean Networks. `http://citeseer.nj.nec.com/281511.html`.

[18] D. Watts, S. Strogatz (1998). Collective dynamics of small-world networks. *Nature. Vol 363.*. pp. 202-204.

[19] R. Mondragón, J. Pitts, D. Arrowsmith (2000). Chaotic Intermitmittency-Sawtooth Map Model of Aggregate Self-Similar Traffic Streams. *Electronic Letters. Vol 36. 2.* pp. 184-186.