Unstable internet routing as oriented hypercubes

Paul Hunter

July 28, 2008

Abstract

We discuss a combinatorial approach to the internet routing problem as presented at WG 2008

1 Introduction

We work from the following model: Autonomous routers are represented as vertices of an undirected graph, with an edge between two routers if there is a communcation pathway between them. Each router has an ordered list of (simple) paths[1] from itself to a single destination (0), specifying a preference between different routing policies, for example if router 1 contained the list: $(10, 120, 130)$ then this would indicate that it would prefer to route packets to the destination (0) via router 2 rather than through router 3, but it would prefer routing directly to 0 over either. Paths that do not appear in the preference list are *invalid* and are considered to be less favourable than any path in the list (and equally favourable with other invalid paths). We assume that a router does not know (or rather utilise) the policy preferences of any other router.

To model asynchronous and synchronous communication (and to some extent commication delay), we *activate* sets of edges at time intervals, allowing routers to change their routing policy and/or disseminate their policy using the newly available paths (based on information known prior to the activation). Asynchronous communication is essentially activating one edge at a time, and synchronicity is modelled by the simultaneous activation of many edges. We assume that only an edge not being "used" (that is, neither of its endpoints is [attempting to] route traffic through it) may be activated.

Taking a snapshot at each time interval gives us the notion of a *state* of the system. Formally a state is a map σ which takes every vertex to one of its neighbours ($\sigma(v)$ is the neighbour of v to which v is routing its traffic). To model delays in communication, we assume a vertex may only make policy decisions (for the next state) based on the current state (and of course the set of activated edges and their list of preferences). Thus a policy change may not be for the better if routers further along the path also make a change. We assume that the current state of the system is known to everyone.

[1] that is, no router is visited more than once

To summarise the situation, we assume we are in some state σ. We select a set E of edges and activate them. Each router then chooses a routing policy based on their preference list, σ and E, resulting in a new state σ', and we repeat. A system is *stable* if for all choices of E, $\sigma = \sigma'$ (that is, no router changes its policy).

Given a system, some of the questions we might be interested in are as follows:

Q1 Does there exist a stable state?

Q2 Does there exist a unique stable state?

Q3 Is there an algorithm describing a (polynomial) sequence of edge activations which results in a stable state?

2 An alternative model

In order to simplify the combinatorial interpretation of this situation, we first provide an alternative for the above model. In this model, the network is described by a *directed* graph, with edges indicating the direction along which packets travel. We still activate sets of edges and a state of this system is a map which takes vertices to some vertex in their out-neighbourhood. This is clearly more general than the above model (we replace edges by a pair of directed edges in each direction), but we can also simulate the directed variant in the undirected case by doubling every edge and then subdividing (that is, replacing the edge AB with the four edges AC, CB, AD, DB where C and D are new vertices); replacing occurrences of AB in all routers' policy lists with ACB, and occurrences of BA with BDA; and adding the appropriate sublist of A's policies to C, and the sublist of B's policies to D.

We may also assume that each vertex has out-degree at most 2: if v has $m > 2$ out-neighbours, we can insert a binary tree rooted at v with m leaves (the out-neighbours of v), and make similar policy adjustments to those above. Labelling the successors of a vertex v with out-degree 2 as v_0 and v_1, we see that a state of this system can be simply represented as a vector $s \in \{0,1\}^n$ (where n is the number of vertices with out-degree 2).

Edge activation in this model is the same as above, however we observe that activating the edge uv allows (only) u to change policy (if desired). Since a policy switch is now a binary decision (either we switch or we stay the same[2]) we can actually view a set of edges to be activated as a set of vertices permitted to change policies.

Since, when a vertex has the opportunity to switch policy, it may choose either of its successors, the choice is only dependent on the policies of all the other vertices. That is, the preference listing of a vertex v (with out-degree 2) can be modelled by a (partial) function $p_v : \{0,1\}^{n-1} \to \{0,1\}$ indicating that

[2]recall we assume that edges being "used" can be considered to be part of the activation set

Figure 1: Oriented 2-cubes

if the policy vector of the other vertices (that is, the state vector excluding the component corresponding to v) is $\mathbf{a}$, then v would prefer to be routing through v_i where $i = p_v(\mathbf{a})$. In order to make this function total, we need to break ties (i.e. when v has no preference between either of its successors). So we define $p_v(\mathbf{a}) = 0$ for these cases.

3 A combinatorial interpretation

Recall that an *n-dimensional hypercube* is a graph H_n with vertex set $V(H_n) = \{0,1\}^n$ and an edge between $\mathbf{a} \in V(H_n)$ and $\mathbf{b} \in V(H_n)$ if and only if $\mathbf{a}$ and $\mathbf{b}$ differ in exactly one component. For $n' \leq n$, an *n'-dimensional subcube (of H_n)* is a subgraph which is isomorphic to an n'-dimensional hypercube. For a set $I \subseteq \{1,\ldots,n\}$ and a vertex $\mathbf{a} = (a_1,\ldots,a_n) \in V(H_n)$, the *subcube containing $\mathbf{a}$ generated by I* is the subcube induced by the vertex set $\{(v_1,\ldots,v_n) : v_i \in \{0,1\}$ and if $i \notin I$ then $v_i = a_i\}$. For $\mathbf{a} = (a_1,\ldots,a_n) \in V(H_n)$ the vertex *antipodal* to $\mathbf{a}$ (in H_n) is the vertex $\mathbf{b} = (b_1,\ldots,b_n) \in V(H_n)$ such that $a_i \neq b_i$ for all i.

An *orientation* of a hypercube H_n is a directed graph with H_n as the underlying undirected graph. See Figure 1 for the four examples describing the complete set (up to isomorphism) of oriented 2-cubes.

Given a routing system (in which every vertex has out-degree at most 2), we associate with it an oriented hypercube H_n in the following way. The dimension, n is the number of vertices with out-degree exactly 2. $V(H_n)$ is the set of states of the system. Let $\mathbf{a} = (a_1,\ldots,a_n) \in V(H_n)$ and $\mathbf{b} = (b_1,\ldots,b_n) \in V(H_n)$ be n-dimensional vectors differing only in the i-th component (that is $a_j = b_j$ for all $j \neq i$ and $a_i \neq b_i$), so $\mathbf{a}$ and $\mathbf{b}$ are adjacent vertices on H_n. Let $\mathbf{c} = (c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_n)$ be the $(n-1)$-dimensional vector common to both $\mathbf{a}$ and $\mathbf{b}$ and assume without loss of generality that $b_i = 1$. Then we orient the edge $\mathbf{ab}$ from $\mathbf{a}$ to $\mathbf{b}$ if $p_{\mathbf{a}}(\mathbf{c}) = 1$, otherwise we orient it from $\mathbf{b}$ to $\mathbf{a}$ (observe that by definition, $p_{\mathbf{a}}(\mathbf{c}) = p_{\mathbf{b}}(\mathbf{c})$). See Figures 2 and 3 for examples of routing systems (in the directed setting) and their associated hypercubes. In these examples, the i-th component corresponds to router i, and for each router v, v_0 is router 0 (i.e. a 0 in the i-th co-ordinate corresponds to the case when router i is routing directly to 0.)

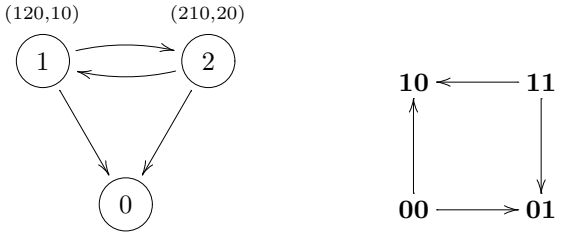

Figure 2: The **bad** gadget and its hypercube

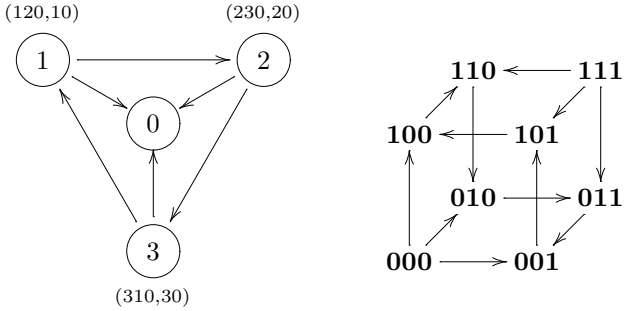

Figure 3: The **awful** (not its real name!) gadget and its hypercube

3.1 Interpreting state transition

So now that we have an oriented hypercube associated with our system, how do we interpret the evolution of the system? Clearly, since each vertex of the hypercube corresponds to a state in the system, we can think of a pebble being placed on a vertex (indicating the current state of the system), and as the system moves from state to state the pebble moves to different vertices in the hypercube. So the question is "can we determine which vertex the pebble will move to next?"

We observe that each vertex in our system now corresponds to a dimension in our hypercube model, so an activation set (of vertices) is therefore a set of dimensions. What happens when we "activate a set of dimensions"? Based on our earlier assumption that the decision of each (activated) router to switch is based solely on the current state of every other router, we observe that router i will switch if and only if there is an outgoing edge (in dimension i) from the current state. Taking into account simultaneous switching of all activated routers, the activation-switch process can be described as follows:

1. Given the system is in state $\mathbf{a} \in V(H_n)$.
2. Let $E \subseteq \{1, \ldots n\}$ be a set of dimensions.
3. Let $E' \subseteq E$ be the set of dimensions in E that correspond to the outgoing edges from $\mathbf{a}$.

4. The next state for the system **b** is the vertex antipodal to **a** on the subcube of H_n containing **a** generated by E'.

The computation of E' in the above process requires only an examination of the neighbourhood of **a**. We call such a query a *local query (in dimensions from E)*. We call the state transition in step 4 a *jump (in the hypercube)*.

3.2 Stability

Given the above interpretation of state transition, we observe the following:

Observation. *A state is stable if and only if it corresponds to a sink in H_n.*

This gives us the following combinatorial interpretations of the questions raised in the original setting.

Q1′ Does H_n contain a sink?

Q2′ Does H_n contain a unique sink?

Q3′ Is there an algorithm which finds a sink in H_n using (a polynomial number of) local queries?

4 The structure of H_n

The theory on oriented hypercubes is extensive, so a natural question to ask is what, if anything, can be said about the structure of an oriented hypercube arising from these routing systems (and some restrictions we consider later in this section). For example, can every oriented hypercube be realized as the hypercube associated with a routing system? The answer to this is no – one can readily verify with brute force that the Loop (see Figure 1) cannot be the oriented hypercube of a routing system. In fact, we have a considerably stronger result:

Theorem 1. *The oriented hypercube associated with a routing system is Loop-free*[3].

Proof. Suppose we have a routing system that is associated with a hypercube which contains a Loop. We can fix the policies of all but 2 of the routers so that state transitions remain within the Loop. Observe that this effectively reduces our routing system to a system with two routers (plus possibly some additional routers which have no choice for their routing policy which may be useful for distinguishing paths) and the associated hypercube to this system is the subcube that we have restricted to. Call these routers 1 and 2 and assume their successors are 1_0, 1_1, 2_0 and 2_1 (labelled in the obvious way). We assume each router is attempting to find a path to 0, and we can assume without loss

[3]That is, it does not contain the Loop as an induced subgraph; or equivalently, every 2-dimensional subcube is either an Eye, a Bow, or a Saddle.

of generality that each of these seven routers are distinct. Since routers 1 and 2 are the only routers with a choice of successors, every path is determined until it reaches 0, 1 or 2.

We make the following observations:

Obs1 Since, for some state, router 1 switches from 1_0 to 1_1, it follows that there must be a (valid) route from 1_1 to 0 (possibly via 2) – router 1 will always choose 1_0 if there is no (valid) path to 0 from either successor. Symmetrically there is a (valid) route from 2_1 to 0 (possibly via 1).

Obs2 If neither of the paths from 1_0 and 1_1 reach 2 then the choice for 1 is independent of the state of 2, so 1 will switch from 1_0 to 1_1 (or from 1_1 to 1_0) for both choices of 2, contradicting the assumption that the associated hypercube is the Loop. Therefore at least one of 1_0 or 1_1 has a path to 2. Symmetrically, at least one of 2_0 or 2_1 has a path to 1.

Obs3 Now if both 1_0 and 1_1 have a path to 2, then from Obs1 the path from 2_1 to 0 is not via 1. It follows from Obs2 that there is a path from 2_0 to 1. But then the choice, for 2, of 2_0 always results in an invalid route (recall we do not allow looping) whereas the choice of 2_1 is a valid route. So router 2 will always switch to 2_1 regardless of the policy of router 1, contradicting the assumption that the associated hypercube is the Loop. Symmetrically, if both 2_0 and 2_1 have a path to 1 we arise at a contradiction. Thus exactly one of 1_0 and 1_1 has a path to 2 and exactly one of 2_0 and 2_1 has a path to 1.

We claim that when router 1 switches to the one successor which routes through 2 then 2 must be routing directly (i.e. not via 1) to 0. If 1_1 is the successor which routes through 2, then from Obs1 and Obs3 it follows that one of 2_0 or 2_1 has a path to 1 and the other has a path to 0. As 1_1 is not the "default" choice for 1, router 2 must be routing directly to 0 when 1 switches. If 1_0 is the successor which routes through 2, then from Obs1 there is a valid (from router 1) path from 1_1 to 0. Since the path from 1_1 is valid (from 1), the route through 1_0 must also be valid. Thus 2 must be routing directly to 0 when 1 switches.

By symmetry, when router 2 switches to the one successor which routes through 1, then 1 must be routing directly to 0. But then, as we have just argued, it is impossible for router 1 to switch (as required by the associated Loop hypercube). This gives us a contradiction, and therefore there can be no routing system which has the Loop as a subcube of its associated hypercube. □

It is an interesting question as to whether the converse to this holds:

Open problem 1. *Is every Loop-free hypercube realizable as a hypercube associated with a routing system?*

I personally expect the answer to this to be negative as there ought to be higher-dimensional analogues of the Loop which contradict the cycle-free nature of routes. However, the example in Figure 3 provides a useful counter-example for some of the more obvious generalizations.

4.1 Restricted classes

We now consider the structure of hypercubes corresponding to some of the resticted classes of routing systems discussed at WG (or what I can remember of them...)

Safety

Our first restriction is to systems that always enter a stable state on various sets of infinite sequences of edge activations. For convenience, we consider sequences of edge (equivalently router) activations which change the state of a system. So for example, if a system always reaches a stable state after any (finite) choice of edge activations, then every sequence of activations is finite. We now show that the hypercubes associated with this class of systems characterized by this property enjoy a nice combinatorial structure, which allows us to show that such systems have a unique stable state.

An *acyclic unique sink ordering (AUSO)* of an n-dimensional hypercube H is an orientation of H such that every subcube of H contains a unique sink. AUSOs (also called *completely unimodal pseudo-boolean functions*) are a well-studied class of structures. They enjoy many nice properties, for example from every vertex there is a path of length at most n to the global sink; and each vertex has a unique set of improving directions. Another property is that it is relatively easy to decide if an orientation is an AUSO thanks to the following theorem of Williamson Hoke:

Theorem 2 (Williamson Hoke 88)**.** *An oriented hypercube is an AUSO if and only if it is acyclic and {Loop,Saddle}-free.*

Note that the latter statement in the above equivalence could be rephrased as "acyclic and every 2-subcube is an AUSO" because the Eye and the Bow are the two 2-dimensional AUSOs.

Using the above result, we can show that if a system always eventually reaches a stable state, then it has a unique stable state. Let us say a system is *very safe* if it has no infinite sequence of activations.

Theorem 3. *The hypercube associated with a very safe system is an AUSO.*

Proof. Let H be the hypercube associated with a very safe system. We show that it is acyclic and Saddle-free. From Theorem 1 it follows that it is also Loop-free, which, with Theorem 2 gives us our result. Suppose H has a cycle $C = v_1 v_2 \cdots v_n$ (where $v_n = v_1$). By the definition of a hypercube for each i, v_i differs from v_{i+1} in exactly one component, say a_i. Assume the system is in the state corresponding to v_1 and consider the activation sequence $S = (\{a_1\} \cdots \{a_n\})^\omega$. From the definition of state transition the state of the system under this sequence moves around C. As C has more than one vertex, this gives us an infinite activation sequence contradicting the very safe-ness of the system. Thus H is acyclic. Now suppose H contains a Saddle. By the definition of subcube, all vertices of the Saddle agree on all but two components. Let b_1 and

b_2 be those components. Assume the system is in a state which corresponds to one of the local sinks of the Saddle and consider the set of activations $\{b_1, b_2\}$. From the definition of state transition, the state will jump to the other local sink of the Saddle. Thus the sequence $(\{b_1, b_2\})^\omega$ is an infinite activation sequence for the system, again a contradiction. Thus H does not contain a Saddle. □

Corollary. *A very safe system has a unique stable state.*

This association of very safe systems with AUSOs raises some interesting questions. The first is whether the converse to Theorem 3 holds.

Open problem 2. *Is every AUSO realizable as a hypercube associated with a very safe routing system?*

The second question stems from a much researched problem associated with AUSOs: "Is there a deterministic algorithm which finds the global sink of an AUSO in a polynomial number of queries?". So far the best known algorithm (Fibonacci See-Saw) uses $O(1.61^n)$ queries, and it is known that the greedy jumping procedure (i.e. jump in all improving directions) may visit $2^{n/2}$ vertices. If the above conjecture is true, then this question is equivalent to Q3 applied to very safe systems. But it may be that very safe systems result in a subclass of AUSOs that can tractably find the sink with a polynomial local search.

Open problem 3. *Is there an algorithm describing a polynomial sequence of edge activations which results in the stable state of a very safe system?*

We now turn to a more general class of systems that only have infinite activation sequences of a certain kind. We say an activation sequence is *fair* if every edge is activated at least once. We say a system is *safe* if every fair sequence is finite. These systems were introduced at WG, and the question was asked whether these systems always have a unique stable state. Although it is straightforward to translate the definitions of fairness and safety to hypercubes, the structure of the hypercubes associated with safe systems is not immediately obvious. However as Figure 4 shows, the class of hypergraphs associated with safe systems includes hypercubes which are not AUSOs. In the section after next we make a conjecture about the structure of "safe" hypercubes.

Greedy algorithm

We now consider classes of systems for which stable states can be found via an algorithm. I cannot recall the greedy algorithm described at WG, however I will consider the structure of the hypercube associated with systems for which stable states can be found with an alternative algorithm. Let us say that a router has a *locally dominant policy* if there is a unique neighbour to which all traffic is preferred to be routed regardless of the state of the system. For example, in Figure 4, router 3 has a locally dominant policy of routing traffic to router 4. The algorithm works as follows:

1) Find a router r with a locally dominant policy.

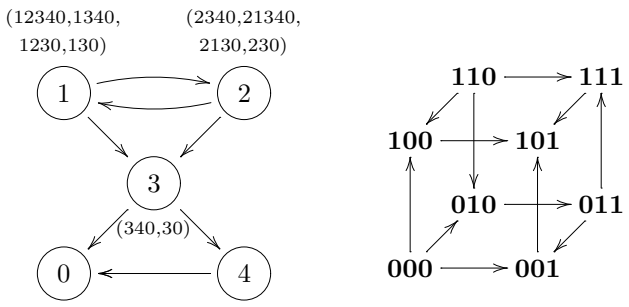

Figure 4: The **not very safe** gadget and its hypercube

2) Remove all arcs from r which do not go to the locally dominant neighbour (thereby reducing the state space).

3) Return to 1) until all vertices have been processed.

We observe that if this algorithm terminates then no router prefers to route via any other path, so the system is stable. Thus it makes sense to classify routing systems based on whether or not this algorithm terminates, and, in line with the terminology of WG, we call such systems *greedy*. We now examine the structure of hypercubes associated with greedy systems.

We say that a digraph G is a *directed union* of (disjoint) digraphs G_1 and G_2 if $V(G) = V(G_1) \cup V(G_2)$ and $E(G) = E(G_1) \cup E(G_2) \cup E$ where $E \subseteq V(G_1) \times V(G_2)$. That is, G is a disjoint union of G_1 and G_2 with possibly some edges from G_1 to G_2. Let us say an oriented hypercube is *separable*[4] if it can be recursively constructed as follows:

1. A 1-dimensional oriented hypercube (i.e. a directed edge) is separable; and

2. An n-dimensional oriented hypercube is separable if it is the directed union of a pair of $(n-1)$-dimensional separable hypercubes.

For the hypercubes of greedy systems, we need a slight weakening of the separable condition. Let us say an oriented hypercube is *semi-separable*[4] if it can be recursively constructed as follows:

1. A 1-dimensional oriented hypercube is semi-separable; and

2. An n-dimensional oriented hypercube is semi-separable if it is the directed union of H_1 and H_2 where H_2 is a $(n-1)$-dimensional semi-separable hypercube.

It should be clear that a semi-separable hypercube has a unique sink (though not necessarily a unique source). Furthermore, the following result follows directly from the definition of the associated hypercube:

[4]this is my own terminology, there may be more usual names for these cubes

Theorem 4. *The oriented hypercube associated with a greedy system is semi-separable.*

The converse to Theorem 4 does not hold as a semi-separable hypercube may contain a Loop. However, it does seem plausible if we restrict to Loop-free (or, if the answer to Open problem 1 is negative, hypercubes realizable from routing systems) semi-separable hypercubes. We leave it as an open problem.

Open problem 4. *Is every Loop-free (or realizable) semi-separable hypercube realizable as the hypercube of a greedy system?*

We observe in passing that for a semi-separable hypercube to be Loop-free (or realizable) it suffices for H_1 in the above definition to be Loop-free (respectively realizable).

Generalizing AUSOs and semi-separable hypercubes

It should be clear that a separable hypercube is an AUSO, as every 2-dimensional subcube must contain a pair of parallel edges. However, as Figure 4 shows, there are semi-separable hypercubes which are not AUSOs. Also it is relatively easy to construct an AUSO which is not semi-separable. We now introduce a generalization of both these classes. Let us say a hypercube is a *semi-seperable unique sink orientation (SSUSO)*[5] if it can be recursively constructed as follows:

1. An AUSO is an SSUSO; and
2. An n-dimensional oriented hypercube is an SSUSO if it is the directed union of H_1 and H_2 where H_2 is a $(n-1)$-dimensional SSUSO.

It is trivial to show that SSUSOs have a unique sink. It should also be clear that SSUSOs satisfy the natural interpretation of the safety constraint discussed earlier. More precisely,

Theorem 5. *If the hypercube associated with a routing system is an SSUSO, then the system is safe.*

There are two ways we could extend this result First we have the question of whether every safe routing system has an SSUSO hypercube.

Open problem 5. *Is the hypercube associated with a safe routing system always an SSUSO?*

The second extension to Theorem 5 is whether every SSUSO is realizable as the hypercube associated with a safe system. Clearly this does not immediately hold as SSUSOs may contain Loops. However, restricting to the Loop-free (or realizable) SSUSOs may be sufficient.

Open problem 6. *Is every Loop-free (or realizable) SSUSO realizable as the hypercube of a greedy system?*

[5] again, this is my own terminology

From our above observations regarding SSUSOs, a positive answer to this last open problem would resolve the question of whether every safe routing system has a unique stable state.