Title

APDV: Making Distance Vector Routing Scale Using Adaptive Publish-Subscribe Mechanisms

Authors

Garcia-Luna-Aceves, J.J.
Li, Q.

Publication Date

2013

Peer reviewed

APDV: Making Distance Vector Routing Scale Using Adaptive Publish-Subscribe Mechanisms

Qian Li*
*Computer Engineering Department
University of California, Santa Cruz
Santa Cruz, CA 95064
liqian@soe.ucsc.edu

J.J. Garcia-Luna-Aceves*†
†Palo Alto Research Center
Palo Alto, CA 94304
jj@soe.ucsc.edu

ABSTRACT

The Adaptive Publish-subscribe Distance Vector (APDV) protocol is introduced as an example of a new approach to allowing distance-vector routing to scale by integrating it with adaptive publish-subscribe mechanisms. APDV combines establishing routes to well-known controllers using distance-vector signaling with publish-subscribe mechanisms. The latter allow destinations to publish their presence with subsets of controllers, and sources to obtain routes to intended destinations from those same controllers. Controllers are selected dynamically using a fault-tolerant distributed election algorithm to ensure that each non-controller node is covered by at least a given number of controllers within a few hops. Extensive simulation experiments are used to compare APDV with AODV and OLSR, which are representative protocols for on-demand and proactive routing. The results show that APDV achieves significantly better data delivery, attains comparable delays for delivered packets, and incurs orders of magnitude less control overhead than AODV and OLSR, even under heavy data loads.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*

Keywords

Publish-subscribe; routing; service discovery; distance vectors; hash table; distributed algorithm; MANET.

1. INTRODUCTION

Traditional routing protocols for wireless ad hoc networks rely on the network-wide dissemination of signaling packets stating either proactive updates to the state of links or distances to destinations, or on-demand requests for routes to destinations. However, as the number of network nodes, connectivity changes, and new traffic flows increase, both approaches tend to incur excessive signaling overhead, and the problem is even worse for the case of mobile ad hoc networks (MANET). Section 2 summarizes the prior work aimed at making routing more scalable in ad hoc networks. This review indicates that routing protocols for ad hoc networks are such that the signaling needed to maintain destination-based routing tables up to date works independently of the functionality needed to map the names of destinations to either their locations or routes to them.

Although considerable work has been reported on service discovery in ad hoc networks, the solutions to date either operate on top of a routing infrastructure, or augment one of the existing routing protocols to support service discovery. Interestingly, none of the prior solutions to make routing more scalable integrate destination-based routing with adaptive publish-subscribe mechanisms in a way that reduces the signaling required for both routing and service discovery.

The main contribution of this paper is presented in Section 3 and consists of introducing a new approach to routing in wireless ad hoc networks based on publish-subscribe mechanisms that is far more scalable than traditional on-demand or proactive routing. The *Adaptive Publish-subscribe Distance Vector* (APDV) protocol is presented as an example of this approach.

APDV integrates three components: (a) electing a subset of nodes to serve as controllers that maintain routes to nearby destinations, (b) maintaining routes to all known controllers using distance vectors, and (c) using publish-subscribe mechanisms with which destinations inform controllers of routes to them and sources obtain routes to destinations.

Section 4 describes the results of several simulation experiments used to compare the performance of APDV to that of AODV [16] and OLSR [9], which are representatives of traditional on-demand and proactive routing, respectively. The three protocols are compared using static and dynamic topologies. The impact that network size, traffic load, multiple access interference, and mobility have on performance are examined using packet-delivery ratio, average end-to-end delay, and signaling overhead as the performance metrics. The results from these experiments show that APDV incurs orders of magnitude less signaling overhead than AODV and OLSR while attaining similar or better packet delivery ratios and average delays.

MSWiM'13, November 3–8, 2013, Barcelona, Spain.

http://dx.doi.org/10.1145/2507924.2507925.

2. RELATED WORK

Prior routing approaches assume that the mapping of destination names to addresses or routes is done independently of routing, and includes using hierarchies, limiting the dissemination of control messages, distributed hash tables (DHT), Bloom filters, virtual or geographical coordinates, or sets of dominating nodes to reduce the size of routing tables or the amount of route signaling.

Hierarchical routing schemes organize nodes into clusters (e.g., [2], [12], [15], [20]) and some reduce signaling of clustering schemes by limiting propagation of control messages based on their distance from an originating point (e.g., HSLS [17]). The limitations of these approaches are that the affiliation of nodes to clusters is easily broken when nodes move, and re-establishing such affiliations incurs considerable overhead, and incorrect routes can result in schemes in which signaling decays based on the distance to the links.

DHT-based schemes (e.g., [14], [18], [23]) are attractive because a DHT grows only logarithmically with the number of intended destinations. However, typical DHT schemes define a virtual topology, and substantial signaling overhead can be incurred to maintain the links of virtual topologies defined in large MANETs. AIR [5] avoids the use of virtual topologies, but requires the use of variable-length prefix labels instead of addresses. Another approach consists of hashing node identifiers of destinations into Bloom filters, which are then used in routing updates [1]; however, such schemes suffer from the existence of false positives, in which case nodes incur considerable overhead.

Routing protocols that use geographical coordinates for routing (e.g., GPSR [11]) are limited by the requirement to have GPS services and still incur signaling overhead discovering the geo-locations of destinations. A number of schemes use virtual coordinates consisting of the distances of nodes to a few reference nodes (e.g., [4], [24]). The main limitation of this type of virtual coordinates is that multiple nodes may be assigned the same virtual coordinates, and there is no inherent uniqueness to a specific vector of distances to beacons. This results in either incorrect routing or the use of additional signaling (typically flooding) aimed at resolving false positives.

There are many proposals attempting to reduce the number of relays that need to forward signaling messages for a given number of destinations. The best known example of this approach is the use of multipoint relays in OLSR [9]. The main limitation of these proposals is that they call for the establishment and maintenance of connected dominating sets, i.e., the nodes selected to forward signaling messages must form a connected subgraph. This tends to require a large subset of nodes, especially in dynamic topologies.

There is a large body of work on resource and service discovery in ad hoc networks [21]. What is striking about this prior work is that all proposals either assume that names are mapped to addresses and routing to those addresses is then done independently (e.g., ADNS [8]), or augment existing routing protocols with service discovery functionality (e.g., LSD [13], AODV-SD [6], L+ [3]). The use of a common hash function to select controllers in APDV is similar to the approaches used in ADNS and L+. However, ADNS and L+ operates on top of the routing protocol assuming that directory sites or landmarks have been selected, while APDV integrates routing with the selection of controllers and the mapping of names to routes.

3. ADAPTIVE PUBLISH-SUBSCRIBE DISTANCE VECTOR (APDV)

3.1 Overview

APDV assumes that each network node is assigned a network-wide unique node identifier, and takes advantage of the broadcast nature of radio links. First, a subset of nodes are selected dynamically to serve as *controllers*. A controller acts as a directory server by maintaining the routes to destinations nearby, with destinations being denoted by their node identifiers. The distributed algorithm used to select controllers ensures that each non-controller node is within a maximum distance r from a minimum number k of *local controllers* for the node, and as a side effect informs each node about the routes to its one- and two-hop neighbors.

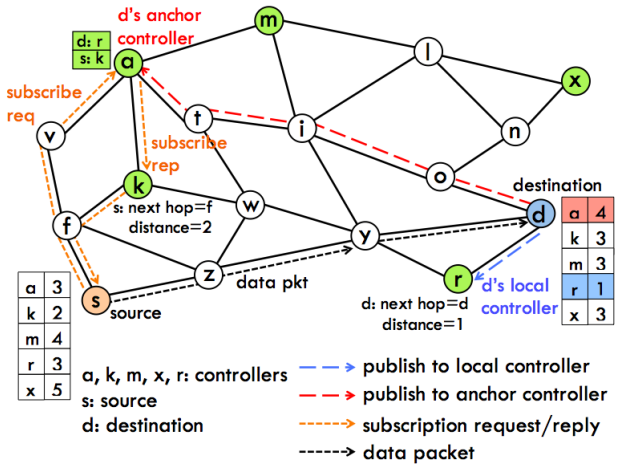

Figure 1: Example of APDV operation

Fig. 1 illustrates the basic operation of APDV assuming that each node has at least one local controller within two hops. In the example of Fig. 1, nodes *a, k, m, x* and *r* are the elected controllers of the network.

All network nodes maintain routes to all controllers using a loop-free distance-vector routing algorithm. For simplicity, we assume in this paper that a node maintains a single route to each controller, but APDV can be extended to provide multiple loop-free paths. Each node contacts each of its local controllers to publish its presence. To do this, node d sends a publish message to each of its local controllers with the mapping $(d, \{l_d^1, ..., l_d^k\})$, where l_d^i $(1 \leq i \leq k)$ is a local controller for node d. Each local controller l_d^i of node d and each relay between d and the controller receiving the publish request from d stores a tuple stating d, the next hop to d, and $\{l_d^1, ..., l_d^k\}$. In addition, node d uses a common hash function to select an *anchor controller* a_d that should store the mapping $(d, \{l_d^1, ..., l_d^k\})$, and sends such a mapping to controller a_d. The relay nodes between d and a_d can cache the mapping information. In the example of Fig. 1, node d has published its presence with its local controller (node r) and its anchor controller (node a); note that controller r has a route to node d while controller a has a route to controller r and the mapping (d, r).

A source requiring a route to destination d uses the same common hash function on the identifier of node d to find the anchor controller for d, a_d, and sends a subscription re-

quest to controller a_d stating d and $(s, \{l_s^1, ..., l_s^k\})$, where l_s^i $(1 \leq i \leq k)$ is a local controller for node s. In turn, controller a_d responds with the mapping $(d, \{l_d^1, ..., l_d^k\})$ and sends that response towards the nearest local controller for source s selected from the set $\{l_s^1, ..., l_s^k\}$. The answer is redirected to source s by either the selected controller l_s^j or the first relay node along the route from a_d to controller k with a route to s. Node s can then send data packets to destination d by sending them towards the nearest controller in the set $\{l_d^1, ..., l_d^k\}$. Those packets will be redirected to d after either reaching the selected controller in $\{l_d^1, ..., l_d^k\}$ or a node along the route from s to the selected controller with an active route to d. In the example of Fig. 1, node s subscribes to node d by contacting anchor a, which maintains the mapping (d, r) and returns it to node s by sending its response towards node k, which is the local controller of node s. Node a also caches the mapping (s, k). Node s sends data packets to d by sending them towards controller r; however, node y is in the route from s to r and also has a route to d, and forwards the data packets directly to d.

The rest of this section presents important details regarding the information and mechanisms used in APDV for selecting controllers, maintaining routes to controllers, and publishing and subscribing to destinations using controllers.

3.2 Information Stored and Exchanged

The information maintained at each node allows the node to select and route to controllers, route to local destinations, and learn the local controllers associated with distant destinations on demand. Node i maintains a *controller table* (CT^i) stating information about all controllers elected in the network; a *neighbor controller table* (NCT^i) stating information reported by each neighbor of node i regarding all controllers elected in the network; a *neighbor table* (NT^i) stating information about all one- and two-hop neighbors of node i; a *neighbor local routing table* $(NLRT^i)$ stating routing information reported by each neighbor regarding *all* destinations within two hops and *some* destinations within r hops; a *local routing table* (LRT^i) stating routing information about *all* destinations within two hops and *some* destinations within r hops; a *neighbor routing table* (NRT^i) stating information reported by each neighbor regarding distant destinations; and a *routing table* (RT^i) stating information about distant destinations.

APDV employs soft-state to operate efficiently in dynamic networks, and a node transmits its HELLOs periodically every 3 seconds and the HELLO includes some or all the updates made to node's tables. A node stores all the information from the HELLOs it receives from its neighbors, and also caches information it receives in subscription or publication requests from neighbors. Entries in RT^i and NRT^i are populated by the publish-subscribe signaling described subsequently. NT^i, CT^i, NCT^i, LCL^i, and $NLRT^i$, LRT^i, are updated by the exchange of HELLOs among one-hop neighbors.

For each controller c selected in the network, CT^i specifies: the identifier of node c (nid_c^i); the distance from i to c (d_c^i); the successors (next hops) from i to c (s_c^i); and a sequence number (sn_c^i) used to avoid routing loops. NCT^i stores the controller tables reported by each neighbor of node i. The entry for controller c reported by neighbor j and stored in NCT^i is denoted by $\{nid_{cj}^i, d_{cj}^i, sn_{cj}^i\}$.

For each neighbor j of node i, NT^i specifies: the identifier of the node (nid_j^i); a sequence number (sn_j^i) created by j and used to determine that the entry is the most recent from node j; a *controller status flag* (cs_j^i) stating whether or not node j is a controller; the *controller counter* (k_j^i) stating the number of controllers within r hops of node j; and the *local controller list* (LCL_j^i) consisting of the identifiers of all controllers within r hops of node j. An entry for neighbor v in NT^j sent in a HELLO to node i is denoted by $\{nid_v^j, sn_v^j, cs_v^j, k_v^j, LCL_v^j\}$, and the same entry stored in NT^i is denoted $\{nid_{vj}^i, sn_{vj}^i, cs_{vj}^i, k_{vj}^i, LCL_{vj}^i\}$.

An entry for destination j listed in LRT^i specifies: the identifier of the node (nid_j^i); a sequence number (sn_j^i) created by j used to avoid routing loops; the distance from i to j (d_j^i); the successor in the route to j (s_j^i); and the local controller list of node j (LCL_j^i), which may be a link to NT^i if the node is within two hops. An update made by neighbor j to LRT^j communicated in a HELLO is denoted by $\{nid_v^j, sn_v^j, d_v^j, LCL_v^j\}$, and the corresponding entry stored at node i in $NLRT^i$ is denoted by $\{nid_{vj}^i, sn_{vj}^i, d_{vj}^i, LCL_{vj}^i\}$. An entry for destination v listed in RT^i simply specifies the identifier of the node (nid_v^i) and the list of local controllers for node v (LCL_v^i), because node i maintains the routes to all controllers in CT^i.

Node i includes its own information in NT^i, i.e., it stores an entry corresponding to nid_i^i, and uses the information in its HELLOs. A HELLO from node i contains: nid_i^i, sn_i^i, cs_i^i, k_i^i, and updates to NT^i, CT^i, and LRT^i. An update to NT^i regarding neighbor j consists of the tuple $\{nid_j^i, sn_j^i, cs_j^i, k_j^i, LCL_j^i\}$. An update to CT^i regarding controller c consists of the tuple $\{nid_c^i, d_c^i, sn_c^i\}$. An update to LRT^i regarding destination v consists of the tuple $\{nid_v^i, sn_v^i, d_v^i, LCL_v^i\}$.

3.3 Selecting and Routing to Controllers

The distributed selection of controllers in APDV amounts to selecting a dominating set C of nodes in the network that serve as controllers, such that every node $u \notin C$ (called *simple node*) is at a distance smaller than or equal to r hops from at least k nodes in C (called *controllers*). A node u is said to be (k, r) dominated (or *covered*) if there are at least k nodes in C within r hops from u. There is a large body of work on dominating sets in graphs [7], and many distributed algorithms exist to approximate minimum connected dominating sets (MCDS) with constraints (e.g., [10]). However, the controller selection scheme in APDV is simply aimed at obtaining a set of controllers that cover all nodes but need not be a MCDS, and maintaining routes to all selected controllers. It is based on HELLO messages exchanged among one-hop neighbors. To keep the selection algorithm and signaling simple, only distances to controllers and node identifiers are used as the basis for the selection of controllers.

3.3.1 Selecting Controllers

The only way to add or delete controllers in the network is for nodes to self-select themselves to become controllers or stop being controllers. A given node i determines to add or delete its own entry in CT^i, respectively, according to the Controller Addition Rule (CAR) and Controller Deletion Rule (CDR) defined below.

Node i is initialized with $CT^i = \phi$ and $NT^i = \phi$, and waits for a few seconds to start receiving HELLOs from nearby nodes. Hence, according to CAR, node i will select itself as a controller when it is first initialized, unless it has received HELLOs from neighbors that prompt it not to include itself as a controller based on CDR. Node i updates an entry for $j \neq i \in CT^i$ according to the rules described in the next subsection, which ensure that no loops are formed for routes to controllers. Once node i has updated NT^i and CT^i by processing the HELLOs it receives from neighbors, it computes its local controller list (LCL^i) from CT^i, such that $v \in LCL^i$ if $d_v^i \leq r$, and sets $k_i^i = |LCL^i|$.

CAR (Controller Addition Rule):
Node i adds itself to CT^i if
$(k_i^i < k) \wedge [\ i = Min\{nid_j^i\ \forall j \in NT^i\ |\ (j \notin CT^i) \wedge (k_j^i < k)\}\ ]$

CDR (Controller Deletion Rule):
Node i deletes itself from CT^i if
$(k_i^i > k) \wedge [\ \forall\ j \in NT^i\ [\ (\ |LCL_j^i - \{i\}| \geq k\ \forall j \notin LCL_i^i\ ) \wedge$
$(nid_j^i < nid_i^i\ \forall j \in LCL_i^i)\ ]\ ]$

Fig. 2 shows an example of controller selection in APDV in a network of six nodes, assuming that each node must be covered by one controller ($k = 1$) within two hops ($r = 2$). For simplicity, the example assumes that HELLO transmissions are synchronized. The figure shows the local controller list (LCL) at each node, and each new state per node in the figure is determined by the reception of HELLOs from all neighbors, followed by the addition or deletion of controllers in the LCL resulting from applying CAR and CDR, deleting controllers that are farther than 2 hops away, or deleting a controller after the successor to that controller sends an update with a deletion of the controller (see RCR and UCR below).

As Fig. 2(a) shows, in this example each node selects itself as a controller after initialization following CAR and sends a HELLO, because nodes do not wait for HELLOs to arrive before using CAR. Fig. 2(b) shows that, after receiving a HELLO from each neighbor, a node may add new controllers reported in the HELLOs, but may delete itself from being a controller based on CDR, which is the case for nodes 30, 50 and 90. Figs. 2(c) to 2(f) illustrate the deletion of a controller entry from LCL at a given node when the successor towards the controller sends a HELLO that deletes the controller, which is the case of node 20 deleting controller 90 and node 18 deleting controllers 30 and 50 in Fig. 2(d), or nodes 30 and 18 deleting controller 20 in Fig. 2(f), for example. Fig. 2(g) illustrates the fact that nodes include in their LCLs only those controllers within $r = 2$ hops, which is the case of node 90 not including controller 10 in its LCL. Fig. 2(g) shows the final state of the LCLs for the example network.

3.3.2 Routing to Controllers

For simplicity of presentation, in this paper we assume that each node maintains a single route to each controller selected in the network using the updates to controller tables included in HELLOs.

APDV uses a distance-vector routing approach to maintain routes to controllers. To guarantee loop-free routes, APDV uses sequence numbers that restrict the selection of next hops towards a given controller by any node, such that

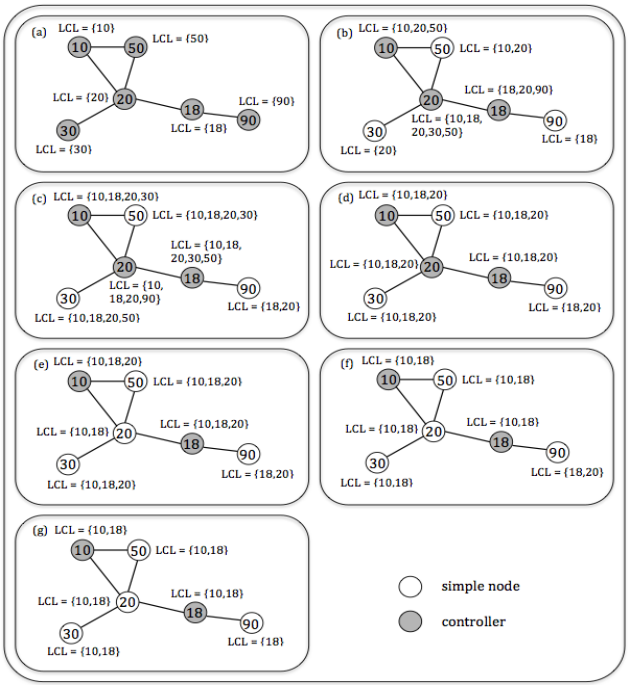

Figure 2: Example of controller selection in six-node network with $k = 1$ and $r = 2$.

only those neighbors with shorter distances to the controller or with a more recent sequence number reported by the controller can be considered as successors. An important aspect of APDV is that entries for controllers can be deleted on purpose as a result of CDR, rather than only as rare occurrences due to failures or network partitions. Together with the transmission of periodic HELLOs, the Reset Controller Rule (RCR) and the Update Controller Rule (UCR) discussed below address this functionality.

Let N^i be the set of one-hop neighbors of node i. Node i updates CT_j^i as a result of HELLOs from neighbor $j \in N^i$ or the loss of connectivity to neighbor j. If node i loses connectivity to node j, the entries in CT_j^i are deleted. Once node i is selected as a controller, it is the only node that can change the sequence number for its own entry in controller-table updates sent in HELLOs.

When a given node i decides to delete itself as a controller based on CDR, its entry must be deleted in the rest of the network. To accomplish this, node i uses RCR to set its self-entry with an infinite distance and an up-to-date sequence number for a finite period of time T, before deleting its self-entry from CT^i to ensure that the rest of the nodes delete the entry for i in their controller tables. If node i receives a HELLO from j or experiences a link failure that makes it update CT_j^i for entry $c \neq i : \{nid_{cj}^i,\ d_{cj}^i,\ sn_{cj}^i\}$, node i updates its entry for c in CT^i according to UCR, which forces node i to propagate a reset update or to select a successor to controller c that is either closer to c or has reported a more recent sequence number from c.

RCR (Reset Controller Rule):
If node i must delete itself from CT^i using CDR then
set $d_i^i = \infty$; $sn_i^i = sn_i^i + 1$; $reset\text{-}timer^i = T$

UCR (Update Controller Rule):

If ($\exists q \in NT^i \mid sn^i_{cq} > sn^i_c$)
  then begin
    if ($(v = s^i_c) \wedge (sn^i_{cv} > sn^i_c) \wedge (d^i_{cv} = \infty)$)
      then set $sn^i_c = sn^i_{cv}$; $d^i_c = \infty$
      else begin
        set $d^i_c = Min\{d^i_{cf} + 1 \mid (f \in NT^i) \wedge (sn^i_{cf} = Max\{sn^i_{cv} \mid v \in NT^i)\}\}$
        set $s^i_c = j \mid (j \in NT^i) \wedge (d^i_{cj} = d^i_c - 1)$
        set $sn^i_c = Max\{sn^i_{cv} \mid v \in NT^i\}$;
  else begin
    set $d^i_c = Min\{d^i_{cf} + 1 \mid (f \in NT^i) \wedge (sn^i_{cf} = sn^i_c) \wedge (d^i_{cf} < d^i_c)\}$;
    set $s^i_c = j \mid (j \in NT^i) \wedge (d^i_{cj} = d^i_c - 1)$

3.4 Publish-Subscribe Mechanisms for Name-to-Route Resolution

Nodes learn about routes to controllers and to one- and two-hop neighbors, but have no routes to destinations many hops away as a result of the exchange of HELLOs.

To allow sources to obtain routes to arbitrary destinations without incurring network-wide dissemination of signaling messages, APDV uses a publish-subscribe mechanism for name-to-route resolution. The use of consistent hashing in APDV is similar to recent proposals for distributed name resolution in MANETs (e.g., ADNS [8]) that use consistent hashing to map the names of destinations to one of several predefined directory sites storing the name-to-address mapping for destinations. The key differences between APDV and this prior work are that: (a) the directories (i.e., controllers) are selected dynamically; (b) a node publishes its presence with multiple controllers; and (c) name resolution is integrated with the selection of and routing to controllers, rather than running on top of routing. Hence, in APDV, controllers maintain name-to-route mappings, rather than storing name-to-address mappings and then using an underlying routing protocol to obtain the routes for known addresses.

For simplicity, we describe the publish-subscribe mechanisms in APDV assuming that node identifiers constitute the names for which routes must be found. However, it should be noted that the same publish-subscribe mechanisms in APDV are applicable to support information-centric networking, such that nodes publish and subscribe to names of destinations, content or services, rather than just node identifiers.

3.4.1 Publishing Destinations

Publishing in APDV consists of having a local controller know the route to a given destination or having an anchor controller know the mapping from a node identifier to a list of local controllers. Subscribing in APDV consists of a node requesting a way to reach a named destination through an anchor controller.

In APDV, node i publishes itself with the k controllers listed in LCL^i, and with one or more anchor controllers. The local controllers in LCL^i are within r hops of node i and serve as the "landmarks" for other nodes to submit data to node i, given that nodes far away from node i do not have routes to node i. Accordingly, a local controller for node i must maintain a route to node i, and it also maintains the mapping (i, LCL^i), so that it can find alternate ways to reach node i if its route to i fails. The anchor controllers are needed for nodes far away from destinations to obtain the mappings between the identifiers of those destinations and their local controllers. For simplicity, in this paper we assume that a single anchor controller is used for any one node.

The anchor controller for node i (denoted by a_i) is obtained by using a network-wide consistent hash function that maps the identifier of node i into the identifier of one of the controllers selected in the network. Controller a_i must store the mapping (i, LCL^i), so that it can provide any node v far away from node i the list LCL^i, with which node v can send data packets towards the local controller in LCL^i that is nearest to node v according to its controller table CT^v.

The forwarding of a publication request from a node to its local controllers is done by the exchange of HELLOs. Given that nodes maintain loop-free routes to all controllers, publication requests directed to local controllers of nodes are forwarded over the reverse loop-free routes already established from local controllers to nodes. The routes maintained by local controllers to nearby nodes are refreshed periodically; each node creates a new publication request by increasing the sequence number included in the LRT self-entry of its own HELLO. If node i receives a HELLO from neighbor j with a publication request originated by node v, which consists of update to LRT^j for destination v ($\{nid^j_v, sn^j_v, d^j_v, LCL^j_v\}$), then node i forwards the request (i.e., it includes the LRT^i entry $\{nid^i_v, sn^i_v, d^i_v, LCL^i_v\}$ in its own HELLO) if it is the successor for node j to any of the controllers listed in LCL^j_v. Once a local controller c receives an entry for destination v and $c \in LCL^v$, then c publishes (i.e., stores) the entry $\{nid^c_v, sn^c_v, d^c_v, s^c_v, LCL^c_v\}$, where s^c_v is the neighbor from which it received the publication request. Controller c may also forward it if it is the successor to another controller in LCL^v for the neighbor from which it received the publication request.

The submission of a publication request from node i to its anchor controller a_i is done by node i using the network-wide consistent hash function on the set of identifiers in CT^i to obtain $hash(i) = a_i$, where $a_i \in CT^i$. After that, node i sends a publication request to its successor towards its anchor controller a_i with the tuple $\{nid^i_i, sn^i_i, d^i_i, LCL^i_i\}$. Each node v in the route from node i to controller a_i forwards the publication request towards a_i and caches the tuple $\{nid^v_i, sn^v_i, d^v_i, s^v_i, LCL^v_i\}$. Once controller a_i receives the request, it stores the tuple $\{nid^{a_i}_i, sn^{a_i}_i, d^{a_i}_i, s^{a_i}_i, LCL^{a_i}_i\}$. Hence, each node processing a publication request learns the route to the node issuing the request, and the anchor controller is able to obtain the mapping needed to redirect nodes sending subscription requests to the local controllers of node i.

3.4.2 Subscribing and Routing to Destinations

The forwarding of subscription requests is handled in much the same way described above for the case of publication requests. When node o has data for destination $j \notin CT^o$, it computes $hash(j) = a_j$, where $a_j \in CT^o$ and sends its subscription request towards a_j. The subscription request from node o regarding destination j states the identifier of node j, its anchor controller a_j, and LCL^o. When a_j receives o's request, it responds with the tuple $\{nid^{a_j}_j, sn^{a_j}_j, LCL^{a_j}_j\}$ and sends the response to the nearest controller it finds in LCL^o. Node o stores the tuple $\{nid^o_j, sn^o_j, LCL^o_j\}$ in RT^o upon receiving the reply to its subscription. Data packets from o are then sent towards the controllers in LCL^o_j

that are the closest to node o. A data packet must specify the sender, the destination, and the selected local controller of the destination. This can be done by encapsulating the header of the packet stating the origin and the destination with a header stating the origin and the selected local controller of the destination. Once the packet reaches a relay node y with an active route for the destination, the packet is forwarded directly to the destination itself, as long as the distance from node y to the destination is at most r hops.

4. PERFORMANCE COMPARISON

We used QualNet [19] (version 5.0) as the discrete event simulator to compare the performance of APDV with the performance of AODV and OLSR, which are representative protocols for the traditional on-demand and proactive routing schemes used in ad hoc networks. We use packet delivery ratio, end-to-end delay, control overhead as our performance metrics. The control overhead is the average number of control packets generated by the routing protocols. We evaluated the three protocols in static and mobile networks. In a static network, nodes are uniformly distributed in the network to avoid disconnected nodes. The random waypoint model was chosen as the mobility model for mobile networks. The routing protocols are tested using the IEEE 802.11 DCF as the underlying MAC protocol, and all signaling packets are sent in broadcast mode. Data sources produced a constant bit rate (CBR) traffic at a rate of 10 packets per second. The three protocols use the same time period to refresh their routing structures. For APDV we used $k = 2$ and $r = 3$ to select controllers. Each simulation ran for 10 different seed values. Unless otherwise stated, the simulation environment details are listed in Table 1.

Table 1: Simulation Environment

MAC Protocol	802.11	Simulation time	300s
Data source	CBR	Data rate	10 packets/s
Pkts. per flow	500	Flow duration	50s
Static Network:			
Total nodes	100-400	Node placement	Uniform
Mobile Network:			
Total nodes	100	Network size	1800 x 1800 m^2
Mobility model	Random waypoint	Pause time	10s
Min.-Max. Vel.	1-10m/s		

4.1 Results for Static Networks

4.1.1 Impact of Increasing Network Size

To evaluate the impact of network size on the performance of the protocols we used static topologies and an ideal physical layer with low data rates per sources in order to limit the impact of Multiple Access Interference (MAI) on the observed performance [22]. The number of nodes is increased from 100 to 400. To avoid having denser networks as the number of nodes is increased, the simulation area is increased proportionally to the number of nodes, so that node density is similar in all cases. The end result is that the 802.11 MAC protocol experiences perfect capture; hence, when multiple packets are received concurrently at a receiver, the receiver decodes one of them successfully. Because traffic load is kept small, MAI due to data traffic is minimum, but the effect of MAI becomes a factor when signaling traffic increases. To exercise the signaling of all protocols, each data flow lasts 50 seconds, and the total number of concurrent data flows is the same in the network at any time. CBR flows are established among randomly selected nodes, and each CBR source generates a total of 500 data packets of 256 B at a rate of 10 packets per second. To avoid bias of traffic load when the network size is changed, we used 5% and 10% of the total number of nodes for the sources with concurrent data flows. The results show that APDV scales much better than AODV and OLSR in every aspect.

The results for 5% of concurrent data flows are shown in Fig. 3(a)-(c). Fig. 3(a) shows that all protocols attain high delivery ratios when the network size is increased subject; however, APDV attains the highest delivery ratios. Fig. 3(b) shows the average control overhead induced by the protocols. APDV incurs the smallest and contrasts with the overhead induced by OLSR, which experiences a steep increase for 400 nodes. APDV incurs limited and fairly constant control overhead because only unicast publish-subscribe requests to anchor controllers are sent other than HELLOs. OLSR erroneously interprets the loss of control packets due to collisions as topology changes, triggering new topology control messages that are diffused in the network and generate more congestion. Fig. 3(c) shows that APDV attains similar end-to-end delays of delivered packets when the protocols have the same delivery ratios. APDV shows slightly higher delays than AODV for 400 nodes, which is mostly due to the fact that APDV delivers more packets than AODV, but in some cases packets may take routes slightly longer than with AODV. OLSR's longer delays are due to the queueing of packets waiting for signaling packets to be sent.

Fig. 4(a)-(c) show the results for 10% traffic load. All three protocols perform similarly up to 200 nodes. Fig. 4(a) shows that APDV scales much better than AODV and OLSR. For 400 nodes, APDV is capable of delivering close to 40% more packets than AODV and 25% more packets than OLSR. The reason behind this behavior can be observed in Fig. 4(b), which presents the average control overhead induced by the protocols. The figure shows that the control overhead incurred by APDV remains constant as the number of nodes increases, while AODV and OLSR incur more overhead as the number of nodes increases. The control overhead in OLSR is a function of the number of nodes in the network. By contrast, Figs. 3(b) and 4(b) show that the control overhead in AODV depends on the traffic load, the heavy traffic load and continuous arrival of new flows forces AODV to constantly flood the network with route requests. Under high congestion, many control packets are lost due to collisions, which is interpreted by AODV as broken links that need to be repaired, and hence nodes react to these packet losses by generating even more route requests, which congest the network even more. Fig. 4(c) shows the end-to-end delays attained by delivered data packets. We observe that APDV performs similar to or better than the other two protocols for all network sizes.

4.1.2 Impact of Increasing Number of Flows

In this scenario we increase the number of concurrent CBR sources from 10% to 40% of the 100 nodes in the static network using a real physical layer. Nodes are uniformly distributed in a simulation area of 1800×1800 m^2. The same CBR flow scheme described in Section 4.1.1 is used. The results are shown in Fig. 5(a)-(c). Fig. 5(a) shows that APDV and AODV attain similar packet delivery for 10 and

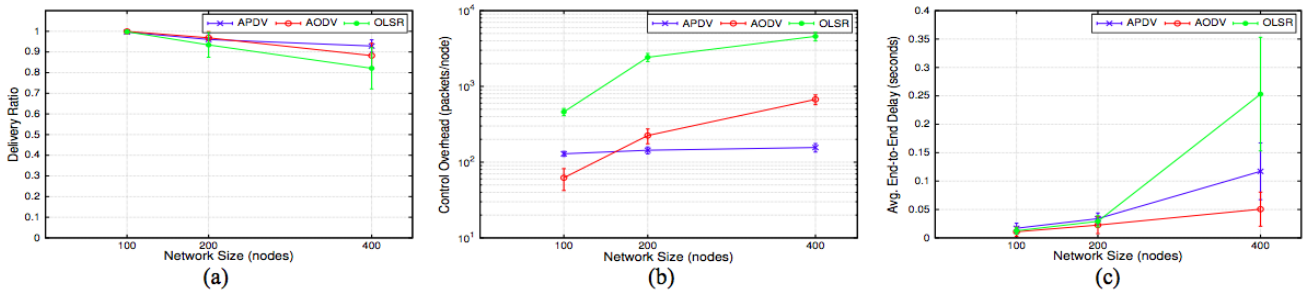

Figure 3: Static network with increasing network size, ideal PHY, and 5% data load: (a) Delivery ratio, (b) average number of control packets sent per node, (c) end-to-end delay of delivered data packets.

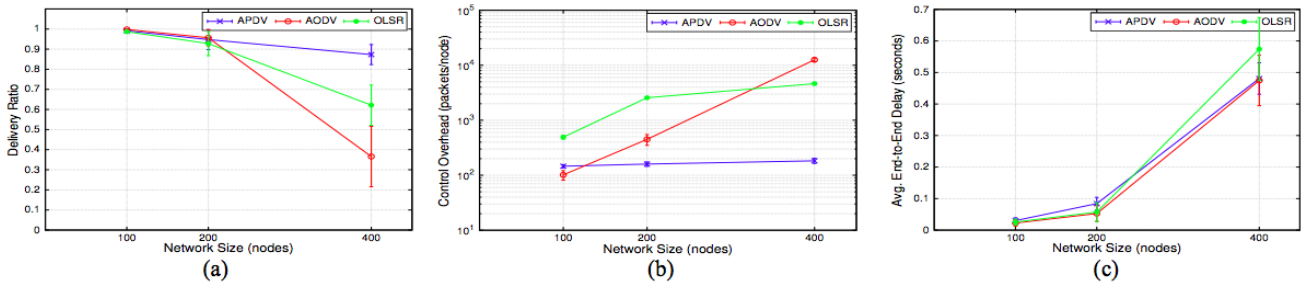

Figure 4: Static network with increasing network size, ideal PHY, and 10% data load: (a) Delivery ratio, (b) average number of control packets sent per node, (c) end-to-end delay of delivered data packets.

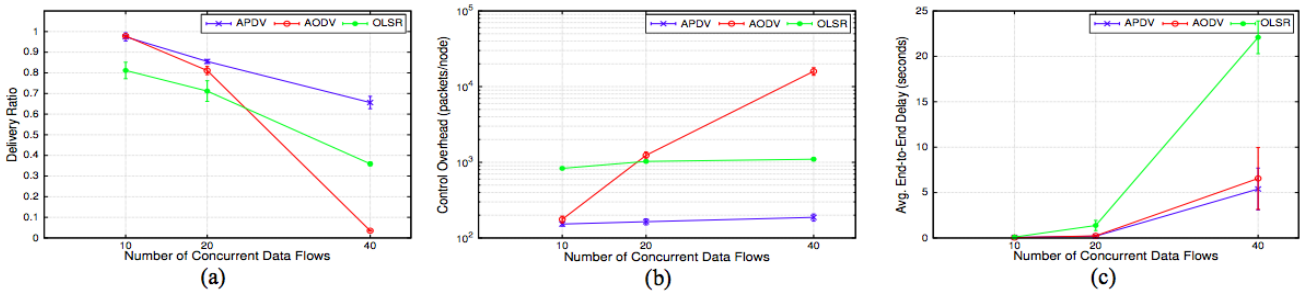

Figure 5: Static 100-node network with increasing number of CBR flows, and a real PHY: (a) Delivery ratio, (b) average number of control packets sent per node, (c) end-to-end delay of delivered data packets.

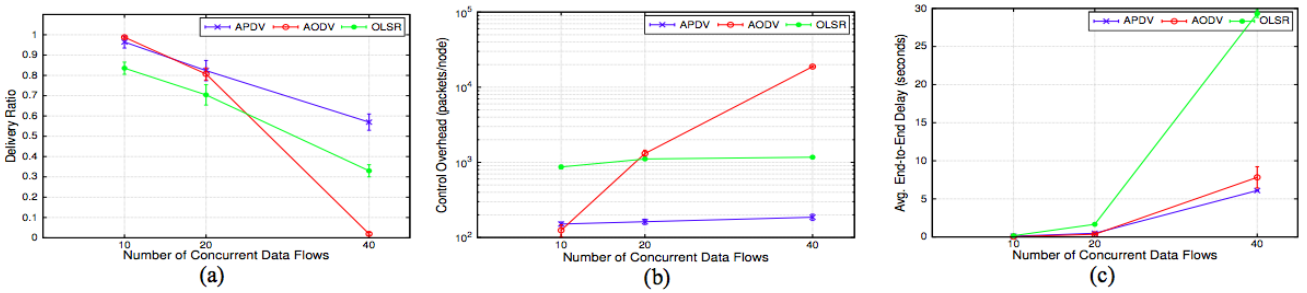

Figure 6: Mobile network with increasing number of CBR flows and real PHY: (a) Delivery ratio, (b) average number of control packets sent per node, (c) end-to-end delay of delivered data packets.

20 concurrent flows, and that APDV is consistently better than OLSR. APDV attains far better packet delivery than AODV and OLSR for 40 flows; APDV delivers close to 30% more packets than OLSR for 40 flows, and AODV can deliver less than 5% of the packets! Fig. 5(b) shows the control overhead induced by the protocols. APDV incurs very limited and fairly constant control overhead, which contrasts with the overhead incurred by AODV for 40 flows. Even though OLSR is able to keep the control overhead almost constant, it is still much higher than in APDV–almost an order of magnitude–even for a small number of data flows. Fig. 5(c) shows that APDV attains the smallest average end-to-end delays of delivered packets for all traffic loads, and that delays in OLSR increase dramatically for 40 data flows, which is a consequence of data packets being queued waiting for signaling packets to be sent. The delays in AODV for 40 flows appear to be better than for OLSR, but these delays are only for 5% of the packets transmitted, compared to more than 60% of the packets in APDV and more than 30% of the packets in OLSR.

4.2 Results for Mobile Networks

In this scenario we evaluate the performance of the protocols in mobile 100-node networks using a real physical layer, with the number of concurrent CBR sources increasing from 10% to 40% of the number of nodes. The simulation environment is described in Table 1. Fig. 6(a)-(c) show the results of these experiments. Fig. 6(a) shows that APDV consistently outperforms AODV and OLSR. Under heavy traffic load, APDV scales better than AODV and OLSR and is capable of delivering close to 30% more packets than OLSR for 40 flows, while AODV is ineffective and delivers less than 3% of the packets. Fig. 6(b) shows that APDV incurs limited and fairly constant signaling overhead in mobile networks, which helps to illustrate the fact that its use of controllers and adaptive publish-subscribe mechanisms is very well suited for MANETs. By contrast, AODV's overhead explodes for 40 flows, which is the reason why data packets cannot be delivered to destinations. Fig. 6(c) shows that APDV attains the smallest end-to-end delays of delivered data packets, which is a consequence of incurring limited signaling overhead that lets data packets flow faster to their intended destinations. By contrast, delays in OLSR explode for 40 flows, because data packets must wait in queues while signaling packets are transmitted.

5. CONCLUSION

We introduced the *Adaptive Publish-subscribe Distance Vector* (APDV) protocol to provide scalable routing in ad hoc networks using distance vectors by integrating routing with the selection of controllers serving as directories, and name-to-route resolution based on publish-subscribe mechanisms. We used simulation experiments to compare its performance with the performance of AODV and OLSR. APDV achieves significantly better data delivery, attains comparable delays for delivered packets, and incurs substantially less control overhead than AODV and OLSR, because it substitutes network-wide dissemination of link states or distances to destinations with publish-subscribe signaling with controllers.

More work is needed to fully exploit the approach advocated in APDV. One important aspect is the use of a hierarchy of controllers to allow destinations to be known locally, regionally or globally. The performance impact of different values of k and r, and the use of multi-path routing and load balancing among multiple loop-free paths should also be explored.

6. ACKNOWLEDGMENTS

This work was supported in part by the Baskin Chair of Computer Engineering at UCSC, and by the US Army Research Office under Grant SUB0700155/SC20070363.

7. REFERENCES

[1] P. S. Almeida, C. Baquero, N. M. Preguiça, and D. Hutchison. Scalable bloom filters. *Inf. Process. Lett.*, 101(6):255–261, 2007.

[2] J. Behrens and J. J. Garcia-Luna-Aceves. Hierarchical routing using link vectors. In *INFOCOM*, 1998.

[3] B. Chen and R. Morris. Scalable landmark routing and address lookup for multi-hop wireless networks, 2002.

[4] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. E. Culler, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *NSDI*, 2005.

[5] J. J. Garcia-Luna-Aceves and D. Sampath. Scalable integrated routing using prefix labels and distributed hash tables for manets. In *IEEE MASS 2009*, 2009.

[6] J. A. García-Macías and D. A. Torres. Service discovery in mobile ad hoc networks: Better at the network layer? In *ICPP Workshops*, pages 452–457, 2005.

[7] T. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, 1998.

[8] X. Hong, J. Liu, R. Smith, and Y.-Z. Lee. Distributed Naming System for Mobile Ad Hoc Network. In *ICWN*, pages 509–515, 2005.

[9] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. Optimized Link State Routing Protocol for Ad Hoc Networks. In *IEEE INMIC 2001.*, pages 62–68, 2001.

[10] D. S. Joshi, S. Radhakrishnan, and C. Narayanan. A fast algorithm for generalized network location problems. In *ACM SIGAPP*, 1993.

[11] B. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *ACM MobiCom*, pages 243–254, 2000.

[12] L. Kleinrock and F. Kamoun. Hierarchical Routing for Large Networks; Performance Evaluation and Optimization. *Computer Networks*, 1:155–174, 1977.

[13] L. Li and L. Lamont. A lightweight service discovery mechanism for mobile ad hoc pervasive environment using cross-layer design. In *IEEE PerCom Workshops*, 2005.

[14] Q. Li and J. J. Garcia-Luna-Aceves. Opportunistic Routing Using Prefix Ordering and Self-Reported Social Groups. In *ICNC Workshop on Computing, Networking and Communication*, 2013.

[15] Y. Mao, F. Wang, L. Qiu, S. S. Lam, and J. M. Smith. S4: Small State and Small Stretch Routing Protocol for Large Wireless Sensor Networks. In *NSDI*, 2007.

[16] C. E. Perkins and E. M. Belding-Royer. Ad-hoc On-Demand Distance Vector Routing. In *WMCSA*, pages 90–100, 1999.

[17] C. A. Santiváñez, R. Ramanathan, and I. Stavrakakis. Making link-state routing scale for ad hoc networks. In *ACM MobiHoc*, 2001.

[18] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy. Scalable routing on flat names. In *ACM CoNEXT*.

[19] S. N. Technologies. Qualnet. http://web.scalable-networks.com/content/qualnet.

[20] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *SIGCOMM*, 1988.

[21] C. N. Ververidis and G. C. Polyzos. Service discovery for mobile ad hoc networks: a survey of issues and techniques. *Commun. Surveys Tuts.*, 10(3):30–45, 2008.

[22] Y. Wang and J. J. Garcia-Luna-Aceves. Collision avoidance in multi-hop ad hoc networks. In *IEEE MASCOTS 2002*, 2002.

[23] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and. Technical report, 2001.

[24] Y. Zhao, Y. Chen, B. Li, and Q. Zhang. Hop ID: A Virtual Coordinate-Based Routing for Sparse Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 6(9), 2007.