Beacon: A Hierarchical Network Topology Monitoring System Based on IP Multicast

Marcos Novaes

IBM T. J. Watson Research Center
30 Saw Mill River Rd., Hawthorne, NY USA
mnovaes@us.ibm.com

Abstract. This paper presents *Beacon*, a technique that exploits the advantages of IP multicast to provide a low overhead system for network topology monitoring. *Beacon* is a self-organizing system which builds a self similar network topology that enables the system to converge very quickly even at a high degree of scalability. Another property of the *Beacon* system is that it does not require any interdomain routing protocol for IP multicast, making its deployment possible even in subnetworks which are interconnected by domains which do not support IP multicast routing or that deploy different routing protocols. One desirable side effect of the deployment of beacon is that it acts as a self configuring IP multicast tunneling facility which provides a distributed system with fault tolerant IP multicast reacheability. A comparison of Beacon with several other routing and gateway protocols is discussed.

Keywords. Network Topology, Network Monitoring, IP Multicast, Fault Tolerant Multicast Routing

1 Introduction

The main subject of this paper is a hierarchical protocol, named *Beacon*, which was designed to provide a Topology Services facility for distributed systems. Some of these terms have been used extensively in the literature, most of the time with slightly different meaning, so it would be wise to briefly state their significance in the context of this paper:

Distributed System: A system that is comprised of a collection of nodes interconnected by a communications network; each node consisting of an instance of an operating system which is reachable via one or more network addresses. One of the most important aspects of a distributed system is the *definition of its scope*, that is, the way in which the collection of nodes is defined. Nodes may be implicitly defined by their physical placement in a particular network, such being the case with nodes which own a port to a bridge or switch. In distributed systems built for fault tolerance or for the consolidation of management tasks, the configuration of the system is done arbitrarily by the administrators of the system. In this paper, we assume an arbitrary configuration of the distributed system, but also take into account the fact that some nodes appear to the Internet Group Management Protocol (IGMP) [1] as network neighbors.

Topology Services: A facility which provides each node with the knowledge of its ability to communicate with the other nodes in the distributed system. In the context of this paper we assume that the communication property is transitive (if A can communicate with B, and B can communicate with C, then A can communicate with C), and reflexive (if A can communicate with B then B can communicate with A). These assumptions are generally true in the realm of layer 3 protocols, which are the subject of this paper. The literature in the field is divided between probabilistic fault tolerance systems and deterministic ones. *Beacon* is a deterministic topology system, where the failure of a path is actually sensed by monitoring nodes. Deterministic topology services systems rely on verification messages (which have in the literature been called *probes*, *keep alive*, *hello* or *heartbeat* messages) sent to monitoring nodes at regular intervals, and due to this fact they are sometimes referred to as *heartbeating systems*. The verification messages used in *Beacon* are (predictably) called *beacon* messages, because they are multicast messages which function as the beacon lights of a transmission tower. The transmission tower does not know who is observing the beacon lights, they are just on constantly, and are therefore always available to interested observers (low flying aircraft). This is the communication model used throughout the *Beacon* system. This communication model can be termed *observational,* because it does not employ rigorous message sequencing.

The goal of Topology Services systems is to support a high number of nodes with a minimum of overhead. Another goal is to optimize the message flow such that failures can be perceived and communicated to potentially all members of the distributed system with maximum efficiency, i.e., in a minimum amount of time. Fault tolerant distributed systems usually have strict requirements for failure notifications, reaching intervals sometimes measured in milliseconds in such applications as fault tolerant multimedia streaming. The *Beacon* system which is discussed in this paper accomplishes these goals leveraging the filtering facility present in most communication adapters which support standard IP multicast, as described in RFC 1112 [1].

2 Advantages of Utilizing IP Multicast

This section reiterates the characteristics IP multicast [1] which make this technology specially suitable in the context of distributed systems management. There are two basic aspects of IP multicast which are of special interest:

2.1 One to Many Message Propagation

As the term implies, a single IP multicast datagram can be received by a plurality of clients. The use of this facility can greatly reduce the amount of network traffic when there is a need to send a single datagram to a large number of clients. In other words, the employment of IP multicast leads to a reduction in the utilization of network bandwidth.

2.2 Protocol Support at the Hardware Level

The second aspect of IP multicast which makes it very attractive for distributed system management applications is the widespread hardware support for the protocol. The communication adapters which support the IGMP standard described in RFC 1112 [1] effectively filter datagrams in each network interface according to the status of IP multicast group subscriptions in the host; prior to the actual receiving of these datagrams by the IP layer. This means that a multicast datagram is only received by any particular host if there is at least one process in that host which has joined that specific multicast group to which the datagram was destined on that particular interface. If there are no such subscribers, than the datagram is not received by the IP layer in the host, which means that IP multicast datagrams do not impact at all the CPU resources for a host that has no processes interested in it.

This hardware support is present in the vast majority of communication adapters in use today, and is the most attractive feature of IP multicast, and is precisely the feature that distinguishes it from similar protocols. As an example, we note that IP broadcast does have the one to many distribution property mentioned in the section above, but it has no support for isolating the set of interested receivers of a datagram. In other words, all hosts which are reachable by an IP broadcast datagram will receive it, even if there are no processes on that host interested in it.

3 Key Elements of Topology Systems

3.1 Monitoring Nodes

An important aspect of any Topology Services system is the way in which the monitoring topology is laid out. The simplest choice is to chose one distinguished node in the topology and assign to it the responsibility of monitoring all the other nodes, resulting in a monitoring topology in the shape of a star. This topology obviously cannot scale very well, as the monitoring node soon becomes overwhelmed with the overhead of receiving all the liveness messages from all other members. Another negative aspect of star monitoring topologies is that the whole system becomes unstable after the loss of the center monitoring node, and a new election has to take place and a new center node selected.

The opposite of the star topology is probably the ring topology, in which nodes are disposed linearly in an ordered stream, each node having an upstream neighbor and a downstream neighbor. Each node is then assigned the responsibility of monitoring either its upstream neighbor or its downstream neighbor. This topology is very desirable because it distributes evenly the task of receiving liveness messages from all the nodes. Nevertheless, ring topologies incur in added recovery complexity, because it is necessary to re-discover neighbors in the case of the failure of a node, and most importantly in the case that two rings have to be merged.

3.2 Leader Nodes

Another role that is commonly assigned to nodes in a Topology Services system is that of group leadership. The group leader is a distinguished node that is responsible for gathering the monitoring information supplied by the monitoring nodes, for combining the partial topology knowledge of each node and for producing a global topology view which is then sent back to all nodes. In a star network monitoring topology, the group leader is usually the center node, since it has direct access to all the topology information anyway. In ring monitoring topologies, electing a leader becomes a real problem. Since each node communicates only with its neighbors, it is necessary to devise an election scheme that traverses the ring, and that proves to be very complex in the case of failures. Therefore, most ring monitoring topologies fall back to a star topology for the purpose of leader communications. All nodes send the topology changes that they perceive (basically the death of a neighbor) to the group leader, and the leader then sends the updates back to all the other nodes. This communication pattern is not so bad, since failures are not expected to happen very often, although the leader node is likely to be overwhelmed during the bring up or shutdown procedure of a large distributed system. But the basic draw back of relying on a star topology for leadership is the cost of leader election and recovery in the case of a leader failure. The more nodes that are involved in the election process, the more complex the election will be (usually requiring the broadcasting of votes to all members). This issue is specially important when we consider the possibility of group leader failure during normal operation of the distributed Topology Services system. Not only does a new leader need to be elected, in addition the topology state must be rebuilt from scratch, usually requiring status of all monitoring nodes to be sent to the new leader, and this can result into a serious overload for the new group leader node.

3.3 Monitoring and Leader Nodes in *Beacon*

In *Beacon*, the monitoring and group leader topologies are the same, as in star topologies. That means that the monitoring nodes are also group leader nodes. That brings the advantage of having the topology information readily available in the leader nodes. The scalability problem of the star monitoring topology is dealt with by employing hierarchical star topologies, and assigning one leader for each star. The group leaders at one level of the hierarchy (also called a *tier*) will then form a star of their own and also elect a leader. Eventually the system stabilizes, having a single leader at the higher tier. This approach has the advantages associated with the simplicity of the star monitoring topology, while maintaining the capability of being able to scale to a large number of nodes.

4 The *Beacon* Protocol

Now that we have defined the scope of the problem that *Beacon* proposes to solve, and hinted at the basic improvements which can be derived from the deployment of IP multicast, and stated which key elements of the Topology System can benefit from it, let's finally explain how *Beacon* does it. In the discussion that follows, it is assumed that there is a *Beacon* process running on each node in the distributed system, which periodically sends a *beacon* on a well known IP multicast address. These addresses are also called *groups* in the context of IGMP[1], but this term makes it difficult to explain the role of nodes in *Beacon*, which forms groups of a different scope. Therefore, the word *channel* is used instead in this paper in reference to multicast addresses.

4.1 Divide and Conquer

The crudest implementation of a multicast based topology services would employ a single group leader for gathering liveness information for all other nodes. This crude implementation would no be very scalable, since the concentration of membership messages to a single node would generate an excessive load in that node. The solution is to partition the nodes into groups, which we label T_0 (tier zero) and which have distinguished leaders. Then we make it a task of the group leaders to also form a group (T_1), and elect a "leader of leaders". Once elected the leaders can forward their partial knowledge of the network topology to their group leader, which will in turn be able to combine the partial topology information collected from the other members of its group into a global view of the network topology. This global view can now reflected back to all the other members by the reverse path: the T_1 leader will send it to all the members of the T_1 group, and each member of T_1, being a leader for the members at the T_0 level, will send it to the members of its T_0 group. We now have a two tiered hierarchy.

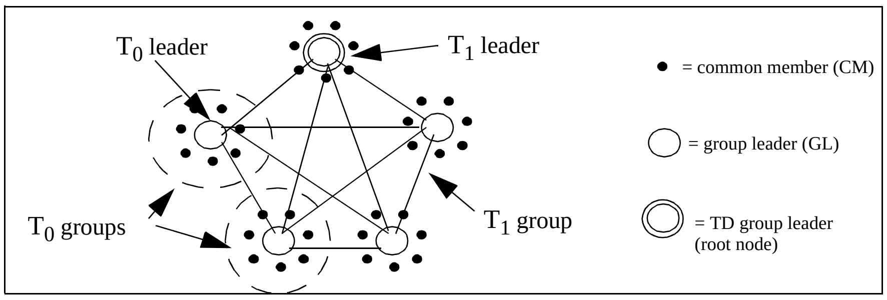

Figure 1: A Two Tiered Topology

While the two tier hierarchy is an improvement, in very large systems there will eventually be too many members at either the T_0 or T_1 levels, so it becomes necessary to employ a multi-tiered architecture. In *Beacon* this is done by limiting the number of nodes (*degree)* that a group in any given tier may have. The maximum number of members per group is presently read from a configuration parameter, but an interesting extension of this work would be to determine this limit adaptively from the state of the system. The lower the group degree, the more tiers the resulting group hierarchy will have. The illustration below shows a fully populated beaconing network of degree 5.

The numbering of tiers in *Beacon* is done as follows:

a. The number 0 is distinguished, and derived from an implicit rule: network neighborhood. An instance of a T_0 group is a group of nodes which are network neighbors.

b. The root node *R* belongs to T_D, where *D* is the *depth* of the spanning tree rooted at *R*. The root node is elected using the election protocols detailed in the sections that follow.

c. Each intermediary tier is numbered by subtracting from *D* the number of edges required to reach the tier departing from *R*.

We note that in the figure below that the root (indicated by a double circle) is a leader of a Tier 2 group, since the depth of the tree is two. It is also a leader of a tier 1 group, which is represented by the smaller star attached to it. And it is also a leader of a T_0 group, as indicated by the cloud of common members which orbit it. The root node is the only node that may be a group leader at three tier levels. The remaining nodes can be group leaders at most at two tier levels.

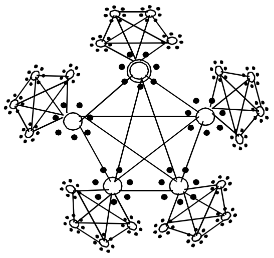

Figure 2: A Multi-tiered Topology

4.2 Tier Zero Groups

Now that we have a tool for partitioning the Topology problem, we wish to deploy it in such a way such that it exploits IP multicast such that the system incurs in minimal overhead. Therefore, we will partition the groups in a way that makes sense for IP multicast.

The group forming process outlined above can be seen as a bottom up procedure that forms a tree. We now need to define how each node is placed as a member of a T_0 group. In *Beacon*, Tier-0 is chosen to be formed of the nodes that are network neighbors in terms of IP multicast. That is, two nodes belong to the same T_0 group if and only if they can exchange IP multicast messages with the Time to Live (TTL) parameter set to 1, i.e., without having the messages traverse any router. This means that the set of possible nodes in each T_0 group is implicitly determined by the way in which the network is configured.

4.3 The *Beacon* Double-Channelled Communication Model

Now that we have established an implicit rule that divides the nodes in T_0 groups, we can now study how to gather topology information at the T_0 level and elect a leader that will forward this information up in the tree. Since we have now reduced the number of nodes that participate in the protocol to a relatively small number, we can now use the simple procedure of having each node send beacon messages to a well known multicast address (channel), and have all nodes monitor each other. But this is clearly a waste of CPU utilization, since it is only necessary for one distinguished node to act as a monitoring agent and report the topology information to the next tier. The other nodes would be wasting CPU cycles receiving beacon messages, since they are not reporting it to the other tiers.

A better approach is to first have the group election, select a group leader and then have the leader be the only node in each T_0 group which actually joins the multicast channel that receives beacon messages. Since this channel is used by the other members to send beacons to the group leader, we label it the Group Leader (GL) channel. By allowing the other members to filter out the beacon messages, we allowed them to save CPU cycles. Nevertheless it is now necessary for all the other nodes in the T_0 group to monitor the health of the group leader, in order to make sure that they have a representative at the higher tiers. Therefore, all nodes in a T_0 group which are not the group leader join a second multicast channel, which we label the Common Member (CM) channel. This channel is used by the group leader to send its own beacon (GL beacon) to the other members and also any updates to the network topology. We are now exploiting the many to one distribution of IP multicast, and thus saving the network bandwidth. Therefore, by using two multicast channels we have reduced the CPU utilization and the network bandwidth used in the protocol. We still have kept the protocol simple enough so that it can be written with a few lines of multithreaded code.

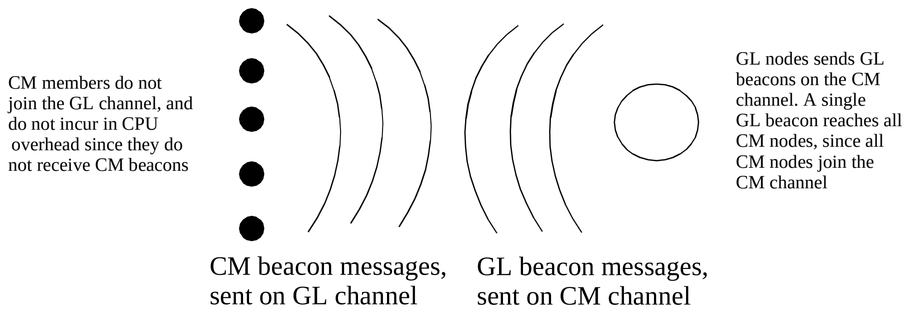
CM members do not join the GL channel, and do not incur in CPU overhead since they do not receive CM beacons
GL nodes sends GL beacons on the CM channel. A single GL beacon reaches all CM nodes, since all CM nodes join the CM channel
CM beacon messages, sent on GL channel
GL beacon messages, sent on CM channel

Figure 3: Two Channel Communication Pattern Used in Beacon

4.4 Election Protocols and Group Formation

Now that we have decided on the communication model that we want to have between the Common Members and their Group Leader, we now need to specify a procedure with which the members of the group can unanimously arrive at such an arrangement.This is done with a very simple election routine that is done at initialization time on every node, and also every time that a CM has not received messages from its GL within a configurable period of time. This election procedure is similar to the root node election used in the IEEE 802.1d link layer protocol. Another similar variation is used in IGMP version 2. The election procedure that follows is an adaptation of this simple procedure to the double channelled communication model:

1. CM members *always* emit periodic beacons on the GL channel. This is elegantly done by having a separate thread dedicated to this task. The beacon messages are used both to report liveness status, and also for election purposes. Beacon, messages are *always* sent with TTL=1, such that they are contained within the T_0 domain.

2. When entering the election procedure, the CM node will join both the CM and the GL multicast channel. This is the only time that a CM node has to join the GL channel.

3. After joining both the CM and GL channels, the CM member waits for a period of time *e*, during which the CM member records the messages that it receives. The outcome of the election will be determined by the beacon messages received within the period *e*. Note that there is no new message type associated with the election, and also that there is no specific election protocol. The election is simply done by having each electing member observe the traffic in both multicast groups for a period of time and then to deduct the state of the system using the following rules:

4. If the CM member receives a beacon in the CM channel, this means that this T_0 domain already has a leader. The CM node than leaves the GL channel and there after

runs in CM mode. This mode basically consists of continuing to beacon periodically to the GL channel and of monitoring the health of the group leader by receiving messages in the CM channel. The CM node will only exit this state if it fails to receive beacons from the GL member, in which case it will re-initiate the election, going back to step 2 above.

5. In case that the CM node does not receive any beacons in the GL channel, then it will assume that the T_0 domain was undergoing election and that the node that should be chosen as the group leader is the node that sent a beacon message from the interface with the lowest IP address.

6. The node that contains the interface with the lowest IP address will immediately start sending beacons on the CM channel. It will also assume the role of Group Leader and will initiate the search for higher numbered tiers.

7. All other nodes resume operation as CM nodes, proceeding as in 4.

4.5 Interdomain Elections

As a result of the tier zero elections, a collection of independent T_0 groups has been formed. The leaders of these groups will now search for each other, with the objective of attaching their T_0 domain to a higher tiered group. This operation will produce beaconing structures which involve multiple T_0 groups. In the discussion below, these beaconing structures are called *domains*.

In order to unite the separate T_0 domains, we will utilize the initial assumption that the list of all IP addresses for all nodes in the system is available from the distributed system management facility. Each GL member which has been elected at tier N will make a search for other tiers consulting this list. We can optimize this search according to the way that we chose T_0. If T_0 corresponds to a subnetwork, it is possible to send a subnet directed broadcast querying for the leader of each T_0 domain. In the following discussion we label the node performing the search *S*, and the nodes responding to search queries *R*:

1. The search space consists of all the nodes in the list. The first step is for the *S* to delete from the search space the nodes that it knows to be located within its own T_0 domain (since these nodes were already discovered by the initial election).

2. An IP address is now selected from the list, giving priority to the addresses which can be reached with the least number of routing hops. A point to point connection to the node is attempted. If it fails, we choose another address and retry. This is repeated until a responsive node *U* is found.

3. Node *S* now sends a query to *U*, requesting its status (CM, GL or undefined).

4. If *R* replies that its status is undefined, it means that it is undergoing election. Then *S* will wait for a period of time *e* and expect to be informed of the new status of the node and proceed as below.

5. If the *R* replies that it has CM status, then *U* will direct *S* to the group leader of the highest numbered tier of its domain. Then *S* will terminate the connection to *U*, and contact the GL member named in the reply and proceed as below.

6. In the remaining possibility, *S* eventually contacts a node which is a GL in another T_0 domain. We label this node *L*. The two nodes then exchange the topology information related to their domains, that is, the list of nodes which they have already discovered. This information exchange is all that is needed for the election. Again, the results are determined by an implicit rule derived from the network addresses
7. Using a rule similar to the one used in the T_0 election, the lowest numbered IP address is used to determine if L or *S* wins this election. The leader of the domain which contains the lowest IP address (and which is by definition a tier leader) will take no action. The other node, will attempt to join the other domain, by selecting a group leader in the other domain that is the closest to it, according to some metrics such as the number of IP hops (easily obtainable, given we have the IP address), and also that has not reached its maximum degree of members. This closest leader is labeled the attachment point, *A*. The join procedure is initiated by the joining node, which sends a join message to *A*.

8. If A can accommodate a new member, then the join proceeds and the joining node assumes the role of a CM node in the higher level tier, being responsible to sending beacon messages to its leader and also being responsible to monitoring the leaders health.

9. If the joining node cannot find any tier leader that it can join, then it will chose a CM node that is already attached to the tree (preferably the CM that is closest to the joining node according to the number of routing hops) and make a request to initiate a new lower level tier. The chosen CM node will then become the GL of the newly formed tier.

Once a searching node finds another domain with a lower IP address, it will join this domain and stop the search. Therefore, each domain will have only one node which continues the search procedure, which is the leader of the highest numbered tier, or *root* node. A root node will stop to search for other members if it detects that it has found all possible T_0 domains, i.e., if any of the following conditions apply:

a. It looses an election to a lower addressed domain, and becomes a CM node in a tier of the winning domain.

b. A node detects that all T_0 domains are represented in it domain. This condition also means that this node is the root of a spanning tree that connects all T_0 domains in the system. The search can then stop because all T_0 domains have been found.

4.6 Node Monitoring in Tiers above T_0

Now that we have glued together the T_0 domains into a hierarchical structure, we need an infrastructure with which members of a higher tiered domain can monitor themselves. Again, the number of nodes in each tier was limited to a tractable number, and so even the least sophisticated procedures would work. Basically it is just necessary for the CM members of a domain to send beacons to their SL, and the SL has to respond with a beacon to all its members. A straight point to point approach would work, but why develop new code when we already solved the problem for the T_0 case? By virtue of being networks neighbors, the T_0 members are able to utilize the two IP multicast channels and save in resource utilization and programming complexity. The logic of the topology structure is self similar, and therefore the code should be reusable. All that is needed is for each member of a domain to establish an IP multicast tunnel to each other member, and make all group leaders run an instance of a standard IP multicast routing protocol, such as DVMRP, or PIM.

We can now run the very same procedure used for T_0 monitoring, and have the same savings, although they will come for different reasons. In the T_0 level, having only the GL node join the GL channel means that all other nodes will filter out the unwanted beacons. In levels above T_0, that are not network neighbors, not having any other node join the GL channel means that the tunneling nodes that have no members for that group will send prune messages back to the source of the beacons, having the effect that beacons are only effectively transmitted to the GL, the only node that is actually interested in them. On the other hand, all nodes join the CM channel and act as tunneling endpoints, and therefore the beacon messages from the GL are routed to all nodes, and the network bandwidth is thus spared by virtue of the one to many transmission pattern.

4.7 Recovering from Failures of Links in the Hierarchical Tree

The most important benefit of the *Beacon* hierarchy is probably the simplicity of the recovery procedure in the case of the loss of nodes. This capability should be studied in the context of the recovery of spanning trees, and may have many applications in layer 2 protocols. Each star in the beacon hierarchy is a separate recovery domain. A failure of a node in a specific domain is dealt with only by the members of that domain, and will make minimal impact to the rest of the tree. For example, consider the failure of the root node. In non hierarchical systems the entire tree would have to be recalculated. Now, let's examine how this is handled in *Beacon.* The root is the leader of the highest tiered domain, T_D. Its failure is directly sensed by all members of T_D, which will then promptly elect a new leader. The tree now has a new root, and no new edges had to be created or recalculated. Nevertheless, the failed leader at the T_D level was also a leader of T_{D-1}, by virtue of the hierarchical construction of the tree. Therefore all members that were connected to the tree via the T_{D-1} domain of the failed leader are now disconnected. The members of T_{D-1} domain also sense the failure of their leader, and will then elect a new leader which in turn will re-attach the disconnected subtree at some point of the main tree. The important aspect here is that neither the main tree nor the disconnected subtree were dissolved, and do not have to be recalculated. Lastly, we note that the members of the T_0 domain that contained the failed leader are still disconnected. They perform a similar procedure, electing a new leader which then re-connects the T_0 domain back to the main tree.

The recovery capability of *Beacon* preserves the calculated leaders and edges even in the face of an arbitrary number of failures. Each failure will cause at most tree disconnected domains, which elect new leaders and attempt to re-connect to a higher priority tree. Eventually all the trees are reconnected with minimal disruption or loss of calculated leader and edges.

5 Relationship to Previous Work

Curiously, the majority of the literature in Topology Services systems does not come from the area of distributed system management, but rather from the area of communications. We mentioned previously the similarities with the IEEE 802.1d protocol [10], which is a layer 2 protocol. We can also find some heartbeat capability added to standard routing protocols, such as RIP [11], OSPF[4], DVMRP[5], PIM-SM[12]. The heartbeating capability of these protocols is usually an added feature to the main protocol, provided to give the protocol the capability of coping with router failures. The main difference is that whereas these protocols require the configuration of redundant routers such that there are alternative paths to route around failures, *Beacon* configures the smallest number of routers such that there is a tree that spans all members. *Beacon* copes with failures by having all nodes in the distributed system acting as stand by routers, and dynamically configures nodes as router nodes as needed. This contrasts with the traditional approach of configuring redundant routers because it saves on the amount of route announcements and router traffic overhead. If we were to configure every single node of a distributed system as a router node, than the amount of router announcements and router traffic would be enormous. Also, it would create looped networks, which would cause most dynamic protocols to converge slowly or not at all.

The proposal in *Beacon* is to precisely abstract out the Topology functionality from the routing domain and put it back in the distributed system management domain, where it can be better controlled. It was mentioned earlier that a distributed system may be comprised of several nodes taken from subsections of network domains that run different layer 2 and layer 3 protocols. *Beacon* allows for the uniform control of the topology parameters (such as the heartbeat interval, which determines the speed with which failures are sensed). Just because we have a system that contains a few nodes from a huge Ethernet LAN that need to be monitored very closely because they are video stream servers, it does not follow that all the routers in this large LAN should be setting their *hello* messages to 10msec intervals. Actually, most protocols recommend intervals that range from 30 sec. to 90 sec., far exceeding the range for most critical applications.

Beacon also relates to border gateway protocols such as BGP[6] and BGMP[9], in the sense that it offers interdomain multicast routing. Actually, *Beacon* could be deployed as an interdomain multicast routing protocol in conjunction with either DVMRP, MOSPF or PIM-SM or PIM-DM. But *Beacon* does not deal with individual multicast groups, it just establishes tunnels and lets another protocol, such as DVMRP deal with the group subscriptions. Finally, *Beacon* is hierarchical, and while an attempt at providing hierarchy for interdomain routing protocols was made with BGP Federations [8], the approaches are very different. The *Beacon* hierarchy is self configurable and self similar, very different from the proposed, statically configured flat hierarchies proposed in [8].

References

1. S. Deering, “Host Extensions for IP Multicasting, IETF RFC 1112, 1989
2. A. Ballardie, “Core Based Trees Multicast Routing Architecture”, IETF RFC, 1997
3. J. Moy, “Multicast Extensions to OSPF”, IETF RFC 1584, 1994
4. J. Moy, “OSPF Version 2”, IETF RFC 2328, 1998
5. D. Waitzman, C. Partridge, S. Deering, “Distance Vector Multicast Routing Protocol”, IETF RFC 1075, 1988
6. Y. Rekhter, “A Border Gateway Protocol 4 (BGP-4)”, IETF RFC 1771, 1995
7. A. S. Thyagarajan, S. E. Deering, “Hierarchical Distance-Vector Multicast Routing for the MBone”, ACM SIGCOMM, 1995
8. P. Traina, “Autonomous System Confederations for BGP”, IETF RFC 1996
9. D. Thaler, D. Estrin, D. Meyer, “Border Gateway Multicast Protocol (BGMP): Protocol Specification”, IETF Internet Draft, 2000
10. R. Perlman, “Interconnections: Bridges, Routers, Switches and Internetworking Procols”, 2nd. ed., Addison-Wesley, 1999
11. G. Malkin, “RIP Version 2”, IETF RFC 1058, June 1998
12. D. Estrin et. al, “Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification”, IETF RFC 2362, June 1998