# Table of Contents

List of Figures vi

List of Tables viii

Abstract ix

Acknowledgments xi

1 Introduction 1
   1.1 Types of Mobile Wireless Networks . . . . . . . . . . . . . . . . . . . . 1
   1.2 Challenges for Routing in Mobile Ad Hoc Networks . . . . . . . . . . . 2
   1.3 Routing in Mobile Ad Hoc Networks . . . . . . . . . . . . . . . . . . . . 3
   1.4 Routing in Information Centric Networks . . . . . . . . . . . . . . . . . 5
   1.5 Research Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2 Related Work 11
   2.1 Routing in Delay Tolerant Networks (DTN) . . . . . . . . . . . . . . . . 11
   2.2 Scalable Routing in MANET . . . . . . . . . . . . . . . . . . . . . . . 14
   2.3 Routing in Information-Centric Networks (ICN) . . . . . . . . . . . . . . 16

3 Automatic Social Routing 18
   3.1 ASR Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
      3.1.1 Network Architecture . . . . . . . . . . . . . . . . . . . . . . . . 19
      3.1.2 Routing within Social Groups . . . . . . . . . . . . . . . . . . . 21
      3.1.3 Routing across Social Groups . . . . . . . . . . . . . . . . . . . 28
      3.1.4 Routing between Physically Disconnected Components . . . . . . 34
   3.2 Performance Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . 34
      3.2.1 Experimental Dataset . . . . . . . . . . . . . . . . . . . . . . . . 36
      3.2.2 Simulation Setup . . . . . . . . . . . . . . . . . . . . . . . . . . 37
      3.2.3 Simulation Results . . . . . . . . . . . . . . . . . . . . . . . . . . 38
   3.3 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
List of Figures

3.1 Node Profile Structure .................................................. 21
3.2 Example of routing in ASR within and across social groups .... 27
3.3 Node Classification ...................................................... 37
3.4 ASR: Performance (a)-(c) delivery ratio with different data flows, (d)-(f) average delay with different data flows, (g)-(i) signaling overhead per node with different data flows, (j)-(l) data forwarding per node with different data flows .................................................. 39

4.1 Example of APDV operation ........................................... 45
4.2 Example of controller selection in six-node network with $k = 1$ and $r = 2$ 52
4.3 Route stretch in APDV ................................................... 61
4.4 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. .......................... 64
4.5 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. .......................... 64
4.6 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. .......................... 65
4.7 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. .......................... 65

5.1 (k, r) Dominating Nodes ................................................. 71
5.2 Example of Multi-Path ................................................... 72
5.3 Static network with increasing value of $k$: (a) Delivery ratio, (b) end-to-end delay of delivered data packets, (c) average number of control packets. 75
5.4 Static network with increasing value of $r$: (a) Delivery ratio, (b) end-to-end delay of delivered data packets, (c) average number of control packets. 77
5.5 Static network with Multi-Path: (a) Delivery ratio, (b) end-to-end delay of delivered data packets, (c) average number of control packets. 79
5.6 Mobile network with Multi-Path: (a) Delivery ratio, (b) end-to-end delay of delivered data packets, (c) average number of control packets. 80

6.1 Example of operation 84

7.1 Example of CORD operation 101
7.2 Mapping of Hierarchical Name 105
7.3 Mapping of Flat Name 106
7.4 Impact of increasing number of content requesting groups: (a) End-to-end delay of delivered data packets, (b) average number of control plane packets sent per node, (c) average number of data plane packets sent per node. 110
7.5 Impact of increasing number of flows: (a) End-to-end delay of delivered data packets, (b) average number of control plane packets sent per node, (c) average number of data plane packets sent per node. 110
7.6 Impact of caching, one consumer each group: (a) End-to-end delay of delivered data packets, (b) average number of control plane packets sent per node, (c) average number of data plane packets sent per node. 113
7.7 Impact of caching, nodes are all consumers in each group: (a) End-to-end delay of delivered data packets, (b) average number of control plane packets sent per node, (c) average number of data plane packets sent per node. 113
List of Tables

4.1 Simulation Environment ........................................... 63
6.1 Signaling Overhead for Routing to NDOs ........................ 96
Abstract

Towards Efficient Routing to Addresses and Names in Computer Communication Networks

by

Qian Li

Traditional routing protocols in mobile ad hoc networks (MANET) rely on routing tables which suffer from the scale-free and self-organizing nature of wireless networks. Routing protocols adopting destination-based routing tables introduce large overhead to maintain the routing tables in wireless networks. Many routing protocols cause flooding in updating and finding routes when topology changes. In information centric network (ICN), prior content routing approaches assume that the entire ICN use the same naming space for named data objects (NDO) and that routing tables list routes to NDOs or name prefixes, which incurs more overhead than routing to address ranges.

In this thesis we present novel approaches to build a new hierarchy for routing in computer communication networks which collaborates information dissemination in networks with publish-subscribe mechanisms, network storage, scalable routing and social networks. We propose a few hierarchical, scale-free routing schemes which scale well with network size. In contrast to prior approaches to disseminate information through wireless networks over routing tables, our approaches exploit the social plane exiting in the wireless network to improve the efficiency to deliver information and eliminates the
use of destination-based routing tables. In addition, different from prior works in adopting social networks into wireless networks that focus on physical aspects of connectivity over time, our approaches emphasize on the underlying social information among the wireless networks to analyze nodes potential delivery capability and efficiently utilize network storage resources. We also design novel use of *Dominating Sets* for improving the problems of broadcasting, information query and storage, topology management, and routing in MANETs and ICNs. Our new approaches allow distance-vector routing to scale by integrating it with adaptive publish-subscribe mechanisms. We propose to apply our approaches to address the existing issues of flooding, scale-free, routing efficiency, and routing table maintenance overhead that are faced by computer communication networks that use traditional routing tables for information dissemination.
Acknowledgments

I want to express my deep gratitude to my adviser, Professor J.J. Garcia-Luna-Aceves for his continuous support, guidance, and understanding during my journey of doctoral study and research. His positive attitude, enthusiasm, good energy, and friendship are an inspiration.

I thank the members of my committee, Professors Bradley R. Smith and Hamid Sadjadpour for their valuable advice and feedback. I would also like to thank Emily Gregg for her friendly help and advice.

A big thanks to my fellow cocos (Spencer Sevilla, Ali Dabirmoghaddam, Sam Wood, James Mathewson, and Turhan Karadeniz) in the Computer Communications Research Group (CCRG) directed by Professor J.J..

I cannot forget to thank my parents, without whom I could not have started this journey.
Chapter 1

Introduction

1.1 Types of Mobile Wireless Networks

There are two types of mobile wireless networks. The first type has wired and wireless gateways. Mobile nodes connect to the nearest base station. When nodes move across the boundary of one base station range towards another base station range, they perform hand-off and connect to another base station. The second type is the mobile ad hoc network (MANET) which is also called “networks without network”.

A MANET can be defined as a system of autonomous mobile nodes that communicate over wireless links without any preinstalled infrastructure. MANETs are useful in places that have no network infrastructure or when that infrastructure is severely damaged. MANETs are suited to applications where rapid deployment and dynamic reconfiguration are necessary. Typical applications are: emergency rescue operations, disaster relief efforts, military operations. MANETs have a few unique natures which

1
makes them differ from wired networks. They have fast and unpredictable topology changes due to nodes mobility. There are no specific dedicated routers to do routing for mobile nodes, instead, nodes themselves work as routers and hosts simultaneously. In other words, the functionality of each node is the same as every other node. Sometimes MANETs need to change channel capacity due to environmental effects. On the other hand, MANETs differ from mobile cellular networks in that with the absence of base stations, MANETs need multi-hop approach to deliver data.

1.2 Challenges for Routing in Mobile Ad Hoc Networks

The biggest challenge in MANETs is to find a path between communicating nodes, that is, the MANET routing problem. The considerations of the MANET environment constrains and the nature of the mobile nodes create further complications which results in the need to develop special routing algorithms to meet these challenges. The important considerations [64] in the performance of MANETs are the nature of the participating nodes, and the mobility of the mobile nodes, as MANETs are composed of power limited devices with a limited transmission range, in which case mostly they will not be able to communicate directly with the destination device. Routing in MANETs is always a dynamic optimization task to provide communication paths that are optimum in terms of some criterion (e.g. minimum distance, maximum bandwidth, shortest delay), and also satisfying some constraints (e.g. limited power of mobile nodes, limited capacity of wireless links). The main two challenges for routing in ad hoc networks are
scalability for a large number of nodes and limited bandwidth and power.

MANETs may also be composed of different types of devices, which have
different transmission ranges, this heterogeneous situation results in various problems.
According to [10] unidirectional link is one outcome of the heterogeneous situation.
Unidirectional links cause problems in route discovery process. A valid path from the
source to the destination might not be valid on the opposite direction due to different
transmission ranges of intermediate nodes.

In addition, MANETs are decentralized networks, nodes are unaware of the
state of the network. In order to become “aware” of the state of the network, nodes
frequently exchange information to support the routing function, which introduces un-
desired signaling overhead.

1.3 Routing in Mobile Ad Hoc Networks

Despite the problems of MANETs, MANETs have a tremendous potential to
be used in various real-world situations where setting up a traditional network infras-
tructure would be implausible. It is crucial to define optimum routing protocols in
MANETs.

Routing in MANETs occurs at the network layer. The objective of routing in
MANETs is to find a path between the source and destination over which packets can
be forwarded. Every routing algorithm in MANETs needs three essential components:
route discovery mechanism, route maintenance mechanism, and route error correction
mechanism. The route discovery mechanism discovers routes between source and destination nodes, the route maintenance mechanism maintains the discovered routes during communication process, and the route error correction mechanism takes care of failed routes.

In general, the approaches taken up to date can be classified by the way they construct the routing protocol: based on the information used to build routing tables, or based on when routing tables are built. When use network information to build routing tables, the network information contains shortest distance and link state. Shortest distance algorithms use the Bellman-Ford algorithm, Ford-Fulkerson algorithm as distance information to build routing tables. Examples of distance-vector routing algorithms include RIP [41] and IGRP [62]. Link state algorithms use connectivity information to build a topology graph that is used to build routing tables. OSPF [59] and IS-IS [54] are two routing algorithms using link state information to build routing tables. Routing protocols also can be classified based on when routing tables are built. Proactive algorithms maintain routes to destinations even if they are not needed, such as DSDV [60], OLSR [44]. On the other hand, reactive algorithms maintain routes to destinations only when they are needed. The well-known ones are AODV [61] and DSR [46]. The third approach hybrid algorithms combine both proactive and reactive algorithms advantages to maintain routes to nearby nodes even if they are not needed and maintain routes to far away nodes only when needed. ZRP [13] is one example of hybrid algorithms.

Proactive routing protocols provide fast responses to topology changes by actively maintaining routing information for all nodes in the network. However, the cost
of signaling overhead incurred in maintaining routing information for all nodes is high especially in large networks where large numbers of nodes have no interest in those routing information. In contrast, reactive routing protocols provide routing information on-demand, which in theory can reduce the signaling overhead incurred in maintaining routing information compared to proactive routing. However, this on-demand approach may incur long setup time in discovering the routes to new destinations. Hybrid algorithms balance both proactive and reactive and take advantages of both. Such as in ZRP, nodes proactively maintain routes to all nodes in intra-zone routing, inter-zone routing uses reactive routing. However, it is very critical to determine the optimal zone and hence owing to the lack of such global knowledge, the protocol converges to suboptimal routes.

1.4 Routing in Information Centric Networks

Several information centric network (ICN) architectures have been proposed [21, 27, 25] as alternatives to the current Internet architecture to address the fact that Internet usage is dominated by peer-to-peer communication and user-generated content. All ICN architectures aim at accessing content and services by name, independently of their location, in order to improve system performance and end-user experience.

In some ICN architectures, the identifiers of named data objects (NDO) or name prefixes are mapped into addresses of servers hosting the content, and IP routing is used to forward requests and content between content consumers and the servers
storing content. By contrast, some other ICN architectures rely on name-based routing of content, which integrates name resolution and content routing. Routers advertise or compute routes to named data objects (NDO) or name prefixes, and content requests for specific NDOs are forwarded towards the nearest routers storing those NDOs. Prior content routing approaches assume that the entire ICN use the same naming space for NDOs and that routing tables list routes to NDOs or name prefixes, which incurs more overhead than routing to address ranges.

1.5 Research Contributions

To realize the real world applications of MANETs and ICN architectures, we need efficient routing algorithms for computer communication networks which can adapt to the dynamic topology and handle delivery of packets to the destination while delivering high performance in terms of scalability, and adaptability to the changing topology. We have shown [11] that the order capacity of a wireless network can increase if the social groups that determine the flow of information in the network tend to involve nodes within small distances of one another, relative to the network size. However, these capacity gains cannot be approached unless the caching and routing mechanisms used in a wireless network take into account the structure of the social groups operating in the network. As Section 1.3 describes, prior proposals for routing and caching in MANETs have focused mostly on physical-level aspects of network connectivity, even when they have attempted to address the social groups in which nodes participate. Fur-
thermore, the vast majority of routing schemes proposed for MANETs rely on the use of destination-based routing tables and the dissemination of information towards destination addresses. Section 1.4 states that most ICN architectures rely on name-based routing of content, which integrates name resolution and content routing. Routers advertise or compute routes to NDOs or name prefixes, and content requests for specific NDOs are forwarded towards the nearest routers storing those NDOs.

We propose to reduce or eliminate the negative performance of routing in computer communication networks that stem from the above design choices. The clou of this dissertation is to design efficient routing in computer communication networks which provides hierarchical, scale-free routing, at the mean time, eliminates traditional routing tables. In this hierarchy, nodes can make forwarding decision more dynamically, locally than previous works. The new hierarchy also eliminates the need for any flooding of link states or per-destination information throughout networks, which provides better utilization of network bandwidth.

In this dissertation we analyze, design, and evaluate a set of distributed protocols for computer communication networks, which address the above mentioned problems and meet routing algorithms essential components. First, We present a new approach for information dissemination in networks subject to disruption to their end-to-end connectivity. In addition, in this new routing hierarchy, we also discover the connection between MANETs and underlying social plane, and apply the social plane into design of scalable routing algorithms in MANETs with efficient network storage utilization. In contrast to prior approaches to disseminate information through wire-
less networks over routing tables, our approaches exploit the social network exiting in the wireless network to improve the efficiency to deliver information and eliminate the use of destination-based routing tables. What is more, our approaches take into account the network storage resources, which provide novel information query and storing schemes for routing in MANETs. Disruption Tolerant Networks (DTN) [33] are networks characterized by time-varying connectivity in which end-to-end paths are not always available. In DTNs, messages must be stored at intermediate nodes in order to establish routes from sources to destinations in space and time, given that nodal encounters (nodes moving into transmission range of each other) provide opportunities information delivery even in those cases where physical end-to-end connectivity does not exist. We integrate the use of social-group information with an approach to routing that eliminates the use of destination-based routing tables. We show that our approach provides correct unicast routing, and compare its performance against that of epidemic routing and an disruption-tolerant, address-based routing scheme operating over networks subject to connectivity disruption. The results from our simulated experiments illustrate that using the social plane to guide how information dissemination occurs in a wireless network offers substantial performance improvements over traditional methods for routing in disruption-tolerant networks.

In Chapter 4 we introduce Adaptive Publish-subscribe Distance Vector (APDV) to substantially improve the scaling and performance properties of MANET routing by eliminating most of the flooding needed to maintain routes to destinations. Our second approach addresses problems related to location of resources or services to accommodate
client demands subject to constraints in ad hoc network. This work consists of main-
taining routing information for destinations at a few nodes whose routes are maintained
throughout the network, and using a common hash function for nodes to determine
where to publish their routes and how to route to destinations. Location problems can
arise whenever clients are looking for useful information or network services. Network
overhead is introduced into routing when searching network resources and services. We
consider this location of resource with one classical location problem consists of com-
puting the dominating sets (DS) of a network, and provide another approach to find the
DS of a network other than comparing node degree. A fast, distributed and distance
vector based routing protocol is designed to cooperate and provide routing for informa-
tion exchanging. 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.

Continue with APDV, we investigate the performance impact of different pa-
rameters in Chapter 5. We conclude the trend when varying the parameters how the
performance is impacted. We also investigate the benefits of having multi-path when
making routing decision.

Based upon APDV for MANETs, we introduce the a new protocol for routing
of NDOs in MANETs. APDV for content routing in Chapter 6 is a new approach to
routing [51] in MANETs that eliminates most of the flooding needed for signaling. We
used a simulation experiment to compare its performance with the performance of on-demand and proactive approaches to name-based routing in wireless networks. The key reason why APDV outperforms the other protocols is that it eliminates most flooding by using directories.

Chapter 7 presents an approach to content routing within autonomous systems in which directory nodes act as intermediaries to establish virtual cords linking consumers of content with content producers or caching sites. The primary objective of using directories between content producers or caches and the consumers of content is to reduce control overhead in the ICN. Instead of having routing tables listing routes to individual NDOs or name prefixes, they only list routes to the directories that maintain the mappings between name prefixes or NDO names and the locations where their copies reside.

The proposed work is trying to solve the routing problems exist in current computer communication networks coming as a result of cognizance of three essential components we discussed in Section 1.3. Our approaches has been shown to be very adaptive and responsive to changing environmental conditions in the problem domains and hence are a good fit for the current Internet architecture.

Several papers [53, 51, 36, 37] based on the above research work have been published or accepted for publication. Chapter 8 summarizes our contribution.
Chapter 2

Related Work

2.1 Routing in Delay Tolerant Networks (DTN)

There is a large body of work on routing for Wireless Mobile Network. The vast majority of the work reported to date has focused on routing to intended addresses or uses physical network infrastructure. However, past studies on human mobility suggest that exploiting social communities may have a positive impact on the performance of information dissemination schemes [29]. In addition, binding content to node addresses is not efficient in networks where end-to-end connectivity to specific nodes may be disrupted. As a result, a number of approaches have been proposed in recent years to support content dissemination in Wireless Mobile Networks, especially in DTNs, based on the names of information objects or the social contacts established by nodes over time.

Ad hoc On-Demand Distance Vector (AODV) [61] Routing is a routing proto-
col for mobile ad hoc networks (MANETs) and other wireless ad-hoc networks. It is a reactive routing protocol establishing a route to a destination only on demand. AODV is, as the name indicates, a distance-vector routing protocol. While operating under a highly mobile network scenario, it fails to utilize the structure built by the control messages during the route discovery process, meanwhile the overhead incurred through periodically flooding can affect its performance significantly [45].

In reactive schemes such as Epidemic routing [70], applications rely on movement that is inherent in the devices themselves to help deliver messages. When nodes are disconnected, outgoing messages are stored by nodes till their reconnection. Since nodes encountering can be unpredictable and rare in large mobile network, this information delivery approach suffers potentially low data delivery rates and large delays. In order to increase delivery rate and reduce delay, nodes flood messages throughout the network, which, however, exacerbates contention for limited buffers in nodes and drains nodes limited energy.

Bigwood et al. [23] found that information dissemination based on self-reported social groups is much simpler than assuming that social groups are detected by the system, and that delivery ratios for the two approaches are very similar. To detect the social groups existing in the wireless network requires complicated computation and large overhead to exchange accumulated physical connection information. On the other side, the self-reported social network can achieve similar accuracy to detected social network by comparing the delivery ratio. This motivates our use of self-reported social groups in our research.
DIRECT [66] is an example of content-based information dissemination schemes. Nodes flood their interest in content items denoted by attribute-value pairs; the interest requests establish routes back to the nodes interested in content, and those nodes with replicas of content that match the attribute values in a request are able to answer the request. This scheme is similar to Directed Diffusion [43] and has been shown to have delivery rates close to epidemic routing but with much smaller overhead. However, the attributes used in DIRECT simply state an object name, its type, its publisher, its size, and a time stamp. Furthermore, DIRECT incurs considerable overhead due to the flooding of interest requests, which becomes an issue in large networks.

There are also a number of routing approaches for specific network type such as DTN network (e.g., [18], [32], [16], [28], [52]).

Hsu et al. [32] classify nodes by their mobility profiles using a set of predefined locations to determine profiles and an association matrix denoting the importance of such locations in a mobility profile. Data are sent to all nodes fitting a profile. The approach is better than epidemic dissemination, but it cannot ensure accuracy and mobility profiles cannot fully represent the social ties among nodes.

Daly et al. [16] propose SimBet Routing for information dissemination in DTNs based on betweenness centrality and similarity of nodes in their social networks to determine which neighbor a node should select as a relay. The past encounters are exchanged to locally calculate the betweenness value and the similarity value. The message is routed to a structurally more central node where the potential of finding a suitable carrier is dramatically increased. Exchanging a summary vector containing a
list of destination nodes they are currently carrying messages for, the similar approach in Epidemic routing [70], is used to calculate the messages request list, which incurs messages forwarding. The authors show this approach outperforms PRoPHET [55] when nodes have low connectivity. However, computing node centrality and similarity require exchanging and updating the encounter history from each node, which incurs excessive overhead. Furthermore, constructing paths based on betweenness centrality can cause congestion when the same nodes are selected for too many paths.

2.2 Scalable Routing in MANET

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., [12], [49], [56], [69]) and some reduce signaling of clustering schemes by limiting propagation of control messages based on their distance from an originating point (e.g., HSLS [63]). 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., [53], [65], [73]) 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 [35] 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 [9]; 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 [48]) 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., [34], [74]). 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 [44]. 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 [71]. 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 [42]), or augment existing routing protocols with service discovery functionality (e.g., LSD [50], AODV-SD [38], L+ [15]).

2.3 Routing in Information-Centric Networks (ICN)

The ICN architectures proposed recently advocate various ways to accomplish name resolution and routing, and all of them use on-path caching of content [27], [25].

Several ICN projects advocate using a link-state routing approach for intra-domain content routing, and adding content prefixes to BGP for inter-domain content routing (e.g., [1, 2, 17, 4, 7]). NLSR [20] and OSPFN [26] are two protocols for name-based routing of content within an autonomous-system. Routers exchange topology information by flooding two types of link states advertisements (LSA). LSAs can describe the state of physical links just as it is done in traditional link-state routing protocols. In addition, routers flood LSAs about prefixes for which they have copies. Gritter and Cheriton [39] proposed the name-based routing protocol (NBRP) as an extension of BGP. In essence, name-prefix reachability is advertised among content routers, and path information is used to avoid permanent loops. The routing approach in the Mobility First project [3] requires using network addresses or source routing or partial source
routing. Several ICN projects (e.g., [5, 7]) have addressed content routing modalities based on distributed hash tables (DHT) running in overlays over the physical infrastructure and accomplish name-based routing on top of link state routing protocols.

Another early development on name-based routing of content was the CBCB [19] (combined broadcast and content based), which consists of two components: a spanning tree of the network or multiple per-source trees based spanning the network are established, and publish-subscribe requests for content based on predicates are sent between consumers and producers of content over the tree(s) established in the network. Bootstrapping [68] is similar to CBCB; however, in [68] nodes only publish or subscribe for control-plane information, and need the topological view of the network to decide the delivery paths for disseminating information.

DONA [30] uses flat names for content and either global or local IP addressing and routing to operate. If only local IP routing is used, content requests (FIND messages) gather autonomous-system (AS) path information as they are forwarded, and responses are sent back on the reverse paths traversed by requests. Within an AS, IP routing is used.

We observe that the content routing approaches proposed to date require one or more of the following types of mechanisms: (a) maintaining paths to named content or using source routes to content; (b) flooding of information about the network topology and the location of replicas of content; (c) flooding of content requests; (d) establishing trees spanning the network over which name-based publish-subscribe signaling is performed; and (e) maintaining overlays for distributed hash tables (DHT).
Chapter 3

Automatic Social Routing

Many approaches have been proposed and implemented in the recent past to support information dissemination in networks subject to end-to-end connectivity disruption. We have shown [11] that the order capacity of a wireless network can increase if the social groups that determine the flow of information in the network tend to involve nodes within small distances of one another, relative to the network size. However, these capacity gains cannot be approached unless the caching and routing mechanisms used in a wireless network take into account the structure of the social groups operating in the network. Prior proposals for routing in Disruption Tolerant Networks (DTN) have focused mostly on physical-level aspects of network connectivity, even when they have attempted to address the social groups in which nodes participate. Furthermore, the vast majority of routing schemes proposed for DTNs rely on the use of destination-based routing tables and the dissemination of information towards destination addresses.
3.1 ASR Description

Automatic Social Routing (ASR) is a new approach for opportunistic information dissemination. In contrast to prior approaches to disruption-tolerant networking that focus on physical aspects of connectivity, ASR is based on exploiting the social plane to improve the efficiency with which the physical network infrastructure is used. It integrates the use of social-group information with an approach to routing that eliminates the use of destination-based routing tables.

Social information is used as an integral part of routing decisions in ASR. From the standpoint of any social group, routing in a DTN can be classified as intra-group and inter-group routing, because information dissemination can take place within or across social groups. In intra-group routing, nodes are from the same social group and are typically virtually connected via a physical path. Inter-group routing may involve routing without existing physical paths.

3.1.1 Network Architecture

We assume that the social networks operating over a DTN consist of users with predefined and self-reported social information. Each user belongs to at least one social group. From the standpoint of the underlying wireless network connectivity, social groups may overlap or be disjoint from each other.

There are two cases to consider for overlapping groups. If nodes belong to more multiple groups, then those groups overlap, because of some of their nodes. Nodes
may each join a single group, but may be physically close to nodes in other groups. To
test the performance of ASR, we set the initial state only contains the second scenario,
but nodes can join another group in simulation. So we will cover both cases. If social
groups are disjoint from each other, information can traverse across groups only if either
nodes move around and encounter nodes in other groups, or nodes forward information
to remote groups along paths consisting of nodes in different social groups. Message
passing inside a social group may also depend on the social profiles of nodes, which we
discuss later.

As our first attempt to model social groups in DTNs, we use a conference
trace file [8], the participators’ information is exported. In the DTN network, each
node stores a structured profile as shown in Fig. 3.1 containing information from the
dataset: \textit{Node.ID} is the node’s identity; \textit{Group.ID} indicates the social group to which
the node belongs. In a conference scenario, \textit{Task} is different talks that the node needs
to attend, \textit{Area} shows different areas visited by the node visits. Also, in a conference
scenario, each participant is interested in a couple of topics, and these topics are called
\textit{Interests} in the node profile. Each node has a \textit{Contacts} list extracted from email or text
messages. \textit{Country} indicates the country from which this participant comes.

In ASR, nodes take advantage of the fact that they report the social groups
to which they belong. Within a social group, routing can be to specific individuals;
however, across social groups, routing can be simplified by first targeting the social
groups to which an intended destination belongs and focusing on individuals once a
given social group is reached. This is very similar to traditional hierarchical routing,
except that no strict clusters or subnets of nodes are maintained.

### 3.1.2 Routing within Social Groups

Each social group elects a labeling root node using a distributed algorithm based on neighbor-to-neighbor signaling among all nodes in a connected component of the network. Each node transmits a *Hello* message which is a neighbor-to-neighbor broadcast message specifying the node profile, and for each social group also states the root identifier of the social group, the label of the node, and the node identifiers and labels assigned to its immediate neighboring nodes. For each social group, if Σ is the finite set of symbols, then the routing label of a node, \( l \), is a string with symbols from \( \Sigma \) such that \( |l| \geq 1 \). The root node has the smallest label. Once a node has a routing label, it assigns a unique suffix \( s_i \) to each of its children, \( i \). The child then assigns itself
the label \( l \odot s_i \), where \( \odot \) is the concatenation operator.

When a node is initialized, it determines for each social group the smallest routing label and root it can attain from the Hello received from its neighbors. If the node does not obtain a routing label for a social group within a local labeling timeout period, it assigns itself as the root node, and sends Hello to its one-hop neighbors and assign labels to them. If a labeled node receives a Hello with a root identifier lower than its own root for a social group, the node accepts the lower root and receives a routing label. Eventually, for each social group, the node with the lowest identifier is elected as the root and all the other nodes are ordered with respect to that root node. The root node is elected such that: (i) each node is assigned a label denoting the relative location of the node with respect to the root; (ii) the labels of a source and a destination define one or multiple valid routes between two nodes; and (iii) node mobility and link or node failures and additions have limited impact on the labels already assigned to other nodes. Nodes send their Hellos periodically to help refresh their labels to cope with node mobility, failures, and additions.

Once the prefix labels are setup, the anchor setup phase begins. Each node in the DAG identifies an anchor to store the mapping between the node profile and associated prefix label of the node [35]. As nodes fail, reboot or move around, their labels change and anchors keep track of these changes through periodical updates from nodes.

Given the prefix labels of a source and a destination in the same social group operating in a connected component of the network, one or multiple routes are defined
Algorithm 1 Intra-Group Routing

1: Data: source, dest, pkt, destLabel, nbrTable
2: nextHop = NULL;
3: if source.socialgroup == dest.socialgroup then
4: /*intra-group routing*/
5: if self == source then
6: /*dest is in neighborhood*/
7: Fn(sendPkt(pkt));
8: return
9: else if destLabel is known then
10: /*dest’s label is known and added into pkt*/
11: nextHop = Fn(findNextHop(nbrTable, destLabel));
12: else if find dest in cached routes then
13: destLabel = cachedLabel;
14: nextHop = Fn(findNextHop(nbrTable, destLabel));
15: else
16: /*get Anchor’s label by hashing dest, send request to Anchor*/
17: anchorLabel = Fn(hash(dest));
18: nextHop = Fn(findNextHop(nbrTable, anchorLabel));
19: end if
20: else if self == dest then
21: numDataReceived++;
22: return
23: else
24: /*relay node*/
25: if findDestInNbr(dest) == TRUE then
26: /*dest is in neighborhood*/
27: Fn(sendPkt(pkt));
28: return
29: else if destLabel is known then
30: /*dest’s label is known and added into pkt*/
31: nextHop = Fn(findNextHop(nbrTable, destLabel));
32: else if find dest in cached routes then
33: destLabel = cachedLabel;
34: nextHop = Fn(findNextHop(nbrTable, destLabel));
35: else
36: /*relay request to Air Anchor*/
37: nextHop = Fn(findNextHop(nbrTable, anchorLabel));
38: end if
39: end if
40: if nextHop == NULL then
41: Fn(keepPkt(pkt));
42: else
43: Fn(sendPkt(pkt, nextHop));
44: end if
45: end if
Algorithm 2 Link Failure Local Repair

1: Data: source, dest, pkt, failedNextHop, destLabel, nbrTable
2: nbr = NULL;
3: nextHop = NULL;
4: Localrepair = TRUE;
5: if Link fails at node A then
6:  /*one hop neighborhood has changed or physical link failure*/
7:  if A.socialgroup == source.socialgroup then
8:    /*node A sends a route error message to source node*/
9:    Fn(sendRouteError(source));
10: else
11:    /*node A performs local repair*/
12:    if A.socialgroup == dest.socialgroup then
13:      if destLabel is known then
14:        /*find nextHop to dest bypassing the failed next hop*/
15:        nextHop = Fn(getNextHop(nbrTable, failedNextHop, destLabel));
16:      else
17:        /*node A sends a new route request to Air Anchor bypassing failed next hop*/
18:        anchorLabel = Fn(hash(dest));
19:        nextHop = Fn(getNextHop(nbrTable, failedNextHop, anchorLabel));
20:      end if
21:    else
22:      /*resolve a new node with closest physical or social connection with destination or
destination group*/
23:      nextHop = Fn(resolveNextHop(nbrTable, dest, failedNextHop));
24:    end if
25:  end if
26:  if nextHop == NULL then
27:    /*local repair fails*/
28:    Fn(keepPkt(pkt));
29:  else
30:    Fn(sendPkt(pkt, nextHop));
31:  end if
32: end if
Algorithm 3 Handling Route Error Message
1: HandleRouteError(pkt, source, dest)
2: {
3:   if self == source then
4:     \text{Fn}(\text{deleteCachedRoute}(\text{dest})));
5:     \text{Fn}(\text{insertNotUseNode}(\text{pkt})));
6:   else
7:     /*relay node*/
8:     \text{Fn}(\text{sendRouteError}(\text{source})));
9:   end if
10: }

automatically from those labels. To route to a given destination in the same social
group, a source must find the prefix label of the destination and the destination must
publish its prefix label, and this is accomplished by means of social anchors. A social
anchor is a node whose own label is the closest match within its two-hop neighborhood
to the hashing of the name of a social group $\kappa_{\text{group}}$ and a set of zero or more interests
$\kappa_{\text{ ints}}$ within that group. To publish its presence, a node hashes its name, identifier
or set of attributes that describe it, using a common hashing function over the terms
selected for publishing. The node then unicasts a publish request towards the prefix
label resulting from the hashing, and the request states the node profile and its prefix
label. To communicate with a destination, a source hashes the name, identifier or
attributes describing the destination and obtains the prefix label of a social anchor; it
then unicasts a subscription request towards the social anchor. This publish-subscribe
signaling takes place among nodes in the same social groups.

For fault-tolerance purposes, multiple social anchors are used to maintain the
information of a social group associated with specific interests. More specifically, the
neighbors of a social anchor also become social anchors for the same group of individual. Therefore, if one social anchor moves away or dies, there are other social anchors around to keep the publish-subscribe process working.

A consistent hash function is used to avoid remapping of nodes. The hash function takes node profiles as input and returns a prefix label of the associated social anchor. Note that, because social groups and individuals may be characterized by a variable number of attributes, all nodes must agree beforehand on the subset of attributes to be used for hashing the description of a given social group or individuals into prefix labels.

Algorithm 1 explains the procedure of the Intra-Group Routing. In the example shown in Fig. 3.2, if $A$ were the intended destination of node $S$, then $S$ would hash $A$’s name to obtain $A$’s social anchor’s prefix label, then obtain $A$’s prefix (122) from social anchor, after which it could route to $A$ based on its own prefix (11) and $A$’s prefix.

We use a route error message to help node track broken routes and provide local repairs. If the route failure happens in the same group as source node, the failed node sends a route error message to source node. When receiving this router error, the source node deletes local cached routes to the destination to force the source node to request a new route in the future. In the request message, there is an entry containing failed nodes that anchors should not use for some time period for the same destination. If the route failure happens in a different group from that of the source node, the relay node performs local repair and finds another route through its neighborhood. If local
repair fails, that node keeps the packet and a new forwarding path is selected using the routing strategy explained in following section. Algorithm 2 is the procedure used to perform Link Failure Repairment. Route error message is handled using Algorithm 3.

If a destination node is unreachable, a packet intended for that destination is stored at the current node (source or relay), and a forwarding path is selected using the routing strategy explained in the following subsection.

A node is aware of the destination’s social group. If source-destination pairs of packet are in the same group, the source sends a request for the route to the proper Anchor. If the destination is in a remote group, the source sends a request to a Social Anchor, Social Anchor will pick the node that is most socially or physically close to that group to forward the packet.
3.1.3 Routing across Social Groups

To route to a destination in a different social group, the node sends a subscription request to a social anchor selected among the nodes in the social groups to which the source belongs. The source uses the name or attributes describing the social group of the destination, rather than the destination itself. We assume that the source is aware of the destination’s social group.

To enable nodes to operate as social anchors for social groups to which they do not belong, nodes that come into physical contact with nodes in other social groups publish their encounters with social anchors in their own social groups. ASR combines physical distance information with social network information for routing across social groups. If a node has either a physical or social connection with nodes in another social group, it publishes the mapping between the connection and the associated social anchor’s prefix label with the corresponding social anchor in its own group. If the publishing request is meant to inform the social anchor of a status change in the social connection status, it consists of the encountered node’s name and its social group, number of common tasks, friendship between each other, number of common friends, and encounter duration in time \( t \). If the publishing request is generated because of a change in physical connection status, in addition to above information, it also contains the physical distance (network hops) to the node with which the encounter happened. Node sets the distance value in the publishing request to be infinite when the physical connection breaks (node moves away or dies); otherwise, it is a finite integer. The
node with a prefix label that is the closest match in its two-hop neighborhood to the
prefix label stated in a publishing request becomes the designated social anchor for the
mapping, then stores it and builds the SDHT. The SDHT consists of two tables storing
published attributes, one lists the nodes currently having physical connection with nodes
in the social group this social anchor representing for. A node with infinite physical
distance value stated in a publishing request is removed from the list by social anchor.
The other table lists the nodes having social connections but no physical connections
with nodes in that social group. This SDHT building policy guarantees that there is
no overlapping of nodes between the two tables. The neighbors of the designated social
anchor also get the publishing request and store it into SDHT structure. The frequency
to update node’s connection status to social anchors is controlled by the change of node’s
physical or social connection with another social group.

The following four node characteristics are used to inform routing decisions
based on information regarding social connections when physical distance information
is not available:

**Similarity:** two users are similar if they have common task. We assume that
because they have common task, they know each other, and they will meet more often
than others in a certain period of time. This similarity is also restricted by time since
‘task’ has time sensitive characteristic. The similarity $Sim(x, y)$ between user $x$ and $y$
was calculated by:

$$Sim(x, y) = |N(x) \cap N(y)|$$  \hspace{1cm} (3.1)
where $N(x)$ and $N(y)$ are the set of tasks of user $x$ and $y$ respectively. If the similarity index is larger than a threshold, we call user $x$ and $y$ are socially similar.

**Social Affiliation:** two users are affiliated or socially close if they are friends in their profile friend list. If put Social Affiliation into a $n \times n$ symmetric matrix, where $n$ is the number of users in the network. The Affiliation matrix has elements:

$$A_{xy} = \begin{cases} 
1 & \text{if } x, y \text{ are friends} \\
0 & \text{otherwise}
\end{cases}$$

We consider contacts to be bidirectional, so if a contact exists between $x$ and $y$ then there is also a contact between $y$ and $x$. This Affiliation value will add one point into users social tie strength calculation if it is positive, otherwise, we ignore this part.

**Spatially Closeness:** two users are spatially close if they have more common friends in their profile friend list or contact list. The spatial closeness $S(x,y)$ between user $x$ and $y$ was calculated by:

$$S(x,y) = |F(x) \cap F(y)|$$

where $F(x)$ and $F(y)$ are the set of friends of user $x$ and $y$ respectively. The value of $S(x,y)$ determines how spatially close user $x$ and $y$ are.

**Temporally Closeness:** two users are temporally close if they meet or communicate often in a certain time, and is was calculated by:

$$T_t(x,y) = \frac{C(x,y)}{t}$$

where $C(x,y)$ is the time of communication happening between $x, y$ in a given period of time, and $t$ is the period of time we calculate in. So $T_t(x,y)$ is the communication
frequency between $x$ and $y$.

These are then composed into a single value, as

$$C_{social} = \omega_{sim} Sim(x, y) + \omega_a A(x, y)$$

$$+ \omega_s S(x, y) + \omega_t T_t(x, y) \quad (3.4)$$

which represents how socially close two nodes are and implies social distance between nodes in reversal. The weights $\omega$ denote the relative importance of each attribute. Their value depends on the application scenario. Due to space limitations, we briefly apply those values on a single scenario which we used to show the performance also demonstrate some crucial properties of our approach. The default values of the weights $\omega_{sim} = 0.45$, $\omega_a = 0.2$, $\omega_s = 0.05$, and $\omega_t = 0.3$ are those providing the best performance in terms of delivery ratio in our simulations.

If there is more than one node having a strong social tie with either an individual destination or other nodes in the social group of the intended destination, the social closeness values of opportunistic nodes are ranked from highest to lowest. The first $k$ nodes are selected as opportunistic contacts to forward message to an intended destination.

The following rules are used to route data across social groups:

**distance**($k$) forwarding path $u \rightarrow v$ is allowed if $v$ is within distance $k$ to $d$ in the current network topology, and satisfies distance $D(v) < D(u)$. 

31
neighbor\((k)\) by comparing node characteristics, forwarding path \(u \rightarrow v\) is allowed if \(v\) and \(d\) are socially close within distance \(k\) (or has the highest ranking among all opportunistic nodes) in social graph with social distance \(S(v) < S(u)\).

**non-increasing-social-distance** with the same distance\((k)\) to \(d\), forwarding path \(u \rightarrow v\) is allowed if social distance from \(v\) to \(d\) is less than the one from \(u\) to \(d\), \(S(v|D) < S(u|D)\).

According to the **non-increasing-social-distance**, when nodes have both social distance and physical distance information available, the physical distance is given higher priority. Equation (1) then is adjusted to be:

\[
C = \omega_\text{sim}Sim(x, y) + \omega_\text{a}A(x, y) + \omega_\text{s}S(x, y)
\]

\[
+ \omega_\text{t}T_t(x, y) + \omega_\text{d}/D(x, y)
\]

(3.5)

where \(\omega_\text{d} = 1000\), so that physical distance dominates the selection. Equation (2) also consists with **non-increasing-social-distance** forwarding rule that with same physical distance, social distance is the criteria to select a relay node.

To route to the destination in another social group, the source uses the name describing the social group of the destination as the hash input to get the social anchor’s prefix label. The source unicasts a subscription request towards the social anchor through prefix routing. The designated social anchor stores the SDHT for nodes who have social distance or physical distance information with nodes in destination’s social group. Social anchor selects first from the table storing nodes with physical distance information to the destination’s group, which provides a direct forwarding path to des-
destination’s group. If that table is empty, social anchor selects nodes from the other table with social distance information to the destination’s group by ranking the nodes according to Equation (1). The relay node selection follows the above forwarding rules. Social anchor sends back a reply to source containing the selected relay node for continuous communication between source and destination. Once the packet is forwarded to a node belonging to destination’s group, routing to destination is attained by the prefix routing approach we described in Section 3.1.2.

Fig. 3.2 illustrates how routing takes place across social groups. The source S is in social group 1 with label ‘11’, and it has packet for node D, which the source knows to be in social group 2 but has no cached route to it. The first criteria to select a relay node is to find a node that has distance information to the destination group. The source asks the node serving as social anchor for group 2 for a relay node. In this example, node H is the social anchor for group 2, it selects first k relay nodes that have distance information or strong social ties with the destination. If no distance information is available in group 1, relay nodes are chosen by their social connection with the destination. There are two nodes between the two groups with direct connection to group 2, and both of them are at one-hop distance to node A. Accordingly, node A is selected as one relay node by social anchor H. Both node B and C has direct connection to group 2, but node B has a stronger social tie with destination D after node A compares their social distance to group 2. In this case, node A selects node B to be next hop and sends the packet to it. Node B forwards the packet to node E which is in group 2. After the packet has been sent to group 2, prefix routing is carried
out to the specific destination. Algorithm 4 explains the procedure of the Inter-Group Routing.

### 3.1.4 Routing between Physically Disconnected Components

End-to-end paths need not be always available. If routing must happen between physically disconnected components within or across social groups, social information is used to select relaying nodes. In the current two-hop neighborhood, the source or a relaying node calculates neighbors’ social distance to destination or destination’s group using four nodes characteristics introduced in previous subsection. The node who has a shorter social distance or who has an immediate neighbor with a shorter social distance is selected as the next hop of the packet. A copy of packet is then sent to the selected relaying node. To achieve fast packet delivery, if a network disconnection happens between two social groups, the relaying node sends a copy of the packet to every member from the destination group during the first encounter, until reaching the maximum copy a node can create. In addition, when the destination is unreachable in the current two-hop neighborhood, the relaying node must keep the packet and perform periodically next hop calculation until the packet lifetime expires, or it meets the destination or nodes having routes to the destination.

### 3.2 Performance Evaluation

We compare ASR with two other data dissemination approaches. We use an Epidemic scheme [70] in which there is no user grouping information and all nodes
Algorithm 4 Inter-Group Routing

1: **Data:** source, dest, pkt, destLabel, nbrTable
2: nextHop = NULL;
3: if source.socialgroup ≠ dest.socialgroup then
4: /*inter-group routing*/
5: if self == source then
6: if findDestInNbr(dest) == TRUE then
7: /*dest is in neighborhood*/
8: Fn(sendPkt(pkt));
9: return
10: else if self is the node selected to carry this pkt then
11: /*self has closest physical or social connection with destination or destination group*/
12: nextHop = Fn(resolveNextHop(nbrTable, dest));
13: else if selected carry node for pkt is known then
14: nextHop = Fn(findNextHop(nbrTable, selecNodeLabel));
15: else
16: /*ask Social Anchor the selected carry node for pkt*/
17: socialanchorLabel = Fn(hash(dest.socialgroup, dest));
18: nextHop = Fn(findNextHop(nbrTable, socialanchorLabel));
19: end if
20: else if self == dest then
21: numDataReceived++;
22: return
23: else
24: /*relay node*/
25: if self.socialgroup == dest.socialgroup then
26: /*intra-group routing*/
27: Fn(IntraGroupRouting(source, dest, pkt));
28: return
29: else if findDestInNbr(dest) == TRUE then
30: /*dest is in neighborhood*/
31: Fn(sendPkt(pkt));
32: return
33: else if self is the node selected to carry this pkt then
34: /*self has closest physical or social connection with destination or destination group*/
35: nextHop = Fn(resolveNextHop(nbrTable, dest));
36: else if selected carry node for pkt is known then
37: nextHop = Fn(findNextHop(nbrTable, selecNodeLabel));
38: else
39: /*relay request to Social Anchor*/
40: nextHop = Fn(findNextHop(nbrTable, socialanchorLabel));
41: end if
42: end if
43: if nextHop == NULL then
44: Fn(keepPkt(pkt));
45: else
46: Fn(sendPkt(pkt, nextHop));
47: end if
48: end if
can be relays. According to [70], we set a message hop limit, and a buffer size limit in the implementation. A message life timer is assigned to each message. Each node periodically deletes timeout messages from its message buffer. This is also the most aggressive forwarding strategy in DTN. We also implemented an efficient disruption-tolerant, address-oriented routing protocol based on that reported in [14]. We denote this protocol by DAR and use it for comparison because it is very efficient for DTN routing towards destination addresses without taking into account social groups. Each node has a message buffer to store route unknown messages till their timeout. These messages are kept until current node encounters a node having route to destination or destination directly.

3.2.1 Experimental Dataset

In this paper, we use a dataset collected from the seventh HOPE (Hackers On Planet Earth) conference held on July 18-20, 2008. Conference attendees received RFID badges that uniquely identified and tracked them across the conference space. The dataset was collected from the three days of the conference, and the content included participants’ location, interest, profile, friend list, and event details.

We selected the 83 most active participants from the dataset, i.e., the most interests listed in their profiles and the most talks attended. We use common interests and social affiliation (e.g., contact with email, text message, their countries and other attributes) to divide them into 4 social groups. The method to classify participants into groups based on their attributes is supported in statistical computing and graphics
language R [6]. The hierarchy is shown in Fig. 3.3.

We separate the dataset into three parts. 50% data is used for training, 10% for tuning, and 40% for simulating. We believe this dataset is large and accurate enough to simulate our scheme.

![Cluster Dendrogram](image)

Figure 3.3: Node Classification

### 3.2.2 Simulation Setup

We use a hi-fidelity event-driven packet level network simulator, Qualnet-v.4.0 [67]. We first import the dataset into Qualnet, and then we set a node position file according to the conference trace file, which means at each second in the simulation, network topology represents the real conference scenario. The trace file was collected using RFID tracking data, and in order to adapt it to 802.11b transmission ranges we increased the coordinate distances in the trace file. Therefore, the coordinate value in the trace file represents 802.11b transmission ranges in our simulation. We collect statistics
from 1 hour up to 10 hours, which only contains daytime node activity. We simulate scenarios with different numbers of concurrent data flows to see the performance of three schemes under different network load. Simulations were instrumented in networks of nodes deployed in a terrain of dimensions 1600m X 1600m. PHY-Model in the nodes was PHY802.11b with transmission range of 300m.

Data sources are generators that produce a constant bit rate (CBR). In the trace file, some nodes are moving fast, in order to fully explore the use of social network information in the fast mobile scenario, we set the data rate of 2 packets per minute and each source was allowed to transmit up to 1200 packets. Data pairs are randomly selected. All the experiments are run multiple times with 10 different seeds to avoid any artifact of pseudo random number generators.

We set same key parameters for three schemes. Hello message interval is set to 3 sec, there is the maximum 100 messages buffer size, all list and buffer flush timers are set to 1 hour, and we set three times hello message interval for neighborhood status check.

3.2.3 Simulation Results

We use the following four metrics for comparison:

Overhead: The number of signaling packets sent per node.

Average Delay: The average time between data generation time and data receiving time at destination. It contains message-holding time in each node’s buffer, which represents DTN network character.
Figure 3.4: ASR: Performance (a)-(c) delivery ratio with different data flows, (d)-(f) average delay with different data flows, (g)-(i) signaling overhead per node with different data flows, (j)-(l) data forwarding per node with different data flows.
**Delivery Ratio (goodput):** The number of data packet received divided by number of data packet sent.

**Data Forwarding:** The number of data packets sent including initiating and forwarding at each node.

Fig. 3.4 summarizes the simulation results for the three schemes under different scenarios. In Fig. 3.4 (a) - (c), all three schemes have lower delivery ratios as the number of data flows increase. Epidemic routing has the highest delivery ratio for the 15- and 50-flows scenarios, but it has lower delivery ratio than ASR and DAR for the 100-flows scenario. One reason for the degradation in performance in epidemic routing with 100 data flows is the large number of transmissions needed to support many data flows, which causes too many collisions. Other contributing factor is that we set the limit of data buffer size, and eight hops is the maximum hop limit to avoid looping. ASR always achieves higher delivery ratio than DAR under different numbers of data flows.

Fig. 3.4 (d) - (f) show the average delay of three schemes. With a small number of data flows, DAR attains lower delays than ASR, because very few routes are needed in the network. In our approach, sources need to send requests to social anchors first, which increases end-to-end delays. With large data traffic, epidemic routing attains the smallest delays for those packets that are delivered because of the characteristic of flooding; however, fare fewer packets get delivered. Under high data load, ASR has lower end-to-end delay than DAR. DAR performs the worst under high data load. When the number of data flows increases from 15 flows to 100 flows, the end-to-end delay of DAR increases 4 times, and is 33% higher than in ASR.
Fig. 3.4 (g) - (i) show the average overhead under different data flows. With low data load, epidemic routing has the highest overhead and this is mainly because of the cost to exchange summary vectors to determine which messages have not been seen by each other and the cost to request copies of those messages. When the number of data flows increases, DAR generates too many control packets, the typical one is route request messages. Its overhead is 4.5 times higher than ASR and 2.25 times higher than epidemic routing with 100 data flows. ASR has the lowest overhead under different data load. In other words, our approach provides similar delivery ratio but much lower overhead than DAR.

Fig. 3.4 (j) - (l) present the data packets forwarded per node. As expected, epidemic routing has the highest data forwarding statistic. ASR has slightly higher forwarding load than DAR. This is because a node carrying data and encountering a member from the destination group must forward the data to that node. This incurs additional data forwarding overhead than DAR. However, the difference between our approach and DAR becomes very small as the number of data flows increases.

### 3.3 Conclusion

We proposed a novel approach to routing in DTNs that takes advantage of social-group information and eliminates the vast majority of the signaling overhead present in traditional routing schemes for DTNs. We used a social network model of social affiliations of group members, and developed an integrated approach to routing.
within and across social groups that operates efficiently even when the underlying net-
work is disconnected by eliminating flooding by means of social clues maintained in a
DHT. From the simulated experiments based on a real-world trace file, we find out that
ASR has similar delivery ratio but far lower end-to-end delay and overhead than DAR,
especially under high data load. It is also far more efficient than epidemic routing, and
yet very resilient. If the network is temporally disconnected or labeling is not up to
date, data packets are stored at relaying nodes, and routing resumes once forwarding
opportunities occur, which saves considerable signaling overhead.
Chapter 4

Adaptive Publish-subscribe Distance Vector Routing

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). The prior work aimed at making routing more scalable in ad hoc networks 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 work is it introduces 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.

4.1 APDV Description

4.1.1 APDV 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.

Fig. 4.1 illustrates the basic operation of APDV assuming that each node has at least one local controller within two hops. In the example of Fig. 4.1, 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 \((d, \{l_1^d, ..., l_k^d\})\), where \( l_i^d \) \((1 \leq i \leq k)\) is a local controller for node \( d \). Each local controller \( l_i^d \) 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. 4.1, 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 request 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. 4.1, 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 \( d \).

### 4.1.2 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 $\{nid_{vj}^i, d_{vj}^i, sn_{vj}^i, cs_{vj}^i, k_{vj}^i, LCL_{vj}^i\}$.

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 $nid^i$, and uses the information in its HELLOs. A HELLO from node $i$ contains: $nid^i$, $sn^i$, $cs^i$, $k^i$, and updates to $NT^i$, $CT^i$, and $LRT^i$. An update to $NT^i$ regarding neighbor $j$ consists of the tuple $\{nid^j, sn^j, cs^j, k^j, LCL^j\}$. An update to $CT^i$ regarding controller $c$ consists of the tuple $\{nid^c, d^c, sn^c\}$. An update to $LRT^i$ regarding destination $v$ consists of the tuple $\{nid^v, sn^v, d^v, LCL^v\}$.

4.1.3 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 [40], and many distributed algorithms exist to approximate minimum connected dominating sets (MCDS) with constraints (e.g., [47]). 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.
4.1.3.1 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 \( d^i_v \leq r \), and sets \( k_i^j = |LCL^i| \).

**CAR (Controller Addition Rule):**

Node \( i \) adds itself to \( CT^i \) if

\[
(k_i^j < k) \land \{ i = \text{Min}\{\text{nid}_j \forall j \in NT^i \mid (j \notin CT^i) \land (k_j^i < k)\} \}
\]

**CDR (Controller Deletion Rule):**

Node \( i \) deletes itself from \( CT^i \) if

\[
(k_i^j > k) \land \{ \forall j \in NT^i \mid (|LCL_j^i - \{i\}| \geq k \forall j \notin LCL_j^i) \land
\]
Fig. 4.2 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. 4.2(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. 4.2(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. 4.2(c) to 4.2(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. 4.2(d), or nodes 30 and 18 deleting controller 20 in Fig. 4.2(f), for example. Fig. 4.2(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. 4.2(g) shows the final state of the LCLs for the example.
network.

Figure 4.2: Example of controller selection in six-node network with $k = 1$ and $r = 2$.

4.1.3.2 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 con-
trollers. 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 only those
neighbors with shorter distances to the controller or with a more recent sequence num-
ber 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 $CT^i_j$ 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 $CT^i_j$ 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 $CT^i_j$ for entry $c \neq i : \{nid^i_{cj},
\quad d^i_{cj}, sn^i_{cj}\}$, 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 $c$. 53
RCR (Reset Controller Rule):

If node $i$ must delete itself from $CT^i$ using CDR then

$$d_i^* = \infty; \quad sn_i^* = sn_i^1 + 1; \quad reset-timer^i = T$$

UCR (Update Controller Rule):

If $(\exists q \in NT^i \mid sn_i^q > sn_i^v)$

then begin

if $(v = s_i^c) \land (sn_i^c > sn_i^v) \land (d_i^c = \infty)$

then set $sn_i^c = sn_i^1; \quad d_i^c = \infty$

else begin

$$d_i^* = \min\{d_i^f + 1 \mid (f \in NT^i) \land (sn_i^f = \max\{sn_i^v \mid v \in NT^i\})\}$$

$$s_i^c = j \mid (j \in NT^i) \land (d_i^j = d_i^c - 1)$$

set $sn_i^c = \max\{sn_i^v \mid v \in NT^i\}$;

else begin

$$d_i^* = \min\{d_i^f + 1 \mid (f \in NT^i) \land (sn_i^f = sn_i^c) \land (d_i^c < d_i^f)\};$$

set $s_i^c = j \mid (j \in NT^i) \land (d_i^j = d_i^c - 1)$

4.1.4 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 mech-
anism 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 [42]) 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.

4.1.4.1 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 \( v \) \( \{nid_v^j, sn_v^j, d_v^j, LCL_v^j\} \), then node \( i \) forwards the request (i.e., it includes the \( LRT^i \) entry \( \{nid_v^i, sn_v^i, d_v^i, LCL_v^i\} \) in its own HELLO) if it is the successor for node \( j \) to any of the controllers listed in \( LCL_v^j \). Once a local controller \( c \) receives an entry for destination \( v \) and \( c \in LCL_v \), then \( c \) publishes (i.e., stores) the entry \( \{nid_c^v, sn_c^v, d_c^v, LCL_c^v\} \), 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 \( \{nid_i^n, sn_i^n, d_i^n, LCL_i^n\} \). Each node \( v \) in the route from node \( i \) to controller \( a_i \) forwards the publication request towards \( a_i \) and caches the tuple \( \{nid_v^n, sn_v^n, d_v^n, LCL_v^n\} \). Once controller \( a_i \) receives the request, it stores the tuple \( \{nid_i^{a_i}, sn_i^{a_i}, d_i^{a_i}, s_i^{a_i}, LCL_i^{a_i}\} \). 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 \( i \).

4.1.4.2 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

57
$j \notin CT^o$, it computes $\text{hash}(j) = a_j$, 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 $LCL^o_j$. When $a_j$ receives $o$'s request, it responds with the tuple $\{\text{nid}_j^{a_j}, \text{sn}_j^{a_j}, LCL_j^{a_j}\}$ and sends the response to the nearest controller it finds in $LCL^o_j$. Node $o$ stores the tuple $\{\text{nid}_j^o, \text{sn}_j^o, LCL_j^o\}$ in $RT^o$ upon receiving the reply to its subscription. Data packets from $o$ are then sent towards the controllers in $LCL_j^o$ 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.

### 4.2 Analysis

The following theorems demonstrate that APDV is loop free at every instant, that all non-directory nodes have at least $k$ local directories within $r$ hops within a finite time after the topology of the network is connected and stable, and that the the route stretch incurred by APDV is small.

**Theorem 4.2.1.** Nodes using UDR to update routes to directories yields loop-free routes to directories at every instant.
Proof: The proof is by contradiction. Assume that a routing loop \( L_c \) for directory \( c \) consisting of \( h \) hops is created when nodes in \( L_c \) change successors according to UDR. Let \( L_c = (n_1, n_2, \ldots, n_h) \), with \( n_{i+1} = s^n_{c_i} \) for \( 1 \leq i \leq h - 1 \) and \( n_1 = s^n_{c_1} \).

According to UDR, for each hop \( n_i \in L_c \) (\( 1 \leq i \leq h \)) it must be true that \( s^n_{c_i} \leq s^n_{c_{i+1}} \) for \( 1 \leq i \leq h - 1 \) and \( s^n_{c_h} \leq s^n_{c_1} \). This result implies that \( s^n_{c_i} = s^n_{c_{i+1}} \) for \( 1 \leq i \leq h - 1 \) and \( s^n_{c_h} = s^n_{c_1} \). Because of UDR, this implies that \( d^n_{c_i} < d^n_{c_{i+1}} \) for \( 1 \leq i \leq h - 1 \) and \( d^n_{c_h} < d^n_{c_1} \), which implies that \( d^n_{c_i} < d^n_{c_i} \) for \( 1 \leq i \leq h \). This is a contradiction and hence the theorem is true. \( Q.E.D. \)

Theorem 4.2.2. APDV is loop-free at every instant.

Proof: From Theorem 1, it follows that all routes to nodes that are selected as directories are loop free. Nodes that serve as local directories for nodes nearby obtain their routes by the propagation of publication requests from those nodes along the reverse loop-free paths to local directories established following UDR. Hence, the routes maintained at local directories to simple nodes, and the routes stored by simple nodes along reverse loop-free paths to local directories must be free of loops. On the other hand, the route from a given source node to a destination far away consists of two concatenated components. The first component is a loop-free route to a local directory, which is loop free. The second component is a route from a relay node within \( r \) hops to the destination that was obtained by the propagation of publication requests from the destination along the reverse loop-free routes to a local directory, which is also loop-free. Therefore, the theorem is true. \( Q.E.D. \)
Assume that APDV is executed in a connected network $G$ with a node set $N$, and further assume that topological changes stop taking place after a given time $t_T$.

**Theorem 4.2.3.** Each simple node in $G$ must be covered by at least $k$ directories located within $r$ hops of the node.

*Proof:* Because nodes communicate their directory lists persistently with every HELLO they transmit, all nodes must have a consistent view of their one- and two-hop neighbors within a finite time $t_H \geq t_T$, for otherwise at least one node is unable to receive the HELLOs from a neighbor. Because all nodes have loop-free routes to any selected directory (Theorem 2), $CT = DT^i \forall i \in N$ within a finite time after any node $v$ changes $DT^v$ using DAR or DDR.

Assume that a given node $u$ is uncovered and has only $k_u^u < k$ directories within $r$ hops at time $t_H$, while the rest of the nodes are covered. Node $u$ can only be a simple node or a directory at time $t_H$.

Consider the case that node $u$ is a directory at time $t_H$. Node $u$ cannot use DDR to stop being a directory at time $t_H$, because then node $u$ itself and all its one- and two-hop neighbors who are simple nodes must be covered by at least $k$ directories other than node $u$. Hence, if node $u$ is a directory at time $t_H$, it must remain being a directory indefinitely. This implies that the theorem is true in this case, because APDV ensures only that simple nodes are covered by $k$ directories within $r$ hops each.

Consider the case that node $u$ is a simple node at time $t_H$. It follows from DAR that node $u$ must become a directory, because $u = Min\{nid^u_j \forall j \in NT^u \mid (j \notin$
Given that \( u \) is the only node in its neighborhood that is not covered. Furthermore, any node \( v \) using DDR at time \( t_S > t_H \) as a result of node \( u \) becoming a directory must be covered when it becomes a simple node. Hence, the theorem is also true in this case. \( Q.E.D. \)

Let \( r \) represent the maximum distance between a simple node and any of its local directories, and \( D \) be the shortest distance between a node and an NDO it requests. The following theorem states an upper bound on the route stretch that can be incurred by APDV, which is defined as the ratio of the distance attained by APDV between a node and a remote NDO divided by the shortest distance \( D \).

**Theorem 4.2.4.** \( APDV \) incurs a route stretch smaller than or equal to \( 1 + 2r/D \).

**Proof:** Consider Fig. 4.3, which shows the case of a node \( s \) requesting an NDO from node \( d \) by using the route provided by APDV pointing to a local directory of \( d, l_d \). Node \( y \) is the first node along the path to \( l_d \) with a valid route to node \( d \), and it may be the case that \( y \) is in fact \( l_d \). The route attained with APDV is the concatenation of the route from \( s \) to \( y \) with the route from \( y \) to \( d \), with lengths \( d_{sy} \) and \( d_{yd} \), respectively. The length of the shortest path from \( s \) to \( d \) is \( D \). Accordingly, the route stretch in APDV
(S) equals

\[ S = \frac{d_{sy} + d_{yd}}{D} \]  \hspace{1cm} (4.1)

Because \( y \) must have a route to \( d \) that is at most \( r \) hops, \( d_{yd} \leq r \). Hence, by the triangle inequality it follows that \( d_{sy} \leq (d_{yd} + d_{sd}) \). This implies that \( d_{sy} \leq (r + D) \) and hence \( d_{sy} + d_{yd} \leq 2r + D \). The proof of the theorem follows by substituting this result in the expression for \( S \). \hspace{1cm} Q.E.D.

From the above, it is clear that the length of the routes attained in APDV tends to be the shortest-path length as \( \frac{r}{D} \) becomes smaller, which is the case of very large networks.

### 4.3 Performance Evaluation

We used QualNet [67] (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 4.1.

Table 4.1: Simulation Environment

<table>
<thead>
<tr>
<th>MAC Protocol</th>
<th>Simulation time</th>
<th>300s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data source</td>
<td>CBR</td>
<td>Data rate</td>
</tr>
<tr>
<td>Pkts. per flow</td>
<td>500</td>
<td>Flow duration</td>
</tr>
</tbody>
</table>

**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

### 4.3.1 Results for Static Networks

#### 4.3.1.1 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 [72]. 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.

![Figure 4.4](image)

Figure 4.4: 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.

The results for 5% of concurrent data flows are shown in Fig. 4.4(a)-(c). Fig. 4.4(a) shows that all protocols attain high delivery ratios when the network size is increased subject; however, APDV attains the highest delivery ratios. Fig. 4.4(b)
Figure 4.5: 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 4.6: 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 4.7: 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.
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. 4.4(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. 4.5(a)-(c) show the results for 10% traffic load. All three protocols perform similarly up to 200 nodes. Fig. 4.5(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. 4.5(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. 4.4(b) and 4.5(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. 4.5(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.

4.3.1.2 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 $1800 \times 1800$ m$^2$. The same CBR flow scheme described in Section 4.3.1.1 is used. The results are shown in Fig. 4.6(a)-(c). Fig. 4.6(a) shows that APDV and AODV attain similar packet delivery for 10 and 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. 4.6(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. 4.6(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.

4.3.2 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 4.1. Fig. 4.7(a)-(c) show the results of these experiments. Fig. 4.7(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. 4.7(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. 4.7(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.

4.4 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.
Chapter 5

Efficient Routing with Dominating Nodes

5.1 (k, r) Value Description

5.1.1 Overview

In APDV [51], we discussed that the different values of parameter $k$ (controllers listed in $LCL$) and $r$ (controllers in $LCL$ are within $r$ hops) may impact the performance of APDV. We are investigating the performance impact of those different values in this chapter.

When selecting controllers, the controllers redundancy is got by choosing a value for the parameter $k$ greater than one, which means one node can have more than one local controller in $LCL$ and when one local controller fails there is alternativity. In addition, more local controllers in $LCL$ may provide shorter path from source to
destination, therefore overall shorter delay in performance.

Meanwhile, the distance parameter $r$ allows increasing local availability by reducing the distance to the controllers. The larger value of $r$, the fewer local controller in the vicinity, which renders fewer total number of controllers in the network. Vise versa, the smaller value of $r$, the more local controller in the vicinity, as only very close by controllers can satisfy the distance requirement, which gives more total number of controllers in the network.

The two parameters can be adjusted appropriately according to the application requirements. We are interested to see the performance impact of different values of $k$ and $r$, and analyze the trend. As we can predict, the more controllers in the vicinity incurs smaller average delays.

### 5.1.2 Performance Optimization with Multi-Path

In APDV [51], there is only one local controller used to send Route Request or receive Route Reply or data packet for every local node, even though there are $k$ con-
controllers listed in $LCL$. By doing this, it is possible that the route from anchor controller to destination or from source to destination is not the shortest as the local controller of destination is the closest controller to destination but not the closest controller to anchor controller or source. In view of the current design of APDV, end-to-end delay may be effected by the selection of local controllers.

![Diagram of Multi-Path](image)

**Figure 5.2: Example of Multi-Path**

Fig 5.2 explains one example of the scenario that shorter path may exist in the network, but may not be selected by current design of APDV. In this example, node $s$ is the source of the data packet, and node $d$ is the destination of the data packet. Node $s$ has local controllers node $f$ and node $z$, and node $d$ has local controllers node $y$ and node $r$.

The local controllers of node $s$ and node $d$ are all one hop away, therefore in APDV both of them can be selected as the local controller to send *Route Request* or
receive Route Reply. However, if node z is selected by node s, when anchor controller a sends subscription reply to s, the route from a to z then to s is not the shortest path in current topology. If we consider the distance from anchor controller a to local controller f and from local controller f to node s, the total distance is shorter than the former path where z is the intermediate node.

When node s sends data packet to destination d, if r is selected as d’s local controller, the route is not the shortest path either. Instead, node y should be selected considering the distance from s to y and y to d. Therefore, the distance on both side of local controller should be considered when we select local controller for end node under the condition that we have multi-path available in APDV. The distance from node to its local controller is kept in publication packet and anchor request packet.

In order to study the performance impact of selection of different local controllers when we make forwarding decision, we introduce multi-path to APDV. Multi-path can provide APDV more choices when selecting forwarding path, which is often the shortest one, and we expect to see the decrease of end-to-end delay in overall performance.

5.2 Performance Evaluation and Analysis

We used QualNet [67] (version 5.0) as the discrete event simulator to evaluate the performance impact of parameters and multi-path on APDV with the performance of original APDV. 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. Each simulation ran for 10 different seed values.

5.2.1 Results for Static Networks

5.2.1.1 Impact of Increasing Parameter $k$

To evaluate the impact of increasing parameter $k$ on the performance of the protocol we used static topologies and an ideal physical layer in order to limit the impact of Multiple Access Interference (MAI) on the observed performance [72]. There are 400 nodes in the network.

To exercise the signaling of APDV, 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 capture the influence of traffic load, we used 5%, 10% and 20% of the total number of nodes for the sources with concurrent data flows.
Figure 5.3: Static network with increasing value of $k$: (a) Delivery ratio, (b) end-to-end delay of delivered data packets, (c) average number of control packets.
The results for 5%, 10% and 20% of concurrent data flows are shown in Fig. 5.3. Fig. 5.3(a) show that when \( r \) remains the same and \( k \) increases, the delivery ratio increases slightly. The reason is when \( k \) increases, there are more controllers in the vicinity, which provides higher possibility that the anchor controller of destination exists locally or more local controllers have routes to destination. The shorter path and more available routes provide higher delivery ratio.

Fig. 5.3(b) show that when \( r \) remains the same and \( k \) increases, the end-to-end delays of delivered packets decrease slightly. More controllers in the vicinity gives higher possibility that some of them have valid routes to destination, therefore data forwarding happens more smoothly, which obtains lower delay.

Fig. 5.3(c) show the average control overhead induced when \( r \) remains the same and \( k \) increases. From the result we can see there is no much difference, which means the control overhead induced by APDV does not increase a lot when traffic load increase and remains when \( k \) increases.

5.2.1.2 Impact of Increasing Parameter \( r \)

In this section, we present the results in the way that we can see the difference with increasing value of \( r \). The results are shown in Fig. 5.4.

The distance parameter \( r \) allows increasing local availability by reducing the distance to the controllers. The larger value of \( r \), the fewer local controller in the vicinity, which renders fewer total number of controllers in the network, vise versa.

Fig. 5.4(a) shows that when \( k \) remains the same and \( r \) increases, the delivery
Figure 5.4: Static network with increasing value of $r$: (a) Delivery ratio, (b) end-to-end delay of delivered data packets, (c) average number of control packets.
ratio does not vary too much. However, Fig. 5.4(b) shows that when $k$ remains the same and $r$ increases, the end-to-end delays of delivered packets increase slightly. Fewer controllers in the vicinity or in other words fewer controllers in the network makes every route look-up become longer and more network congestion may happen at anchor controllers. When $r$ is comparably large, one controller can cover more nodes, with distance vector routing we need the local controller to keep routing path to every node in the region, the routing table size increases as well. In addition, the longer path from local controller to nodes the longer delay in end-to-end communication as the path may not be the shortest path between source and destination.

From Fig. 5.4(c), we can see that the average control overhead induced when $k$ remains the same and $r$ increases does not vary a lot.

### 5.2.1.3 Impact of Multi-Path

In this scenario we investigate the performance impact of selection of different local controllers by local nodes, we introduce multi-path to APDV. We set $k$ to be 4, and increase $r$ to see the performance of APDV with impact of multi-path and increasing $r$. Nodes will register with at least four local controllers if they have enough nodes around. When nodes need to send packets to the local controller of source or destination, the sum of distance from themselves to the local controller and distance from the local controller to the source or destination is used as the criteria when we select which local controller to forward packets. The local controller who is on the shortest path is selected.

The results are shown in Fig. 5.5. The solid lines represent the results from
Figure 5.5: Static network with Multi-Path: (a) Delivery ratio, (b) end-to-end delay of delivered data packets, (c) average number of control packets.

multi-path, and the dotted lines are results from Fig. 5.4. Fig. 5.5(a) shows that when multi-path is available, when $k$ remains the same and $r$ increases, the delivery ratio does not vary too much.

Fig. 5.5(b) shows that when $k$ remains the same and $r$ increases, the end-to-end delays of delivered packets increase slightly. The reason was explained above. When we compare the results of single path scenario (dotted lines) with multi-path (solid lines) we can see that the end-to-end delays with multi-path decrease a little, as with multi-path APDV uses the shortest path to forward data and control packets, which provides shorter end-to-end delay.

5.2.2 Results for Mobile Networks

In this scenario we investigate the performance in mobile 400-node networks with the number of concurrent CBR sources increasing from 5% to 20% of the number of nodes.

The results of these experiments are shown in Fig. 5.6. The solid lines represent
Figure 5.6: Mobile network with Multi-Path: (a) Delivery ratio, (b) end-to-end delay of delivered data packets, (c) average number of control packets.

the results of mobile network with multi-path, and the dotted lines are results from static network with multi-path. Fig. 5.6(a) shows that when \( k \) remains the same and \( r \) increases, the delivery ratio decreases a little. The reason is that with the larger \( r \), controllers can not get updated information soon when nodes are moving. Some stale information makes the data packets transmission unsuccessful.

Fig. 5.6(b) shows the end-to-end delays of delivered packets increase slightly. And the reason is described before as longer path may be stored in controllers when nodes are moving.

From Fig. 5.6(c) we can see that the average control overhead does not vary much when \( k \) remains the same and \( r \) increases in the mobile network scenario.

5.3 Conclusion

From this study, we have investigated the performance impact of different values of \( k \) and \( r \) in APDV, and analyzed the trend. We have found that in static net-
works, more controllers in the vicinity provides better performance with higher delivery ratio, lower end-to-end delays of delivered packets and does not introduce more control overhead. On the other hand, less controllers in the vicinity incurs longer end-to-end delays.

In mobile networks, the situation is similar to the static networks. Larger $r$ brings less controllers in the vicinity, which incurs longer end-to-end delays. And because of nodes movement the delivery ratio decreases a little.

With the multi-path selection, APDV incurs shorter end-to-end delays of delivered packets compared with single path scenario.

We conclude that the different values of $k$ and $r$ do have impact on performance. Therefore, the two parameters can be adjusted appropriately according to the application requirements, and multi-path selection provides better performance in terms of end-to-end delays.
Chapter 6

Content Routing in MANETs

Given the limitations of traditional routing to destination addresses, many proposals have been made for information centric networking (ICN) [22] in which content is routed by names and routers cache content opportunistically. However, relatively few results have been reported on routing of named data objects (NDO) in mobile ad hoc networks (MANET). The challenges posed by node mobility have been considered [24]; however, few proposals have been made to support name-based routing of content in wireless networks. Interestingly, the protocols proposed to date for routing to NDOs in MANETs rely on the establishment of overlays, the flooding of link-state information, or network-wide dissemination of content requests. These approaches incur excessive signaling overhead as the number of network nodes or NDOs increase. We introduce the Adaptive Publish-subscribe Distance Vector (APDV) protocol for routing of NDOs in MANETs. APDV is a new approach to routing [51] in MANETs that eliminates most of the flooding needed for signaling.
The following section describes APDV as it applies to routing of NDOs. It consists of three components: (a) electing a subset of nodes to serve as directories that maintain routes to nearby destinations, (b) maintaining routes to all known directories using distance vectors, and (c) using publish-subscribe mechanisms with which directories are informed of routes to NDOs and consumers obtain routes to NDOs from directories.

6.1 APDV for Routing of NDOs in MANETs.

APDV (Adaptive Publish-Subscribe Distance Vector) assumes that each network node has a network-wide unique name, and each piece of content is an NDO. The names assigned to NDOs are structured, with each name of an NDO containing a publisher name component that uniquely denotes a publisher, and an object name component that is unique for a given publisher. For example, “soe.ucsc.edu” in the name “soe.ucsc.edu/papers/apdv-NOM13 paper.pdf” is the name of the publisher of the NDO.

A subset of routers are selected dynamically to serve as directories. A directory maintains the routes to nodes nearby, as well as some mappings of NDO names to the names of nodes. The distributed algorithm used to select directories ensures that each non-directory node is within a maximum distance $r$ from a minimum number $k$ of local directories for the node. Fig. 6.1 illustrates the basic operation of APDV assuming that each node has at least one local directory within two hops. In the example of Fig. 6.1,
nodes $a, k, m, x$ and $r$ are the elected directories of the network.

![Figure 6.1: Example of operation](image)

All network nodes maintain loop-free routes to all directories using sequence-numbered distances. For simplicity, we assume that a node maintains a single route to each directory. Each node contacts each of its local directories to publish its presence. To do this, node $d$ sends a publish message to each of its local directories with the mapping $(d, \{l_1^d, ..., l_k^d\})$, where $l_i^d$ ($1 \leq i \leq k$) is a local directory for node $d$. Each local directory $l_i^d$ of node $d$ and each relay between $d$ and the directory receiving the publish request from $d$ stores a tuple stating $d$, the next hop to $d$, and $\{l_1^d, ..., l_k^d\}$.

A node $d$ uses a common hash function on the publisher portion of the name of an NDO it stores to publish it with an anchor directory. The anchor directory is the directory whose name is the closest match to the hash of the NDO name among all directories. A node may publish an NDO individually or may publish all NDOs sharing
a given name prefix using a single entry. For convenience, we use the term NDO to
denote an individual NDO or a name prefix.

Node $d$ provides the anchor directory of the NDO it publishes using name $ndo$
with the tuple $(ndo : d, \{l_{d_1}^1, ..., l_{d_k}^k\})$, which maps the name of the NDO to the name of a
node storing it and the local directories near the storage node. Relay nodes between $d$
and anchor directories may cache the information published by $d$ for a period of time.
In the example of Fig. 6.1, node $d$ has two NDOs, it publishes its presence with its local
directory (node $r$), and publishes NDOs $o_1$ and $o_2$ with node $a$, which is the anchor
directory for both NDOs. Directory $r$ has a route to node $d$, while directory $a$ has a
route to directory $r$ and the mappings $(o_1 : \{(d, r), (l, \{m, x\})\})$, and $(o_2 : (d, r))$. A
node requiring NDO $o_i$ uses the common hash function on the publisher’s portion of the
name of $o_i$, and sends a subscription request to the resulting anchor directory. In turn,
the anchor directory for NDO $o_i$ replies with the mapping of $o_i$ to the nodes that store it
and the local directories near those nodes. In the example of Fig. 6.1, a node requesting
NDO $o_1$ would contact node $a$ and receive the mapping $(o_1 : \{(d, r), (l, \{m, x\})\})$.

### 6.1.1 Routing Information

Node $i$ maintains two tables with information about directories. A directory
table ($DT^i$) states, for each directory $c$ in the network, its name ($n_i^c$); the distance from
$i$ to $c$ ($d_i^c$); the next hops from $i$ to $c$ ($s_i^c$); and a sequence number ($sn_i^c$) creed by $c$ and
used to avoid routing loops. Node $i$ also maintains a neighbor directory table ($NDT^i$),
which stores the directory tables reported by each neighbor of node $i$. The entry for
directory $c$ reported by neighbor $j$ and stored in $NDT^i$ is denoted by $\{n^i_{cj}, d^i_{cj}, sn^i_{cj}\}$.

Node $i$ maintains a neighbor table ($NT^i$) with information about neighboring nodes and used to select those nodes that should be directories. For each neighbor $j$ of $i$, $NT^i$ stores: the name of the node ($n^i_j$); a sequence number ($sn^i_j$) created by $j$ and used to determine that the entry is the most recent from node $j$; a directory status flag ($ds^i_j$) stating whether node $j$ is a directory; the directory counter ($k^i_j$) stating the number of directories within $r$ hops of node $j$; and the local directory list ($LDL^i_j$) consisting of the names of all directories within $r$ hops of node $j$.

A node transmits HELLOs periodically every few seconds and a HELLO includes some or all the updates made to its 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. Node $i$ maintains routing information about NDOs and nodes in a routing table ($RT^i$) based on HELLOs containing publication and subscription requests it receives from neighbors. $NT^i$, $DT^i$, $NDT^i$, $LDL^i$, and $RT^i$ are updated by the exchange of HELLOs among neighbors.

An entry for node $j$ listed in $RT^i$ specifies: the name of the node ($n^i_j$); a sequence number ($sn^i_j$) created by $j$ and used to avoid routing loops; the distance from $i$ to $j$ ($d^i_j$); the successor in the route to $j$ ($s^i_j$); and the local directory list of node $j$ ($LDL^i_j$), which may be a link to $NT^i$ if the node is within two hops. An entry for NDO $o_j$ listed in $RT^i$ specifies: the name of the NDO, the name of the node that publishes it ($j$); and the local directory list of node $j$ ($LDL^i_j$), which may be a link to $NT^i$ if the node is within two hops.
Node $i$ includes its own information in $NT^i$, i.e., it stores an entry corresponding to $n^i_i$, and uses the information in its HELLOs. A HELLO from node $i$ contains: $n^i_i$, $sn^i_i$, $ds^i_i$, $k^i_i$, and updates to $NT^i$ and $DT^i$. An update to $NT^i$ regarding neighbor $j$ consists of the tuple $\{n^i_j, sn^i_j, ds^i_j, k^i_j, LDL^i_j\}$. An update to $DT^i$ regarding directory $c$ consists of the tuple $\{n^i_c, d^i_c, sn^i_c\}$. An entry for neighbor $v$ in $NT^j$ sent in a HELLO to node $i$ is denoted by $\{n^i_{vj}, sn^i_{vj}, ds^i_{vj}, k^i_{vj}, LDL^i_{vj}\}$, and the same entry stored in $NT^i$ is denoted $\{n^i_{ij}, sn^i_{ij}, ds^i_{ij}, k^i_{ij}, LDL^i_{ij}\}$.

6.1.2 Building Path to Directories

Nodes select a dominating set $C$ of nodes that serve as directories, such that every node $u \notin C$ (called simple node) is at a distance smaller than or equal to $r$ hops from at least $k$ directory nodes in $C$. A node $u$ is said to be $(k, r)$ dominated (or covered) if there are at least $k$ directories in $C$ within $r$ hops from $u$. The directory selection scheme is based on HELLO messages exchanged among one-hop neighbors. To keep the selection algorithm and signaling simple, only distances to directories and node names are used as the basis for the selection of directories.

6.1.2.1 Selecting Directories

Nodes self-select themselves to become or stop being directories. A given node $i$ determines to add or delete its own entry in $DT^i$ according to the Directory Addition Rule (DAR) and directory Deletion Rule (DDR) defined below.

Node $i$ is initialized with $DT^i = \phi$ and $NT^i = \phi$, and waits for a few seconds to
start receiving HELLOs from nearby nodes before selecting directories. Hence, according to DAR, node $i$ selects itself as a directory when it is first initialized, unless it receives HELLOs from neighbors that prompt it not to include itself as a directory based on DDR.

Let the number of entries in a list $L$ be donated by $|L|$, and the lexicographic value of a name $n$ be donated by $|n|$.

Once node $i$ has updated $NT^i$ and $DT^i$ by processing the HELLOs from its neighbors, it computes $LDL^i$ from $DT^i$, such that $v \in LDL^i$ if $d_{iv} \leq r$, and sets $k^i_i = |LDL^i|$.

**DAR (Directory Addition Rule):**

Node $i$ adds itself to $DT^i$ if

$$(k^i_i < k) \land [ |i| = \text{Min}\{|n^i_j| \forall j \in NT^i \mid (j \notin DT^i) \land (k^i_j < k)\} ]$$

**DDR (Directory Deletion Rule):**

Node $i$ deletes itself from $DT^i$ if

$$(k^i_i > k) \land [ \forall j \in NT^i \{ (|LDL^i_j| - |i|) \geq k \land \forall j \notin LDL^i_j \} \land

\{ |n^i_j| < |n^i_i| \forall j \in LDL^i_j \} ]$$

### 6.1.2.2 Routing to Directories

For simplicity, we assume that each node maintains a single route to each directory selected in the network using the updates to directory tables included in HELLOs.

To guarantee loop-free routes, APDV uses sequence numbers that restrict the
selection of next hops towards a given directory by any node. Only those neighbors with shorter distances to a directory or with a more recent sequence number reported by it can be considered as next hops to the directory. An important aspect of APDV is that entries for directories can be deleted on purpose as a result of DDR, rather than only as rare occurrences due to failures or network partitions. Together with the transmission of periodic HELLOs, the Reset Directory Rule (RDR) and the Update Directory Rule (UDR) discussed below address this functionality.

Let $N^i$ be the set of one-hop neighbors of node $i$. Node $i$ updates $DT^i_j$ 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 $DT^i_j$ are deleted. Once node $i$ is selected as a directory, it is the only node that can change the sequence number for its own entry in directory-table updates sent in HELLOs.

When node $i$ decides to delete itself as a directory based on DDR, its entry must be deleted in the rest of the network. Node $i$ uses RDR 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 $DT^i_i$ to ensure that the rest of the nodes delete the entry for $i$ in their directory tables. If node $i$ receives a HELLO from $j$ or experiences a link failure that makes it update $DT^i_j$ for entry $c \neq i : \{n^i_{cj}, d^i_{cj}, s^i_{cj}\}$, node $i$ updates its entry for $c$ in $DT^i_i$ according to UDR, which forces node $i$ to propagate a reset update or to select a successor to directory $c$ that is either closer to $c$ or has reported a more recent sequence number from $c$.

**RDR (Reset Directory Rule):**
If node $i$ must delete itself from $DT_i$ using DDR then

$$d_i' = \infty; \quad sn_i' = sn_i' + 1; \quad reset-timer_i = T$$

**UDR (Update Directory Rule):**

If ( $\exists q \in NT_i \mid sn_{cq} > sn_{c}$ )

then begin

if ( $(v = s_{ic}) \land (sn_{cv} > sn_{ic}) \land (d_{ic} = \infty)$ )

then set $sn_{ic}' = sn_{cv}'$; $d_{ic}' = \infty$

else begin

set $d_{ic}' = Min\{d_{ic}'+1 \mid (f \in NT_i) \land (sn_{cf} = Max\{sn_{cv} \mid v \in NT_i\})\}$

set $s_{ic}' = j \mid (j \in NT_i) \land (d_{ic}'' = d_{ic}' - 1)$

set $sn_{ic}' = Max\{sn_{cv} \mid v \in NT_i\}$;

else begin

set $d_{ic}' = Min\{d_{ic}'+1 \mid (f \in NT_i) \land (sn_{cf} = sn_{ic}') \land (d_{ic}' < d_{ic}''\})$; $sn_{ic}' = Max\{sn_{cv} \mid v \in NT_i\}$;

set $s_{ic}' = j \mid (j \in NT_i) \land (d_{ic}'' = d_{ic}' - 1)$

6.1.3 Publish-Subscribe Mechanisms for NDOs

Nodes learn about routes to directories and to one- and two-hop neighbors, but have no routes to NDOs. To allow nodes to obtain routes to arbitrary NDOs without incurring network-wide dissemination of signaling messages, APDV uses a publish-subscribe mechanism for name-to-route resolution.
6.1.3.1 Publishing NDOs

Publishing an NDO in APDV consists of making one or more local directories know the route to the node storing the NDO and having an anchor directory know the mapping from the name of the NDO to the name of the storing node and its associated local directories.

Node $i$ publishes itself with the $k$ directories listed in $LDL^i$. The local directories in $LDL^i$ are within $r$ hops of node $i$ and serve as the “landmarks” for other nodes to communicate with node $i$, given that nodes far away from node $i$ do not have routes to node $i$. Accordingly, a local directory for node $i$ must maintain a route to node $i$, and it also maintains the mapping $(i, LDL^i)$, so that it can find alternate ways to reach node $i$ if its route to $i$ fails.

The anchor directory for NDO $o_i$ (denoted $a_{o_i}$) is obtained by using a network-wide consistent hash function that maps the publisher name component of the name of the NDO into the name of one of the directories selected in the network, which are listed in the directory table. Directory $a_{o_i}$ must store the mapping $(o_i, LNDO_i)$, where $LNDO_i$ is a list of tuples corresponding to the nodes that have published NDO $o_i$. Each tuple specifies the name of a node storing the NDO and the local directory list (LDL) of the node. This allows a node $v$ that obtains $LNDO_i$ to request NDO $o_i$ by sending its request towards the nearest local directory (according to its directory table) of a node storing NDO $o_i$.

Nodes can publish themselves by using their names to select the anchor di-
rectories using the network-wide consistent hashing function. The anchor directory for node \( i \) (\( a_i \)) stores the mapping (\( i, LDL^i \)), so that it can provide any node \( v \) far away from node \( i \) the list \( LDL^i \), with which node \( v \) can send data packets towards a directory in \( LDL^i \) that is nearest to node \( v \).

The forwarding of a publication request from a node to its local directories is done by the exchange of HELLOs. Given that nodes maintain loop-free routes to all directories, publication requests are forwarded over the reverse loop-free routes already established from directories to nodes. The routes maintained by local directories to nearby nodes are refreshed periodically; each node creates a new publication request by increasing the sequence number 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 \( v \) (\( \{nid_v^j, sn_v^j, d_v^j, LDL_v^j\} \)), then node \( i \) forwards the request (i.e., it includes the \( LRT^i \) entry \( \{nid_v^i, sn_v^i, d_v^i, LDL_v^i\} \) in its own HELLO) if it is the successor for node \( j \) to any of the directories listed in \( LDL_v^j \).

Once a local directory \( c \) receives an entry for destination \( v \) and \( c \in LDL_v \), then \( c \) publishes (i.e., stores) the entry \( \{nid_v^c, sn_v^c, d_v^c, s_v^c, LDL_v^c\} \), where \( s_v^c \) is the neighbor from which it received the publication request. Directory \( c \) may also forward it if it is the successor to another directory in \( LDL_v \) for the neighbor from which it received the publication request.

Node \( i \) can publish itself as a destination with its anchor directory \( a_i \) by node \( i \) using the network-wide consistent hash function to map its name to the name of one
of the directories in $DT^i$ to obtain $hash(i) = a_i$, where $a_i \in DT^i$. After that, node
$i$ sends a publication request to its successor towards its anchor directory $a_i$ with the
tuple $\{n_i^i, sn_i^i, d_i^i, LDL_i^i\}$. Each node $v$ in the route from node $i$ to directory $a_i$ forwards
the publication request towards $a_i$ and caches the tuple $\{n_v^i, sn_v^i, d_v^i, s_v^i, LDL_v^i\}$. Once
directory $a_i$ receives the request, it stores the tuple $\{n_{a_i}^i, sn_{a_i}^i, d_{a_i}^i, s_{a_i}^i, LDL_{a_i}^i\}$. Hence,
each node processing a publication request learns the route to the node issuing the
request, and the anchor directory is able to obtain the mapping needed to redirect
nodes sending subscription requests to the local directories of node $i$.

Similarly, node $i$ publishes NDO $o_k$ with an anchor directory using the network-
wide consistent hash function on the set of directories in $DT^i$ to obtain $hash(o_k) = a_{o_k}$,
where $a_{o_k} \in DT^i$. Node $i$ then sends a publication request to its successor towards $a_{o_k}$
with the tuple $\{o_k, n_i^i, LDL_i^i\}$, where $n_i^i$ is node $i$’s name. Each node $v$ in the route
from node $i$ to directory $a_{o_k}$ forwards the publication request towards $a_{o_k}$ and caches
the tuple. Once directory $a_i$ receives the request, it adds the mapping $(n_{a_i}^i, LDL_{a_i}^i)$ and
adds $n_i^i$ to the list of nodes storing $o_k$. Hence, $a_{o_k}$ learns over time the nodes that store
NDO $o_k$ and the local directories of those nodes.

6.1.3.2 Subscribing and Routing to NDOs

Subscribing to an NDO in APDV consists of a node requesting a way to reach
an NDO by contacting the anchor directory of the NDO, and the anchor directory pro-
viding the requesting node with a list of nodes storing the NDO and the local directory
lists for those nodes. The forwarding of subscription requests is handled in much the
same way described above for the case of publication requests.

Node $p$ requests NDO $o_k$ by computing $\text{hash}(o_k) = a_{o_k}$, where $a_{o_k} \in DT^p$ and sends its subscription request towards $a_{o_k}$. The subscription request states the name of NDO ($o_k$), the anchor directory ($a_{o_k}$), and the list of local directories for node $p$ ($LDL^p$). Directory $a_{o_k}$ responds to node $p$ with the mapping of $o_k$ to the list of names of nodes storing the NDO and their associated local directory lists. The response is sent to the nearest directory it finds in $LDL^p$.

Similarly, if node $p$ has data for destination node $j \notin DT^p$, it computes $\text{hash}(j) = a_j$, where $a_j \in DT^p$ and sends its subscription request towards $a_j$. The subscription request states the identifier of node $j$, its anchor directory $a_j$, and $LDL^p$. Directory $a_j$ responds with the tuple $\{n_j^{a_j}, sn_j^{a_j}, LDL_j^{a_j}\}$ and sends the response to the nearest directory it finds in $LDL^p$. Once node $p$ receives the reply to its subscription, it stores the tuple $\{n_j^p, sn_j^p, LDL_j^p\}$ in $RT^p$. Data packets from $p$ are then sent towards the directories in $LDL_j^p$ that are the closest to node $o$. A data packet must specify the sender, the destination, and the selected local directory 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 directory 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.
6.2 Performance Evaluation

We compare APDV against representative approaches of content routing protocols proposed for MANETs, with all the protocols using opportunistic content caching at each router. To isolate the impact of node mobility from content mobility (i.e., caching), and due to space limitations, we assume static network topologies in this paper. We made this choice given that prior results [51] show that APDV performs far better than traditional routing protocols (AODV [61] and OLSR [44]) in large MANETs.

We implemented AODV-NDO by modifying the AODV QualNet implementation [67] as an example of on-demand content routing. AODV-NDO operates like DIRECT [66] and LFBL [58], with routers flooding requests for NDOs on demand, followed by the delivery of NDOs from routers with local copies of the NDOs. We also implemented OLSR-NDO by modifying the OLSR QualNet implementation [67] as an example of proactive content routing. OLSR-NDO operates like NLSR [20], with routers flooding link state advertisements for physical links as well as for NDOs stored locally.

The data plane used for both OLSR-NDO and APDV is the same as that is advocated in recent ICN architectures [22]. An NDO request is forwarded towards the nearest router that has advertised the NDO, and the NDO is sent back over the reverse path by the first router with a local copy of the NDO being requested.

We focused on the average number of control packets generated by the routing protocols as the performance metric. All protocols use the same time period to refresh their routing structures. For APDV we used $k = 2$ and $r = 3$ to select directories.
Table 6.1: Signaling Overhead for Routing to NDOs

<table>
<thead>
<tr>
<th>Protocol</th>
<th>10%</th>
<th>20%</th>
<th>40%</th>
</tr>
</thead>
<tbody>
<tr>
<td>APDV</td>
<td>162.73</td>
<td>176.04</td>
<td>199.18</td>
</tr>
<tr>
<td>AODV-NDO</td>
<td>157.85</td>
<td>1041.90</td>
<td>26591.93</td>
</tr>
<tr>
<td>OLSR-NDO</td>
<td>698.68</td>
<td>1441.54</td>
<td>2268.49</td>
</tr>
</tbody>
</table>

Each simulation ran for 10 different seed values. We used static networks of 100 nodes, with nodes being uniformly distributed in the network to avoid disconnected nodes. The routing protocols were tested using the IEEE 802.11 DCF as the underlying MAC protocol, and all signaling packets are sent in broadcast mode. We increased the number of nodes requesting NDOs from 10% to 40% of the 100 nodes of a static network. Nodes are uniformly distributed in a simulation area of 1800×1800 m². Each node originally publishes 10 NDOs, with the name of each NDO consisting of a hierarchical name including the name of its publisher and the hashing to find anchor directories is based on the publisher name of the NDO name.

Table 6.1 shows the results of the simulation experiment. It is clear that, as the rate of content requests increases, the signaling incurred in the flooding of content requests (AODV-NDO) or the updating of all nodes on the locations of new copies of NDOs (OLSR-NDO) becomes excessive. On the other hand, APDV signaling overhead increases sub-linearly with the rate of content requests because the use of directories eliminates most of the flooding in the MANET.
6.3 Conclusion

We introduced the Adaptive Publish-subscribe Distance Vector (APDV) protocol to provide efficient routing to named data objects in MANETs using distance vectors. We used a simulation experiment to compare its performance with the performance of on-demand and proactive approaches to name-based routing in wireless networks. The key reason why APDV outperforms the other protocols is that it eliminates most flooding by using directories. More work is needed to fully understand and exploit the approach advocated in APDV. One important aspect is the fact that many NDOs need to be known only within a given horizon, and hence a hierarchy of directories can be used to allow NDOs to be known locally, regionally or globally. The combined effect of node and content mobility must also be analyzed.
Chapter 7

Content Oriented Routing with
Directories

The shift in Internet usage patterns from host-to-host communication to peer-to-peer and user-generated content has led to several information centric network (ICN) architectures being proposed [21, 27, 25] as alternatives to the current Internet architecture. All ICN architectures aim at accessing content and services by name, independently of their location, in order to improve system performance and end-user experience.

Most ICN architectures rely on name-based routing of content, which integrates name resolution and content routing. Routers advertise or compute routes to NDOs or name prefixes, and content requests for specific NDOs are forwarded towards the nearest routers storing those NDOs. As our review of prior work indicates, prior content routing approaches assume that the entire ICN use the same naming space for NDOs and that
routing tables list routes to NDOs or name prefixes, which incurs much more overhead than routing to address ranges.

7.1 CORD Description

CORD (Content Oriented Routing with Directories) is an approach to content routing within autonomous systems in which directory nodes act as intermediaries to establish virtual cords linking consumers of content with content producers or caching sites. The primary objective of using directories between content producers or caches and the consumers of content is to reduce control overhead in the ICN. Instead of having routing tables listing routes to individual NDOs or name prefixes, they only list routes to the directories that maintain the mappings between name prefixes or NDO names and the locations where their copies reside. This type of indirection in routing was first discussed by McQuillan [57] in the early days of the ARPANET.

CORD consists of three main elements: (a) maintaining multiple loop-free routes to directories that maintain the mappings from NDOs and name prefixes to the addresses of sites storing the content; (b) maintaining loop-free routes from directories to destination nodes nearby; and (c) publish-subscribe mechanisms for publishers and consumers of content to advertise and request content.

7.1.1 Basic Operation

CORD assumes that: (a) each router or host is assigned a flat or hierarchical name; (b) each piece of content is a named data object (NDO) that can be requested
by name; (c) NDOs can be denoted using flat or hierarchical naming, with multiple naming conventions possibly being used in the same ICN; and (d) routers cache content opportunistically.

All routers maintain multiple loop-free routes to directories using sequence-numbered distances. Router \(d\) maintains this information in its directory table \((DT)^d\), and uses it to contact a subset of directory nodes within a maximum distance \(r\) from itself to publish its presence. To do this, router \(d\) sends a publish message to its selected local directories with the mapping \((d, \{l_{d1}, ..., l_{dk}\})\), where \(l_{di} (1 \leq i \leq k)\) is a local directory for router \(d\). Each local directory \(l_{di}\) of \(d\) and each relay between \(d\) and the directory receiving the publish request from \(d\) stores a tuple stating \(d\), the next hop to \(d\), and \(\{l_{d1}, ..., l_{dk}\}\). Router \(i\) maintains a local directory list \((LDL)^i\) stating the information about its local directories. In addition, each router \(i\) maintains a neighbor table \((NT)^i\) stating routing information communicated by its neighbor routers, and a content store table \((CST)^i\) listing all content cached by \(i\). Directories also maintain routing information for those routers that select them as local directories.

The anchor of a name prefix or NDO is a directory responsible for maintaining the mappings between the name prefix or NDO to the locations where copies of the prefix or NDO are stored. Directories advertise to the entire network the name prefixes and intervals of NDOs for which they serve as anchors, and hence all routers know which directory to contact regarding any NDO or name prefix.

Routers exchange control information using HELLO messages sent periodically every 3 seconds. A HELLO includes some or all the updates made to the sending node’s
Figure 7.1: Example of CORD operation

tables. Each node stores all the information from the HELLOs it receives from its neighbors, and also caches content it receives. Entries in $CST^i$ are populated by the publish-subscribe signaling described subsequently.

Fig. 7.1 illustrates the basic operation of CORD. In this example, nodes $a$, $k$, and $r$ maintain directories, and all nodes maintain routes to such directories.

In contrast to the content routing approaches in prior ICN architectures, hierarchical naming and flat naming can be used in the same ICN running CORD. This is attained by stating the name space in which a prefix name or NDO name is defined as part of the advertisements sent by directories, as well as the publish-subscribe requests exchanged with directories.

The data plane in CORD is slightly different than many prior approaches, in that it is assisted by the publish-subscribe signaling between routers and directories to support content requests without routing tables listing entries for NDOs or name
prefixes. The router of the producer or a caching site of an NDO publishes the local copy of the NDO by sending a publish message to its local directories and the anchor of the NDO. The message states the namespace used, the NDO name in that namespace, the router identifier, and its local directories.

A consumer of content asks for an NDO by sending a content request to its local router. In turn, the router follows a two-step process to request the NDO. To request content, a router first sends a subscription request to one or more of its local directories. If the local directory or directories cannot provide a mapping, the router sends its subscription request to the known anchor of the NDO, as exemplified in Fig. 1. The subscription request specifies the namespace used, the NDO name, the identifier of the consumer, and the local directories for the requesting router. A directory with the requested mapping sends the subscription reply to one of the local directories of the requesting router, and the reply is sent to the requesting router from that local directory or a router with a route to the requesting router.

Once a router receives a subscription reply from a directory, it knows the names or identifiers of one or multiple sites hosting the NDO. It can then select the site whose local directory is closest, and can send a content request based on the mechanisms defined for the data plane of the network. The operation of CORD in the control plane is independent of the data plane mechanisms once a router obtains the identifier or name of a site hosting the required content. In the data planes assumed in most prior ICN architectures (e.g., [31, 4]), an NDO request specifies the name of the NDO, does not state the name of the requestor, and is forwarded by routers towards the nearest site
known to store the NDO. The NDO is sent back to the consumer by the NDO producer or a caching site over the reverse path traversed by the content request.

Hence, CORD supports name-based content routing with routers having to know only how to reach directories, and directories having to know how to reach some routers.

7.1.2 Updating Routing Information

CORD uses a distance-vector routing approach to maintain routes to directories. To guarantee loop-free routes, CORD uses sequence numbers that restrict the selection of next hops towards a given directory, such that only those neighbors with shorter distances to the directory or with a more recent sequence number reported by the directory can be considered as successors. Algorithm 1 (DSU) is used to update routing information for directories.

**Algorithm 5 DSU: Directory Status Update**

1: Input: $DT^i, NT^i$;
2: if $\exists q \in NT^i \mid sn_{cq}^i > sn_{c}^i$ then
3: if $(v = s_c^i) \land (sn_{cv}^i > sn_{c}^i) \land (d_{cv}^i = \infty)$ then
4: $d_{c}^i = \infty; sn_{c}^i = sn_{cv}^i$;
5: else
6: $d_{c}^i = \min\{d_{cf}^i + 1 \mid (f \in NT^i) \land (sn_{cv}^i = \infty) \}$;
7: $s_c^i = j \mid (j \in NT^i) \land (d_{cj}^i = d_{c}^i - 1)$;
8: $sn_{c}^i = \max\{sn_{cv}^i \mid v \in NT^i\}$;
9: end if
10: else
11: $d_{c}^i = \min\{d_{cf}^i + 1 \mid (f \in NT^i) \land (sn_{cv}^i = sn_{c}^i) \land (d_{cf}^i < d_{c}^i) \}$;
12: $s_c^i = j \mid (j \in NT^i) \land (d_{cj}^i = d_{c}^i - 1)$;
13: end if
Let $N^i$ be the set of one-hop neighbors of node $i$. Node $i$ updates $DT^i_j$ 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 $DT^i_j$ are deleted. The node, which is predefined as a directory, is the only one that can change the sequence number for its own entry in directory table updates sent in HELLOs.

If node $i$ receives a HELLO from $j$ or a link failure occurs that makes it update $DT^i_j$ for entry $c \neq i : \{nid^{i}_{cj}, d^{i}_{cj}, sn^{i}_{cj}\}$, node $i$ updates its entry for $c$ in $DT^i$ according to Algorithm 1 (DSU), which forces node $i$ to propagate a reset update or to select a successor to directory $c$ that is either closer to $c$ or has reported a more recent sequence number from $c$.

### 7.1.3 Mapping of Content to Directories

All routers maintain multiple loop-free routes to directories using sequence-numbered distances. The number of directories in the network is related to the network size and traffic load, which means the larger network and heavy traffic load the more directories are needed in the network to support scalability and provide highly efficient query processing. With the utilization of consistent hashing, changes of directories size do not affect the correctness and efficiency in CORD. CORD allows directories to announce to the entire network the name prefixes or intervals of NDOs for which they serve as anchors through the HELLO messages they send periodically. A router with content to publish extracts the name space of the NDO, and finds the corresponding anchor from its $DT^i$.
If hierarchical names are used, directories send updates about the list of the name prefixes for which they are anchors. For instance, a directory could announce being the anchor for the name prefix “/ucsc/ccrg/*” and a content request for the NDO with name “/ucsc/ccrg/liqian/paper/CORD.pdf” would be sent to that anchor.

If flat names are used, directories announce the range of NDO identifiers they serve. Assuming an NDO has flat name 100, a directory could advertise a range from 0 to 1,000. Fig. 7.2, 7.3 illustrate the consistent mapping to directories using NDO’s hierarchical name and flat name when publishing and subscribing to content.

![Diagram of mapping of hierarchical name](image)

### 7.1.4 Publish-Subscribe Mechanisms

#### 7.1.4.1 Publishing Content

Publishing in CORD consists of having a few local directories know the routes to a given node storing the NDOs in a name prefix or a given range of flat names, and
having an anchor directory know the mapping from an NDO range or name prefix to a list of local directories.

**Algorithm 6 Publishing to Directories**

1: Input: $o^k, DT^i, LDL^i, CST^i, NT^i$;  
2: if $o^k \notin CST^i$ then  
3: /* data object $o^k$ is new to node $i$ */  
4: $CST^i \leftarrow \{o^k\}$;  
5: if $i$ is origin of $o^k$ then  
6: $i$ publish $o^k$ to $a_k$ with $(i \leftarrow \{o^k\}, LDL^i)$;  
7: /* $a_k$ is the anchor directory for $o^k$ */  
8: end if  
9: $i$ publish $o^k$ to $l_k = \text{hash}(o^k)$ with $(d \leftarrow \{o^k\}, i, LDL^i)$;  
10: /* $l_k$ is the selected local directory from $LDL^i$ for $o^k$ */  
11: end if

Each non-directory node $i$ publishes itself with the $k$ directories listed in its local directory list ($LDL^i$). If node $i$ is attached to a content producer, it also publishes the existence of the content with one or more anchor directories, and with its local directories. The local directories in $LDL^i$ are within $r$ hops of node $i$ and serve as the
“landmarks” for other nodes to reach node $i$, given that nodes far away from node $i$ do not have routes to node $i$. Accordingly, a local directory for node $i$ must maintain updated routes to node $i$, and it also maintains the mapping $(i \leftarrow \{o^1_i, ..., o^n_i\}, LDL^i)$, so that it can find alternate ways to reach node $i$ if its route to $i$ fails, and it can resolve subscription requests for content stored at $i$. The anchor directories are needed for nodes far away from content to obtain the mappings between the content and the local directories of the original content producer as a way to obtain the content. Algorithm 2 explains when and how to publish content to directories. For simplicity, we assume that a single anchor directory is selected for any one original content producer.

The forwarding of a publication message from a node to its local directories is done by the exchange of HELLOs. The routes maintained by local directories to nearby nodes are refreshed periodically based on a HELLO interval. A node that is the original source of an NDO publishes the existence of the NDO by sending a publication message to the anchor directory known to be in charge of the name prefix or NDO range to which the NDO belongs. An anchor directory stores the mapping from the identifier of a node $i$ to the name prefixes or NDO ranges corresponding to NDOs stored at node $i$, and the list of local directories for node $i$ $(LDL^i)$. Nodes caching an NDO do not publish the NDO with the anchor directory of the NDO; they simply inform their local directories.

The submission of a publication message from node $i$ to an anchor directory regarding NDO $o^k_i$ is done by node $i$ sending the message with the mapping $(i \leftarrow \{o^k_i\}, LDL^i)$ towards anchor $a$. Each node $v$ in the route from node $i$ to directory $a$
forwards the publication packet towards \(a\) and caches the mapping. Hence, the anchor directory and each node processing a publication message is able to redirect nodes sending subscription requests for NDO \(o^k_i\) to the local directories of node \(i\).

### 7.1.4.2 Subscribing to Content

When node \(t\) needs to request NDO \(o^k\), it first sends a subscription request to a local directory in \(LDL^t\), which is selected using a hash function that computes \(\text{hash}(o^k) = l_t\), where \(l_t \in LDL^t\), and sends its subscription request towards \(l_t\).

If directory \(l_t\) has received publication from local nodes regarding \(o^k\) before, it replies with the identifier of a node \(p\) where \(o^k\) exists, as well as the local directories of \(p\). If \(l_t\) does not know about \(o^k\), it sends a negative reply to node \(t\). At that point, node \(t\) sends its request to the anchor directory \(a_k\) for \(o^k\), based on the name prefixes and NDO ranges advertised by anchor directories. In turn, anchor directory \(a_k\) responds with the mapping \((d \leftarrow \{o^k_d\}, LDL^d)\) towards the nearest local directory of node \(t\) selected from \(LDL^t\). The answer is redirected to \(t\) by either the selected directory \(l^t_d\) or the first relay node along the path from \(a_k\) to directory \(l^t_d\) with a route to \(t\).

Hence, node \(t\) obtains a subscription response from either one of its local directories or the anchor directory of \(o^k\). At that point, node \(t\) is able to send a content request acceding to the data-plane mechanisms defined for the ICN in which it operates.
7.2 Performance Evaluation

We implemented CORD and other content routing protocols using the discrete event simulator QualNet [67] (version 5.0). We ran simulation experiments using the 154-node AT&T topology, which is well known.

We compared the performance of CORD with that of NLSR, which is based on the link-state approach, and a loop-free distance-vector approach to content routing. The distance vector approach uses sequence numbers to ensure that routes to destination nodes are loop-free, and nodes learn about all the replicas of each NDO in the ICN.

We use end-to-end delay, control plane overhead, data plane overhead, and packet delivery ratio as our performance metrics. The control plane overhead is the average number of control packets generated by the routing protocols, and the data plane overhead is the average number of data plane packets generated by the routing protocols, including subscription requests sent from data consumers to producers and forwarded content packets. We evaluated the three protocols in wired networks. The three protocols used the same time period to refresh their routing structures. For CORD we used a maximum distance of 3 hops to select local directories. Each simulation ran for 10 different seed values.

In order to evaluate the performance of three protocols in ICN, a few scenarios were involved in the simulation experiments. We randomly selected a few groups in the AT&T topology, and within each group nodes are connected with each other through different paths. Nodes in selected groups can be content consumers requesting content.
Each requested NDO exists in the network in such way that it is generated by one original content producer, but may also be cached anywhere else in the network. We investigated the impact of different scenarios on the performance of these three content routing protocols.

Figure 7.4: Impact of increasing number of content requesting groups: (a) End-to-end delay of delivered data packets, (b) average number of control plane packets sent per node, (c) average number of data plane packets sent per node.

Figure 7.5: Impact of increasing number of flows: (a) End-to-end delay of delivered data packets, (b) average number of control plane packets sent per node, (c) average number of data plane packets sent per node.
7.2.1 Impact of Number of Content-Requesting Group

In this scenario we evaluated the impact of increasing the number of consumer
groups on the performance of the protocols. We started from 5 groups up to 10 groups,
and the number of nodes in each group varied from 10 to 20. To minimize the influence
from other parameters, there was only one content consumer from each group in this
scenario. In addition, the number of content producers varied from 1 to 10, so that the
number of content flows grew with the number of consumer groups at the same pace to
avoid content sparsity in the network.

The results of this scenario with performance metrics used in comparison are
shown in Fig. 7.4, where “1-data” means only one content producer existing in the
network, and “5-group” means there are five content requesting groups. Given that the
three approaches we simulated attain close to 100% delivery in all cases, so we do not
show that metric in our results due to space limitations.

Fig. 7.4-a shows that CORD attains similar end-to-end delays of delivered
contents when the protocols have the same delivery ratios. Fig. 7.4-b shows the aver-
age control plane overhead induced by the protocols. CORD incurs the much smaller
overhead and contrasts with the overhead induced by NLSR, which experiences a steep
increase for 10 groups. CORD incurs limited and fairly constant control overhead,
because only unicast publish-subscribe requests to directories are sent other than HEL-
LOs. By contrast, NLSR needs to flood link state advertisements (LSA) regarding the
existence of new copies of NDOs, and the distance-vector routing approach needs to
flood the network about the existence of new copies of NDOs at different nodes. Fig. 7.4-c shows the data plane overhead induced by three protocols. We can see that as the number of groups increases, the other two protocols incur more data plane overhead than CORD.

### 7.2.2 Impact of Increasing Number of Flows

In this scenario we increased the number of content consumers, which is the other way to increase the number of content flows, to see the impact on performance. We used the same selected consumer groups; however, nodes in each group are all content consumers and request NDOs at the same time.

Fig. 7.5 shows the results in this scenario. Fig. 7.5-a shows that CORD incurs slightly higher end-to-end delays to deliver NDOs than the other two protocols, which is mostly due to the fact that in some cases packets may take routes that are slightly longer than the shortest paths attained with the other two approaches. Fig. 7.5-b shows the control plane overhead induced by the protocols. CORD incurs very limited and fairly constant control overhead, which contrasts with the overhead incurred by other two for 10 groups with 10 data producers in the network. Fig. 7.5-c shows that the data plane overhead induced by three protocols is similar, even with an increasing number of flows. The key reason why CORD is more efficient than the other approaches is that CORD eliminates the need to communicate information about the network or content replicas.
Figure 7.6: Impact of caching, one consumer each group: (a) End-to-end delay of delivered data packets, (b) average number of control plane packets sent per node, (c) average number of data plane packets sent per node.

Figure 7.7: Impact of caching, nodes are all consumers in each group: (a) End-to-end delay of delivered data packets, (b) average number of control plane packets sent per node, (c) average number of data plane packets sent per node.
7.2.3 Impact of Caching Scheme

To evaluate our caching scheme as well as compare the performance with the impact of caching, this scenario took in-network caching into consideration. We considered “path caching,” where all nodes cache content opportunistically when they forward NDOs to requesting nodes; and “edge caching,” in which only those nodes that request NDOs cache them. We evaluated the impact of these two caching schemes using the same scenarios described above.

Fig. 7.6, 7.7 show the results of using “path caching”, and Fig. 7.4, 7.5 present the results of using “edge caching”. In Fig. 7.6, there is only one content consumer in each group, whereas nodes are all content consumers in each group in Fig. 7.7. When we compare Fig. 7.4, 7.5 with Fig. 7.6, 7.7, “path caching” helps the three protocols to attain slightly lower end-to-end delays to deliver NDOs; however the difference is very small. When we compare the control plane overhead and data plane overhead in the figures, we find the difference by using two caching schemes is very small. Hence, the results indicate that “edge caching” provides most of the advantages of “path caching” with far less storage overhead.

7.3 Conclusion

We introduced CORD (Content Oriented Routing with Directories) as an alternative to content routing in ICNs. CORD eliminates the need for content-based routing tables by establishing routes to directories using distance-vector signaling and by map-
ping name prefixes or names of NDOs to directories using publish-subscribe mechanisms. CORD constitutes the first approach for name-based content routing based on distance information to directories.

We used simulation experiments to compare its performance with that of name-based routing of content using the link-state approach advocated in NLSR [20] for ICN and a loop-free distance-vector approach. CORD achieves the same high data delivery, attains comparable delays to deliver NDOs, and incurs substantially less control plane overhead than the alternatives. The key reason why CORD outperforms the other name-based routing approaches is that it eliminates the need to maintain topology information or routing information for all the replicas of the same content.
Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, we present new hierarchies for routing in computer communication networks which provide hierarchical, scale-free routing, at the mean time, eliminates traditional routing tables. In this hierarchy, nodes can make forwarding decision more dynamically, locally than previous works. The new hierarchies also eliminate the need for any flooding of link states or per-destination information throughout networks, which provides better utilization of network bandwidth.

We present a novel approach for information dissemination in networks subject to disruption to their end-to-end connectivity. In addition, in this new routing hierarchy, we also discover the connection between MANETs and underlying social plane, and apply the social plane into design of scalable routing algorithms in MANETs with efficient network storage utilization. We integrate the use of social-group information
with an approach to routing that eliminates the use of destination-based routing tables. We show that our approach provides correct unicast routing, and compare its performance against that of epidemic routing and an disruption-tolerant, address-based routing scheme operating over networks subject to connectivity disruption.

Then we introduce APDV to substantially improve the scaling and performance properties of MANET routing by eliminating most of the flooding needed to maintain routes to destinations. APDV addresses problems related to location of resources or services to accommodate client demands subject to constraints in ad hoc network. The proposed approach is shown to outperform the best existing heuristics presented in the literature.

Motivated by the successful application of dominating sets and publish-subscribe mechanisms, we present a study on the performance impact of different parameters. We conclude the trend when varying the parameters how the performance is impacted. We also show the benefits of having multi-path when making routing decision.

To address the existing problems in ICN architectures which rely on name-based routing of content, integrating name resolution and content routing, we present two approaches to content routing within autonomous systems in which directory nodes act as intermediaries to establish virtual linking consumers of content with content producers or caching sites. The primary objective of using directories between content producers or caches and the consumers of content is to reduce control overhead in the ICN. Instead of having routing tables listing routes to individual NDOs or name prefixes, they only list routes to the directories that maintain the mappings between name prefixes.
or NDO names and the locations where their copies reside. This distributed approach achieves the same high data delivery, attains comparable delays to deliver NDOs, and incurs substantially less control plane overhead than the best known ICN protocols presented in the literature.

### 8.2 Future Work

The multi-path routing mechanisms could use different constraints (e.g., energy consumption, interference, and load balancing) when deciding the set of forwarding nodes, not only the distance. It will be interesting to investigate how different constraints relate to each other, and how they compare in terms of efficiency and reliability for the forwarding of data packets.

It will be interesting to investigate the converge time of APDV when nodes join the network at any time, and current nodes may leave. We can study the time elapsed till nodes have common agreement of elected controller nodes. Since APDV operates without any concept of round, it tolerates transient modifications to the topology.

APDV and CORD do not employ any load balancing when electing controller or directory nodes. It will be interesting to investigate solutions to incorporate load balancing when building a \((k, r)\)-DS of the network. Investigate how often nodes should trade positions as controller or directory nodes, and how to better distribute the traffic load among the elected nodes.
Bibliography


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


[60] Perkins, C. E., and P. Bhagwat. Highly dynamic destination-sequenced distance-


