A Dynamic Neighbourhood Discovery Protocol for Active Overlay Networks

Sylvain Martin and Guy Leduc

Research Unit in Networking, Université de Liège, 4000 Liège 1, Belgium
{martin, leduc}@run.montefiore.ulg.ac.be
http://www.run.montefiore.ulg.ac.be/

Abstract. *d-RADAR*[1] is a neighbourhood discovery protocol for overlay network environments designed for (but not limited to) active network overlays. The core of the algorithm is an expanding ring-search based on the IP routing table content augmented with traffic-based and dynamic refreshing techniques that allows it to react to virtual topology changes (nodes joining/leaving the overlay) as well as IP topology changes (broken and repaired link, route changes and moving nodes). This paper presents how the protocol detects overlay candidate nodes using probing capsules and the algorithms needed to select neighbours among the candidates. We also show how *d-RADAR* keeps the neighbouring table up to date and learns topology changes while keeping a low discovery and refresh overhead. A short summary of simulations carried out with our active network simulator illustrates how these algorithms actually behave.

1 Discovering Neighbours in an Overlay Network

1.1 Introducing the Problem

In several new network technologies like *ip multicast*, *IPv6* or *active networks*, we have to face the situation of a heterogenous network made of routers that understand the new protocol and *legacy IP routers* for which new packets are invalid. One solution to incrementally deploy a new routing protocol while keeping backward compatibility with the existing solution is to build *virtual links* or *tunnels* between routers that support the new technology (for instance using IP over IP encapsulation [11]).

The mesh obtained with these tunnels can then be considered as a regular network by the new routing protocol, and will be referred to as the *overlay network*. Experimental overlay networks have already been set up for the routing technologies cited above, respectively MBONE [6], 6BONE [7] and ABONE [4,5].

Several parameters distinguish overlay networks from 'real' networks, like the fact that link costs may change at any time (due to a change in the underlying topology), or the fact that there's usually no broadcast facility to discover peer routers.

If we consider the setup of a new router in a 'real' network, one of the first steps is to discover neighbour routers that are directly reachable through the router's interfaces.

[1] *This work has been partially supported by the Walloon Region in the framework of the WDU programme (ARTHUR project), and by the Belgian Science Policy in the framework of the IAP programme (MOTION P5/11 project). Sylvain Martin is a Research Fellow of the Belgian National Fund for Scientific Research (FNRS).*

This discovery usually involves simple packets using a conventional ("all routers") IP destination address that are broadcast on the link/LAN to which the router is connected. In overlay networks, this technique can be used to maintain links once they're established, but not to setup the virtual links the router will use.

1.2 Approaches to Neighbourhood Discovery

In most existing frameworks, overlay networks use manually-provided lists of neighbours for their local domains and connect to a large scale backbone using a manually designated access router [4,5]. While this is affordable to interconnect small testbeds, it cannot reasonably be used to deploy a new technology in a large or frequently changing network.

Alternative solutions have been proposed, mainly based on the use of the local *DNS* server to locate candidate neighbours for a given protocol, or to locate database maintainers of these candidate neighbours. Once the DNS has been used as a rendez-vous point, the neighbourhood discovery is reduced to probe active[2] routers listed by the registry and select the most interesting ones. An algorithm like TAO [9], for instance, creates clusters of close nodes and selects a *leader* for each cluster, which is the sole router having virtual links to other clusters.

Even though DNS-based solutions may be useful in some circumstances, maintaining and updating DNS mapping usually requires human intervention which make them unappropriate for dynamic environments.

Other works expect active nodes to join a given multicast group used to advertise that they are active and listen for other nodes' messages. However, assuming that all the routers of a domain are multicast-enabled is a very restrictive hypothesis, and we can't assume that the active router is member of a multicast overlay, of course.

The approach we selected is completely distributed, and avoid the need for a single node which would register and list active routers. Candidate neighbours simply come from the IP routing table, which is used to feed an *expanding ring search* (we refer to this technique as *table-driven* discovery), and from *previous hop* information carried by traffic that uses the same execution environment (*traffic-based* discovery).

Our previous work on RADAR [10] has shown that this technique could be used to create an overlay network that guarantees that if node A is a neighbour of B on the overlay, no other node of the overlay receives the messages B sends to A.

Solutions developped for peer-to-peer overlays like *CAN*[16] or *Pastry*[15] are not well suited for building overlays of *routers*, but they could help *end-systems* to join an overlay built by *d-RADAR*.

1.3 Challenges in Neighbourhood Discovery

As a ring-based discovery combines the search for active routers and the check for neighbourhood in one single operation, special care must be taken to make sure the discovery is not flooding the network with scans.

[2] without loss of generality, we'll restrict the discussion to overlays of active routers for the sake of readability

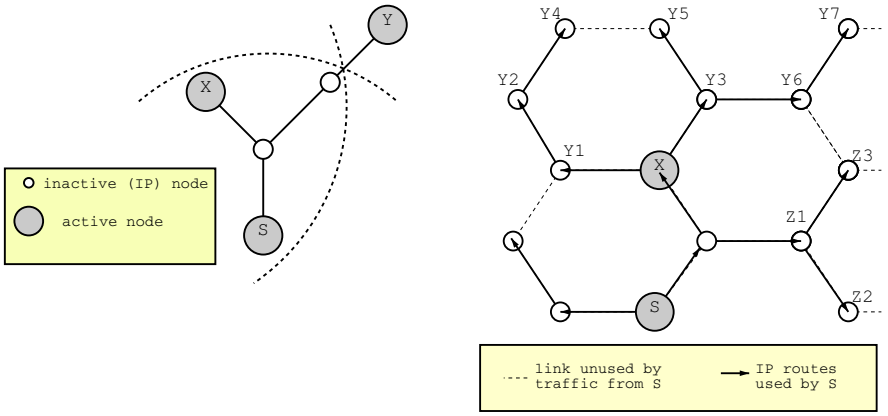

Fig. 1. *(a) stopping at the first discovered neighbour may isolate some nodes from the topology, (b) illustrating the problem of hidden targets*

- One neighbour per physical interface is usually not enough. If you look at fig. 1.a, you'll notice that stopping the search at the first neighbour discovered on an interface will result in the impossibility for both X and S to discover Y.
- Scanning targets in the whole domain will usually be too costly, but on the other side, it is not possible to tell *a priori* how far the scan should go. Depending on the density of active nodes in a domain, the distance needed to find N neighbours may change, as well as the amount of neighbours needed to keep routing efficient.

Note that registry-assisted discovery will suffer from the same kind of problem if the amount of active nodes in the domain becomes pretty large. As soon as a neighbour X is discovered by S, all the targets $Y_1 \dots Y_n$ (see fig. 1.b) that require crossing X to be reached will actually be *hidden* by X, in the sense that as X will see any capsule sent to Y_i, the routing decision could have been deferred to X. Identifying hidden routers and avoiding to scan them as soon as possible will help reducing the unwanted discovery overhead.

Moreover, as soon as one considers *dynamic topologies* – i.e. networks where the subset of active nodes and the routes to targets may change frequently[3], *hidden* nodes become even more important as knowing the *hidden-by* relationship will help telling which previously discarded target need to be re-scanned due to a topology change.

It should also be taken into account that active routers may join or leave the overlay without a topology change at the IP level. Indeed, active routers may stop supporting a given execution environment due to administrative decision, while the router itself (at IP level) is still present in the network.

This means that a neighbourhood discovery technique that will be used to build active network overlays will *have* to check the state of the current neighbours periodically,

[3] this will especially be the case in wireless networks, but may be extended to any network in which such a topology change is not considered as a exceptional event

even if no state change is announced by the IP layer, rather than waiting for an update of the IP table to learn that a neighbour is down.

Even though pro-active checking of the neighbours' state for non-active overlays is not mandatory, it may be an interesting property as it allows the overlay to recover situations (like selecting an alternative route) *quicker* than the underlying network.

1.4 Probing Targets

d-RADAR[4] uses *active probes*, known as *AYA*[5] capsules that it sends to neighbour candidates in order to check whether they're active or not. Compared to *ICMP echo* packets, *AYA* capsules also allow to check if the target is currently responding to specific execution environment's messages.

When no active router is met on the road to a non-active target X, the *AYA* capsule is lost when reaching X. But if any router on the road is active, it will intercept the *AYA* capsule and store its own address in it before sending it back to its source. This technique of allowing a capsule that has X as its IP destination to be intercepted by a node other than X - referred to as *capsule grabbing* - is detailed in *RADAR*[10].

While it isn't a usual behaviour for a non-active router, *grabbing* could also be implemented in non-active networks by means of techniques like IP router alert option [12]. However, router alerts usually slow down packets a lot on every node while a grabbable capsule is processed at top-speed by a legacy router.

2 Required Environment

d-RADAR has been implemented for the *ANTS* execution environment [1,2,3] which itself relies on NodeOS infrastructure [8]. Small modifications to the ANTS code are needed to implement our solution, mainly in order to allow emission and reception of *grabbable capsules* and to access the IP routing table. However, *d-RADAR* uses very lightweight capsules and could be easily adapted to overlays that use only passive packets.

d-RADAR expects the NodeOS to be able to deliver the network layer routing table for *table-driven discovery*, and it also requires a method to get notified of changes (added/removed entries and cost changes) to this table. It does not, however, make changes to the IP table and therefore its presence is transparent to legacy routing. The only system table alterred by *d-RADAR* is the *overlay's neighbourhood table* which may be used by active protocols that directly use neighbourhood information or an active routing protocol. Our discovery technique also expects that route table entries will at least contain the *route cost*, preferably expressed as a *hops count*, and an identifier of the interface used (for the purpose of grouping targets based on the interface they use).

It is important that the IP routing table holds enough information about the local domain. If some routers are not listed in that table, the only chance to detect them is by *grabbing* part of the traffic they send to other nodes. In particular, a *host* node is unlikely

[4] Dynamic Ring-based Adaptive Discovery of Active neighbour Routers
[5] for "Are You Active?"

to discover any neighbour (but its default gateway) if it does not first start sending (or receiving) traffic from another active node. To make this possible, the *overlay routing table* will always contain a "default" entry that will create direct tunnels to any IP destination on demand.

The routing protocol used to build the IP routing table has little significance, but *d-RADAR* may be made much more efficient if the table contains information about the *last router* crossed before a destination D is reached. With this information, we can retrieve the whole sink tree and deduce for some targets whether they can be reached or not without actually probing them. In the absence of such information, for instance if IP was running a distance vector or if we receive the *forwarding* table rather than the *routing* table, we can either try to get the information by sending *ICMP echo* packet that would have the *Route Record* option set, or simply assume that every target *must* be probed.

Finally, in order to support *traffic-based discovery*[6], it is mandatory that packets processed by the execution environment carry a *previous hop* information. While this is virtually true in every active network environment (as the previous hop is the node that will be asked for code download if the protocol of a packet is unknown by the current node), it's less direct in a IPv6 or multicast overlay, but the source address of the current tunnel should be a good candidate.

3 d-RADAR Approach

d-RADAR is mainly based on a ring-search discovery that adapts the maximum searching distance (a.k.a. the discovery *threshold*) to the density of neighbours. Targets are grouped by their cost[7] in *rings* and then probed by increasing ring cost. Everytime a new neighbour or a hidden target is found, the threshold is reduced multiplicatively by the α constant, and everytime a ring completes, the threshold is increased by the amount of remaining targets. In other words, if T_i is the threshold value after ring i discovery has been completed, we have

$$T_i = T_{i-1} \cdot \alpha^{a_i} + (n_i - a_i) \tag{1}$$

where a_i is the amount of positive replies (neighbours and hidden targets) for ring i and n_i is the total amount of targets on ring i. If there were enough replying targets on ring i, it may occur that $T_i < i$, which stops the discovery. Rather than excluding discovered neighbours that have a cost c such as $T_i < c < i$, we then use $max(i, T_i)$ as the effective threshold.

Our previous work on static topologies has shown that this approach offers good results, especially when $T_0 = 2,\ \alpha \in [0.5 \ldots 0.8]$ and that the exact value chosen for α has little significance on the overall mesh obtained[8]. One interesting property of

[6] *traffic-based discovery* consists of selecting targets to probe by looking at the intercepted data capsules

[7] Here comes the main reason why *hops count* costs are preferred: we need to ensure that a target on ring $i + 1$ can only be reached through a target on ring i.

[8] a higher α value will result in more neighbours per node, but it usually does not degenerate into a full mesh and α does not need to be fine-tuned for a particular network (see [10]).

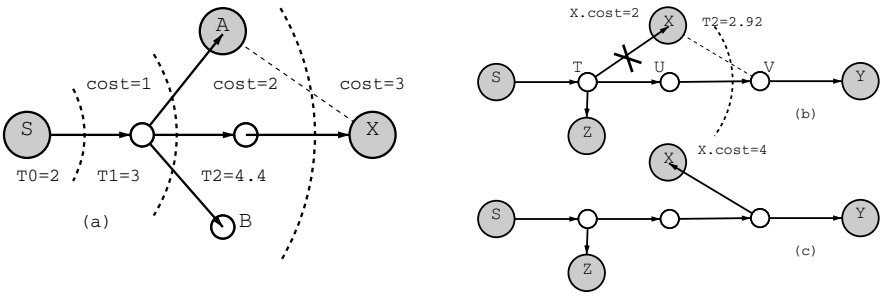

Fig. 2. *assuming $\alpha = 0.8$, (a) Initial treshold after ring 2 detection is complete is $T_2 = 3 \cdot 0.8 + 2$, which is above 3 and allows S to scan X. If B now becomes active, we have $T_2 = 3 \cdot 0.64 + 1$, which excludes X from the neighbourhood of S. (b) if the link on default route to X is broken, X can no longer respond and is excluded from the neighbourhood of S. T_2 then raises to 4.4 which allows Y to join the neighbourhood. (c) when the IP routes are updated, X appears at level 4 and will come back to the neighbourhood.*

this threshold computation technique is that, once a neighbour is found, the search will stop quicker when the neighbour *hides* more targets (and thus potentially serves more destinations).

3.1 Neighbours Come and Go ...

The changes of the "activity" state of a router (i.e. an Execution Environment starts or stops) as well as routing updates that affect the cost of neighbours will be reflected on the set of neighbours a node can use. Not only because of trivial modifications – of course if X stops being an active node, X must be removed from the set of active neighbours – but also because it affects the global environment of the considered active router S. As explained in section 1.3 and illustrated in fig. 1.b, when a neighbour X leaves, a set of targets $\{Y_1 \dots Y_n\}$ leaves the *hidden* state[9] and become potentially reachable. These new neighbour candidates must be re-scanned and the closest active ones have good chance to be included in the new set of neighbours.

The departure of X will also lower the active nodes density in S's surrounding and may require that S increases the maximum distance at which neighbours should be scanned (i.e. increasing the threshold in *d-RADAR*).

Similar changes may occur when an *inactive* (and not hidden) node with a cost that is lower than the actual threshold becomes active (see fig. 2.a) :

- some other inactive or neighbour nodes may become hidden.
- as a result of newly responding targets (the newly active one and nodes moving from *inactive* to *hidden* state), the threshold may be reduced, excluding some of the furthest neighbours from the new neighbourhood set.

The case of an IP route change (modification of the route's next hop or cost) is simply handled by removing every information we had about the route's destination

[9] a node Y is *hidden* if *AYA* capsules sent to Y are intercepted and replied by another node X.

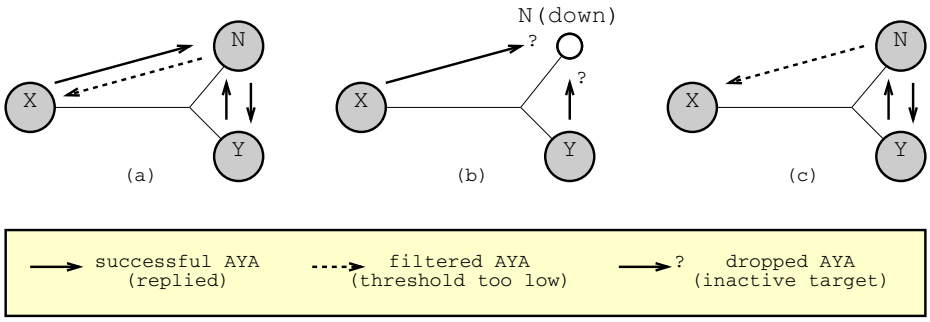

Fig. 3. *A problematic scenario if down neighbours are not re-scanned periodically: (a) N is initially discovered by X which has a high threshold but X can't be discovered by N because it's out of its scope, (b) when N crashes, both X and Y notice it after their refresh* AYA *gets dropped, (c) after N recovered, only Y is able to detect it from N's boot-time scanning. X being out of scope, it believes N is still down*

and then re-inserting the new route as if it was a completely new target (see fig. 2.b and c).

d-RADAR stores, for each output interface, the *history* of the boot-time discovery as a list of (T_{i-1}, a_i, n_i) parameters that can be used to recompute the current threshold from T_k when a change occurs on n_{k+1} or a_{k+1}, by simply re-applying (1) on $T_k, T_{k+1}, \ldots$ until we reach $l : T_l \leq l$. As soon as the new threshold l is defined, its value is committed to the scanning processes that use this interface and will result in some probes being filtered out (if their cost is above the new threshold) or granted, and thus removing or reintegrating some active nodes in the neighbourhood set.

3.2 The Soft-State Refresh Mechanism

In order to maintain an up-to-date neighbourhood set, *d-RADAR* associates every entry X in the *neighbour* table with an *expiration date* E_X, which defines how long the entry will remain valid. Once the entry expires, it will trigger a *refreshing scan* of X, which may either confirm the current state of X or detect that X is no longer a responding neighbour, but that it has become a *down* target[10].

A similar timer is kept for every *down* target so that we can check whether an old neighbour has recovered or not.

To understand the role of *down* targets refresh, it is important to remember that a neighbourhood relationship may not be symmetric. Therefore, as figure 3 illustrates, it is possible that a target N, neighbour of X, will not send *AYA* capsules to X after it recovered from a crash, which could lead X to conclude that N is still down while it could actually be reintegrated in X's neighbourhood.

[10] a *down* target is a target that has been a neighbour earlier, but which no longer responds to *AYA* capsules. This state is different from the *inactive* node which never replied to any *AYA*.

The duration of the *validity period*, i.e. the period between two refreshes for a given target, is not constant in *d-RADAR*. Instead, it depends on the *age* of the information. In other words, for a neighbour X, if T_X is the time at which X has entered the neighbourhood for the last time, and if C_X is the time at which the last refresh for X occurred, the current expiration for X will be E_X, given by (2).

$$E_X = C_X + k \cdot (C_X - T_X) \tag{2}$$

Everytime a target moves between the *neighbour* and the *down* state, its last join/leave time T_X is reset. As a result, the more stable a neighbour is, the more we will *trust* it and assume it's unlikely to disappear in a near future. On the opposite side, a new (or recently recovered) neighbour will be scanned more often.

Defining the expiration time this way can be interesting to protect against short node failures. When a node stops responding, we first assume this is a temporary situation and that it is likely to respawn in a near future. As the time goes by, chances are that it was rather a permanent failure and refreshes will become more spaced. How fast a temporary situation will be considered as permanent will depend on the k value.

Another context in which this behaviour may be useful is the field of *ad hoc* networks where we could have a low refresh rate for permanent neighbours and keep a high refresh rate for more mobile neighbours.

In order to prevent the validity period from becoming too long in the case of very old entries and keep a useful failures detection time, a maximal refresh period (typically a few tenths of seconds) is enforced regardless of the age of the information.

3.3 Freshness Through Traffic Monitoring

In addition to the periodic neighbour state refresh controlled by E_X, it may be interesting to use traffic received from a neighbour X to monitor X's activity. As *d-RADAR* already catches every incoming capsules for the sake of *traffic-based discovery*, this additional monitoring would virtually come with no cost at all. A smart routing protocol that wishes to offer a low packet loss probability might compare those activity reports and select the route which goes through the most recently (or the most frequently) refreshed neighbour.

To help building such protocols, *d-RADAR* offers information about the last time a capsule has been received from neighbour X in the neighbourhood table (R_X) and maintains the average inter-arrival time $\overline{\Delta T_X}$ according to (3).

$$\overline{\Delta T_X} = \beta \cdot (now - R_X) + (1 - \beta) \cdot \overline{\Delta T_X} \tag{3}$$

An active protocol that would like to ensure to be notified of node failures at last t seconds after the failure could check $now - R_X$ before sending a capsule to X and require X to send back an acknowledgement if $now - R_X > t$.

In classical networks, the incoming traffic on an interface can be used as a *replacement* of the heartbeat that maintains neighbourhood information up to date. If IP receives traffic from a link, the routing protocol can safely assume that the router at the end of that link is still up and running without even monitoring it. It can also decide to make heartbeats more frequent when there's not enough traffic to keep the information

refreshed, as the monitoring traffic will *replace* the normal traffic without competing with it.

Even though information about incoming capsules from X provide a kind of *freshness* information about X, some neighbours in an overlay network cannot be monitored that way.

First, because our neighbourhood relationship isn't symmetric, we may receive no traffic from a neighbour X as well as we may receive traffic from a node Y that is not a neighbour. Indeed, when IP routes $N \rightarrow X$ and $X \rightarrow N$ are different, it is possible that an active node Y hides N from X while the messages sent by N directly reach X. We thus have to keep on sending *AYA* refreshes regardless of the amount of traffic we receive from our neighbours.

Moreover, on an overlay network, we should be very careful before we increase the *AYA* refresh rate. One should keep in mind that active capsules may represent only a small part of the traffic received from a given link and that other traffics like UDP, TCP or capsules from another execution environment could suffer from extra refreshes. Another possible problem comes from the fact that one single network interface card may be used to connect to a great number of neighbours $\{X_1 \dots X_n\}$. A simultaneous increase of the probing traffic for those n neighbours could lead to an excessive scanning overhead on links close to the scanner.

Finally, *d-RADAR* lacks information about the requirements of the active data flows like minimal neighbour freshness or tolerated response time to a neighbour failure. All it could offer is a generic service which could be insufficient for some application and unneeded by other ones. We will thus defer the decision of whether a refresh is required or not to the active routing protocol (or to autonomous active transport protocols) which will base it on the statistics provided in the neighbourhood table, and possibly on other statistics gathered by the execution environment or the NodeOS.

4 Simulation results

All the simulations have been run using our *Run Active Network Simulator*, which provides an ANTS platform for a generic network simulator (so far, both *SSFNet* [13] and *javasim* [14] are supported).

We have first run the *d-RADAR* on 4 series of 5 random networks, each consisting of 60 nodes, with a varying density of active nodes. Fig. 4 show how the network resource consumption evolves after every active node boot simultaneously, which is the worst possible situation. As one could expect, both the amount of AYA capsules sent by a node and the average travelled distance are higher when the density decrease. However, if we consider the global cost (summing the individual costs over all the active nodes), we can see on fig. 4b that dense networks consume more bandwidth than scarse ones.

Another interresting fact is that the *refreshing* traffic is quite independent from the considered density.

In addition, we ran a collection of proof-of-concept qualitative simulations to ensure that each of the possible topology change was identified by the protocol and that the proper updates were made to neighbourhood sets and discovery thresholds:

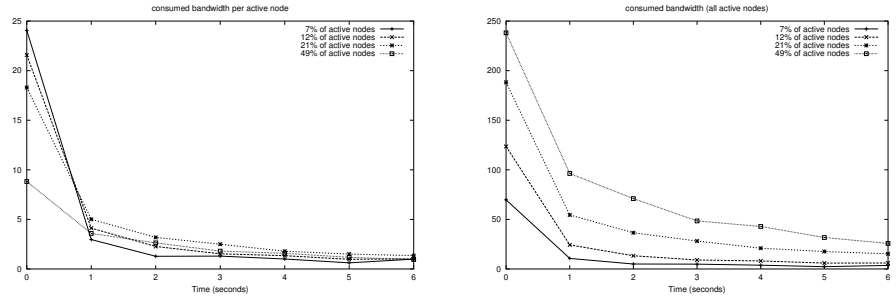

Fig. 4. *Average bandwidth (in Kbps) consumed by AYA over all the links in random 60 nodes network. (leftmost) AYA traffic generated by one average active node, (rightmost) traffic generated by all the active nodes together*

- stopping and resuming a neighbour's execution environment while keeping the IP layer of that neighbour up and running, including the case where the neighbour was hiding other active nodes (cf fig. 1.b),
- forcing active node X to be excluded from the neighbourhood set by activating other routers either on the path to X or not, but at a small enough distance to trigger threshold reduction (cf fig. 2.a),
- breaking and restoring links on the path to X, forcing *d-RADAR* to remove X from the neighbourhood until the IP routing table announces a new route with a different cost for X (cf fig. 2.b),
- making an active neighbour unreachable by breaking the sole path that reaches it or by stopping its IP layer.

In every situation that involved a change at IP level, *d-RADAR* has offered better response time due to its adaptive and pro-active behaviour (from 0.5 to 10 seconds to detect the loss of a neighbour X depending on the maximal value defined for D_X, respectively 5 to 25 seconds, compared to OSPF's 60 seconds heartbeat).

5 A word about the implementation

5.1 The structure

We developped *d-RADAR* as an active application that runs on top of a slightly modified *ANTS* [1,3] framework. Special care has been taken to make the solution as modular as possible, mainly regarding to the dependence to *ANTS* and the NodeOS.

As fig. 5 shows, the core class `NeighbourApplication`, with its helper class `IP-RouteTable`, is the only component which depends on (and interfaces with) the execution environment. It will take care of sending capsules created by the scanners and dispatching capsules received to the appropriated scanner. The core also carries every communication between the other internal components it is connected to, which reduces the inter-components dependency to a small set of "pipes".

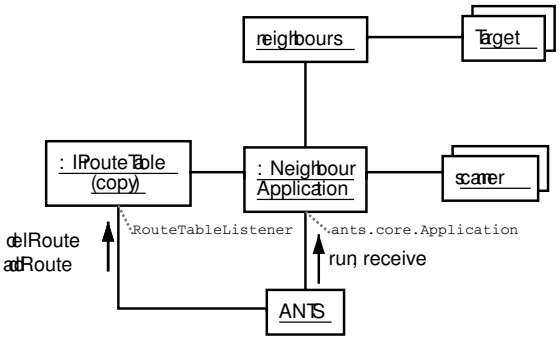

Fig. 5. *our software design*

The protocol's policy is implemented in the `Scanner` classes, which decide when and to which target the *AYA* capsules should be sent. They are also responsible for modifying the threshold of their respective network interface appropriately when capsules come back or when timer for unreplied *AYA* expire. Decisions of the scanners are transmitted as new `Target`-based objects which will join the `Neighbours` hashtable.

This table receives clock ticks from the core and will send the expired objects back to the core for refresh, which will enqueue them to the appropriated scanner. the `Neighbours` table is the class where all the soft-state part of the algorithm takes place.

5.2 Extensions facilities

Unlike its former static version, *d-RADAR* can deal with several parallel scanners for the same interface, and share the same threshold result, which makes the encoding of the algorithms clearer and more straightforward. So far, only two types of scanners have been used (pro-active and refresh scanner), but the modular nature of the actual code allows *d-RADAR* to be easily extended by new types of scanners (such as a *reverse* scanner that would scan one far target from time to time when the "bottom up" ring expansion does not give good results), or even virtual scanners (forcing the state of some targets or reordering the search based on informations gathered from other neighbours).

This just requires a new extension of the *scanner* class to be written, overloading the behaviour of the periodic "time for scanning" and of the "capsule replied" events delivered by the core.

6 Conclusion and Future Work

We proposed a distributed algorithm that builds an overlay dynamically, adapting to the density of active routers in the network, and that can evolve to follow topology changes. This algorithm require no special support from non-active routers and its principles can be translated to overlays other than active networks.

The feature implemented by *d-RADAR* can be seen as a service that can be reused by several active routing protocols, decoupling the problem of neighbourhood discovery from the routing itself. Therefore, some of the routing-specific decisions, like adapting the refreshing rate to applicative flow requirements or filtering oscillating neighbours, are delegated to the routing protocol itself: *d-RADAR* simply includes required timing information into the neighbourhood table.

Simulations have been carried on our active networks simulator, which completes our previous results on static topologies ([10]) with response to events such as routing table updates, active node failure, etc. An appropriate refresh mechanism should also allow *d-RADAR* to keep track of mobile neighbours or handle ad hoc networks, even though no specific simulations have been performed in that area so far.

Several optimization techniques can still be envisioned, and the existing framework can host them without changing the overall code structure. For instance, the initial discovery cost could be reduced through the *reverse scanner* quickly introduced in section 5.2, or *d-RADAR* could be made more conservative on discovered neighbours by using an *hysteresis* mechanism based on two thresholds (T_{in} used by the *active scanner* to accept new neighbours and $T_{out} = \gamma \cdot T_{in}$ used by the *refresh scanner* to reject existing neighbours).

References

1. D. Wetherall, A. Whitaker : ANTS - an Active Node Transfer System. version 2.0.
2. D. Wetherall : Service Introduction in an Active Network. *http://www.cs.washington.edu/-research/networking/ants/ants-thesis.ps.gz*
3. D. Wetherall, J. Guttag, D. Tennenhouse : ANTS - A Toolkit for Building and Dynamically Deploying Network Protocols. *IEEE OPENARCH'98*, April 1998
4. S. Berson, B. Braden : DANTE : Dynamic Topology Extension for the ABone. *ABone: Technical Specs - http://www.isi.edu/abone/DOCUMENTS/dante2.ps*
5. S. Berson, B. Braden, L. Ricciulli : Introduction to the ABone. *http://www.isi.edu/abone/-DOCUMENTS/ABoneIntro.pdf*
6. H. Eriksson : MBONE : the multicast backbone, *Communications of the ACM, vol. 37 issue 8 pp. 54-60* (1994)
7. I. Guardini, P. Fasano, G. Girardi : IPv6 Operational Experience within the 6bone. *http://-www.isoc.org/inet2000/cdproceedings/1e/1e_1.htm*
8. L. Peterson (Editor) : NodeOS Interface Specification. *DARPA AN NodeOS Working Group Draft*, 1999.
9. A. Collins, R. Mahajan, and A. Whitaker : The TAO Algorithm for Virtual Network Management. *Unpublished work*. December 1999. *http://citeseer.nj.nec.com/collins99tao.html*
10. S. Martin, G. Leduc : RADAR: Ring-Based Adaptive Discovery of Active Neighbour Routers. *Lecture Notes in Computer Science 2546, "Active Networks", Springer, 2002 (IWAN 2002)*, pp 62-73
11. D. Farinacci et al. : RFC 2784 - Generic Routing Encapsulation (GRE). *IETF, March 2000*
12. D. Katz (cisco Systems) : RFC 2113 - IP Router Alert Option, *IETF, February 1997*
13. SSFNet : Scalable Simulation Framework for modeling the Internet. *http://www.ssfnet.org*
14. Java-integrated, component based network simulation environment. *http://www.j-sim.org/*
15. M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron: Topology-aware routing in structured peer-to-peer overlay networks, *Tech. Rep. MSR-TR-2002-82*
16. S. Ratnasamy et al.: A Scalable Content-Addressable Network, *Proceedings of ACM SIGCOMM 2001*