Virtual Direction Routing for Overlay Networks

141

***Abstract*— The explosion of peer-to-peer systems in recent years has prompted research into finding scalable and robust seeding and searching methods to support these overlay networks. Initial work relied on network flooding to find information and while robust, lacked scalability. In effort to scale large networks, many have looked at structured approaches to the problem by imposing some sort of structure to the network topology and routing based on that structure. To support search queries, a robust overlay network with routing policies must be in place as search fails to address the actual data traversal path. In much the same way, routing in overlay networks have evolved from pure flooding techniques to structured techniques. In this paper, we attempt to drop the need for imposing a specific structure on the overlay network and introduce a technique to scalably route packets through an *unstructured* overlay network. We introduce Virtual Direction Routing (VDR). VDR is a lightweight, but scalable overlay network routing protocol that uses the concept of *virtual directions* to efficiently perform node information seeding and lookup. State information is replicated at nodes along *virtual orthogonal lines* originating from each node and periodically updated. When a path lookup is initiated, instead of flooding the network, query packets are also forwarded along *virtual orthogonal lines* until an intersection with the seeded state occurs. We show that VDR achieves high reachability with relatively low seed and search packet TTL even under high network churn. We also show that VDR scales well without imposing DHT-like graph structures (eg: trees, rings, torus, coordinate-space, etc.) and the path stretch compared to random-walk protocols is very good. The tradeoff is added latency by choosing suboptimal paths.**

I. Introduction

The explosion of peer-to-peer (P2P) systems in recent years for distribution of content has prompted research into finding scalable and robust seeding, searching and routing methods to support these overlay networks. Peer-to-peer systems are attractive for several reasons including 1) its distributed nature, 2) shared overhead, 3) relatively quick response to dynamic network changes, and 4) ease of joins and leaves. One of the biggest challenges in peer to peer systems is information replication/dissemination and discovery in environments of high dynamism. In order to locate where items resolved in a network of peers, various strategies for query propagation and information location need to be implemented. To support search queries robust overlay networks with routing policies must be in place as P2P systems often assume an underlying overlay network.

Peer-to-peer networks are broadly characterized into two major types based on whether or not strict overlay topologies are enforced: *unstructured* and *structured*. Unstructured P2P systems make little or no requirement on how overlay topologies are established and are often easy to build and maintain while being robust to churn [1], [16]. Unfortunately, they tend to have difficulty in finding rare objects and because overlay topologies tend to move toward a power-law distribution when it comes to node degrees, high load is often placed on high degree nodes. Early unstructured systems like Gnutella [8] queried for objects by simply flooding the network with search queries until an item was found. Flooding and even limited flooding techniques (e.g. normalized flooding [5]), tend to be prohibitive in large-scale networks as limited available bandwidth and the large number of nodes limit scalability. Several techniques have been been examined to attempt to address the lack of scalability in flood-based techniques citechawathe-sigcomm03.

Because of the inherent lack of scalability in flood-based schemes, researchers have looked at several hierarchical and structured based approaches [6], [7]. Hierarchical approaches like Kazaa [9] relied on certain nodes to house more information and coordinate data for a specific subset. Although effective in their own right, hierarchical approaches require reorganization in the event of node failure of local leader nodes. Recently, researchers have utilized novel distributed hash table (DHT) techniques to build virtual structures on overlay networks by mapping nodes to a specific structure be it a CHORD [6] or a coordinate space [7]. In these self-organizing overlay networks, neighborhood relations are more strictly controlled than in unstructured networks and search queries are propagated along the structure until a match is found. Maintaining the structure, however, makes DHT approaches brittle to attacks and churn as repair techniques that detect the failure and replicate the lost data or pointers incur substantial overhead [16].

In recent years, researchers have witnessed a large move yet again from hierarchical and structured systems to unstructured, flat, yet scalable techniques to perform search [5], [1], [16], [17]. As mentioned before, an underlying overlay network with specific routing strategies must already in place for each of the search techniques to work. Routing issues are differ from search issues in that 1) search does not deal with path selection but simply with finding objects and 2) search assumes an underlying overlay network. In this paper, we present Virtual Direction Routing (VDR), a light-weight node information dissemination and location routing technique in unstructured P2P systems. VDR places no restrictions on the underlying overlay topology and utilizes a novel concept we call *virtual directions* to provide efficient node query lookup.

In VDR, each node forms a set of *virtual interfaces* (int_i) and assigns immediate neighbors to an interface based on a hash of their unique node IDs (e.g. moded with the number of interfaces). State information is replicated at nodes along *virtual orthogonal lines* originating from each node and periodically updated. When a lookup is initiated, instead of flooding the network, query packets are also forwarded along

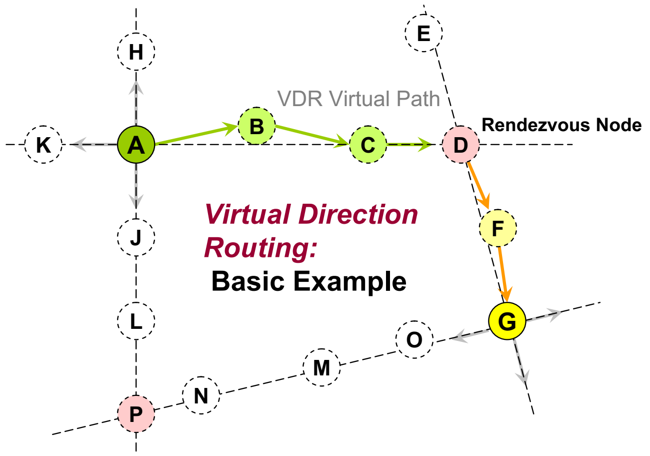

Fig. 1. Virtual Direction Routing Basic Example

virtual orthogonal lines until an intersection with the seeded data occurs. If more than one neighbor is assigned to a virtual interface, ties are broken by selecting the neighbor with the ID closest to the search ID. In this way, seed and query packets automatically "gravitate" toward each other increasing the likelihood of intersect.

Key contributions of VDR include:

- Introduction of the concept of Virtual Directions to eliminate the need for virtual coordinate space or DHT structures to locate items in structured-based approaches to provide routing.
- A flat, highly scalable, and resilient to churn routing algorithm.

We will show that:

- VDR performs much better in state dissemination and reach than random walk
- VDR scales much better than flood-based techniques such as normalized flooding techniques [1]
- VDR performs especially well in dense connectivity situations where the number of neighbors is high. This is valuable as the P2P overlay networks can easily achieve dense connectivity by installing links to several other peers/nodes
- In dynamic network environments where nodes frequently go on and off, VDR significantly outperforms its counterparts in terms of end-to-end reachability and throughput.

To achieve these goals, VDR trades off the end-to-end path stretch required compared to flood-based techniques. Since most real-world topologies have order $log(N)$ reach, it is expected that simple flooding techniques will find shorter paths from source to destination. VDR provides a *scalable* alternative to pure flooding and normalized flooding techniques. Additionally, state is not evenly distributed network-wide due to the biasing of dissemination packets.

The rest of the paper is organized as follows: Section II outlines the concept of VDR including a detailed explanation of information replication and lookup. Section III evaluates VDR against several protocols under varying conditions of churn and TTL. Finally, section IV presents some concluding thoughts and ideas on future work.

II. Virtual Direction Search

The concept of Virtual Direction Search (VDR) is simple: in flat networks, a pair of orthogonal lines centered at different points will intersect at two points at minimum. By seeding state information along orthogonal lines and performing node lookups along those same *virtual directions*, one can ensure successful node lookup in an unstructured manner without flooding the network as these seed and search packets intersect. In this section, we outline VDR and discuss various techniques for mapping neighbors to interfaces in a globally consistent manner, requiring low maintenance manner under various topologies as well as state dissemination and lookup techniques.

A. Virtual Interface Assignment

In this section, we define the concept of *virtual interfaces* as used in VDR. Traditionally, interfaces are physical devices that offer points of connection between other devices. These devices can be physical connectors or wireless antennas that negotiate links between neighbors. In VDR, each node partitions its set of one hop (or low latency) neighbors into a set number (n) of virtual interfaces. The total number of virtual interfaces per node can be fixed or varied but the partitioning strategy (i.e. hash functions) must be globally consistent. We will assume for now that the total number of virtual interfaces a node has (n) is fixed and globally consistent (i.e. all nodes decide on the same number of virtual interfaces and this number does not change).

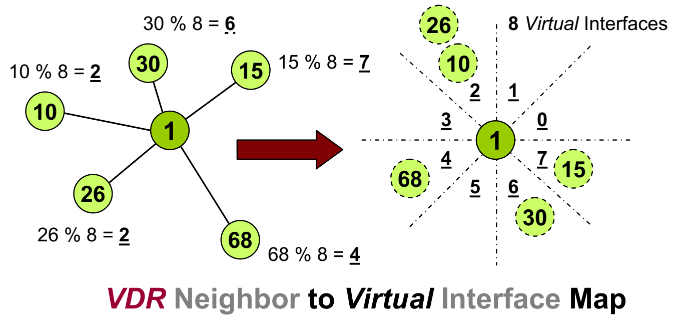

Fig. 2. VDR Virtual Interface Assignment

Each virtual interface is assigned an ID from 0 to $n-1$ and each one hop neighbor (as determined by physical neighbors or by a latency constraint) is assigned to a specific interface. In assigning nodes to an interface, it is important to keep the assignment globally consistent even in the presence of high churns. In other words, nodes assigned to a specific interface should always be assigned to the same interface even if they are unreachable for a certain amount of time. This will minimize the dynamism and make replicated data less susceptible to network dynamism.

Assuming each node has a unique identifier (e.g. IP address), we employ a simple heuristic to assign neighbors to an

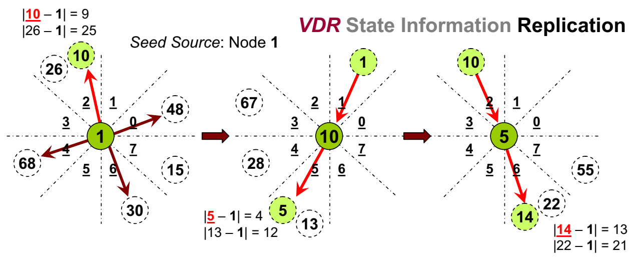

Fig. 3. VDR State Information Seeding Example

interface: 1) Hash each neighbor node ID to 160 bit IDs using SHA-1 [10] and 2) Mod the resulting value by the number of interfaces and assign the node to the interface ID with the resulting value. By assigning neighbors in the preceding manner, we are able to consistently map neighbors to the same interface despite network churn. It is important to note that with these conditions, some interfaces might have more neighbors assigned to them than others. We evaluate another technique whereby we first perform the hash but attempt to make sure that no interface is assigned additional neighbors until all other interfaces have the same number of neighbors in section III.

After all the neighbors have been assigned to a virtual interface, a *virtual north* is randomly chosen for each node. This is done by randomly selecting an interface to be the *virtual north*. This selection is important because information is later forwarded out orthogonal directions with respect to this *virtual north*.

B. State Information Dissemination

In order to minimize network flooding, each node disseminates its own ID to specific neighbors in the network to make itself easier to locate. To do this, each node periodically *seeds* its own ID to nodes along orthogonal paths with respect to its own virtual north. Each node will select 4 interfaces that are orthogonal to each other and choose the neighbor along that virtual interface which has the closest hashed ID match to the source node's hashed ID.

When the neighbor node receives this *seed* packet, it will note the previous hop and source of the packet in its routing table (storing the source as the destination and the previous hop as the next hop) and forward the packet out the interface that is virtually opposite of the receiving interface. The packet is not flooded to all neighbors assigned that that virtual interface, however, but the neighbor that has a hashed ID closest to the source's hashed ID. This will ensure that the packet forward is biased toward nodes that are closer in ID to the source so searching for nodes will form a much higher level of convergence. The packet is forwarded until the TTL is reached. Algorithm 1 gives the algorithm for VDR State Dissemination.

A secondary heuristic (in addition to pure random walk) is used for comparison in our simulations: randomly choosing a neighbor in each virtual direction rather than biasing it toward the ID of the source.

Algorithm 1 VDR State Information Dissemination

```
SendStatePacket($p$)
1:  // Check if we are the source - forward opposite if not
2:  if $p \rightarrow Src = ID$ then
3:      // We are the source, forward orthogonally
4:      // Hash Node ID to 32 Bit SHA-1
5:      $\Psi \leftarrow SHA1(ID)$
6:      // Get Interface ID of Virtual North
7:      $j \leftarrow$ GetVirtNorthIntID
8:      $\alpha \leftarrow$ NumInterfaces
9:      // Send out Orthogonal Directions
10:     for $i$ = 1, i $\leq$ 4, $i$++ do
11:         $\Phi \leftarrow$FindClosestHashedNeighbor($j$)
12:         // Send to Neighbor
13:         send($\Phi$)
14:         $j \leftarrow ((j + \alpha/4)\%\alpha)$
15:     end for
16: else
17:     // We are forwarding - only forward opposite
18:     // Hash Packet Source ID to 32 Bit SHA-1
19:     $\Psi \leftarrow SHA1(p \rightarrow Src)$
20:     // Get received interface ID
21:     $j \leftarrow (p \rightarrow Recv\_Int\_Id)$
22:     // Get opposite interface $j \leftarrow ((j + \alpha/2)\%\alpha)$
23:     $\Phi \leftarrow$FindClosestHashedNeighbor($j$)
24:     // Send to Neighbor
25:     send($\Phi$)
26: end if
```

C. Route Query

When a node wants to do a search for another node in the network, it generates a search request (SREQ) packet and forwards it along virtually orthogonal interfaces with respect to its virtual north. Upon receipt of the packet, each neighbor will update its routing table with a "destination - next-hop" entry based on the SREQ packet's source and previous hop and check to see if it has a routing entry to the node the source is searching for. If not, it will forward the node to the interface virtually opposite the receiving interface until it reaches a node that has information to the search destination or reaches its own TTL.

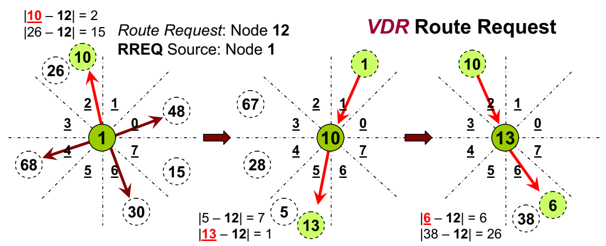

Fig. 4. VDR RREQ Path Illustration: Packets are biased toward destination ID.

If, however, an entry to the search destination exists, the node will prepare a search reply (SREP) packet which contains

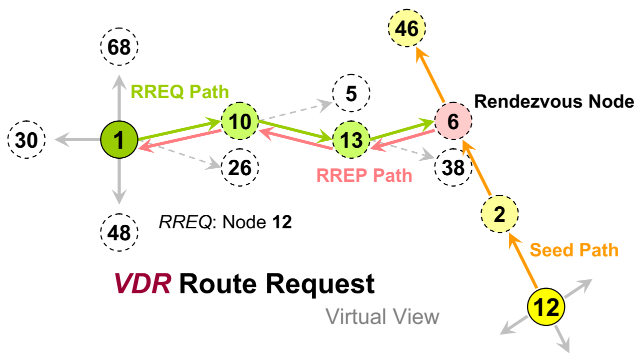

Fig. 5. VDR Dissemination and Route Request Virtual View

the number of hops to the search destination and send it in the reverse direction, relying on routing table entries of the reverse path to get back to the source. Under network churn, if a node in the reverse path is no longer active, VDR will re-select a node in the same virtual direction that has the closest hashed ID match to the original source of the SREQ packet to forward to. This ensures a globally consistent biasing of the packets toward the intended destination despite path breakages due to network churn. The algorithm for route queries is similar to Algorithm 1 except that instead of hashing the packet source to Ψ, the packet query destination is hashed to Ψ.

Algorithm 2 VDR Find Closest Hashed Neighbor

```
FindClosestHashedNeighbor($j$)
1:  // Finds neighbor in virtual direction w/ closest ID match
2:  $\gamma_{min} \leftarrow$ 0xFFFF
3:  // Get Each Neighbor in Virtual Interface
4:  for all $k \in$ Neighbor List($j$) do
5:      $\Theta \leftarrow SHA1(k)$
6:      // Check Hash Distance
7:      $\gamma \leftarrow abs(\Theta - \Psi)$
8:      if $\gamma \leq \gamma_{min}$ then
9:          $\gamma_{min} \leftarrow \gamma$
10:         // Store Send Next Hop
11:         $\Phi \leftarrow k$
12:     end if
13: end for
14: Return($\Phi$)
```

D. Path Deviation

There are instances when nodes wishing to forward in a specific interface find that no neighbors are assigned that virtual interface. VDR employs a strategy to correct for path deviations in an attempt to maintain virtual straight lines. The strategy is fairly straight-forward and employs an angle correction method based on encoding a *multiplier* in the header based on the number of interfaces deviated from the intended send direction. More information can be found in [11].

III. Performance Evaluation

In this section, we provide performance evaluations of VDR under various parameters and against some basic random-walk techniques and flooding techniques. The simulations were performed using PeerSim [13] under a cycle-driven model. We wire our topology such that each node has a K out-degree. Because links are bidirectional, it is expected that each node has an average of $2K$ one hop neighbors. Although internet topology is power-law (many nodes have few connections while some nodes have a large number of neighbors), we can assume this topology because 1) peer-to-peer systems are overlay networks and connections are often virtual, 2) 1 hop neighbors can be physical one hop neighbors or links with the lowest latency to the source, and 3) peer-to-peer systems represent a subset of the whole network and small-world examples show relatively flat topologies [2], [3].

The performance metrics evaluated include *reachability*, *path stretch vs. shortest path*, and network-wide *state distribution*. We examine these metrics under conditions of varying seed and search *TTL* and *strategies*, *average number of immediate neighbors*, *number of virtual interfaces*, and *percentage of network churns*. All simulations were averaged over 10 runs under random topologies and 95% confidence intervals were mapped. Unless otherwise stated, 1000 randomly generated source and destination queries were generated to start somewhere between cycle 30 to 100. Table I outlines our default simulation parameters.

TABLE I
Default Simulation Parameters

Parameter	Values
Number of Nodes	50,000
Number of Virtual Interfaces	8
Simulation Cycles	150
Churn percentage	0% - 50% every 5 cycles
Seed/Search TTL	10 - 100 hops
Seed Entry Expiry	10 Cycles (in churn environments)
Interface Assignment	VDR Hash, VDR Hash w/ NB Shift
Seed/Search Strategy	VDR, VDR-Random, Random Walk
Number of Queries	1000 Randomly Generated

Interface assignment refers to the strategy used to assign neighbors to virtual interfaces. The techniques used consist of the standard VDR hash strategy as described in Section II and a modified heuristic that attempts to evenly distribute the neighbors to each interface (VDR w/ NB Dist). The purpose behind this is to make sure one interface doesn't have a lot of neighbor assignments while the others have none.

The search and seed strategies used include VDR, VDR-Random (VDR-R), and Random Walk Routing (RWR). VDR is the exact strategy described in Section II while VDR-Random (VDR-R) utilizes the same node to interface assignment technique, but randomizes the node forwarding in a specific direction. In short, if a virtual interface has multiple nodes assigned to it, VDR-Random will choose a random neighbor associated with that interface rather than choose the neighbor with the hash closest in distance from the source node (for seed packets) or query-search node (for search packets). The random walk strategy is not a pure random walk but its built around the same concept. For the random walk strategy, 4 "walkers" are used with each source node seeding and search for information by sending out 4 random neighbors. Each of the walkers are essentially random walk packets and are dropped after a certain TTL.

A. Evaluation of VDR in Churn-less Environments

In this section, we examine the effect of search and seed packet TTL, number of virtual interfaces, and average number of neighbors per node on reachability, path stretch, and state distribution under the three seed strategies as listed above (VDR, VDR-Random, and Random Walk Routing) in a fixed, no churn environment. Each node utilizes 8 virtual interfaces with out-degree k assigned to 10. Because links are bi-directional, this means that each node has an average of about 20 neighbors with the deviation from the average to be quite small.

For all cases, seed information is sent only once and the expiry time for each entry is set to the number of simulation cycles as we assume that the network is not dynamic and continual send is redundant. This is also important because under the random walk search (RWR) technique, continual sending of the seed packets lead to different neighbors chosen each time leading to huge confusion in path choices (essentially, all nodes in the network would know a source after a set time if the expiry was set high).

1) Effect of Seed and Search TTL: In this subsection, we examine the effect of search and seed packet TTL on reachability, path stretch, and state distribution under three seed and search strategies. We expect that VDR should provide higher connectivity and lower path stretch than the other strategies (VDR-Random and Random Walk) under lessened seed/search TTL simply because it biases the packets toward a specific ID. Figures 6-8 give our results.

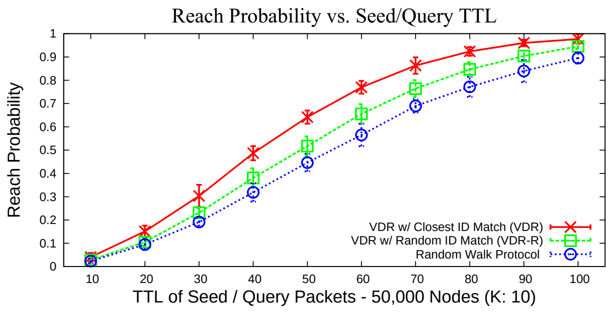

Fig. 6. VDR-R achieves better reachability within less TTL in comparison to RWR. Additional consistency reinforcement with closest ID match (VDR) improves the reachability further.

It can be seen in figure 8 that VDR is able to find information with a higher success rate with less search and seed TTL. This is beneficial because lower TTL lowers the amount of packets traveling network-wide and frees up the links for actual traffic. It is interesting to note that even with a TTL of 100, VDR achieves almost 100% reachability in a network of 50,000 nodes. The random walk search (RWR) technique, as expected, converged the slowest, requiring a much higher TTL to even come close to VDR. The reason that RWR even comes close to VDR is because of the fixed network environment. Under network churns, however, state maintenance would grow dramatically simply because seed dissemination would no longer be sent to the same nodes.

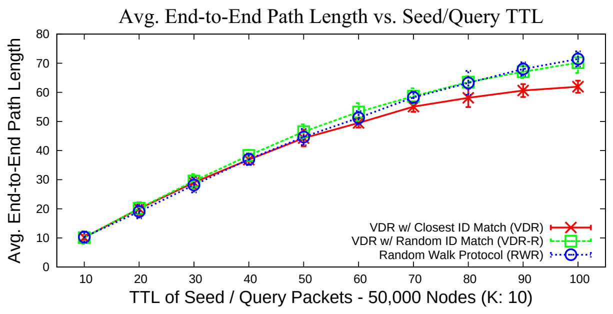

Fig. 7. VDR finds information seeds with 25% less hops than random walk.

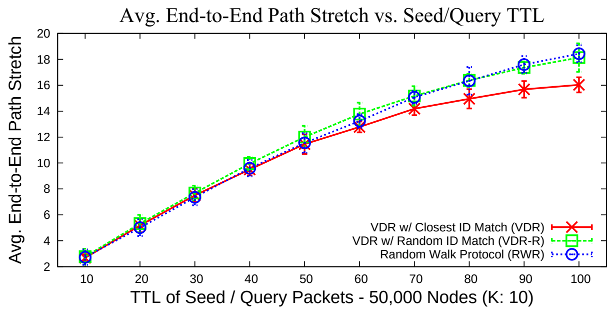

Fig. 8. In VDR, path stretch from source to actual data (destination) is roughly 15% less than with random walk.

We see from figure 7 that the path from the source to a seed node is also much shorter in VDR. Again, this is due to sent packets being biased toward the ID with the closest match. Path stretch (figure 8) shows similar results. It is interesting to note the substantially high number of hops traversed through VDR, VDR-Random, and RWR as compared to shortest path. The shortest path in a wired network grows on order of $Log(N)$ where N is the number of packets in the network. Therefore, it is expected that with 50,000 nodes in the network, the shortest path should be roughly 4.7 hops. It makes sense that these path lengths increase with increased TTL because source and destination pairs that are now farther away can be reached and so the average path length increases with increased reach.

2) Effect of Number of Virtual Interfaces: In this section, we examine the effect of modifying the number of virtual interfaces on reach probability, end-to-end path stretch, and number of states maintained network-wide. With finer granularity (more virtual interfaces), it is expected that the difference between VDR and VDR-R will become smaller because the randomness in neighbor selection for each interface will be reduced as there would only be 1 neighbor per interface. In our simulations, we ran a 50,000 node network with each node having an average of 20 neighbors. Figures 9-10 show our results for simulating VDR and VDR-R with a search/seed TTL of 50 and 100.

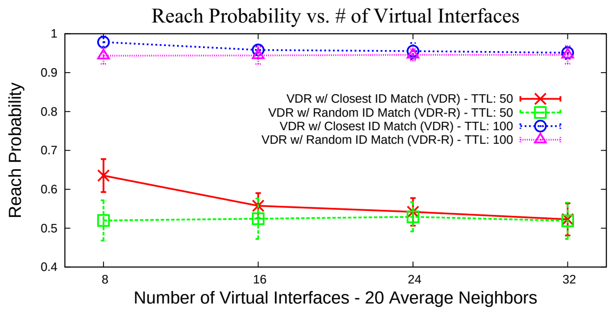

Fig. 9. When the number of interfaces are much less than the number of neighbors, the biasing effects of VDR are more pronounced leading to higher reach.

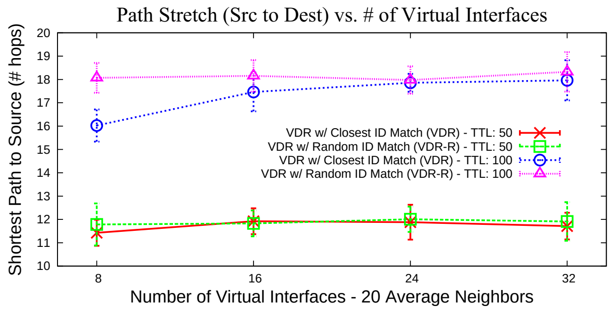

Fig. 10. Biasing effects are greater seen when there are more neighbors assigned to a virtual interface, generating shorter paths.

As shown in figure 9, VDR has much higher reach probability with lower number of virtual interfaces. This is due to the biasing of IDs such that there is a better convergence. The results are more pronounced at lower seed/search TTL simply because there isn't a saturation of states. The closer VDR gets to 100% reach, the less TTL will affect the packet reach probability resulting in less difference in reach. One of the reasons for greater reach is the lowered path length required for VDR as compared to VDR-R. This again, is due to the biasing of packet IDs. It is interesting that the lower the TTL, the lower the path stretch observed. This is because there is a smaller fraction of delivery success and only the paths that succeed (the shorter ones) are measured.

3) Effect of Number Neighbors: In overlay networks, neighbor nodes are often assigned randomly based on the latency from a specific node rather than physical links. Because of this flexibility in neighbor assignment, it becomes interesting to examine how increasing the number of neighbors per node affects reach, path stretch, and state distribution in networks utilizing VDR, VDR-Random, and RWR.

In these simulations, we fix the virtual interfaces to 8 and increase the k constant (the number of out-degrees) from 5 (average of 10 neighbors/node) to 20 (average of 40 neighbors/node). Because as k is increased, a greater number of neighbors will be assigned to each interface, it is expected that the biasing effect in VDR will yield much more beneficial results over VDR-Random for larger k values. As the k is increased, we also expect to observe increased path stretch under lower search/seed TTL simply because the number of nodes in the network are fixed and if each node has more neighbors, paths to each node is inherently shorter (lower shortest path yielding higher path stretch). One would also expect higher reach with increased k because end-to-end paths to all nodes are essentially shorter.

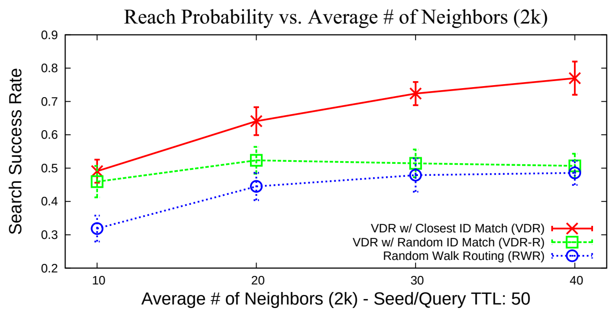

Fig. 11. VDR has higher reachability than VDR-R and RWR with increased neighbors because it and search/seed TTL of 50 hops because of biasing packets toward the query destination.

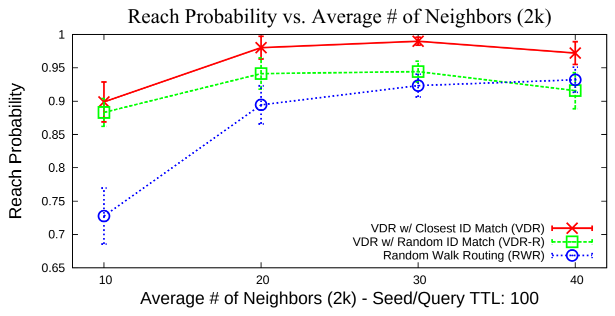

Fig. 12. VDR has higher reachability than VDR-R and RWR with increased neighbors because it and search/seed TTL of 100 hops because of biasing packets toward the query destination.

Figures 11 and 12 show our results for reachability/search success while increasing k for each of the search and seed strategies at 50 and 100 TTL. It can be seen that with VDR, as the number of neighbors increase, higher reach occurs. Under the same conditions, we see that VDR-Random and RWR yield significantly *less* reach than VDR. Comparing VDR to VDR-Random, we see that as the number of neighbors increase, VDR-Random reach remains relatively constant. This is due in part to the forwarding mechanism found in VDR-Random. In VDR-Random, although the number of neighbors (and thus the number of neighbors assigned to each interface) increases, its decision-making strategy is still to choose a random neighbor in a specific virtual interface direction.

The assignment of nodes to a virtual interface negatively impacts the options available to send and therefore the gains by simply having more neighbors (and thurs shorter end-to-end paths) are offset by the losses due to assigning neighbors to rigid virtual interfaces. Because VDR-Random still randomly chooses nodes in a specific interface direction, this results in a relatively constant reach even under increased k.

Figures 13 and 14 show the results for end to end path stretch while increasing k for a query and seed TTL of 50

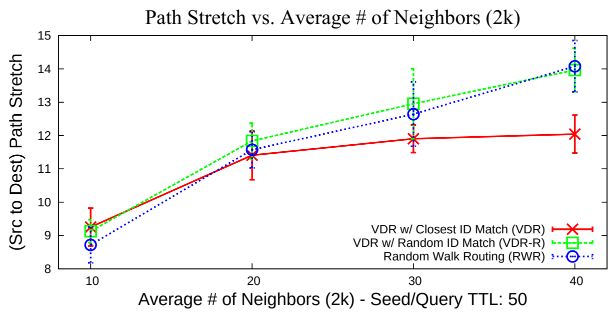

Fig. 13. Path stretch increases with more neighbors because in a network of fixed number of nodes, with more connections to and from each node, the average end to end shortest path decreases.

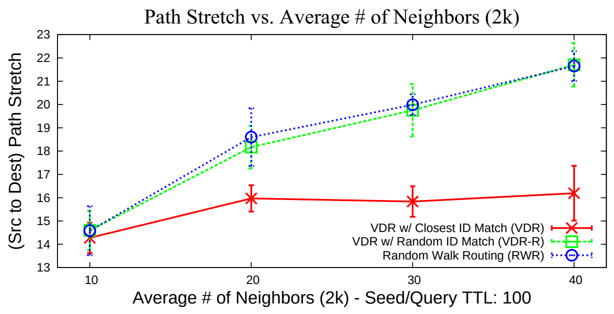

Fig. 14. Path stretch increases with more neighbors because in a network of fixed number of nodes, with more connections to and from each node, the average end to end shortest path decreases.

and 100. It's interesting that overall, the path stretch increases with increased number of neighbors. This makes sense because paths chosen are less efficient due to the greater number of neighbors assigned to each interface. Comparatively, however, VDR still yields only slightly shorter path stretch than VDR-Random and RWR with increased number of neighbors. This is due to the biasing effect of forwarding.

4) Evaluation of State Distribution: Its interesting to examine how evenly the state is spread network-wide because in flat topologies, even distribution suggests no single point of failure. Because VDR is essentially a biased random-walk technique, it is expected that state is fairly evenly distributed throughout the network. To simulate state distribution, we generated a fixed overlay network with an average of 20 neighbors each. Keeping this overlay network fixed, we ran the simulation 10 times with varying initial *virtual orientations* and took snapshots of the state throughout the simulation, averaging the state per node for each run over all 10 runs. A histogram of the frequency of a average states maintained is shown in figure 15.

As figure 15 shows, the average states maintained is less evenly distributed in VDR compared to VDR-R and RWR. This suggests that some nodes have more information than other nodes. We suspect this is due to certain nodes with hashed IDs closer to the average being chosen as an appropriate "next hop" more than the other nodes.

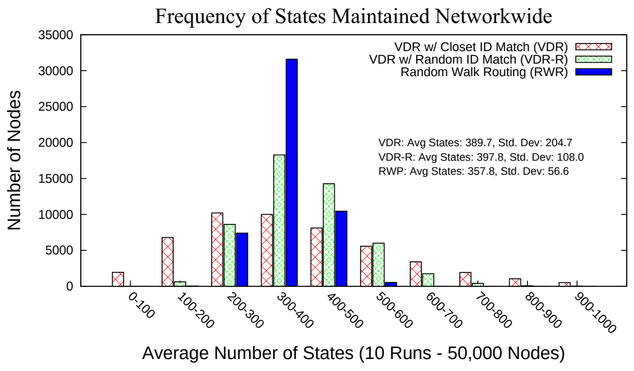

Fig. 15. VDR has high state distribution deviation suggesting an uneven distribution of state networkwide.

5) Evaluation of Network Load Distribution: It has been shown that network congestion can be controlled and limited by routing packets using two-phase routing algorithms [15] [14]. Current overlay measure route cost through hop count and at times, load. In high-traffic networks, by choosing the shortest path, nodes with many connections will become saturated with packets. Busch et al. [15] has shown that by drawing a perpendicular bisector between source and destination and forwarding packets from source to a random point on the perpendicular bisector which in-turn forwards to destination when that point is reached, load can be balanced across the network. In much the same way, VDR inherently implements a seemingly two-phase routing algorithm because it provides rendezvous abstractions whereby the source sends search packets until it rendezvous with seed information. As a result, it is interesting to see the distribution of load network-wide.

In this subsection, we measure network load by taking snapshots of queue lengths of 50,000 nodes at specific intervals in time. Essentially, we fix the wiring of the overlay network with the only variable for each run being the *virtual orientation*. We then run the simulation 10 times and average out the instantaneous queue lengths per snapshot per simulation run for *each* node. By understanding the variation from the mean number of packets in the queues per node, we can see how evenly distributed the load is across the network.

Figure 16 shows the histogram for the number of nodes with queue sizes in the intervals given. It can be seen that there is greater spread of load using VDR compared to VDR-R and RWR suggesting that some nodes incur heavier load than other nodes. This is to be expected because VDR chooses shorter paths and constrains neighbor sending to virtual interfaces. As can be seen, RWR performs the best because random walk models are known to distribute load fairly evenly.

B. Evaluation of VDR in Dynamic Environments

In this section, we examine the effect of network churn on reachability, end-to-end path stretch, overall network load and state distribution under the three seed/search strategies as listed above (VDR, VDR-Random, and Random Walk). We simulate churn in the following manner: First, all nodes are

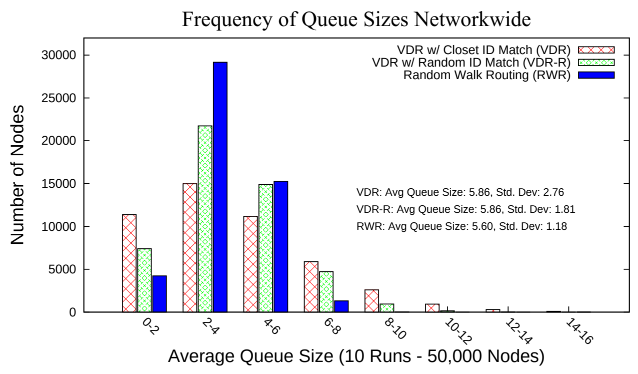

Fig. 16. VDR has high queue size distribution deviation suggesting an uneven distribution of load networkwide.

connected by assigning an average of k out nodes from each node. Because the links are bi-directional, each node generally has roughly $2k$ neighbors. We then "turn off" half the nodes in the network probabilistically essentially dropping the average number of neighbors to k. The inactive nodes now serve as "raw material" for new connections and nodes currently in the original set can be either turned off or on per simulation cycle.

For our simulations, we fix the number of nodes active to be a constant at half the total available nodes and every 5 cycles, randomly activate a percentage of nodes with respect to the active nodes and deactivate the same number of nodes randomly. When nodes are deactivated, all the packets in their incoming queue are dropped and routing tables emptied. When they are activated, the connections that were originally formed with neighbor nodes remain the same. Thus, nodes can be active and inactive at any point in the simulation and have essentially maintain the same state.

The simulator keeps track of all the nodes that have *ever* been active and queries are generated based on any node that has *ever* been active. This makes sense as in an overlay network, resources that have never been allocated will never be able to be found. Expiry time for each routing entry is set 10 cycles which is the same as the seed/announcement packet send interval. As per the VDR algorithm, search queries are sent out virtual orthogonal directions until they intersect a node with a path to the destination in their routing table. When this occurs, a search reply packet is generated and sent in the reverse path. In the event of reverse path nodes no longer being up, a node in the same virtual direction is chosen with an ID closest in match to the source of the search query (the destination of the search reply). Under RWR, another node is randomly chosen. In our scenarios, we simulated 25,000 active nodes with a total pool of 50,000 nodes under various churn percentages. The TTL of the seed/announcement packets was set to 150 and each node contained an average of 20 one-hop neighbors.

1) Effect of Churn on Search Success: In this subsection, we examined how the percentage of network churns affect search success. We consider a successful search to have occurred when a search query is initiated and it receives a search reply. It's expected that VDR outperforms VDR-R and RWR simply because it orders neighbors into a more structured fashion with virtual interface assignments. With the RWR, four "walkers" are sent out to random neighbors in search for seed information planted by four seed "walkers". These seed packets are sent out periodically to different neighbors so while at some point there might be more state network-wide, the expiry of the routing information removes stale routes quickly. Our results are shown in figures 17 and 18.

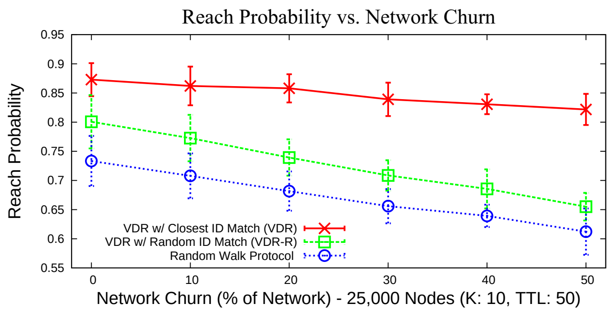

Fig. 17. VDR maintains much higher reachability than VDR-R and RWR with increased percentage of network churn. It also much more robust to network churn, dropping only 5% reach for 50 seed/search TTL compared to VDR-R and RWR which dropped 12-15% going from 0% to 50% network churn for a seed/search TTL of 50.

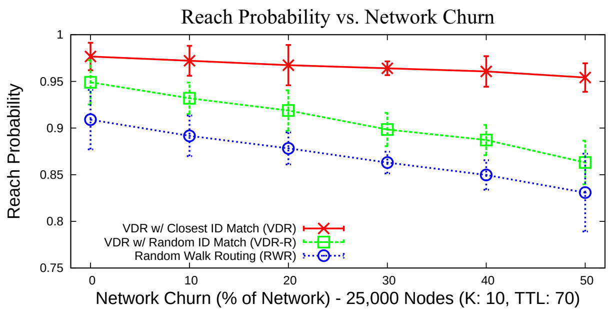

Fig. 18. VDR maintains much higher reachability than VDR-R and RWR with increased percentage of network churn. It also much more robust to network churn, dropping only 2% reach for 70 seed/search TTL compared to VDR-R and RWR which dropped 7-8% going from 0% to 50% network churn for a seed/search TTL of 70.

As figures 17 and 18 show, VDR has the highest percentage of search success/reach under the same network churn rate compared to VDR-R and RWR. It outperforms VDR-R because of the biasing effect of the neighbor send. Because each node has about 15 neighbors and 8 virtual interfaces, there is a possibility that if a neighbor is down (or swapped), VDR will choose another neighbor that is atleast biased toward the search query source (search reply destination) whereas VDR-R will simply randomly choose a node. VDR outperforms RWR simply because in sending search replies, if a previous hop is no longer available, then it must randomly choose a neighbor to forward. If it was forced to perform a random walk until it reached the search query source, it would most definitely result in a packet loss the majority of the times. However, because the random walk need only intersect a node with a path in its routing table to the search query source, there is still relatively high reach (∼81% even for 50% network churn with a search/seed TTL of 70).

It is also important to understand the rate at which search success/reach drops with respect to the percentage of churns. As can be seen from figures 17 and 18, VDR drops only 5% in reach from 0% to 50% network churn for a search/seed TTL of 50 and only 2% for a search/seed TTL of 70. This is important because even with 50% nodes turning off and new ones being added, there is still a high degree of reach and robustness to search. VDR-R and RWR, on the other hand, drops about 12-15% in reach for 50 TTL and 7-8% in reach for 70 TTL simply because of the random nature of their send as described before: if a search query reply packet finds the next hop inactive, it must retrace its path without any kind of "hints".

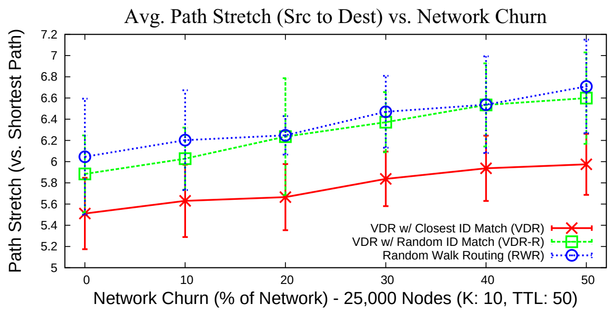

Fig. 19. VDR performs with the shortest amount of path stretch as compared to VDR-R and RWR because of a consistent virtual direction and biasing effect of packets.

2) Effect of Churn on Path Stretch: Figure 19 shows our result on path stretch under churn for VDR, VDR-R, and RWR with a search/seed TTL of 50. It can be seen that as network churn increases, the path stretch increases. This is consistent with expectations as when reverse paths are not reachable, new neighbors must be chosen resulting in often longer paths.

What is interesting, however, is that VDR actually generates much lower path stretch compared to VDR-R and RWR despite the fact that VDR-R and RWR have lower search success and reach. In general, paths with shorter hops are less affected by network churn and therefore it is expected that with lower reach/search success, the paths are generally shorter in general. We see therefore, that VDR not only provides higher path reach, but that it also finds shorter paths.

3) Effect of Churn on Network Load: It has been shown that network congestion can be controlled and limited by routing packets using two-phase routing algorithms [15] [14]. Current overlay measure route cost through hop count and at times, load. In high-traffic networks, by choosing the shortest path, nodes with many connections will become saturated with packets. Busch et al. [15] has shown that by drawing a perpendicular bisector between source and destination and forwarding packets from source to a random point on the perpendicular bisector which in-turn forwards to destination when that point is reached, load can be balanced across the network. In much the same way, VDR inherently implements a seemingly two-phase routing algorithm because it provides rendezvous abstractions whereby the source sends search packets until it rendezvous with seed information. As a result, it is interesting to see the distribution of load network-wide.

In this subsection, we measure network load by taking snapshots of queue lengths of nodes that are active at specific intervals in time. What we measure is the deviation from the average queue length (in our case, 19.7 average packets per node). A higher deviation means that certain nodes have more packets on average than other nodes. One possible issue with measuring instantaneous queue sizes and averaging it over several snapshots network wide is that it does not take into consideration nodes that have just become active compared to nodes that have been active for an extended amount of time. We balance this issue out by taking the RWR as the base case simply because under random walk strategies, it is known that state is relatively well distributed. Figure 20 shows our results for a seed/search TTL of 50.

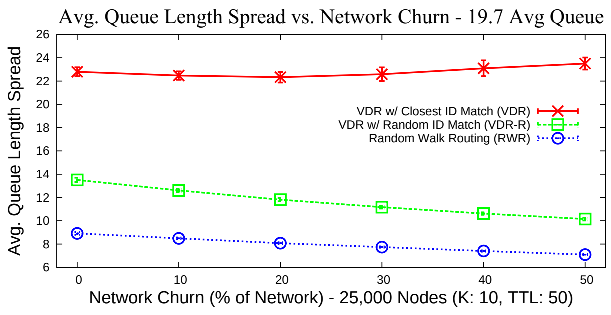

Fig. 20. Effect of network churn percentage on VDR queue size distribution. VDR has a large spread in queue length distribution suggesting that load is not evenly balanced network-wide (about .65X more than VDR-R and 1X more than RWR).

It can be seen that as expected, VDR has the highest deviation from the average of 14.33 average packets in the queue per node. This is expected as again, certain nodes have preferred paths due to closer ID matching. VDR-R has a higher deviation than RWR because it also makes the routing more rigid by mapping neighbor nodes to virtual interfaces. We see that the queue length spread per node for VDR is about .65X more than VDR-R and 1X more than RWR. As network churn increases, more queues are emptied per churn interval resulting in fewer packets per node in the queues overall.

4) Effect of Churn on State Distribution: Similar to subsection III-B.3, it is interesting to understand the state distribution network-wide. Comparing VDR to the other techniques, we see that in general, the states network-wide are fairly consistent at about 171 average states per node. It is interesting, therefore, to understand how those states are distributed to see whether or there are more single points of failure with using VDR as compared to the other strategies. The same issues with network churn apply as in subsection III-B.3 and we likewise address this issue by comparing to the baseline random walk technique (RWR) since it is well known that random walk techniques distribute state fairly evenly. Figure 21 shows our results for a search/seed TTL of 50.

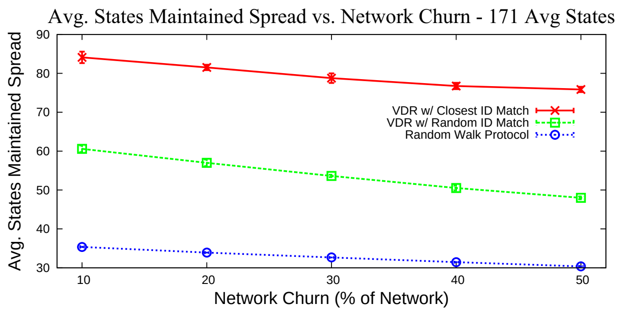

Fig. 21. Effect of network churn percentage on VDR state distribution. VDR showed the highest spread in states maintained network-wide suggesting that certain nodes have more states maintained than other nodes (about .56X higher than VDR-R and 1.5X higher than RWR).)

As figure 21 depicts, VDR has the highest deviation from the average states maintained per node. This suggest that state is not very evenly distributed network-wide. However, when compared to RWR (the baseline), we see that VDR state spread is only about 1.5X higher than RWR. This again, is due to VDR biasing random walks such that sending favors certain nodes (ie: announcement packets generally travel through the same neighbors).

C. *Summary of VDR Performance Evaluations*

Below we summarize our findings in performance evaluations for VDR:

- VDR reaches 3.5% more nodes than VDR-R and 9% more nodes than our modified random walk routing strategy (RWR).
- VDR-R produces the same reach and path stretch results with increasing number of virtual interfaces. This is due to the randomization of sends. VDR, however, increases reach with fewer number of virtual interfaces because of its biasing technique. The gains disappear if the number of neighbors is smaller than the number of interfaces.
- Increasing the number of neighbors generally increased reach and end-to-end path stretch. This was probably due to more node choices per neighbor to bias information.
- VDR states and queues/load are not well distributed.
- VDR shows a 3-4X reach retention rate going from 0% to 50% network churn compared to VDR-R and RWR, showing itself to be much more robust to network churn.
- VDR, even under churn, does not spread state or load evenly.
- VDR paths exhibit high path stretch compared to shortest path but good path stretch compared to pure random walk techniques.

IV. Conclusion

In this paper, we presented Virtual Direction Routing (VDR), a scalable overlay network routing protocol that uses the concept of *virtual directions* to efficiently perform node information dissemination and lookup. State information is disseminated to nodes along *virtual orthogonal lines* originating from each node and periodically updated. When a path lookup is initiated, instead of flooding the network, query packets are also forwarded along *virtual orthogonal lines* until an intersection with the seeded state occurs. Both seed and search packets are biased toward the originator ID and search ID respectively. We show that in a small-world, unstructured, flat topology, VDR provides high reach even with low seed/search TTL ($\sim$98% reach for a TTL of 100 for a 50,000 node network) and that VDR is robust to churn (dropping only 2% in reach going from 0% to 50% network churn). We also show that VDR scales well without imposing DHT-like graph structures (eg: trees, rings, torus, coordinate-space, etc.) and the path stretch compared to random-walk protocols (the traditional method to route in unstructured overlay networks) is very good. VDR trades off the gains by not having a very even distribution of state. This is due to the biasing of state dissemination packets such that certain neighbors consistently receive state information while others do not. Path choices are also suboptimal compared to flooding techniques due to the two phased-biased random-walk nature of VDR.

References

[1] C. Ghantsidis, M. Mihail, and A. Saberi. "Hybrid search schemes for unstructured peer-to-peer networks." Proceedings of IEEE INFOCOM, 2005.
[2] A. Iamnitchi, M. Ripeanu, and I. Foster. "Small-world filesharing communities." Proceedings of IEEE INFOCOM, 2004.
[3] J. Kleinberg. "Navigation in a small world." Nature, page 845, 2000.
[4] A. Kumar, J. Xu, and E. W. Zegura. "Efficient and scalable query routing for unstructured peer-to-peer networks." Proceedings of IEEE INFOCOM, 2005.
[5] H. Guclu and M. Yuksel. "Scale-Free Overlay Topologies with Hard Cutoffs for Unstructured Peer-to-Peer Networks." Proceedings of IEEE ICDCS, 2007.
[6] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Frans Kaashoek, F. Dabek, and H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications." IEEE Transactions on Networking, February 2003.
[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. "A Scalable Content-Addressable Network." Proceedings of ACM SIGCOMM 2001.
[8] Gnutella home page. http://gnutella.wego.com.
[9] Kazaa home page. http://www.kazaa.com.
[10] SHA-1 RFC. http://tools.ietf.org/html/rfc3174.
[11] B. Cheng, M. Yuksel, S. Kalyanaraman, "Orthgonal Rendezvous Routing Protocol for Wireless Mesh Networks," Proceedings of ICNP 2006.
[12] M. Mitzenmacher, "Compressed bloom filters," IEEE/ACM Transactions on Networking, vol. 10, no 5, pp. 604-612, 2002.
[13] PeerSim: A Peer-to-Peer Simulator. http://peersim.sourceforge.net/.
[14] M. Kodialam, T. V. Lakshman, and Sudipta Sengupta, "Efficient and Robust Routing of Highly Variable Traffic", Third Workshop on Hot Topics in Networks (HotNets-III), San Diego (USA), November 2004.
[15] C. Busch, M. Ismail, and J. Xi, "Oblivious Routing on Geometric Networks", Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp 316-324, Las Vegas, Nevada, July 2005.
[16] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, "Making Gnutella-like P2P Systems Scalable", Proceedings of SIGCOMM 2003, August 2003.
[17] W. Terpstra, J. Kangasharju, C. Leng, A. Buchmann, "BubbleStorm: Resilient, Prbabilistic, and Exhaustive Peer-to-Peer Search", Proceedings of SIGCOMM 2007, August 2007.