UNIVERSITY OF CALIFORNIA
SANTA CRUZ

EFFICIENT ROUTING IN WIRELESS NETWORKS AND INFORMATION-CENTRIC NETWORKS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Ehsan Hemmati

December 2018

The Dissertation of Ehsan Hemmati
is approved:

Prof. J.J. Garcia-Luna-Aceves, Chair

Prof. Martine Schlag

Prof. Brad R. Smith

Lori Kletzer
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Efficient Routing in Wireless Networks and Information-Centric Networks

by

Ehsan Hemmati

The current Internet architecture was designed at the 1960s and 70s, to address communication needs of that time: sharing limited, expensive, and static computer resources. Since then, the Internet usage pattern has been shifted from conventional host-centric model to a flexible content-oriented model, in which users and contents are distributed and mobile. Internet of Things is becoming a new paradigm to describe global access to services and information offered by billions of heterogeneous devices, "things", ranging from resource-constrained to powerful devices in an interoperable way.

The first part of this dissertation studies the routing in Information-Centric Networks (ICN). ICN has been recently proposed and is inspiring the design of the future Internet architecture. The goal in these architectures is to provide a cost-efficient, scalable and mobile content distribution networking by adopting a content-based model of communication. ICN not only addresses the change in the Internet usage pattern but also matches the IoT applications, since they target data regardless of the identity of the object that stores or originates them.

In this dissertation, Named-data network (NDN) and Content-Centric Networking (CCNx) are presented and routing strategies for them are evaluated. A comprehensive performance evaluation is done through the simulation experiments. We enhanced the performance of link-state routing by introducing a new protocol called LSCR, link-state content routing protocol, a loop-free name-based routing algorithm that propagates link-state information selectively and provides

multi-path routing to content that may be replicated in different locations. We also introduced the first content routing protocol based on the diffusing computation, DNRP. DNRP provides multiple loop-free routes to the nearest instances of a data using only distance information and without requiring periodic updates, knowledge of the network topology, or the exchange of path information.

MANET paves the way for the development of brand new IoT communication platforms with a high potential for a wide range of applications in different domains. Each layer in the design model require redefinition or modifications to function efficiently in MANETs every is mobile and usually has limited resources on computation, storage, power, etc.

Routing in Mobile Ad-Hoc Networks is studied in the second part of this dissertation. We introduce ODVR (Ordered Distance Vector Routing), that provides loop-free routes at every instant based solely on distances to destinations maintained by nodes and reference distances included in route requests. Routing state is established on demand by means of route requests stating the reference distance and replies from nodes that satisfy this distance. To make the routing more efficient in an IoT environment with a base or gateway nodes, we introduce ADRP, a hybrid routing algorithm that takes advantage of the strengths of reactive routing algorithms as well as the benefits of proactive ones. ADRP uses the same signaling for both reactive and proactive routing.

To my beloved wife Faeze,
my extraordinary parents Masoumeh and Hossein,
and amazing brothers Amir and Arash.

Acknowledgments

First and foremost I want to thank my advisor, Professor J. J. Garcia-Luna-Aceves, for the patient guidance, encouragement and advice he has provided throughout my time as his student. He's the funniest advisor and one of the smartest people I've ever met. J. J. has supported me not only by providing a research assistantship, but also academically and emotionally through the rough road to finish this thesis. I hope that one day I could be as lively, enthusiastic, and energetic as he is and to be able to command an audience as well as he can.

I would also like to thank my committee members, professor Martine Schlag and professor Brad Smith. to serve in my committee despite their busy schedule and and for all their valuable feedback.

I would like to extend my thanks to Emily Gregg and Tracie Tucker for all their great help through my PhD program. Finally, I would like to thank my awesome friends in CCRG (Computer Communication Research Group) Maziar M. Barijough, Ali Dabirmoghaddam, Turhan Karadeniz, James Mathewson, and Spencer Sevilla who were always there for me, in the hard and fun research and personal times.

Last but not least, I would like to thank my mom and dad, Masi and Hossein to whom I owe the happy life that I have led so far. Thank you for your endless love, support, and encouragement.

Chapter 1

Introduction

The Internet usage pattern has been shifted from conventional host-centric model to a flexible content-oriented model, in which users and contents are distributed and mobile. On the other hand, Internet of Things (IoT) is becoming a new paradigm to describe global access to services and information offered by billions of heterogeneous devices (or things), ranging from resource-constrained to powerful devices (and/or virtualized everyday life objects) in an interoperable way. A number of Information-Centric Networking (ICN) architectures have been developed to address the increasing demand for user-generated content [50, 16, 22, 40] in the Internet.

ICN architectures are based on location-independent content naming rather than location-oriented host naming. ICN architectures are implemented based on name resolution and content routing, to provide a cost-efficient, mobile, and scalable content distribution. In such architectures, content providers, i.e., producers, create named data objects (NDOs), and publish name prefixes associated with these content objects. A name prefix, or simply a prefix, is a location independent routable name that is associated with a content object and advertised in the network by the publisher. A consumer of the content asks for the NDO

or name prefix by sending a request or interest that is routed along intermediate nodes, i.e., content routers, toward the publisher(s).

The producers or any caching site that has the original or replicated copy of the requested NDOs satisfy those interests and send back the corresponding *Data* messages. Each content router in the network forwards Interests according to its *Forwarding Information Base* (FIB), which stores next hop information towards name prefixes. A routing protocol is needed to maintain these routes and populate the FIB entries. An important feature of many such architectures is the caching of named data objects that can be done everywhere and for everything in the network. Designing a reliable routing algorithm that copes with adding and deleting prefixes in caches is a challenging task.

In this thesis, we evaluate different routing and forwarding algorithms of current ICN architectures. Most specifically we study routing algorithms for CCNx and NDN, the most popular ICN architectures. We show by analysis and through simulation that current routing algorithms for these ICN architectures face different problems and can not scale well with respect to the number of contents. To address such problems, we introduce new protocols that perform much better and scale well, compared to current methods.

The ICN approach can be particularly beneficial in an ad-hoc networking environment. Mobile nodes can communicate based on what data they need. Many MANET protocols exist for "data delivery" from resource discovery, to content distribution. Resource discovery is a non-trivial operation in a MANET. Resources in a MANET include nodes, content, and services. Initially, researchers proposed to use a centralized directory where resource information could be stored.

In chapter 2 we summarize previous studies on routing problem in both ICN and MANETs. We study different ICN architectures and routing algorithms intro-

duced for these architectures. We summarize approaches that have been proposed for content routing based on the names of objects that may be replicated in different nodes of a network. Finally, we study the operation of the Named-data Link State Routing protocol (NLSR) [52] protocol and the Distance-based Content Routing (DCR) protocol [33], two well-known routing algorithms for CCNx/NDN architecture. NLSR is a good representative of content routing based on link states, and DCR is the first example of loop-free content routing based on distance information. As we summarized in chapter 2 no prior work has been reported using partial information of publishers when multiple routers advertise the same prefix.

Chapter 3 presents **LSCR** (*link-state content routing*), a loop-free approach to name-based routing based on link-state and publisher information to create routing tables pointing to the nearest instances of the NDOs or name prefixes in an ICN. LSCR is a pure name-based routing algorithm that relies on router names instead of IP addresses. Like other link-state routing algorithms, LSCR uses a flooding mechanism to propagate link-state information regarding physical link characteristics and builds a map of the network topology at each router. However, instead of flooding publisher information, as it is done in NLSR [53] and OSPFN [43], LSCR propagates publisher information by diffusing the information selectively, based on distributed computation of preferred publishers. Hence, less communication overhead incurred in LSCR.

The Diffusive Name-based Routing Protocol (DNRP) is introduced in chapter 4 for efficient name-based routing in information-centric networks (ICN). DNRP establishes and maintains multiple loop-free routes to the nearest instances of a name prefix using only distance information. DNRP eliminates the need for periodic updates, maintaining topology information, storing complete paths to

content replicas, or knowing about all the sites storing replicas of named content. DNRP is suitable for large ICNs with large numbers of prefixes stored at multiple sites. It is shown that DNRP provides loop-free routes to content independently of the state of the topology and that it converges within a finite time to correct routes to name prefixes after arbitrary changes in the network topology or the placement of prefix instances. The result of simulation experiments illustrates that DNRP is more efficient than link-state routing approaches.

The second part of the dissertation, we focus on solving the routing problem in mobile networks. A large number of routing protocols for mobile ad-hoc networks (MANET) have been proposed over the years and the key motivation for their development has been the looping and counting-to-infinity problems inherent in any distributed routing algorithm in which a node simply updates its routing state to a destination by selecting the shortest distance through any available neighbor. Our study shows, even well-known standardized routing protocols suffer from looping issue.

The Ordered Distance Vector Routing (ODVR) protocol is introduced in chapter 5 for destination-based loop-free routing in mobile ad-hoc networks. ODVR maintain loop-free routes using only distance information independently of the state of the topology, how long a router maintains routing state for any destination, or the timing or fate of signaling messages. In contrast to all prior on-demand routing protocols, ODVR does not require source or destination sequence numbers, sequence numbers for requests, path information, source routing, or having a router wait for replies from all its neighbors before making changes to its routing table.

Although ODVR as an on-demand protocol shows a great performance, a better routing protocol is able to take advantage of the strengths of reactive routing

algorithms as well as the benefits of proactive ones and to adapt its behavior at the appropriate time and for the appropriate scope of the network. This motivates us to propose a hybrid routing protocol, **ADRP**, (*Ad-Hoc Distance based Routing Protocol*), a hybrid routing algorithm for mobile ad-hoc networks that calculates loop-free routes at every node based on the distances to the destination and reference distances maintained by nodes. Chapter 6 presents *ADRP* and describe its operation.

Chapter 2

Routing in Communication Networks

In this chapter, we summarize Information Centric Networking architectures, their characteristics, and their routing protocols. We also study previous works on routing in Mobile Ad-Hoc Networks.

2.1 Routing in ICN

Several ICN architectures have been proposed to handle name resolution and routing [9, 60]. In general, these architectures implement one or some of the following mechanisms to construct a path for acquiring data from a producer to the consumer: 1) flooding the requests throughout the whole network; 2) flooding the network topology information and the location of publishers; 3) using source routes to content; 4) creating spanning tree using publish-subscribe signaling.

A number of content routing approaches rely on flooding of content requests to cope with the fact that nodes requesting content by name do not know the locations of copies of the content.

Directed Diffusion for Sensor Networks: [44] was one of the first proposals for name-based routing of content. Requests for named content (called interests) are diffused throughout a sensor network, and data matching the interests are sent back to the issuers of interests. DIRECT [57] uses an approach similar to directed diffusion for name-based content routing in ad hoc wireless networks subject to connectivity disruption. Nodes use opportunistic caching of content and flood interests persistently within and across connected network components.

Data Oriented Network Architecture (DONA) [49] is an ICN architecture that uses name resolution to map the flat names to corresponding IP addresses that can be local or global. It relies on an external and fast name resolution system called Global Name Resolution Service (GNRS) to map the data object names, (i.e., Globally Unique Identifier (GUID)) to network addresses. Name-based routing in DONA is accomplished using traditional IP routing and forwarding.

Mobility First [4] introduces a mobility-centric and trustworthy internet architecture that uses geo-distributed names and a global name service to resolve names to flexible attributes. MobilityFirst enables mobility through a separation of names and addresses and enhances security by representing them using intrinsically verifiable identifiers. The routing approach in the Mobility First project requires using either network addresses or source routing or partial source routing.

Publish Subscribe Internet Technology (PURSUIT) [3] (formerly known as Publish-Subscribe Internet Routing Paradigm (PSIRP) [10]) each data object has a unique name that is mapped to the publisher. A Topology Manager (TM) in the network, that runs a distributed routing protocol [i.e. link-state routing] to discover the network topology, is responsible to calculate a route between the publisher and the consumer.

NetInf [66] proposes the use of a *Name Resolution* service. In this schema,

each provider publishes Information Objects (IOs) alongside their locators so consumers can locate them. It adopts content routing modalities based on distributed hash tables (DHT) running in overlays over the physical infrastructure to accomplish content routing. The NR service is underpinned by a DHT (or more precisely Multi-level DHT - MDHT), allowing global content lookups on flat identifiers whilst also supporting local resolution. A destination is assigned a home location in the DHT that nodes can determine by using a common hash function from the namespace of content or even meta-data to the namespace of nodes in the DHT. DHT nodes can cache known mappings to improve efficiency, and the DHTs are built using underlying routing protocols that discover the network topology.

Named Data Network (NDN) [5] is a developed version of **Content-Centric Network architecture** (CCN) [1], which shares the same concepts of the ICN paradigm for the future Internet architecture. The objective of CCN is to completely redesign the Internet protocol stack by replacing IP address with the name of content chunks as a universal component of transport. Content naming is based on hierarchical names encoded with the publisher, content identifier, data digest, version number and segment information (segments are equivalent to packets). In these architectures, a content request is issued by sending an Interest packet, which is routed through the network to a publisher node that hosts that content. These messages are routed using a route-by-name paradigm. Each content router in CCN forwards Interests according to its *Forwarding Information Base* (FIB), which stores next hops toward the provider of name prefixes. Any node having the original or replicated copy of that content satisfy the Interest by responding with the corresponding Data, which traverse the way back to the requester.

CCN and NDN use distributed routing protocols to build the routes over

which interests are forwarded. NLSR and OSPFN are two routing algorithms designed for NDN network. They use link-state routing as topology based shortest path routing to calculate shortest path between every router in such systems. In NLSR, two types of link-state advertisements (LSAs), *AdjacencyLSA* and *PrefixLSA*, propagate topology and publisher information in the network. Each router uses topology information and runs an extension of Dijkstra algorithm to calculate the next hops for each router, then maps the prefix to the name of the publisher and create routing table for each name prefix. NLSR propagates information of all anchors of all prefixes in the network and does not provide any mechanism to rank publishers of the same prefix.

The Distance-based Content Routing (DCR) protocol is an example of distance routing algorithm, which enables routers to maintain multiple loop-free routes to the nearest instances of a named data object or name prefix, and establish content delivery trees over which all or some instances of the same named data object or name prefix can be contacted. In the next section, we study NLSR and DCR in details and compare their performances.

2.2 Elements of NLSR and DCR Operation

2.2.1 NLSR

NLSR uses the link-state approach to compute the shortest paths from each router to every other router and replica of a name prefix. It uses two types of link-state advertisements (LSA): Adjacency LSAs and Prefix LSAs. Based on the information available in LSAs, each node creates a Link-State Data Base (LSDB) and ranks its interfaces to forward interests toward a name prefix and fills the FIB.

An adjacency LSA advertises topology information and contains the name of the router, its neighbors, and the list of all active links connecting the router to its neighbors. Each router sends Adjacency LSA at startup and whenever it detects a change in the links to which it is connected. NLSR sends “info" messages periodically to its neighbors to detect changes in the topology. Prefix LSAs advertise name prefixes that are available locally and contains the name of the router and one name prefix. If the router has multiple local name prefixes, it advertises several prefix LSAs, with each such LSA carrying one prefix update. A prefix LSA contains a flag called *isValid* indicating the status of the prefix. A node sends the prefix LSA with *isValid=1* if the prefix is registered in that node and *isValid=0* whenever a name prefix is de-registered. An NLSR node receiving this LSA will update the name prefix in the LSDB and update its FIB accordingly.

A router stores the latest version of LSAs it receives or creates in its LSDB and uses this information to find the best next-hops to reach each name prefix and ranks its interfaces. Based on link-state information, each node creates the topology of the whole network and runs an extension of Dijkstra’s shortest-path first (SPF) algorithm to calculate multiple paths to each router and ranks the next hops to reach each destination in the network. To calculate the cost of a path using a specific neighbor, the node removes all the links connected to itself except the one that connects the node to that neighbor. Then it runs Dijkstra’s SPF algorithm to calculate the distance to each destination through that neighbor. This process is repeated for each neighbor. Then NLSR uses this information and prefix information to map a name prefix to the name of a router and creates a routing table entry for each name prefix. Routes calculated by this mechanism are not loop-free and NLSR does not provide any mechanism to rank multiple replicas of the same name prefix.

LSAs are propagated in the network using hop-by-hop synchronization approach (*Sync*) [8]. Each router periodically exchanges its hash of the LSDB with its neighbors. In this way, each router can detect the inconsistencies between its own database and the databases of its neighbors and updates its database with the latest LSA information. Sync allows NDN components and applications to define collections of named data in Repositories or *Repo*, called slices. Each router computes a hash tree over the data stored in the LSDB slice and sends this root hash to its neighbors. If the root hash values of two neighbors do not match, routers exchange the hash values on the next tree level until they detect the inconsistency. Each router sends special Interest messages, called Root Advise, containing Root Hash Value of its LSDB slice to its neighbor. The neighbor router sends its own root hash value back using Root Advise Reply. If the hash values do not agree the router will recursively request for next level hash values until they find the mismatch in their databases. Then the NLSR router requests data by sending Content Interest and the neighbor router sends back the data in a Content Reply. The router will update its LSDB with up-to-date information in the repository and will run NLSR algorithm to update the FIB.

NLSR uses a hierarchal naming schema for both routers and LSAs. A router in the network has a name in format of: $/ < network > / < site > / < router >$. where *network* and *site* are assigned based on the network and specific site the router belongs to and *router* is a unique name in that network and site. Each LSA has the name of format of: $/ < network > / < site > / < router > /NLSR/LSA$ followed by the message specific naming. This naming for Adjacency LSA is $/LsType.1/ < version >$ and for prefix LSA is $/LsType.2/LsId. < ID > / < version >$. $Version$ field indicate the ordering and can be a sequence number. $LsId$ is a unique LSA Id assigned for each prefix.

2.2.2 DCR

DCR is the first name-based content routing approach based on distance information that can find routes to any or some replicas of an NDO or name prefix. Routers running DCR do not need to know the network topology, complete paths to destinations, or all the replicas of the desired content. A router that runs DCR maintains three tables for the purposes of routing to the nearest replicas of content: a *link table* stores the list of all neighbors and link cost connecting the router to its neighbors, a *neighbor table* keeps track of routing information reported by each neighbor (including the router itself), and a *routing table* stores routing information for each known prefix.

A router advertising a zero distance to a name prefix is called an *anchor* of the prefix. Each router sends periodic update messages to its neighbors containing a list of updates to the distances to name prefixes. Each update states the distance to a name prefix, the closest anchor for the prefix, and a sequence number assigned to the prefix by the anchor. DCR uses the Successor-Set Ordering Condition (SOC) to select valid next hops in a way that no routing-table loops are created in the network. Based on the information reported by each neighbor and SOC, router i can select its neighbor router k as a next hop to reach name prefix j if and only if:

1. Router k reports a new anchor of prefix j that node i does not know or the most recent sequence from a previously known anchor.

2. If router i has a finite distance to prefix j: Router k has shorter distance to j, or routers i and k have the same distance to j, but i is a lexicographically smaller name than k. If router i has an infinite distance to prefix j: Router k offers the smallest finite distance to j among all its neighbors, and also has the lexicographically smallest name among neighbors offering the smallest

distance to j.

Each router uses SOC to rank its neighbor for each name prefix, whenever it detects a change in link costs or its neighbor table. Router i computes the distance to a prefix as the minimum of distance of paths using neighbors that satisfy SOC. Each router sends the updated routing information to its neighbors using update messages.

The naming schema for DCR depends on the ICN architecture in which DCR is implemented. A flat or hierarchical naming schema can be used for both routers and update messages. If a hierarchical schema is used, each router is named in the following format: ĂIJ/ $< network > / < site > / < router > /$. The update messages can use a naming schema like this: $/ < network > / < site > / < router > /DCR$.". DCR requires a mechanism to compare router names and rank them lexicographically.

2.2.3 Performance Comparison

Protocol implementation

We implemented DCR and an optimized version of NLSR using the ns3 simulator tool with extensions for content-centric networks [54]. SCoNet supports CCNx v.1 specifications and messages based on the TLV format.

The signaling overhead incurred in the transmission of an LSA in NLSR is much larger than the overhead incurred by DCR, which is based on sender-initiated signaling and incurs a single transmission per update message.

To eliminate the differences in performance due to sender-initiated or receiver-initiated modalities, we implemented DCR and NLSR using sender-initiated signaling, in which control messages are simply Interest messages that carry a payload containing control information. As a result, our implementation of NLSR

in the simulation uses a single transmission per LSA, rather than sending them as a result of Interests after neighbor routers determine the differences in their local databases. NLSR propagates LSAs using intelligent flooding. We denote the optimized implementation of NLSR by *i-NLSR*.

In our simulation of DCR, whenever a node receives an update, it checks its information against the information stored in its neighbor table. If it detects any changes in the anchor, distance or sequence number of a name prefix, it updates the information in the neighbor table and schedules a routing update. A router waits to receive updates from other neighbors before changing its routing table. A router reports the updated routing information to its neighbors in its next update message.

For simplicity, our implementation of DCR is such that each update message contains the information regarding all the prefixes known to the router. In practice, only those name prefixes that have changed since the last update message was sent would need to be reported. Each anchor sends a new sequence number on each update message it sends for locally-available prefixes.

For the case of i-NLSR, whenever an LSA is received, the router updates its LSDB and schedules the routing update to rank its interfaces to reach the destination. If the received LSA is an adjacency LSA the router calculates multiple next-hops for each destination and updates its routing table using the NLSR multi-path calculation mechanism described in Section 2.2. If the router receives a prefix LSA, it maps the name prefix to the destination and ranks the next-hops based on routing table information for that destination, calculated in the previous phase. If two or more routers advertise the same name prefix, the faces are ranked based on distance to closest router advertising the name prefix. Adjacency LSA and prefix LSA carry the information described in Section 2.2, as specified in [53].

Simulation Scenario and Parameters

Network model: The AT&T core network topology shown in Fig. 2.1 was used, which is often used as a realistic topology for simulations [42]. The AT&T topology has 154 nodes and 184 links. A node has 2.4 neighbors on average, and there are 14 nodes with only one neighbor. In our simulation model, each node has a unique identifier. The hop count is measured as the distance to a destination; therefore, the path cost is the number of hops between a source and a destination. In the implementation, the existence of a link-level protocol assures that every node detects the loss or recovery of connectivity with its neighbor in a finite time after a router fails to receive the proper control messages a repeated number of times. All messages, link failure, and link recovery are processed one at a time in the order in which they occur and within a finite time.

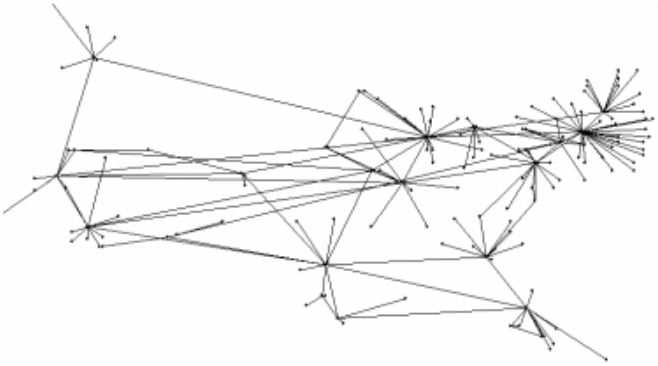

Figure 2.1: AT&T Network Topology, adapted from [42]

In our scenario, 30 nodes are selected as anchors of 180 different name prefixes and all anchors have a prefix list of the same size. Each prefix may have more than one anchor. The anchors are selected randomly, and two or more anchors may have some prefixes in common. For instance, a network with an average of three replicas has 540 NDOs and each anchor publishes 18 unique name prefixes.

The simulations were run 20 times and different seeds and several quantities are measured in the network. On each run, the input event generated was a single

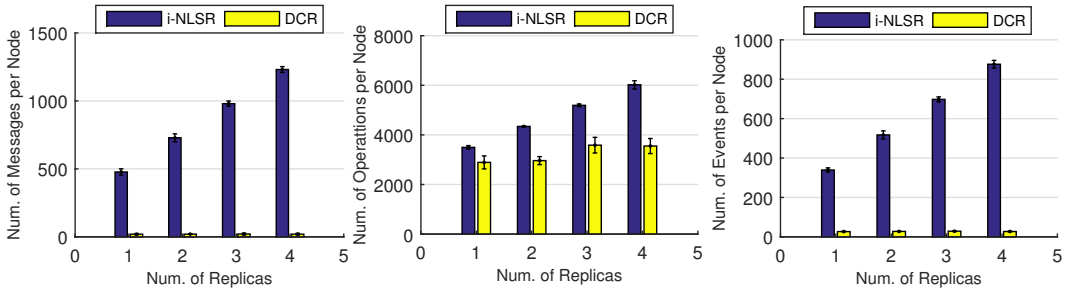

Figure 2.2: Initialization: average number of messages sent per node, average number of operations done per node, average number of events per node

link failure or recovery, and a single prefix addition or deletion. After each of the events, the protocols are allowed to converge to the steady state, which means all the messages are processed and no further changes are made to the routing tables. The links or prefixes that are deleted or added, are selected randomly. Routers perform their computations in zero time. i-NLSR sent *info* messages every 10 seconds and sends LSA whenever it selects a change in topology or local prefixes. DCR sends update messages each 10-second interval. A neighbor is considered as unreachable if the node does not receive four consecutive *info* messages, in the case of i-NLSR or *update* messages, in the case of DCR.

The simulation started from a topology in which all the links and anchors are operational, and all prefixes are attached to the anchors. The network was in a steady state before any changes. All the quantities were measured from the time that a router detected a change until the protocol converged. The performance metrics are:

- *Messages:* The total number of control messages transmitted over the network. The number of messages for i-NLSR includes the number of Hello messages, adjacency LSAs, and prefix LSAs. In DCR this metric indicates the total number of update messages transmitted as a result of any changes.

- *Events:* The total number of updates that must be processed by the protocol,

including changes in neighbor table in the case of DCR and changes in link status or prefix status in the case of i-NLSR.

- *Operations:* The total number of operations performed by each protocol to calculate the routing table. The operation count was incremented whenever an event occurs, and whenever the statements within a loop are executed.

The flooding needed in i-NLSR was implemented in such a way that a router forwards the LSAs toward those neighbors that did not send the LSA. Therefore, the control messages in this study give us a lower bound of the total number of messages that would be required by NLSR.

For the case of i-NSLR, the Dijkstra-SPF algorithm inserts data into a priority queue. The number of operations required to do so is estimated to be $log_2 N$, where N is the number of nodes. The simulation used the BOOST library with the time complexity of $O(NlogN + E)$ [56].

Performance Results

The results of our simulation experiments are shown in Figures 2 to 6. The mean and the standard deviation of the value distribution are given. In each graph, the horizontal axis is the average number of replicas of a name prefix, and the average quantity (number of messages, events, and operations) is presented for each router.

The results for the number of operations and events for DCR are an upper bound, given that our implementation sends all name prefixes, rather than just those that changed since the last update.

Figure 2 shows the results for the initialization phase. Routers do not have any information about the topology, anchors, or prefixes at the beginning, except for those prefixes that are available locally. Routers start sending their local information for a random time after starting the simulation. The results give an

upper bound for real events, such as refreshing the LSDBs in all nodes or adding a new node to the network.

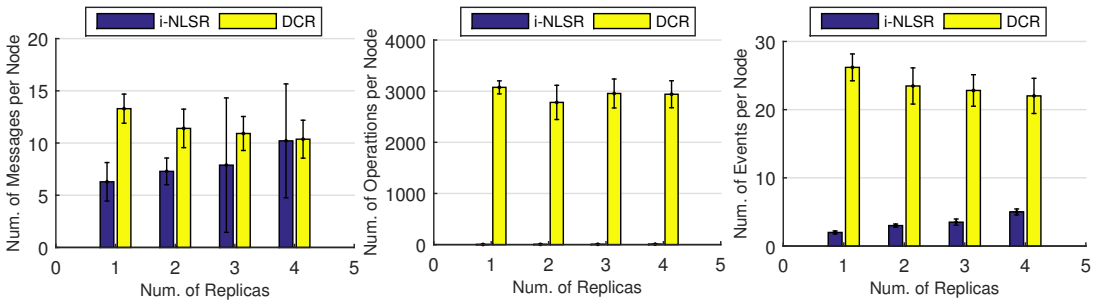

Figure 2.3: Adding a prefix: Average number of messages sent, operations done, and events processed per node

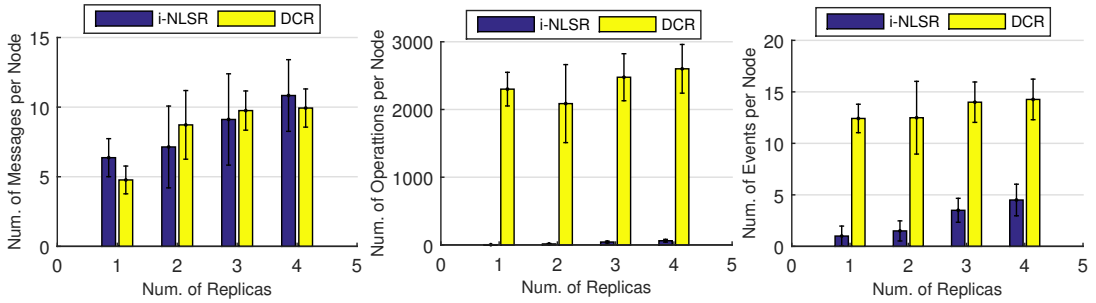

Figure 2.4: Deleting a prefix: Average number of messages sent, operations done, and events processed per node

In DCR, a router sends information regarding the closest anchor to each known prefix to its neighbors. Therefore, the total number of messages for DCR does not change as number of replicas increases, but the number of prefix LSAs in NLSR is proportional to the total number of prefixes and increases as the average number of anchors per prefix increases. The operation count for the Dijkstra-LS algorithm (replicated at each node) was substantially higher than for the operation count in DCR. NLSR updates the LSDB whenever it receives an up-to-date LSA and schedule routing update; therefore, the number of events per node increases as the number of messages increases.

The results of adding a new prefix to the network are depicted in Figure 3.

Routers running DCR exchange more messages than NLSR to converge when the number of replicas is small. However, starting with three replicas, the number of messages is comparable, and the number of messages in NLSR grows with the number of replicas, while the opposite is true for DCR. For the case of prefix deletion, the number of messages needed in DCR remains the same after two replicas, while it keeps increasing in NLSR.

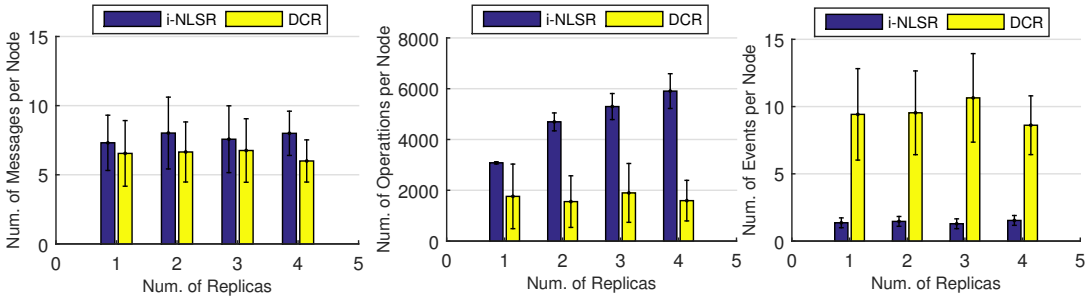

Figure 2.5: Link failure: Average number of messages sent, operations done, and events processed per node

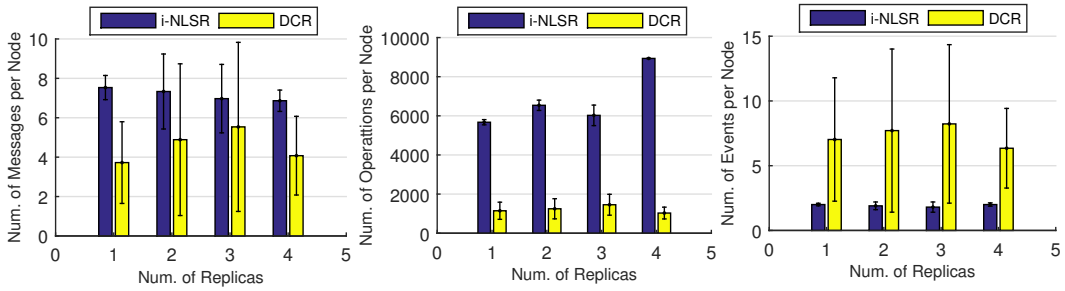

Figure 2.6: Link recovery: (Average number of messages sent, operations done, and events processed per node

Figures 5 and 6 show the results for link failures and recoveries. Each router that runs NLSR has to process LSA updates corresponding to each direction of a link, unless the link connects to a leaf node in the network. The number of messages exchanged per node is smaller in DCR than in NLSR, independently of the number of replicas of prefixes maintained in the network. A router executing NLSR runs Dijkstra on every node and every single neighbor; therefore, the

computation due to running Dijkstra's SPF algorithm is dominant on both link failure and recovery, and the computation cost is higher than in DCR.

2.3 Routing in Mobile Ad-Hoc Networks

MANET paves the way for the development of brand new Internet of Things (IoT) communication platforms with a high potential for a wide range of applications in different domains. The IoT enables "things" to communicate and exchange data among themselves and with existed or evolving commination infrastructure. The IoT is fast becoming a global phenomenon and many issues are arising such as standardization, sensors energy requirements, and efficient routing among others.

It is widely recognized that MANETs are key technologies for several IoT application domains [97]. Their suitability is also boosted by their localized and self-configuring capabilities, which can enable easier large-scale deployments. In addition to academic interest, market research shows that many applications will soon adopt MANETs, mainly for public safety, localization, environmental monitoring, and etc.

In general, the IoT network transmits the message from sensors to the corresponding control nodes. Hence, we can define two types of nodes in an Ad-Hoc network for IoT application: the sensor/edge nodes and sink/control nodes. The edge nodes consist of distributed control nodes as well as IoT network and sensors. The sink nodes consist of local management and centralized data center nodes as well as the gateway nodes. In a typical scenario, edge nodes will send the data to relay nodes, and then the relay nodes forward traffic information along the optimized path to the sink node that is one or more hops away.

The limited resources in Ad-Hoc networks have made designing of an efficient and reliable routing protocol a very challenging problem. Many routing algorithms

have been proposed for MANETs aiming to achieve good performance in terms of high throughput, low control overhead, short delay, low energy consumption, scalability, etc. Both proactive and reactive routing protocols have their advantages and disadvantages. The motivation for proactive routing approaches is the ability to have routes readily available whenever sources have data to send to any destination. The price paid for such responsiveness is that signaling overhead is incurred even for those destinations that are not popular. On-demand routing approaches address this concern by requiring signaling overhead only for active destinations at the expense of incurring slightly longer latencies while routes are established.

Researchers have studied MANET routing features in designing reliable and efficient routing protocols for the Internet of Things. Songfan et al. [98] compares the performance of two typical routing protocols, AODV and DSR, in a real multi-hop environment. In their study, they compared the performance of AODV and DSR in terms of some applications based on IoT, such as Radio Frequency Identification (RFID) service, voice service, and temperature monitoring service.

Several comparative analyses of the performance of on-demand versus proactive routing schemes have been provided and to this date, it is not clear that either approach is substantially better. However, some studies [67, 95] do show that on-demand routing protocols can end up incurring more overhead than proactive routing protocols in MANETs when topology changes that impact existing data flows increase. As it has been pointed out in [74], to a large extent this is due to the fact that typical on-demand routing schemes require each route request (RREQ) to be disseminated throughout the network independently of others.

Many MANET routing protocols have been proposed and many surveys and comparative studies exist (e.g., [88, 79, 95]). These protocols can be categorized

as proactive or reactive, with hybrid approaches using proactive and on-demand mechanisms. Proactive approaches maintain routing information for all destinations, regardless of whether traffic exists for them. On-demand or reactive approaches maintain routing information for only those destinations for which traffic exists and rely on flood search mechanisms to establish routes to destinations. Furthermore, several of these protocols use various mechanisms to reduce signaling overhead, such as geographical coordinates [72], virtual coordinates, connected dominating sets [84], address aggregation [82], and clustering [88, 79, 76]. However, all MANET routing protocols must employ one or more mechanisms to detect or eliminate routing-table loops, and this is the focus of our brief overview of related work.

A simple way to detect or avoid routing-table loops consists of including path information in routing-table updates and many on-demand and proactive routing protocols have used this approach. Path information can be stated incrementally by stating the second-to-last hops to destinations [70, 89] or with complete paths. The latter is the approach taken in DSR [71], which is one of the most popular on-demand routing protocols. The limitation with using path information in updates is that many more or much larger signaling messages may be needed for on-demand routing compared to approaches that simply state destination information.

A less common approach to prevent routing-table loops consists of synchronizing routing-table updates among nodes so that no node changes its next hop to a destination before receiving feedback from all its neighbors about its current distance [92]. This approach has been successful in wired networks [68]; however, the reliable transmissions needed for feedback messages becomes too onerous in a MANET with high mobility.

Corson and Ephremides [65] introduced one of the first proposals for MANET

routing. Their specific approach incurs excessive signaling overhead because replies to route requests are flooded. However, their use of sequenced route requests has been adopted in many subsequent on-demand routing protocols based on distances.

Many proactive and on-demand routing protocols use sequence numbering of some type. OLSR and other MANET routing protocols based on link-state information [64] use sequence numbers to denote the freshness of a link-state update. Although permanent loops are eliminated, link states must be refreshed periodically and temporary routing loops are possible in any routing approach based solely on sequence-numbered link-state updates.

DSDV [90] was arguably the first proactive routing approach that advocated the use of destination sequence numbers to cope with the looping problems associated with traditional routing based solely on distance information. A node accepts a new next hop to a destination only if the destination sequence number stated by that neighbor is smaller than the number held by the node or the destination sequence number is the same but the distance offered by the neighbor is smaller than the current node distance to the destination.

2.3.1 On-Demand Routing

The Ad-hoc On-Demand Distance Vector (AODV) [91] was the first on-demand routing approach based on destination sequence numbers. To find a route to an intended destination, a source broadcasts a route request stating the source and destination nodes, the most recent sequence number known for each, a broadcast identifier and a hop count to the source. Nodes maintain state for the requests they originate or forward, and discard subsequent copies of requests that they have forwarded. The intended destinations or nodes with valid routes to destinations

reply to route requests following the paths traversed by the requests in reverse. A reply states the destination and source of the request, the destination sequence number, and the hop count to the destination. A node receiving a reply establishes a route to the destination stating the destination sequence number, the next hop, and the neighbors using the route (precursors). Nodes forward only the first copies of replies based on the destination sequence numbers. A router that forwards a request for the first time creates a record for the RREQ stating the source and BID pair of the RREQ; and a reverse route to the source of the RREQ stating the next hop and hop count to the source, and the sequence number of the source. It maintains any RREQ record and reverse route for a finite time. Link failures can be recognized in AODV by the absence of HELLO messages sent periodically between neighbors. When a node detects a link failure, it sends a route error to all neighbor nodes that are precursors of a route that is broken because of the link failure. Nodes receiving a RERR message invalidate all routes that were using the failed link and propagate the RERR message to their precursor nodes.

Looping Problem of AODV

While AODV avoids some of the limitations of approaches based on path information or synchronization, a number of looping problems have been identified over the years with the original AODV proposal in [91] and subsequent versions of AODV [62, 69, 94]. Various approaches have been proposed to make sequence numbering more resilient [69, 80] than in the original AODV version [91] and proposals have also been made to provide multiple paths per destination [73]. However, AODV and subsequent proposals based on destination sequence numbers do not address in detail why the protocols work correctly when nodes experience the loss of routing state and signaling messages are lost.

Consider the example shown in Fig. 2.7, which focuses on destination d.

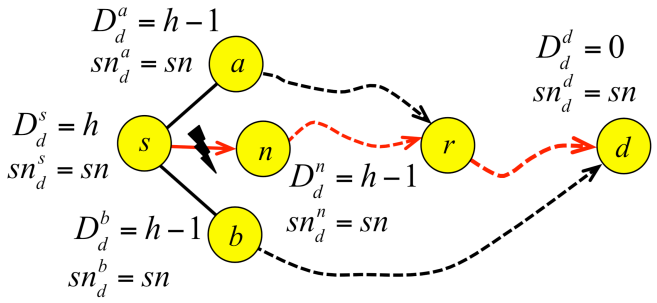

Figure 2.7: Route requests are flooded because nodes must satisfy global constraints

The distance from node x to d is denoted by D_d^x and the sequence number maintained by node x for d is denoted by sn_d^x. In the example, node s uses node n as its next hop to destination d, nodes a and b have valid routes to d, and all nodes have the same sequence number value of sn for destination d. Assume that the AODV version of [91] is used. If link (s, n) breaks and node s is a source of data for d, then node s must increase its sequence number to $SN_d^s = sn+1$ before it broadcasts a route request (REQ) for d stating the new sequence number $sn+1$ for d. The REQ from s must be flooded and destination d is the only node that can respond to the REQ from s because only destination d can satisfy the *global constraint* of $SN_d^s = sn+1$ imposed on d by increasing its own sequence number. If a *local constraint* could be used, nodes a and b could respond to the REQ, or s could determine that a and b were valid alternatives to n if multipath routing were in place.

Next, we consider the routing-table looping problem associated with routing protocols based on destination sequence numbers, and AODV in particular. The conventional wisdom since the introduction of AODV [91] has been that AODV is free of routing-table loops because of the use of destination sequence numbers. However, as we show below, the proof given in [91] is invalid. Routing-table loops may occur in such protocols as a result of complex interactions of events involving

one or more nodes losing routing state (for any reason), signaling messages being lost, topology changes taking place, and the behavior of sources of traffic. Fig. 2.8 illustrates the routing-table looping problems in AODV with a simple example.

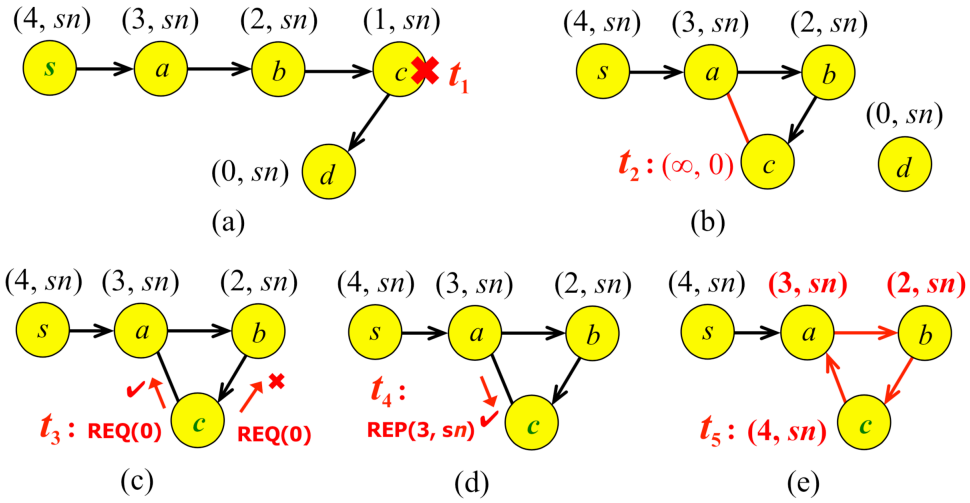

Figure 2.8: Routing-table looping in AODV resulting from deleted routing state

Fig. 2.8(a) shows the distance and sequence number pairs attained by nodes s, a, b, and c to destination d. Node s was the original source that requested the routing state. Node c crashes at time t_1 and node b does not immediately detect the crash due to the absence of data traffic. As Fig. 2.8(b) shows, node c quickly reboots at time t_2 with the corresponding loss of routing state. Furthermore, due to mobility, node d is out of transmission range from node c and within transmission range of nodes a and b. Fig. 2.8(c) shows that node c has traffic for d at time t_3 and sends a REQ with a 0 sequence number. This request is not received by node b but is received by node a. Accordingly, node a sends a reply to node c at time t_4 as shown in Fig. 2.8(d). As shown in Fig. 2.8(e) the reply makes node c adopt a as its next hop to d thus creating a routing-table loop.

A number of prior works have pointed out different routing-table looping problems in the original AODV proposal and subsequent versions (e.g., [62, 69, 94]).

The common thread in our example and the problems identified in prior work is the complex interaction between the control plane (signaling), the data plane (data traffic), and the physical layer (e.g., topology, failures, success of transmissions).

The proof provided by Perkins and Royer [91], which is similar to the proof in [73], attempts to establish a contradiction on the existence of a routing-table loop L involving $n > 0$ nodes. The proof first states that $sn_d^1 \leq sn_d^2 \leq ... \leq sn_d^n \leq sn_d^1$ and hence every node $i \in L$ must hold the same sequence-number value. Given that proposition, the proof states that node i makes node $i+1$ its next hop only if $D_d^i = D_d^{i+1} + 1$. Based on the two propositions, the conclusion is reached that $D_d^1 = D_d^n + (n-1)$, which together with the fact that $D_d^n = D_d^1 + 1$ leads to the contradiction that $n = 0$.

The fallacy in this argument, as well as the proof in [73], is that it *assumes* that the protocol signaling must enforce the condition that $sn_d^1 \leq sn_d^2 \leq ... \leq sn_d^n \leq sn_d^1$ if loop L is created. However, this assumption is not true if at least one node i that helps create L has erased its routing state for d for any reason just before it helps create L by choosing a new next hop. According to the operation of AODV in [91] and subsequent versions of AODV, $sn_d^i = 0$ if node i has no routing state for d. This state allows node i to accept a response from node $i+1$ if $sn_d^{i+1} > 0$ independently of the value of D_d^{i+1}. As a consequence, as our second example illustrates and prior work [62, 69, 94] discusses, a node can select a neighbor with $D_d^i \neq D_d^{i+1} + 1$ when a node deletes routing state due to failures or timeouts, which can lead to routing-table loops.

Zhou et al. [96] provided the sketch of a proof showing that AODV signaling cannot create routing-table loops, provided that the destinations are the only nodes that answer route requests. This restriction does render failsafe behavior;

unfortunately, it makes the signaling overhead of on-demand routing more onerous than the signaling overhead incurred with proactive routing.

Bhargavan et al. [62] showed that the looping problems they identified in AODV can be eliminated, provided that nodes *never* delete routing-table entries and *immediately* detect when a neighbor node restarts its routing process. While their proof is valid, the assumptions needed for guaranteed loop freedom are impractical.

Two lessons can be learned from this discussion. First, a valid proof that AODV avoids routing-table loops is still missing. Such a proof should assume that intermediate nodes may answer route requests, nodes may experience the loss of routing state, and signaling messages may be lost. Second, sequence numbers by themselves are not sufficient to avoid routing-table loops. This has been pointed out before in the context of AODV [94]; however, our assertion is not based on whether or not AODV is correct. Even when no node ever erases its routing state for a given destination, nodes along a loop can have the same sequence number for a destination. As a consequence, an additional constraint is *required* to prevent routing-table looping. In AODV and similar on-demand routing protocols, this additional constraint is based on distance values. Therefore, if distance values are used to ensure loop freedom, then at least in principle an on-demand routing protocol could be designed that eliminates using destination sequence numbers, uses only constraints on distances to destinations, and consequently is simpler than AODV and similar protocols.

2.3.2 Hybrid routing

There is a large design space for hybridization between various basic proactive and reactive protocols, and many hybrid MANET routing schemes have been

proposed in the literature. These schemes can be classified into cluster-centric and node-centric [99]. In the cluster-centric schemes [102, 101, 100], explicit clusters are formed and maintained as efficient control structures for abstracting and propagating routing information, and the boundary of clusters is the switching point between different routing strategies. By contrast, in the node-centric routing schemes [105, 104, 103, 93] each node makes use of an implicit control structure that is naturally associated with itself, that can be its k-hop neighborhood, the application running on the node, or the amount of traffic routed to that node. Such a structure is constructed and maintained as a by-product of exchanging regular routing information among nodes.

The zone routing protocol (ZRP) [105] is a hybrid routing protocol that proactively maintains routes within a local region of the network (which is referred to as the routing zone). In ZRP, route discovery is based on a reactive route request/route reply scheme. A node proactively propagates LSUs to all the nodes in its k-hop neighborhood, where k is called Zone Radius (ZR). This querying can be performed efficiently through the proactive maintenance of a local routing zone topology. The two subprotocols of ZRP are IntrA-zoneRouting Protocol(IARP) and Inter-zoneRouting Protocol(IERP). The IARP (proactive routing) is generally used within the routing zone of the node and IERP (reactive routing) is used between routing zones. Intra-zone routing protocols keep an up to date information of the zone topology, which results in no initial delay while sending information to the destination nodes within the zone.

TZRP [99] is a Two-Zone Hybrid Routing Protocol for Mobile Ad Hoc Networks. In contrast to the original ZRP where a single zone serves a dual purpose, TZRP aims to decouple the protocol's ability to adapt to traffic characteristics from its ability to adapt to mobility. In TZRP each node maintains two zones: a

Crisp Zone and a Fuzzy Zone. By adjusting the sizes of these two zones independently, a lower total routing control overhead can be achieved.

A node-centric hybrid routing protocol is proposed in [93] that divides network nodes to normal and special nodes. Special nodes are called netmarks and are more popular than others. Netmarks force common nodes to discover and maintain path to them proactively, while routes between ordinary nodes are set up on demand. Similar to [93], SHARP [106] also is based on the assumption of existence of hot-spot nodes in an Ad-Hoc network. A proactive zone is defined around each hot-spot node. Nodes within the proactive zone maintain routes proactively only to the central node x. The nodes that are not in the proactive zone of a given destination use the reactive component (AODV with the optimization mechanisms of route caching and expanding ring search) to establish routes to that node. It is interesting to note that SHARP's proactive zone is far more light-weight than ZRP's routing zone.

AntHocNet [109] is a different hybrid routing algorithm for mobile ad hoc networks that is Inspired by Ant Colony Optimization and consists of both reactive and proactive phases: reactive in route discovery and proactive in route maintenance. Multiple paths are set up between the source and destination of a data session during the reactive path setup phase, and during the course of the communication session, ants proactively maintain current paths while exploring the new ones. During the course of a session, the AntHocNet algorithm continuously sample possible paths with ant-like agents, and to indicate the quality of paths by means of artificial pheromone variables.

Chapter 3

Link State Content routing

A new approach for link-state name-based routing in Information-Centric Networks (ICN) is presented in this chapter. Link state content routing (LSCR) finds multiple next hops for each named data object (NDO) or name prefix in the network. LSCR uses two types of link-state advertisements (LSAs): a *RouterLSA* that contains information about links connected to each router, and an *AnchorLSA* that carries information regarding a name prefix and the router that advertise that name prefix, also called the anchor of the prefix. *AnchorLSA*s are propagated selectively based on a diffusing mechanism. LSCR creates routing tables with no permanent routing loops, provides a ranking mechanism to find multiple routes to multiple instantiations of name prefixes, and has better performance compared to other link-state routing algorithms when a name prefix is replicated at multiple sites in the network.

3.1 LSCR Operation

LSCR relies on two mechanisms: name resolution and topology-based routing. Like other link-state routing protocols, LSCR propagates link-state adver-

tisements (LSA) to create a local copy of the network topology and a mapping schema from name prefixes to router identifiers (ID) at each router. Based on topology and anchor information, LSCR creates a lexicographic ordering among neighbors and calculates multiple routes to the nearest replica(s) of prefixes. Each router in the network has a unique name or identifier, that can be flat or hierarchy and a lexicographic value is assigned to the name. Routers are assumed to operate and store information correctly. Each router receives LSAs from its neighbors correctly and processes them one at a time within a finite time. Link costs can vary in time but cost values are always positive.

Every piece of content in the network is a named-data object (NDO), represented by name prefix or simply $prefix$. Prefixes can be simple and human-readable or more complicated and self-certifying or even a cryptographic hash of the content. An anchor is a content publisher, i.e., a router that has some or all parts of the content, corresponding to the prefix and advertises that prefix in the network. Both router and content can use flat or hierarchical naming schema.

3.1.1 Messages and Data Structures

LSCR propagate two type of LSAs: 1) "*RouterLSA*" that advertise the presence of the router and its link configuration including link costs; 2) "*AnchorLSA*" that advertise the anchor and its prefixes. A sequence number is associated with each LSA to identify the message and its order. The RouterLSA can be initiated by any router that runs LSCR. The RouterLSA sent by router i is denoted by $RLSA^i$ and consists of the name of the router i; a message sequence number (msn^i); and a list of links connected to the router i and their costs.

The *AnchorLSA*s can be initiated only by anchors and replicas of the prefix and intermediate routers can forward, drop, or HOLD these LSAs. The *AnchorLSA*

sent by anchor m regarding prefix j is denoted by $ALSA_j^m$ and consists of the name of the anchor (m); and one "$PrefixUpdate$" (PU^m). PU^m states the prefix name j; the sequence number that is assigned to the prefix by the anchor (usn_j^m); and the "ValidFlag" or *vFlag* indicating if name prefix j is attached to anchor m or detached (vf_j^m). Anchor m sends just one prefix update per *AnchorLSA* because of two considerations: first, prefixes are in different length and can be too large to fit several prefixes in a single message, second, every intermediate router that runs LSCR, process the *AnchorLSA* based on the prefix and decide whether to forward or HOLD it. A router may forward one *AnchorLSA* with a specific prefix and HOLD another *AnchorLSA*that advertises different prefix from the same anchor.

A link between router i and its neighbor n is denoted by (i, n) and its cost is denoted by l_n^i. N^i represents a set containing router i and its neighbor routers. The lexicographic value of a neighbor router n is denoted by $|n|$. Router i maintains a Link Cost Table LT^i storing the cost of the link from router i to each of its adjacent routers. Each LSCR router exchanges periodic $Hello$ messages with its neighbors to detect link and/or node failure as well as any changes in the link cost. A predefined parameter defines the time interval between sending ĂHello messages. If a router does not receive a Hello message for a specified amount of time, time-out, from its neighbor, that link will be considered as down. Afterward, router will send recovery Hello message to detect the recovery with time intervals relatively longer than normal Hello message interval. Router can not detect if the link is down or the neighbor node has failed. However, this distinction does not affect the algorithm since in either case, that neighbor should not be used to forward traffic through. Each router that run NLSR sends an update R̂outerLSA at startup and whenever it detects a change in one of its links.

Each LSCR router maintains a *Forwarding Table* FT^i, storing the set of valid

next hops for each destination. The row for destination p in FT^i specifies: (a) the name of the router p; (b) the sequence number $rsn(p)$ reported by router p; (c) the Distance List (RD_p^i) consist of the set of shortest distance from neighbor routers $n \in N^i$ to destination p (rd_{pn}^i); (d) shortest distance to router p (rd_p^i); and (e) the set of neighbors that are valid next hops toward destination p (RS_p^i);

Router i updates FT^i based on *RouterLSA*s received from other routers in the network. The RLSA form router k received by router i is denoted by $RLSA_k^i$. The information stored in $RLSA_k^i$ is router sequence number rsn_j^i and cost value of the link between router j and each of its neighbors.

Router i stores information of prefixes and corresponding anchor(s) in its *Prefix Table* PT^i. The information regarding prefix j is denoted by PT_j^i and consist of: the name of the prefix j and the prefix anchor information of prefix j (PAI_j^i). Each entry of the PAI_{jm}^i consists of: (a) the anchor m of the prefix j; (b) a "$valid$" flag; vf_jm for anchor m, indicating if router m advertises the prefix j or not; and (c) the sequence number (sn_{jm}) reported by anchor m for prefix j;

Router i updates PT^i based on *AnchorLSA*s that are received from anchors. The ALSA form anchor m received by router i regarding prefix j is denoted by $ALSA_{mj}^i$ and consists of the name of the anchor (m); the prefix name j; the sequence number assigned to the prefix by the anchor (usn_{mj}^i); and the *vFlag* indicating if name prefix j is attached to anchor m or detached (vf_{mj}^i).

Router i also maintains a *Routing Table* (RT^i), that stores routing information for each known prefix. The information stored in RT^i regarding prefix j is denoted by RT_j^i, and consist of routing information for the nearest anchor of the prefix j. The routing information stored in routing table includes: (a) the name of prefix j; (b) shortest distance d_j^i to nearest anchor of prefix j; (c) the set of neighbors that are valid next hops for prefix j (S_j^i); (d) the king anchor: that is the anchor

with the smallest legicographic name among those anchors that are at the same shortest distance to j (k_j^i); and (e) a neighbor that is the best next hop in the shortest path to anchor k_j^i of j ($s_j^i \in S_j^i$);

Router i will update PT^i and RT^i based on the other two tables, LT^i and FT^i, and the information available in *AnchorLSA*. The *AnchorLSA* received by Router i sent by anchor m regarding prefix j is denoted by $ALSA_{jm}^i$. The information extracted from $ALSA_{jm}^i$ is the sequence number assigned to the prefix by the anchor (usn_{jm}^i) and the vFlag uvf_{jm}^i.

3.1.2 Routing to Nearest Replicas

A router calculates the best routes to nearest copies of a prefix in two steps: first, calculates valid next hops for all the anchors advertise that prefix, second, selects some of the neighbors from previous step as valid next hops to the prefix. For every anchor in the network, the result of the first phase is a directed acyclic graph tree.

Next-Hop Ordering Condition (NOC)

Every router keeps track of the sequence numbers reported by the routers in the network. Whenever a router receives a *RouterLSA* from another router, it checks the sequence number. If the message sequence number is greater than stored sequence number for that router, it will update the topology information and also forward it to its neighbors, otherwise, the router will drop the message. Using the sequence number and a termination detection mechanism similar to those used in OSPF prevents advertisement messages from circulating in the network forever.

Based on the information received from other routers in the network, router i creates the network topology NT^i and calculates path cost to every destination

p in the network from each of its neighbors as well as the router i itself. Router runs Dijkstra's algorithm, or any other shortest-path algorithm, on the network topology to construct a source graph, which constitutes shortest-path trees to every destination from every neighbor. The results will be stored in the Distance List of the Forwarding Table (RD_p^i). The router also stores the shortest distance d_p^i. Router i will select a subset of its neighbors as valid next hops to reach destination p based on the following condition. We will show that no permanent routing loop can be created if routers use the following condition to select next hops to forward messages to each destination.

Router i can select its neighbor $n \in N^i$ as a valid next hop to reach destination p if:

$$rd_{pn}^i < \infty \wedge (rd_{pn}^i < rd_p^i \vee (rd_{pn}^i = d_p^i \wedge |n| < |i|)) \tag{3.1}$$

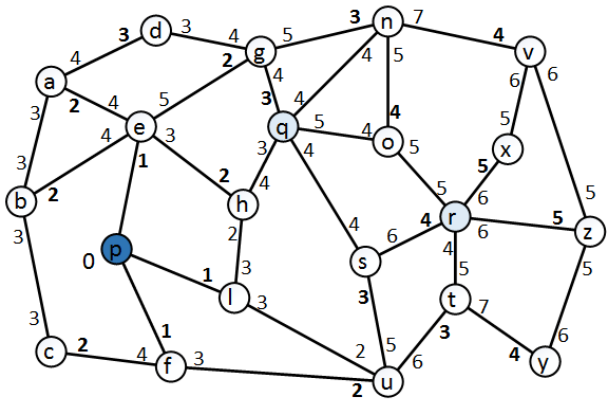

Figure 3.1: Sample Network

Router i selects router n as next hop to reach destination p if the neighbor n is closer to the destination or neighbor n and router i are at the same distance to p and $|n| < |i|$. Figure 3.1 shows the network topology. For each router the distance

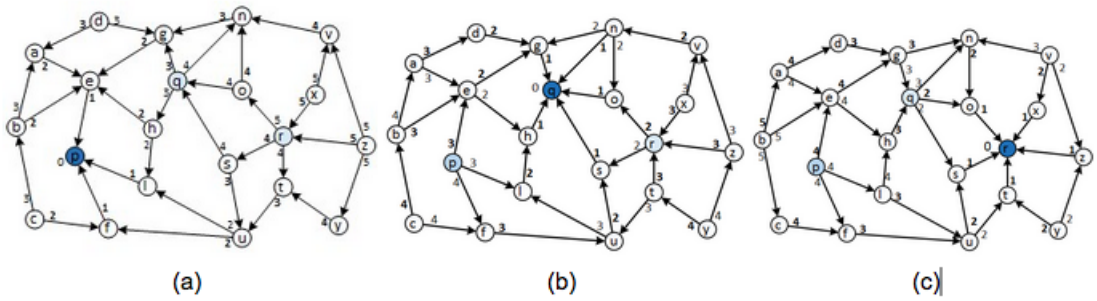

Figure 3.2: Valid next hops to destinations: (a) destination p, (b) destination q, (c) destination r

to the destination p is represented by the number next to the node. Figures 3.2 a,b, and c show the valid next hops for each router to reach destinations p, q, and r respectively. The arrowheads are pointed to valid next hop of each destinatin. For instance, router u can be selected as next hop in routers s and t to reach destination p, and u itself can select routers l and f to forward messages toward p.

Algorithm 1 illustrates how a router updates its forwarding table when it receives a *RouterLSA*. We assume that router waits for a reasonable time since it received the last Router LSA, before starts algorithm 1, and router will not process any new *RouterLSA* while executing the algorithm.

King-Anchor Selection Condition (KSO)

Based on forwarding table and the information available in the *AnchorLSA* received by router i, the router looks up the forwarding table and ranks the next-hops for each prefix based on their costs to reach the anchor(s). Each router calculates the *king* anchor among all the anchors advertise the same prefix. If two or more anchors are at the same distance, router will select the lexicographically smallest anchor as the king anchor. Whenever router receives a new attache *AnchorLSA*, i.e., *AnchorLSA* with the up-to-date sequence number and valid flag

Algorithm 1 Update FT^i

Input: $RLSA^i_k$, LT^i, FT^i

if $rsn^i_k > rsn(k)$ **then**
  $rsn(k) = rsn^i_k$
  Create the Network Topology T^i
  Run Dijkstra's algorithm and Update rd^i_p
  for (every $n \in N^i$) **do**
    Run Dijkstra's algorithm on n;
    for (every router $p \in T^i$) **do**
      Update rd^i_{pn}
    end for
  end for
  for every router $p \in T^i$ **do**
    $RS^i_p := \varnothing$
    if $rd^i_p < \infty$ **then**
      for every $n \in N^i$ **do**
        if $(rd^i_{kn} < rd^i_k) \vee (d^i_{kn} = d^i_k \wedge |n| < |i|)$ **then** $RS^i_p = RS^i_p \cup \{n\}$
        end if
      end for
    end if
  end for
end if

Algorithm 2 Update PT^i

Input: $ALSA^i_{mj}$, PT^i

if $usn^i_{mj} > sn^i_{mj}$ **then**
  $sn^i_{mj} = usn^i_{mj}$
  $vf^i_{mj} = uvf^i_{mj}$
end if

equal to 1, it will update the Prefix Table and calculate the king anchor. Algorithm 2 illustrates how router i updates its Prefix Table PT^i, when it receives a fresh *AnchorLSA* from anchor m regarding prefix j.

The router forwards the *AnchorLSA* and set the forwarded flag, if the anchor is the king anchor, otherwise it HOLDs the LSA, i.e., do not propagate the *AnchorLSA*, but if anytime in the future, that anchor becomes the king anchor, because of topology changes or the current king anchor stops publishing that prefix, then the router will restore the *AnchorLSA* and forward it as well as setting the forwarded flag in the Prefix Table. Router keeps track of forwarded LSAs and uses this forwarded flag information to avoid sending duplicate *AnchorLSAs*. Detach *AnchorLSA*s (*AnchorLSA* with $vFlag = 0$) will be forwarded all the time, regardless of whether it is from king anchor or not.

Anchor m can be selected as king anchor (k^i_j) if the following statement is true:

$$\begin{aligned} vf^i_{mj} = 1 \wedge \forall [a, vf^i_{aj}] \in PI^i_j,\ vf^i_{aj} = 1 \wedge \\ [rd^i_m < rd^i_a \vee (rd^i_m = rd^i_a \wedge |m| < |a|)] \end{aligned} \tag{3.2}$$

The king anchor is an active anchor that is smallest closest anchor among all active anchors. The distance to prefix j is the minimum of distances to anchors advertising j and is equal to the distance to king anchor. $d^i_j = Min\{rd^i_m | m \in PAI^i_j\} = rd^i_{k^i_j}$. If no active anchor advertise prefix j or non of the active anchors are reachable then $PAI^i_j = \varnothing$ and $d^i_j = \infty$ and prefix j will be marked as unreachable. Router i uses algorithm 3 to select the king anchor. After king selection, router will select valid next hops. A neighbor can be selected as valid next hop for a prefix, if it is valid next hop for the anchor that advertise that prefix and that neighbor is closer to the prefix or it is at the same distance but has lexicographically smaller names. If the following condition is satisfied, then the neighbor will

be selected as a valid next hop.

Algorithm 3 King selection for prefix j

Input: FT^i, PT^i

1: $k_j^i := null; d_j^i := \infty$
2: **for** ecery $m \in PI_j^i$ **do**
3: **if** $vf_{mj}^i = 1$ **then**
4: **if** $rd_m^i < d_j^i \vee (rd_m^i = d_j^i \wedge |m| < k_j^i$ **then**
5: $d_j^i = rd_m^i$
6: $k_j^i = m$
7: **end if**
8: **end if**
9: **end for**

Successor-Set Ordering Condition (SOC)

We define the distance from neighbor to prefix j n (d_{jn}^i) as the minimum of distances to anchors of j known by router i:

$$d_{jn}^i = Min\{rd_{mn}^i | m \in PAI_j^i\} \tag{3.3}$$

Router i selects router n as a valid next hop to prefix j if the following statement is true:

$$d_{jn}^i < \infty \wedge (d_{jn}^i < d_j^i \vee (d_{jn}^i = d_j^i \wedge |n| < |i|)) \tag{3.4}$$

Neighbor n can be selected as next hop toward prefix j if the neighbor is closer to the prefix or router i and its neighbor n are in the same distance from destination and the neighbor has lexicographically smaller name than the router i. Figure 3.3 shows the final state of executing LSCR assuming routers p, q, and r, all advertising prefix j. The bold numbers in figure 3.3 indicate links pointing to best next hop of each router. For instance, both f and l offers paths of distance two to router u, since $|f| < |l|$, router f is selected as best next hop at router u

toward destination p. Each tuple on a link represents the closest smallest anchor and distance to that anchor through that link. In this figure, minimum distance from router d to prefix j is through neighbor g and costs two. Consider router a, router d can reach p in three hops via its neighbor a, because a is not a valid next hop for those destinations and also a is two hops away from prefix j (anchor p) and $|a| < |d|$. These conditions satisfy the Eq. 5 therefore neighbor a is selected as next hop to reach prefix j.

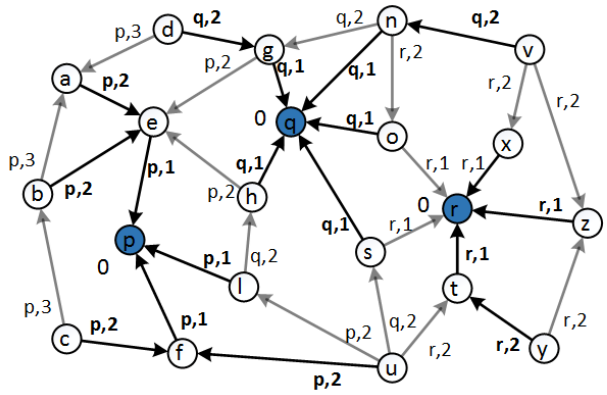

Figure 3.3: Valid next hops to prefix j

Whenever a router receives an up-to-date LSA indicating a change in the network, it runs algorithm 1 or 2, whether they received *RouterLSA* or *AnchorLSA*, and then algorithm 4 to update their routing tables. In section 4 we prove that no permanent routing-loop will be formed if routers choose their next hops based on the SOC conditions.

3.2 correctness

The following theorems prove that LSCR is correct for routing to the nearest anchor of each prefix and routing tables do not contain any permanent loops. It has been verified that sequence numbers can be used correctly to validate updates

Algorithm 4 Update RT_j^i

Input: LT^i, FT^i, PT^i, prefix j

```
1: Ecexute Algorithm 3;
2: if d_j^i < ∞ then
3:     for (every n ∈ N^i) do
4:         d_min^n = ∞
5:         for (every m ∈ PAI_j^i) do
6:             if rd_mn^i < d_min^n then
7:                 d_min^n = rd_mn^i
8:             end if
9:         end for
10:        if d_min^n < ∞ then
11:            if (d_jn^i < d_j^i) ∨ (d_jn^i = d_j^i ∧ |n| < |i|) then
12:                S_j^i = S_j^i ∪ {n}
13:            end if
14:        end if
15:    end for
16: end if
```

[18]. In the rest of this section we assume that there is a finite number of link cost and anchor changes up to time t_0, and no more changes occur after that time, and routers can determine which updates are more recent than others. Also every router has correct information about network topology.

Lemma 1. *The king anchor is the same for all the routers along the shortest path from each router to the king anchor.*

Proof. The proof is by contradiction. Assume that the shortest path P_i from router i to king anchor of prefix j m_j^i, resulted from Dijkstra, consists of h hops and each router selects next hop according to equations 3 and 4. Let $P_i = \{n_1, n_2, S, n_h\}$, where $n_1 = i$, $n_h = m_j^i$, and $n_{k+1} \in RS_m^n$ for $1 \leq k \leq h-1$. We know $rd_m^i = rd_{n_k}^i + rd_m^{n_k} \forall k, 1 \leq k \leq h-1$. Assume that router n_p selected router $a \neq m$ as king router therefore either $rd_a^{n_p} < rd_m^{n_p}$ or $rd_a^{n_p} = rd_m^{n_p} \wedge |a| < |m|$. If $rd_a^{n_p} < rd_m^{n_p}$ then $rd_{n_p}^i + rd_a^{n_p} < rd_{n_p}^i + rd_m^{n_p}$. Therefore, $rd_a^i < rd_m^i$ hence a is the king anchor which contradict our assumption that m is the king anchor. If

$rd_a^{n_p} = rd_m^{n_p} \wedge |a| < |m|$ then $rd_{n_p}^i + rd_a^{n_p} = rd_{n_p}^i + rd_m^{n_p} \wedge |a| < |m|$. Therefore, $rd_a^i = rd_m^i \wedge |a| < |m|$. Again a is the king anchor which contradict our assumption that m is the king anchor. □

Lemma 2. *Each router receives a AnchorLSA from the closest anchor, advertises that prefix.*

Proof. There are two types of *AnchorLSA*s based on the *vFlag* parameter. *DetachLSA*s, i.e., *AnchorLSA*s with $vFlag = 0$, are propagated in the network using flooding mechanism and termination detection is based on sequence number. Therefore every router receives *DetachLSA* from each anchor including the closest one. *AnchorLSA*s with $vflag = 1$ are propagated using diffusion mechanism. Based on lemma 1, all the routers along the shortest path from the king anchor to the router has the same king anchor. Based on the LSA forwarding mechanism, each router forwards *AnchorLSA* form the king anchor. Assume that router i didn't get any *AnchorLSA* from its new king anchor m. Also assume that the shortest path P_i from router i to king anchor of prefix j, m_j^i, resulted from Dijkstra algorithm, consists of h hops. Asume that $P_i = \{n_1, n_2, S, n_h\}$, is such a path. King anchor k is the king anchor of $n_p, 1 \leq p \leq h-1$. Also assume that a router $n_m \in P_i$ did not forward the *AnchorLSA* from its king anchor. This is in contradiction to forwarding mechanism of *AnchorLSA*. Therefore, router i should already received the *AnchorLSA* form its king anchor. □

Theorem 3. *All the routers in the network will converge to the shortest distance to the nearest anchor of the prefix in a finite time after* t_0.

Proof. First note that there is a finite number of prefixes and there is a finite number of anchors and each router process and forward each unique LSA message only once. Without loss of generality, we focus on a specific prefix j. The prefix

can be considered as a virtual node connected to its anchor via a virtual link. Prefix detachment and attachment to the anchor can be considered as link failure and link recovery respectively. For each direction of a link, there is one router (one head of the link) that detects and reports the change in the link in that direction. Therefore for any link l_i, that can be a physical link or virtual link, each router sends at most one LSA for that link after t_0.

Consider an arbitrary router r_0 that never terminates LSCR. Therefore it must send infinite LSA messages after time t_0. Because the network is finite r_0 must processes updates from at least one link l_f. Router r_0 forwards infinite number of LSAs regardingl_f if it receives infinite number of messages regarding L_f, in other words, it is not the case that a router send infinite number of messages regarding l_f because of receiving infinite messages regarding link l_h. It is a direct conclusion of the statement that each router processes and forwards each LSA message only once. If router r_0 receives finite number of LSAs regarding l_f, it will forward finite number of LSAs too. Therefore at least one of its neighbors, call it r_1, must send infinite number of update messages containing an update for link l_f that makes r_0 processes and forward unlimited messages. Neighbor r_1 can send an infinite number of LSAs regarding l_f if at least one of its neighbors, called r_2 forwards an infinite number of LSAs regarding l_f. It is impossible to continue the same line of argument, since the head node of any link can generate at most one update for that link after time t_0 and the network is finite. Therefore, our algorithm can produce only a finite number of update messages for a finite number of link and/or prefix changes and must stop within a finite time after t_0. □

Theorem 4. *No routing-table loops can be formed if NOC is used to select the next hops to anchor at each router.*

Proof. The proof is by contradiction. Assume that a routing loop L_m for anchor

m is formed at time $t_1 > t_0$ when routers update their next-hops satisfying NOC condition. Assume $L_m = \{r_0, r_1, S, r_{q-1}\}$, consisting of q routers is such a loop. We can consider this loop as a path $P_m = \{r_0, r_1, S, r_{q-1}, r_q\}$, where $r_0 = r_q$. Note that $rd^{r_0}_{r_q} = rd^{r_0}_{r_0} = 0$ and $rd^{r_q}_m = rd^{r_0}_m$. Router n_i selects its next hop from the $RS^{n_i}_m$, therefore for $0 \leq i \leq q-1, r_{i+1} \in RS^{n_i}_m$. Based on the NOC for every router $n_i \in P_m$ it must be true that $rd^{n_i}_{mn_{i+1}} < rd^{n_i}_m$ or $rd^{n_i}_{mn_{i+1}} = rd^{n_i}_m$ and $|n_{i+1}| < |n_i|$. Based on definition, $rd^{n_i}_{mn_{i+1}}$ is the distance from $n+1$ to m calculated at i. Routers i and n has the same topology information therefore, $rd^{n_i}_{mn_{i+1}} = rd^{n_{i+1}}_m$. Which results: :

$$(rd^{n_{i+1}}_m < rd^{n_i}_m) \vee (rd^{n_{i+1}}_m = rd^{n_i}_m \wedge |n_{i+1}| < |n_i|) \tag{3.5}$$

Note that $\forall r_i \in P_m, rd^i_m \neq \infty$. Next hop can not have infinite distance to destination, otherwise it contradicts NOC. Therefore, for any two routers $r_u, r_v \in P_m$ and $0 \leq u < v \leq q$ we have:

$$(rd^{r_v}_m < rd^{r_u}_m \vee (rd^{r_v}_m = rd^{r_u}_m \wedge |r_v| < |r_u|) \tag{3.6}$$

The equation is valid for any two routers $r_u, r_v \in P_m$ including r_0 and r_q. therefore it must be true that $rd^{r_0}_m < rd^{r_q}_m$, which is a contradiction, or $rd^{r_0}_m = rd^{r_q}_m \wedge |r_0| < |r_q|$,which is also a contradiction for our assumptions. □

Lemma 5. *For every router i and its neighbor n and for any arbitrary prefix j it is true that $d^i_{jn} = d^n_j$*

Proof. First note that the distance of route n from prefix j is distance of route n from its king anchor: $d^n_j = rd^n_{k^n_j}$. Every router forward the *AnchorLSA* from its king anchor. Therefore, router i is aware of anchor k^n_j, hence, $k^n_j \in PAI^i_j$. From Eq. 3 we have, $d^i_{jn} = Min\{rd^i_{mn} | m \in PAI^i_j\} = rd^n_{k^n_j}$. The last equality is derived from lemma 2, Eq. 2, and the fact $(k^n_j \in PAI^i_j \wedge k^n_j \in PAI^n_j)$. Therefore,

$d^i_{jn} = d^n_j$. □

Theorem 6. *No routing-table loops can be formed if all routers in the network uses LSCR to calculate routes to prefixes.*

Proof. If the prefix is advertised by just one anchor, based on theorem 4 no loops can be formed. If more than one anchor advertise prefix j,using lemma 5 and an argument similar to the proof of theorem 4, we can conclude the statement. □

3.3 Naming

Naming schema depends on the ICN architecture that LSCR is implementing. A hierarchical naming schema such as the one introduced for NLSR can be used for both routers and signaling messages. According this schema, each router is named in the following format: $/ < network > / < site > / < router > /$, where *network* and *site* are assigned based on the network and specific site the router belongs to and *router* is a unique name in that network and site.

LSA messages uses the naming schema like this: */<network>/<site>/<router>/ LSCR/LSA/TypeID/<sequence num>*. The first part is the name of router initiating the LSA, typeID field distinguishes between *RouterLSA* and *AnchorLSA* and *sequencenum* is the sequence number assigned by the router for that LSA.

3.4 Routing Complexity

In this section the communication complexity (i.e. number of messages needed for all routers to have the required information to calculate correct distances), time complexity (maximum time a router needs to receive the information) and storage complexity of LSCR, NLSR, and for shortest-path calculation are investigated. In

the following argument, N and E denote the number of routers and links in the network respectively. The number of distinct anchors available in the network is denoted by D, the average number of instances of the same destination is denoted by R, the average number of neighbors per router is l, and the network diameter is d. C is the number of distinct prefixes in the network.

The number of messages that must be transmitted that a routing algorithm can compete correct routing table for all the destinations is called the communication complexity of the routing algorithm. The storage complexity is the amount of information that should be stored at each router. The time complexity of a routing algorithm is the maximum time needed for all routers to have correct routing information for all destinations.

3.4.1 Traditional Link-State Routing (LSR)

In LSR both network topology and prefix information are broadcasted to entire network. A router that runs LSR, sends adjacency LSA and prefix LSAs, one per each local prefix, and each LSA must be sent to all the other routers in the network, resulting in communication complexity of $O(ERC + lEN)$. Every router stores the complete topology information as well as all instances of all prefixes in the network, therefore, the storage complexity of LSR is $O(RC + E)$. The maximum distance between a source and a destination is d hop, hence the time complexities of LSR is $O(d)$.

3.4.2 Loop-free Distance-Vector Routing (LDVR)

Because of looping problem of traditional distance-vector routing algorithms, the traditional distance-vector routing algorithms can not be used for routing in ICNs with multi-instantiated prefixes. Therefore a loop-free version of distance-

vector routing algorithms should be used. In this approach, each router sends a message for each prefix that locally is available as well as each anchor that prefixes are attached. Hence the communication complexity is $O(RDE + NE)$. The storage complexity of an LDVR is $O(RC + N)$, since each router stores all anchors of all prefixes and destination nodes in the network. The time complexity of the LDVR is same as traditional distance-vector algorithm, which is $O(N)$.

3.4.3 Link-State Content Routing (LSCR)

The information required for LSCR to find correct shortest path to nearest anchor is the complete topology and the prefix information from the nearest anchor including anchor name and sequence number created by that anchor. Therefore, LSCR communication and storage complexity is independent of the number of anchors advertising each prefix. The storage complexity of the algorithm is $O(C+E)$. The increase in the number of replicas will result in distance decrease to nearest anchor. The number of messages exchanged to create the complete topology is $O(EN)$. Each router need to know its nearest anchor and based on the diffusion mechanism, LSCR only propagate specific valid *AnchorLSA*s, the communication complexity is $O(C)$, whoever the communication complexity of *AnchorLSA*s for deletion of a prefix is (CER), where R is number of replicas for a given name prefix. We are working on a diffusion propagation mechanism for propagating the deletion *AnchorLSA* to reduce the communication complexity. Every router needs the complete topology information. The time for receiving such information is the time that a message traverses across the longest shortest path between any two router in the network, d. Therefor, time complexity is $O(d)$.

3.5 Simulation

The LSCR and a routing algorithm similar to NLSR are implemented using SCoNET-Sim, an NS-3 based simulator for content centric networks [54]. In the simulation, NLSR propagates the messages using flooding mechanism. The AT&T core network topology [42], consists of 154 nodes, is used to run simulation experiments. In the following simulation scenarios, we measured the number of messages, number of events, number of operations, and average number of replica per prefix stored in each router. We compared the results for LSCR and NLSR to evaluate the algorithm performance. The quantities are measured for initialization process, link failure and recovery, and attached and detached of a prefix to an anchor.

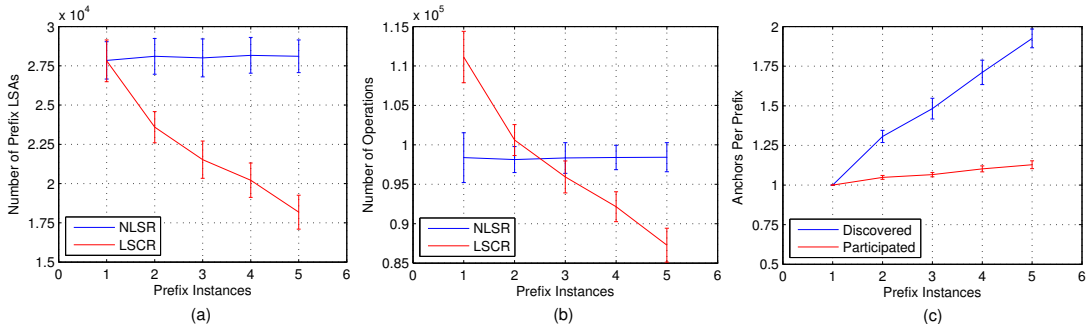

Figure 3.4: LSCR Performance: (a) average number of LSAs packets sent, (b) average number of operations, (c) average number of discovered anchor and number of anchors selected for routing per prefix per node

In the scenarios, we use a total of 210 content objects and 30 anchors. The quantities are also measured as a parameter of the number of replicas for each name prefix or prefix instances. Number of prefix instance indicates the average number of nodes advertise same prefix. For instance prefix instance of 2 means on average two anchors advertise each unique prefix. Different nodes may advertise common prefixes but the whole prefix list of local prefixes of one anchor is different to the list of the other anchor. LSCR performance has been evaluated in five

scenarios. In the first one, each prefix is advertised by only one anchor. In second scenario, for every prefixes, two anchors in the network advertise that anchor. Number of replicas increases in each scenario. In the last one, the average number of replica per prefix is five. Figure 3.4 shows the result of simulation for these five scenarios. Number of operations is the total number of operations performed by each algorithm and is incremented whenever an event occurs, and whenever the statements within a for or while loop are executed. Both LSCR and NLSR run Dijkstra's algorithm to find shortest path to destination, and both run it $|n|$ times for each node, where $|n|$ is the number of neighbors that node has. Therefore Dijkstra just shift the number of operations.

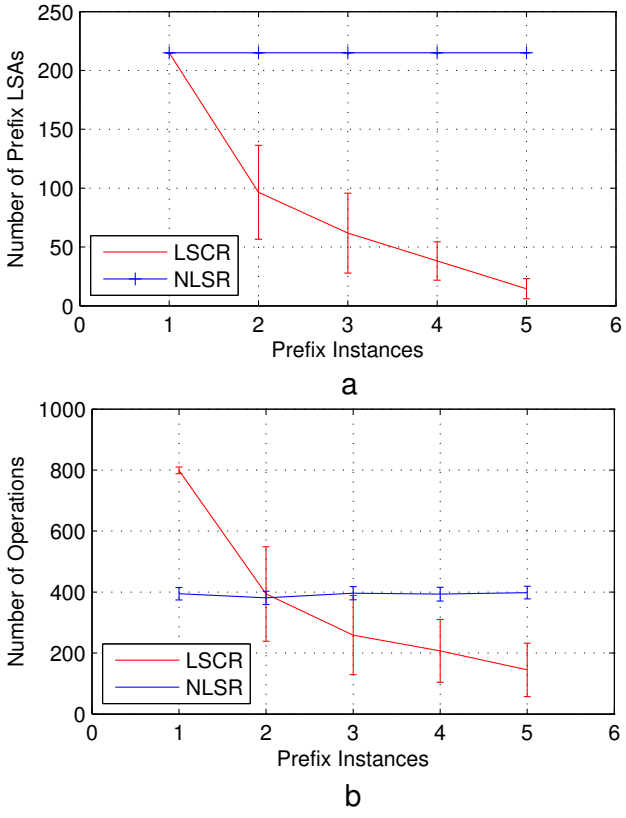

Figure 3.5: Impact of adding a new prefix: (a) average number of LSAs packets sent, (b) average number of operations

Figure 3.4-c illustrates the average number of anchors that LSCR stores for

each prefix and average number of anchors that are participated in routing per prefix per node. As the number of replicas increases the number of replicas stored in the node also increases but even in a network that 5 anchors advertise a prefix, LSCR stores less than two anchor in average, while NLSR stores information of all 5 anchors.

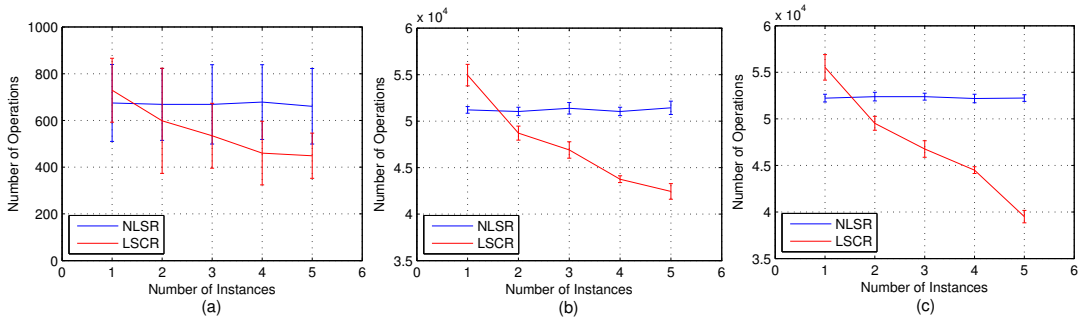

Figure 3.6: Computation overhead of LSCR: (a) after prefix deletion, (b) link failure, (c) link recovery

Figure 3.5 shows the number of LSAs propagated and the number of operations execrated after a new anchor advertises a prefix. The new anchor can be a new node that starts advertising a new content or an intermediate node that temporary caches and advertises that content. As the number of replicas increases, number of LSAs needed to propagate the new anchor information decreases. The number of propagated LSAs in LSCR is almost half the number of LSAs in NLSR when the number prefix instances is two. The computation overhead also decreases as the number of replica for the prefix increases. For instance, the computation overhead of LSCR is half the computation overhead of NLSR when the number prefix instances is 4.

Figure 3.6 illustrate the computation overhead of LSCR compare to NLSR in the case of prefix deletion, link failure and in recovery. In these cases, the performance are almost the same, however, LSCR has better performance in the case that the number of replicas are more than three.

3.6 Summary

The Link State Content Routing (LSCR) protocol is proposed for name-based routing in an ICN architecture. LSCR provides multiple paths to nearest replicas of an NDO or name prefix, and it creates a directed acyclic graph to reach nearest anchors. LSCR relies on full topology information and partial information about prefix replicas; therefore, communication and storage complexity are smaller compared to traditional name-based link state routing algorithms like NLSR and OSPFN.

Routers exchange two types of information: Topology information and anchor information. Each router builds a complete network topology based on topology information. A shortest-path routing algorithm is used to calculate the distance to a given destination through each neighbor of a router. Then neighbors are ranked lexicographically based on their distances. Routers that run LSCR do not receive or store the routing information for all the replicas of the same content. In LSCR routers forward the anchor information selectively based on a distributed computation of preferred publishers. We showed that routes to prefixes are loop-free, even when the prefixes have multiple replicas.

Chapter 4

Diffusion based Content Routing

Clearly, an efficient name-based content routing protocol must be used for any ICN architecture to succeed using name-based forwarding of Interests and requested content. This paper focuses on an approach that avoids the need for periodic messaging by means of diffusing computations [68]. In this chapter we present **DNRP** (*Diffusive Name-based Routing Protocol*), a name-based content routing protocol for ICNs. DNRP provides multiple loop-free routes to the nearest instances of a named prefix or to all instances of a named prefix using only distance information and without requiring periodic updates, knowledge of the network topology, or the exchange of path information. DNRP routing table lists the shortest distances to the nearest anchors of NDOs or name prefixes through one or multiple neighbors. DNRP selects a set of neighbors as valid next hops in such a way that routes to every prefix would be loop-free in every instance.

4.1 DNRP operation

DNRP finds the shortest path(s) to the nearest replica(s) of name prefixes. To ensure that loop-free routes to named prefixes are maintained at every instant

independently of the state of the network or prefixes, DNRP establishes a lexicographic ordering among the routes to prefixes reported and maintained by routers. The lexicographic ordering of routes is based on two sufficient conditions for loop freedom with respect to a given prefix that allow for multiple next hops to prefixes along loop-free routes. DNRP diffuses the computation of new loop-free routes when the loop-free conditions are not satisfied.

Every piece of data in the network is a *Named-Data Object* (*NDO*), represented by a name that belongs to a name prefix or simply a *prefix* advertised by one or more producer(s). Name prefixes can be simple and human-readable or more complicated and self certifying, or may even be a cryptographic hash of the content. Content names can be flat or hierarchical.

A router attached to a producer of content that advertises a name prefix is called an *anchor* of that prefix. At each router, DNRP calculates routes to the nearest anchor(s) of known name prefixes, if there is any, and selects a subset of the neighbors of the router as valid next hops to reach name prefixes, such that no routing-table loop is created at any router for any name prefix. Caching sites are not considered content producers and hence routes to cached content are not advertised in DNRP. Our description assumes that routers process, store, and transfer information correctly and that they process routing messages one at a time within a finite time. Every router has a unique identifier or a name that can be flat or hierarchical.

4.1.1 Messages and Data Structures

Each router i stores the list of all active neighbor routers (N^i), and the cost of the link from the router to each such neighbor. The cost of the link from router i to its neighbor n is denoted by l_n^i. Link costs can vary in time but are always

positive.

The routing information reported by each of the neighbors of router i is stored in its *neighbor table* (NT^i). The entry of NT^i regarding neighbor n for prefix p is denoted by NT^i_{pn} and consists of the name prefix (p), the distance to prefix p reported by neighbor n (d^i_{pn}), and the anchor of that prefix reported by neighbor n (a^i_{pn}). If router i is the anchor of prefix p itself, then $d^i_{pi} = 0$.

Router i stores routing information for each known prefix in its routing table (RT^i). The entry in RT^i for prefix p (RT^i_p) specifies: the name of the prefix (p); the distance to the nearest instance of that prefix (d^i_p); the feasible distance to the prefix (fd^i_p); the neighbor that offers the shortest distance to the prefix (s^i_p), which we call the successor of the prefix; the closest anchor to the prefix (a^i_p); the state mode (mf^i_p) regarding prefix p, which can be *PASSIVE* or *ACTIVE*; the origin state (o^i_p) indicating whether router i or a neighbor is the origin of query in which the router is active; the update flag list (FL^i_p); and the list of all valid next hops (V^i_p).

FL^i_p consists of four flags for each neighbor n. An update flag (uf^i_{pn}) denotes whether or not the routing message should be sent to that neighbor. A type flag (tf^i_{pn}) indicates the type of routing message the router has to send to neighbor n regarding prefix p (i.e., whether it is an *UPDATE*, *QUERY*, or *REPLY*). A pending-reply flag (rf^i_{pn}) denotes whether the router has sent a *QUERY* to that neighbor and is waiting for *REPLY*. A pending-query flag (qf^i_{pn}) is set if the router received a *QUERY* from its neighbor n and has not responded to that *QUERY* yet.

Router i sends a routing message to each of its neighbors containing updates made to RT^i since the time it sent its last update message. A routing message from router i to neighbor n consists of one or more updates, each of which carries

information regarding one prefix that needs updating. The update information for prefix p is denoted by U_p^i and states: (a) the name of the prefix (p); (b) the message type (ut_p^i) that indicates if the message is an *UPDATE*, *QUERY*, or *REPLY*; (c) the distance to p; and (d) the name of the closest anchor.

The routing update received by router i from neighbor n is denoted by U_n^i. The update information of U_n^i for prefix p, u_{pn}^i, specifies the prefix name (p), the distance from n to that prefix (ud_{pn}^i), the name of the anchor of that prefix (ua_{pn}^i), and a message type (ut_{pn}^i).

4.1.2 Sufficient Conditions for Loop Freedom

The conditions for loop-free routing in DNRP are based on the feasible distance maintained at each router and the distances reported by its neighbors for a name prefix. One condition is used to determine the new shortest distance through a loop-free path to a name prefix. The other is used to select a subset of neighbors as next hops to a name prefix.

Source Router Condition (SRC): Router i can select neighbor $n \in N^i$ as a new successor s_p^i for prefix p if:

$$(\ d_{pn}^i < \infty \ ) \ \wedge (\ d_{pn}^i < fd_p^i \vee [d_{pn}^i = fd_p^i \wedge |n| < |i|] \ ) \ \wedge$$
$$(\ d_{pn}^i + l_n^i = Min\{d_{pv}^i + l_v^i | v \in N^i\} \ ). \ \square$$

SRC simply states that router i can select neighbor n as its successor to prefix p if n reports a finite distance to that prefix, offers the smallest distance to prefix p among all neighbors, and either its distance to prefix p is less than the feasible distance of router i or its distance is equal to the feasible distance of i but $|n| < |i|$. If two or more neighbors satisfy *SRC*, the neighbor that satisfies *SRC* and has the smallest identifier is selected. If none of the neighbors satisfies *SRC* the router

keeps the current successor, if it has any. The distance of router i to prefix, d_p^i, is defined by the distance of the path through the selected successor.

A router that has a finite feasible distance ($fd_p^i < \infty$) selects a subset of neighbors as valid next hops at time t if they have a finite distance to destination and are closer to destination. The following condition is used for this purpose.

Next-hop Selection Condition (NSC): Router i with $fd_p^i < \infty$ adds neighbor $n \in N^i$ to the set of valid next hops if:

$$(\ d_{pn}^i < \infty \ ) \wedge (\ d_{pn}^i < fd_p^i \vee [d_{pn}^i = fd_p^i \wedge |n| < |i|] \ ).\Box$$

NSC states that router i can select neighbor n as a next hop to prefix p if either the distance from n to prefix p is smaller than the feasible distance of i or its distance is equal to the feasible distance and $|n| < |i|$. *NSC* orders next hops lexicographically based on their distance to a prefix and their names. It is shown that no routing-table loops can be formed if *NSC* is used to select the next hops to prefixes at each router. Note that the successor is also a valid next hop. The successor to a prefix is a valid next hop that offers the smallest cost.

SRC and *NSC* are *sufficient conditions* that, as we show subsequently, ensure loop-freedom at every instance but do not guarantee shortest paths to destinations. DNRP integrates these sufficient conditions with inter-nodal synchronization signaling to achieve both loop freedom at every instant and shortest paths for each destination.

4.1.3 DNRP Operation

A change in the network, such as a link-cost change, the addition or failure of a link, the addition or failure of a router, the addition or deletion of a prefix, or the

addition or deletion of a replica of a prefix can cause one or more computations at each router for one or more prefixes. A computation can be either a *local computation* or a *diffusing computation*. In a local computation a router updates its successor, distance, next hops, and feasible distance independently of other routers in the network. On the other hand, in a diffusing computation a router originating the computation must coordinate with other routers before making any changes in its routing-table entry for a given prefix. DNRP allows a router to participate in at most one diffusing computation per prefix at any given time.

A router can be in *PASSIVE* or *ACTIVE* mode with respect to a given prefix independently of other prefixes. A router is *PASSIVE* with respect to prefix p if it is not engaged in any diffusing computation regarding that prefix. A router initializes itself in *PASSIVE* mode and with a zero distance to all the prefixes for which the router itself serves as an anchor. An infinite distance is assumed to any non-local (and hence unknown) name prefix.

Initially, no router is engaged in a diffusing computation ($o_p^i = 0$). When a *PASSIVE* router detects a change in a link or receives a *QUERY* or *UPDATE* from its neighbor that does not affect the current successor or can find a feasible successor, it remains in *PASSIVE* mode. On the other hand, if the router cannot find a feasible successor then it enters the *ACTIVE* mode and keeps the current successor, updates its distance, and sends *QUERY* to all its neighbors. Table 4.1 shows the transit from one state to another. Neighbor k is a neighbor other than the successor s.

Algorithm 5 shows the processing of messages by a router in *PASSIVE* mode. Algorithm 6 shows the steps taken in *ACTIVE* mode. Algorithm 7 shows the steps taken to process a routing update.

Handling A Single Diffusing Computation: Routers are initialized in *PASSIVE* mode. Each router continuously monitors its links and processes the

Table 4.1: State transit in DNRP

Mode	*State*	Event	*Next State*
Passive	0	Events from a neighbor k, SRC satisfied	0
		Events from a neighbor k, SRC not satisfied	1
		$QUERY$ from the *Successor*	3
Active	1	Receives last $REPLY$	0
		Change in distance to *Successor*	2
		$QUERY$ from the *Successor*	4
	2	Receives last $REPLY$, SRC satisfied	0
		Receives last $REPLY$, SRC not satisfied	1
		$QUERY$ from the *Successor*	4
	3	Receives last $REPLY$	0
		Change in distance to the *Successor*	4
	4	Receives last $REPLY$, SRC satisfied	0
		Receives last $REPLY$, SRC not satisfied	3

routing messages received from its neighbors. When router i detects a change in the cost or state of a link, or a change in its neighbor table that causes a change in its distance to prefix p (d_p^i), it first tries to select a new successor that satisfies SRC. If such a successor exists, the router carries out a local computation, updates its distance, successor, and closest anchor, and exits the computation. In a local computation, router i computes the minimum cost to reach the destination and updates $d_p^i = min\{d_{pn}^i + l_n^i | n \in N^i\}$. If its distance changes, router i sends a routing message with $ut_p^i = UPDATE$. Router i also updates its feasible distance to equal the smaller of its value and the new distance value, i.e., $fd_p^i(new) = min\{fd_p^i(old), d_p^i\}$.

An $UPDATE$ message from a neighbor is processed using the same approach stated above. If a router receives a $QUERY$ from its neighbor other than its successor while it is in $PASSIVE$ mode, it updates the neighbor table, checks for a feasible successor according to SRC and replies with d_p^i, if it succeeds. If router i cannot find a neighbor that satisfies SRC after a change in a link or neighbor-table entry, then it starts out a diffusing computation by setting the new distance as the distance through its current successor, enters the $ACTIVE$ mode ($mf_j^i = ACTIVE$) and sets the corresponding flag ($rf_p^i n$) for each neighbor

n. After entering the *ACTIVE* mode, router i sets the new distance as the cost of the path through the current successor ($d_p^i = d_{ps_p^i}^i + l_{s_p^i}^i$) and sends a routing message with $ut_p^i = QUERY$. Router i uses the pending reply flag (rf_{pn}^i) to keep track of the neighbors from which a *REPLY* has not been received. When a router becomes *ACTIVE* it sets the update flag ($uf_{pn}^i = 1$), and also sets the type flags ($tf_{pn}^i = QUERY | \forall n \in N^i$) and sends the routing messages to all its neighbors.

Algorithm 5 Processing routing messages in *PASSIVE* mode

INPUT: $RT^i, NT^i, l_n^i, u_{pn}^i$;
[o] verify u_{pn}^i;
$d_{pn}^i = ud_{pn}^i$; $d_{min} = \infty$;
for each $k \in N^i - \{i\}$ **do**
  if $(d_{pk}^i + l_k^i < d_{min}) \vee (d_{pk}^i + l_k^i = d_{min} \wedge |k| < |s_{new}|)$ **then**
    $s_{new} = k; d_{min} = d_{pk}^i + l_k^i$;
  end if
end for
if $(d_{ps_{new}}^i < fd_p^i \vee [d_{ps_{new}}^i = fd_p^i \wedge |s_{new}| < |i|])$ **then**
  if $s_p^i \neq s_{new}$ **then** $s_p^i = s_{new}$; $a_p^i = a_{ps_{new}}^i$
  if $d_p^i \neq d_{min}$ **then**
    $d_p^i = d_{min}; fd_p^i = min\{fd_p^i, d_p^i\}; V_p^i = \phi$;
    for each $k \in N^i - \{i\}$ **do**
      $uf_{pk}^i = 1$; $tf_{pk}^i =$*UPDATE*;
      if $(d_{kp}^i < fd_p^i \vee [d_{pk}^i = fd_p^i \wedge |k| < |i|])$ **then**
        $V_p^i = V_p^i \cup k$;
      end if
    end for
    if $ut_{pn}^i = QUERY$ **then** $tf_{pn}^i =$*REPLY*;
  end if
else
  $mf_p^i =$ *ACTIVE*; $d_p^i = d_{ps_p^i}^i + l_{s_p^i}^i$;
  if $(n = s_p^i \wedge ut_{pn}^i = QUERY)$ **then** $o_p^i = 3$; **else** $o_p^i = 1$;
  for each $k \in N^i - \{i\}$ **do** $uf_{pn}^i = 1$; $tf_{pn}^i =$*QUERY*;
end if

When a router is in *ACTIVE* mode, it cannot change its successor or fd_p^i until it receives the replies to its *QUERY* from all its neighbors. After receiving all replies (i.e. $rf_{pn}^i = 0 | \forall n \in N^i$), router i becomes *PASSIVE* by resetting its feasible distance. The router then selects the new successor and sends *UPDATE* messages to its neighbors. More specifically, router i sets $fd_p^i = \infty$ which insures that the router can find a new successor that satisfies *SRC* and then sets $fd_p^i = d_p^i = min\{d_{pn}^i + l_n^i | n \in N^i\}$ and becomes *PASSIVE*.

Algorithm 6 Processing routing messages in *ACTIVE* mode

INPUT: RT^i, NT^i, u^i_{pn};
[**o**] verify u^i_{pn};
$d^i_{pn} = ud^i_{pn}$;
if $ut^i_{pn} = REPLY$ **then**
  $rf^i_{pn} = 0; lastReply = true$;
  for each $k \in N^i - \{i\}$ **do**
    if $rf^i_{pk} = 0$ **then** $lastReply = false$;
  end for
  if $lastReply = true$ **then**
    if $o^i_p = 1 \vee o^i_p = 3$ **then** $fd^i_p = \infty$
    Execute Algorithm 7
  end if
else if $ut^i_{pn} = QUERY$ **then**
  if $(o^i_p = 1 \vee o^i_p = 2)$ **then**
    if $n \neq s^i_p$ **then** $uf^i_{pn} = 1$; tf^i_{pn} =*REPLY;* **else** $o^i_p = 4$;
  end if
  if $(o^i_p = 3 \vee o^i_p = 4)$ **then** $uf^i_{pn} = 1$; tf^i_{pn} =*REPLY;*
end if

If router i receives a *QUERY* from a neighbor other than its successor while it is *ACTIVE*, it simply replies to its neighbor with a *REPLY* message stating the current distance to the destination. The case of a router receiving a *QUERY* from its successor while it is *ACTIVE* is described subsequently in the context of multiple diffusing computations. *UPDATE* messages are processed and neighbor tables are updated, but the successor or distance is not changed until the router receives all the replies it needs to transition to the *PASSIVE* mode. While a router is in *ACTIVE* mode, neither a *QUERY* nor an *UPDATE* can be sent.

Handling Multiple Diffusing Computations: Given that a router executes each local computation to completion, it handles multiple local computations for the same prefix one at a time. Similarly, a router handles multiple diffusing computation for the same prefix by processing one computation at a time. An *ACTIVE* router i can be in one of the following four states: (1) router i originated a diffusing computation ($o^i_p = 1$), (2) metric increase detected during *ACTIVE* mode ($o^i_p = 2$), (3) diffusing computation is relayed ($o^i_p = 3$), or (4) successor

Algorithm 7 Update RT_p^i

INPUT: $RT^i, NT^i, l_n^i, u_{pn}^i$;
$d_{min} = \infty$;
for each $k \in N^i - \{i\}$ **do**
  if $(d_{pk}^i + l_k^i < d_{min}) \vee (d_{pk}^i + l_k^i = d_{min} \wedge |k| < |s_{new}|)$ **then**
    $s_{new} = k; d_{min} = d_{pk}^i + l_k^i$;
  end if
end for
if $(d_{ps_{new}}^i < fd_p^i \vee [d_{ps_{new}}^i = fd_p^i \wedge |s_{new}| < |i|])$ **then**
  $o_p^i = 0; mf_p^i = PASSIVE$;
  if $s_p^i \neq s_{new}$ **then** $s_p^i = s_{new}$;
  if $d_p^i \neq d_{min}$ **then**
    $d_p^i = d_{min}; fd_p^i = min\{fd_p^i, d_p^i\}; V_p^i = \phi$;
    for each $k \in N^i - \{i\}$ **do**
      $uf_{pk}^i = 1$; $tf_{pk}^i =$*UPDATE*;
      if $(d_{pk}^i < fd_p^i \vee [d_{pk}^i = fd_p^i \wedge |k| < |i|])$ **then**
        $V_p^i = V_p^i \cup k$;
      end if
    end for
    if $qf_{ps_p^i}^i(old) = 1$ **then** $tf_{pn}^i =$*REPLY;*
  end if
else
  if $o_p^i = 2$ **then** $o_p^i = 1$ **else** $o_p^i = 3$;
  for each $k \in N^i - \{i\}$ **do** $uf_{pn}^i = 1$; $tf_{pn}^i =$*QUERY*;
end if

metric changed during *ACTIVE* mode ($o_p^i = 4$). If the router is in *PASSIVE* mode then its state is 0 (i.e., $o_p^i = 0$).

Consider the case that a router i is *ACTIVE* and in State 1 ($o_p^i = 1$). If the router receives the last *REPLY* to its query, then it resets its feasible distance to infinity, checks *SRC* to find the new successor, and sends an *UPDATE* to all its neighbors. On the other hand, if router i detects a change in the link to its successor then it updates its neighbor table and sets $o_p^i = 2$.

If router i is in State 2, receives the last *REPLY*, and can find a feasible successor using *SRC* with the current feasible distance, then it becomes *PASSIVE* and sends an *UPDATE* to all its neighbors($o_p^i = 0$). Otherwise, it sends a *QUERY* with the current distance and sets $o_p^i = 1$.

Router i uses the pending query flag (qf_{pn}^i) to keep track of the replies that

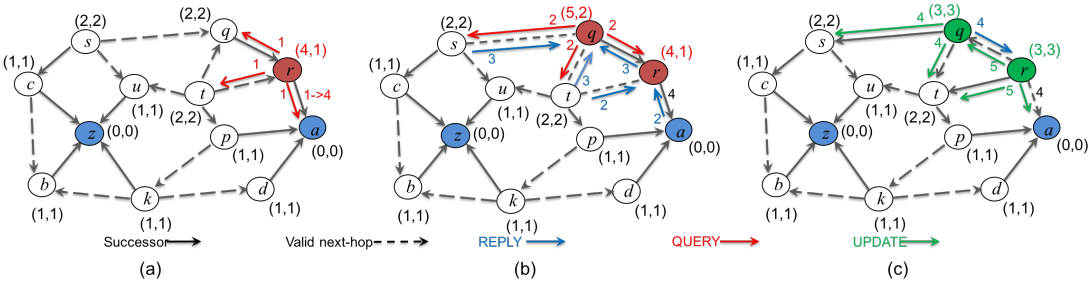

Figure 4.1: DNRP Operation Example

have been received for its *QUERY* regarding prefix p. If router i is in either State 1 or 2 and receives a *QUERY* from its current successor to the prefix, then it sets $qf^i_{ps^i_p=1}$ and transitions to State 4 (i.e., it sets $o^i_p = 4$).

If a router in *PASSIVE* mode receives a *QUERY* from its successor, it searches for a new successor that satisfies *SRC*. If it cannot find such a successor then it keeps the current successor, updates its distance, and becomes *ACTIVE*. Then, the router sends *QUERY* to all of its neighbors and sets $o^i_p = 3$.

When router i in state 3 receives *REPLY* from all of its neighbors, it resets its feasible distance, $fd^i_p = \infty$, selects a new successor, updates the V^i_p and sends *UPDATE* to its neighbors and *REPLY* to its the previous successor. If the router detects a link failure or a cost increase in the link to its current successor, the router sets $o^i_p = 4$ to indicate that a topology change occurred while the router is in *ACTIVE* mode. A router handles the case of the failure of the link with its successor as if it had received a *REPLY* from its successor with $d^i_{ps^i_p} = \infty$.

If router i is in State 4, ($o^i_p = 4$) and it receives replies from all its neighbors, then it tries to find a feasible successor that satisfies *SRC* with the current value of fd^i_p. If such a successor exists, the router updates its successor, distance, and next hops for prefix p, and sends an *UPDATE* message to its neighbors as well as *REPLY* to the previous successor. Otherwise, it sets $o^i_p = 3$ and sends a *QUERY*

with the new distance.

While router i is in *ACTIVE* mode regarding a prefix, if a *QUERY* is received for the prefix from a neighbor other than the current successor, the router updates the neighbor table and sends a *REPLY* to that neighbor. If a router in *PASSIVE* mode receives a *QUERY* from a neighbor other than the current successor, the router updates its neighbor table. If the feasibility condition is not satisfied anymore, the router sends a *REPLY* to the neighbor that provides the current value d_p^i before it starts its own computation.

4.1.4 Example of DNRP Operation

Figure 4.1 illustrates the operation of DNRP with a simple example. The figure shows the routing information used for a single prefix when routers a and z advertise that prefix and each link has unit cost. The tuple next to each router states the *distance* and the $feasible\ distance$ of the router for that prefix. The red, blue, and green arrows represent the *QUERY*, *REPLY*, and *UPDATE* messages respectively and the number next to the arrow shows the time sequence in which that message is sent. Figure 4.1 (a) shows the change in the cost of link (r, a). Router r detects this change and becomes *ACTIVE* and sends *QUERY* to its neighbors.

Router q receives the *QUERY* from its successor and cannot find a feasible successor (Figure 4.1(b)). Therefore, it becomes *ACTIVE* and sends a *QUERY* to its neighbors. Router r receives *REPLY* from a and t, and a *QUERY* from q. Given that q is not a successor for router r, r sends *REPLY* to q. After receiving *REPLY* from routers r, s and t, router q becomes *PASSIVE* again and sends its *REPLY* to its previous successor, r. In turn, this means that r receives all the replies it needs, becomes *PASSIVE*, and resets its feasible distance. The operation

of DNRP is such that only a portion of the routers are affected by the topology change.

4.1.5 Routing to all instances of a prefix

DNRP enables routers to maintain multiple loop-free routes to the nearest anchor of a name prefix. In some ICN architectures, such as NDN and CCNx, an anchor of a name-prefix may have some but not necessarily all the content corresponding to a given prefix. Therefore, simply routing to nearest replica may cause some data to be unreachable, and the ability to contact all anchors of a prefix is needed. To address this case, a multi-instantiated destination spanning tree (*MIDST*) can be used alongside DNRP to support routing to all anchors of the same prefix. A MIDST is established in a distributed manner. Routers that are aware of multiple anchors for the same prefix exchange routing updates to establish the spanning tree between all anchors of a prefix. Once the *MIDST* is formed for a given prefix, the first router in the MIDST that receives a packet forwards it over the *MIDST* to all of the anchors. The details of how a *MIDST* can be established in DNRP are omitted for brevity; however, the approach is very much the same as that described in [35].

4.2 Correctness of DNRP

The following theorems prove that DNRP is loop-free at every instant and considers each computation individually and in the proper sequence. From these results, the proof that DNRP converges to shortest paths to prefixes is similar to the proof presented in [30] and due to space limitation is omitted. We assume that each router receives and processes all routing messages correctly. This implies that

each router processes messages from each of its neighbors in the correct order.

Theorem 7. *No routing-table loops can form in a network in which routers use NSC to select their next hops to prefixes.*

Proof. Assume for the sake of contradiction that all routing tables are loop-free before time t_l but a routing-table loop is formed for prefix p at time t_l when router q adds its neighbor n_1 to its valid next-hop set V_p^q. Because the successor is also a valid next hop, router q must either choose a new successor or add a new neighbor other than its current successor to its valid next-hop set at time t_l. We must show that the existence of a routing-table loop is a contradiction in either case.

Let L_p be the routing-table loop consisting of h hops starting at router q, ($L_p = \{q = n_{0,new}, n_{1,new}, n_{2,new}, \ldots, n_{h,new}\}$) where $n_{h,new} = q$, $n_{i+1,new} \in V_p^{n_i}$ for $0 \leq i \leq h$.

The time router n_i updates its valid next-hop set to include $n_{i+1,new}$ is denoted by t_{new}^i. Assume that the last time router n_i sent an *UPDATE* that was processed by its neighbor n_{i-1}, is t_{old}^i. Router n_i revisits valid next hops after any changes in its successor, distance, or feasible distance; therefore, $t_{old}^i \leq t_{new}^i \leq t_l$ and $d_{pn_{i+1}}^{n_i}(t_l) = d_{pn_{i+1}}^{n_i}(t_{old})$. Also, by definition, at any time t_i, $fd_p^i(t_i) \leq d_p^i(t_i)$, and $fd_p^i(t_2) \leq fd_p^i(t_1)$ if $t_1 < t_2$. Therefore,

$$fd_p^i(t_2) \leq d_p^i(t_1) \text{ such that } \; t_1 < t_2 \tag{4.1}$$

If router n_i selects a new successor at time t_{new}^i then:

$$d_{pn_i}^{n_{i-1}}(t_l) = d_p^{n_i}(t_{old}) \geq fd_p^{n_i}(t_{old}) \geq fd_p^{n_i}(t_{new}) \tag{4.2}$$

Using *NSC* ensures that

$$\begin{aligned}(fd_p^{n_i}(t_{new}) > d_{pn_{i+1}}^{n_i}(t_l)) \\ \vee(fd_p^{n_i}(t_{new}) = d_{pn_{i+1}}^{n_i}(t_l) \wedge |n_i| > |n_{i+1}|)\end{aligned} \tag{4.3}$$

From Eqs. (4.2) and (4.3) we have:

$$\begin{aligned}(d_{pn_i}^{n_{i-1}}(t_l) > d_{pn_{i+1}}^{n_i}(t_l)) \\ \vee(d_{pn_i}^{n_{i-1}}(t_l) = d_{pn_{i+1}}^{n_i}(t_l) \wedge |n_i| > |n_{i+1}|)\end{aligned} \tag{4.4}$$

Therefore, for $0 \leq k \leq h$ in L_p it is true that:

$$\begin{aligned}(d_{pn_1}^{n_0}(t_l) > d_{pn_{k+1}}^{n_k}(t_l)) \\ \vee(d_{pn_1}^{n_0}(t_l) = d_{pn_{k+1}}^{n_k}(t_l) \wedge |n_0| > |n_k|)\end{aligned} \tag{4.5}$$

If $d_{pn_i}^{n_{i-1}}(t_l) > d_{pn_{i+1}}^{n_i}(t_l)$ in at least one hop in L_p then it must be true that, for any given $k \in \{1, 2, \ldots, h\}$, $d_{pn_{k+1}}^{n_k}(t_l) > d_{pn_{k+1}}^{n_k}(t_l)$, which is a contradiction. If at any hop in the L_p it is true that $d_{pn_i}^{n_{i-1}}(t_l) = d_{pn_{i+1}}^{n_i}(t_l)$, then $|k| > |k|$, which is also a contradiction. Therefore, no routing-table loop can be formed when routers use *NSC* to select their next hops to prefix p. □

Lemma 8. *A router that is not the origin of a diffusing computation sends a REPLY to its successor when it becomes PASSIVE.*

Proof. A router that runs DNRP can be in either *PASSIVE* or *ACTIVE* mode for a prefix p when it receives a *QUERY* from its successor regarding the prefix. Assume that router i is in *PASSIVE* mode when it receives a *QUERY* from its successor. If router i finds a neighbor that satisfies *SRC*, then it sets its new successor and sends a *REPLY* to its old successor. Otherwise, it becomes *ACTIVE*, sets $o_p^i = 3$,

and sends a *QUERY* to all its neighbors. Router i cannot receive a subsequent *QUERY* from its successor regarding the same prefix, until it sends a *REPLY* back to its successor. If the distance does not increase while router i is *ACTIVE* then o_p^i remains the same (i.e. $o_p^i = 3$). Otherwise, router i must set $o_p^i = 4$. In both cases router i must send a *REPLY* when it becomes *PASSIVE*.

Assume that router i is in *ACTIVE* mode when it receives a *QUERY* from its successor s. Router s cannot send another *QUERY* until it receives a *REPLY* from all its neighbors to its query, including router i. Hence, router i must be the origin of the diffusing computation for which it is *ACTIVE* when it receives the *QUERY* from s, which means that $o_p^i = 1$ or $o_p^i = 2$. In both cases router i sets $o_p^i = 4$ when it receives a *QUERY* form its successor s and s must send a *REPLY* in response to the *QUERY* from i because, i is not the successor for s. After receiving the last *REPLY* from its neighbors, either router i finds a feasible successor and sends a *REPLY* to s ($o_p^i = 0$) or it propagates the diffusing computation forwarded by s by sending a *QUERY* to its neighbors and setting $o_p^i = 3$. Router i then must send a *REPLY* to s when it receives the last *REPLY* for the *QUERY* it forwarded from s.

Hence, independently of its current mode, router i must send a *REPLY* to a *QUERY* it receives from its successor when it becomes *PASSIVE*. □

Lemma 9. *Consider a network that is loop free before an arbitrary time t and in which a single diffusing computation takes place. If node n_i is PASSIVE for prefix p at that time, then it must be true that* $(d_{pn_i}^{n_{i-1}}(t) > d_{pn_{i+1}}^{n_i}(t)) \vee (d_{pn_i}^{n_{i-1}}(t) = d_{pn_{i+1}}^{n_i}(t) \wedge |n_i| > |n_{i+1}|)$ *independently of the state of other routers in the chain of valid next hops $\{n_{i-1}, n_i, n_{i+1}\}$ for prefix p.*

Proof. Assume that router n_i is *PASSIVE* and selects router n_{i+1} as a valid next

hop. According to *NSC* it must be true that:

$$\begin{aligned}(d^{n_i}_{pn_{i+1}}(t) < fd^{n_i}_p(t) \leq d^{n_i}_p(t)) \vee \\ (d^{n_i}_{pn_{i+1}}(t) = fd^{n_i}_p(t) \leq d^{n_i}_p(t) \wedge |n_{i+1}| < |n_i|)\end{aligned} \tag{4.6}$$

Assume that n_i did not reset $fd^{n_i}_p$ the last time $t_{new} < t$ when n_i became *PASSIVE* and selected its successor s_{new} and updated its distance $d^{n_i}_p(t_{new}) = d^{n_i}_p(t)$. If router n_{i-1} processed the message that router n_i sent after updating its distance, then: $d^{n_{i-1}}_{pn_i}(t) = d^{n_i}_p(t_{new})$. Substituting this equation in 4.6 renders the result of this lemma.

On the other hand, If router n_{i-1} did not process the message that router n_i sent after updating its distance and before t, then $d^{n_{i-1}}_{pn_i}(t) = d^{n_i}_p(t_{old})$. Based on the facts that router n_i did not reset its feasible distance and Eq. 4.1 holds for this case. Therefore:

$$d^{n_{i-1}}_{pn_i}(t) = d^{n_{i-1}}_{pn_i}(t_{old}) > fd^{n_i}_p(t) \tag{4.7}$$

Now consider the case that n_i becomes *PASSIVE* at time t_{new} and changes its successor from s_{old} to s_{new} by reseting its feasible distance. The case that n_{i-1} processed the message that router n_i sent after becoming *PASSIVE* is the same as before. Assume that n_{i-1} did not process the message that n_i sent at time t_{new}. Furthermore, assume that router n_i becomes *ACTIVE* at time t_{old}, with a distance $d^{n_i}_p(t_{old}) = d^{n_i}_{ps_{old}} + l^{n_i}_{s_{old}}$. Router n_i cannot change its successor or experience any increment in its distance through s_{old}; hence, $d^{n_i}_p(t_{new}) \leq d^{n_i}_p(t_{old})$. On the other hand, the distance through the new successor must be the shortest and so $d^{n_i}_p(t_{new}) = d^{n_i}_{ps_{new}} + l^{n_i}_{s_{new}} \leq d^{n_i}_p(t_{old})$. Router n_i becomes *PASSIVE* if it receives a *REPLY* from each of its neighbors including n_{i-1}. Therefore, n_{i-1} must

be notified about $d_p^{n_i}(t_{old})$. Therefore:

$$d_{pn_i}^{n_{i-1}}(t) = d_p^{n_i}(t_{old}) \geq d_p^{n_i}(t_{new}) = d_p^{n_i}(t). \tag{4.8}$$

Substituting this equation in 4.6 renders the result of this lemma. Therefore, the lemma is true in all cases. □

Lemma 10. *Consider a network that is loop free before an arbitrary time t and in which a single diffusing computation takes place. Let two network nodes n_i and n_{i+1} be such that $n_{i+1} \in V_p^{n_i}$. Independently of the state of these two nodes, it must be true that:*

$$\begin{aligned} (fd_p^{n_i}(t) > fd_p^{n_{i+1}}(t)) \vee \\ (fd_p^{n_i}(t) = fd_p^{n_{i+1}}(t) \wedge |n_i| > |n_{i+1}|) \end{aligned} \tag{4.9}$$

Proof. Consider the case that router n_i is *PASSIVE*, then from Lemma 9 and the fact that routers select their next hops based on *NSC*, it must be true that:

$$\begin{aligned} (fd_p^{n_i} > d_{pn_{i+1}}^{n_i}(t)) \vee \\ (fd_p^{n_i} = d_{pn_{i+1}}^{n_i}(t) \wedge |n_i| > |n_{i+1}|) \end{aligned} \tag{4.10}$$

Consider the case that router n_{i+1} is *ACTIVE*. Router n_{i+1} cannot change its successor or increase its feasible distance. If router n_i processed the last message that router n_{i+1} sent before time t, then: $d_{pn_{i+1}}^{n_i}(t) = fd_p^{n_{i+1}}(t)$ and the lemma is true. Assume router n_i did not process the last message that router n_{i+1} sent before time t. Router n_i must send a *REPLY* to n_{i+1} the last time that router n_{i+1} became *PASSIVE* at time t_p reporting a distance $d_p^{n_{i+1}}(t_{old}) = d_{ps_{old}}^{n_{i+1}} + l_{s_{old}}^{n_{i+1}}$.

If router n_{i+1} did not reset its feasible distance since the last time it became

passive, $fd^{n_{i+1}}$, then, $d_p^{n_{i+1}}(t_{old}) \geq fd_p^{n_{i+1}}(t)$. Consider the case that router n_{i+1} resets $fd^{n_{i+1}}$ the last time before t that it becomes *PASSIVE*. Router n_{i+1} cannot change its successor or experience any increment in its distance through its old successor, s_{old}. Hence, $d_p^{n_{i+1}}(t_{new}) \leq d_p^{n_{i+1}}(t_{old})$. On the other hand, the distance through the new successor must be the smallest among all neighbors including the old successor and so $d_p^{n_{i+1}}(t_{new}) = (d_{ps_{new}}^{n_{i+1}} + l_{s_{new}}^{n_{i+1}}) \leq d_p^{n_{i+1}}(t_{old})$. Router n_{i+1} becomes *PASSIVE* if it receives a *REPLY* from each of its neighbors, including n_i. Therefore, n_i must be notified about $d_p^{n_{i+1}}(t_{old})$. Therefore,

$$d_{pn_{i+1}}^{n_i}(t) = d_p^{n_{i+1}}(t_{old}) \geq d_p^{n_{i+1}}(t_{new}) \geq fd_p^{n_{i+1}}(t_{new}) \tag{4.11}$$

The feasible distance $fd_p^{n_{i+1}}(t_{new})$ with $t_{new} < t$ cannot increase until router n_{i+1} becomes *PASSIVE* again; therefore,$fd_p^{n_{i+1}}(t_{new}) \geq fd_p^{n_i}(t)$. The result of the lemma follows in this case by substituting this result in Eqs. (4.11) and Eq. (4.10).

Now consider the case that router n_{i+1} is *PASSIVE*. If router n_i processed the last message that router n^{i+1} sent before time t, then $d_{pn_{i+1}}^{n_i}(t) = d_p^{n_{i+1}}(t) \geq fd_p^{n_{i+1}}(t)$ and the lemma is true. Now consider the case that router n_i did not process the last message router n^{i+1} sent before time t. If router n_{i+1} did not reset $fd^{n_{i+1}}$ then $d_p^{n_{i+1}}(t_{old}) \geq fd_p^{n_{i+1}}(t)$. On the other hand, if router n_{i+1} resets $fd^{n_{i+1}}$ then we can conclude that $fd_p^{n_{i+1}}(t_{new}) \geq fd_p^{n_i}(t)$ and $|n_i| > |n_{i+1}|$ using an argument similar to one we used for the *ACTIVE* mode. Hence, the lemma is true for all cases. □

NSC and *SRC* guarantees loop-freedom at every time instant. If we consider the link form router i to its valid next hop with respect to a specific prefix as a directed edge, then the graph containing all this directed links is a directed

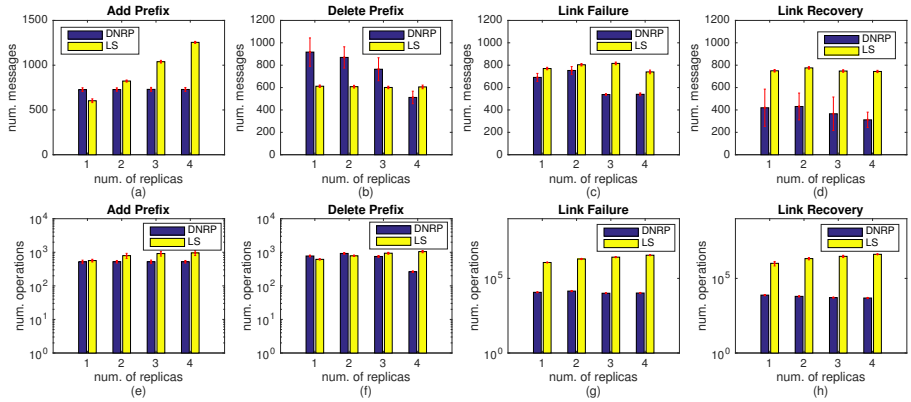

Figure 4.2: Simulation results showing average number of messages and average number of operations vs number of replicas

acyclic graph (DAG) with respect to that specific prefix. The DAG representing the relationship of valid next hops regarding prefix p is denoted by D_p.

Lemma 11. *If routers are involved in a single diffusing computation then D_p is loop-free at every instant.*

Proof. Assume for the sake of contradiction that D_p is loop-free before an arbitrary time t and a loop L_p consisting of h hops is created at time $t_l > t$ when router q updates V_p^q after processing an input event. Assume that $L_p = \{n_1, n_2, \ldots, n_h\}$ is the loop created, where $n_{i+1} \in V_p^{n_i}$ for $1 \leq i \leq h$ and $n_1 \in V_p^{n_h}$. If router n_1 changes its next hop as a result of changing its successor, it must be in *PASSIVE* mode at time t_l because an *ACTIVE* router cannot change its successor or update its next-hop set.

If all routers in L_p are *PASSIVE* at time t_l, either all of them have always been *PASSIVE* at every instant before t_l, or at least one of them was *ACTIVE* for a while and became *PASSIVE* before t_l. If no router was ever *ACTIVE* before time t_l, it follows from Theorem 7 that updating V_p^n cannot create loop. Therefore, for router n_1 to create a loop, at least one of the routers must have been *ACTIVE*

before time t.

If all routers are in *PASSIVE* mode at time t, traversing L_p and applying Theorem 4.10 leads to the erroneous conclusion that either $d_p^{n_1} > d_p^{n_1}$ or $|n_1| > |n_1|$. Therefore updating $V_p^{n_1}$ cannot create a loop if all routers in the L_p are *PASSIVE* at time t.

Assume that only one diffusing computation is taking place at time t_l. Based on Lemma 10 traversing loop L_p leads to the conclusion that either $fd^{n_i} > fd^{n_i}$ or $|n_i| > |n_i|$, which is a contradiction. Therefore, if only a single diffusing computation takes place, then L_p cannot be formed when routers use *SRC* and *NSC* along with difusing computations to select next hops to reach the destination prefix. □

At steady state, the graph containing the successors and connected links between them, must create a tree. The tree containing successors that are *ACTIVE* regarding prefix p and participating in a diffusion computation started form router i at time t are called diffusing tree ($T_{pi}(t)$).

Theorem 12. *DNRP considers each computation individually and in the proper sequence.*

Proof. Assume router i is the only router that has started a diffusing computation up to time t. If router i generates a single diffusion computation, the proof is immediate. Consider the case that router i generates multiple diffusing computations. Any router that is already participating in the current diffusing computation (routers in the T_{pi}, including the router i) cannot send a new *QUERY* until it receives all the replies to the *QUERY* of the current computation and becomes *PASSIVE*. Note that each router processes each event in order. Also, when a router becomes *PASSIVE*, it must send a *REPLY* to its successor, if it has any. Therefore, all the routers in T_{pi} must process each diffusing computation individually and in the proper sequence.

Consider the case that multiple sources of diffusing computations exist regarding prefix p in the network. Assume router i is *ACTIVE* at time t. Then either router i is the originator of the diffusing computation ($o_p^i = 1$ or 2), or received a *QUERY* from its successor ($o_p^i = 3$ or 4). If $o_p^i = 1$ or 3, the router must become *PASSIVE* before it can send another *QUERY*. If the router is the originator of the computation ($o_p^i = 1$ or 2) and receives a *QUERY* form its successor, it holds that *QUERY* and sets o_p^i =4. Therefore, all the routers in the T_{pi} remain in the same computation. Router i can forward the new *QUERY* and become the part of the larger $T_p s$ only after it receives a *REPLY* form each of its neighbors for the current diffusing computation. If router a is *ACTIVE* and receives a *QUERY* from its neighbor $k \neq s_p^a$, then it sends a *REPLY* to its neighbor before creating a diffusing computation, which means that T_{pa} is not part of the *ACTIVE* T_p to which k belongs. Therefore, any two *ACTIVE* T_{pi} and T_{pj} have an empty intersection at any given time, it thus follows from the previous case that the Theorem is true. □

4.3 Performance Analysis

We compare DNRP with a link-state routing protocol given that NLSR [51] is based on link states and is the routing protocol advocated in NDN, one of the leading ICN architectures. We implemented DNRP and an idealized version of NLSR, which we simply call ILS (for ideal link-state), in ns-3 using the needed extensions to support content-centric networking [54]. In the simulations, ILS propagates update messages using the intelligent flooding mechanism. There are two types of Link State Advertisements (LSA): An adjacency LSA carries information regarding a router, its neighbors, and connected links; and a prefix LSA advertises name prefixes, as specified in [51]. For convenience, DNRP sends

HELLO messages between neighbors to detect changes in the sate of nodes and links. However, HELLO's can be omitted in a real implementation and detecting node adjacencies can be done my monitoring packet forwarding success in the data plane.

The AT&T topology [42] is used because it is a realistic topology for simulations that mimic part of the Internet topology. It has 154 nodes and 184 links. A node has 2.4 neighbors on average. In the simulations, the cost of a link is set to one unit, and 30 nodes are selected as anchors that advertise 1200 unique name prefixes. We generated test cases consisting of single link failure and recovery, and a single prefix addition and deletion.

To compare the computation and communication overhead of DNRP and ILS, we measured the number of routing messages transmitted over the network and the number of operations executed by each routing protocol. The number of messages for ILS includes the number of HELLO messages, Adjacency LSAs, and Prefix LSAs. For DNRP, this measurement indicates the total number of all the routing messages transmitted as a result of any changes. The operation count is incremented whenever an event occurs, and statements within a loop are executed.

The simulation results comparing DNRP with ILS are depicted in Figure 4.2. In each graph, the horizontal axes is the average number of anchors per prefix, i.e., the number of anchors that advertise the same prefix to the network. We considered four scenarios: adding a new prefix to the network; deleting one prefix from one of the replicas; a single link failure; and a single link recovery. Figures (4.2a - 4.2d) show the number of messages transmitted in the whole network while Figures (4.2e - 4.2h) show the number of operations each protocol executed after the change. The number of operations in the figure is in logarithmic scale.

ILS advertises prefixes from each of the replicas to the whole network. As

the number of replicas increases, the number of messages increases, because each replica advertises its own Prefix LSA. In DNRP, adding a new prefix affects nodes in small regions and hence the number of messages and operations are fewer than in ILS. Deleting a prefix from one of the replicas results in several diffusing computations in DNRP, which results in more signaling. However, the number of messages decreases as the number of replicas increases, because the event affect fewer routers. In ILS one Prefix LSA will be advertised for each deletion. The computation of prefix deletion is comparable; however, DNRP imposes less computation overhead when the number of replicas reach 4.

DNRP has less communication overhead compared to ILS after a link recovery or a link failure. The need to execute Dijkstra's shortest-path first for each neighbor results in ILS requiring more computations than DNRP. DNRP outperforms NLSR for topology changes as well as adding a new prefix.

4.4 Summary

We introduced the first name-based content routing protocol based on diffusing computations (DNRP) and proved that it provides loop-free multi-path routes to multi-homed name prefixes at every instant. Routers that run DNRP do not require to have knowledge about the network topology, use complete paths to content replicas, know about all the sites storing replicas of named content, or use periodic updates. DNRP has better performance compared to link-state routing protocols when topology changes occur or new prefixes are introduced to the network. A real implementation of DNRP would not require the use of HELLO's used in our simulations, and hence its overhead is far less than routing protocols that rely on LSA's validated by sequence numbers, which require periodic updates to work correctly.

Chapter 5

Ordered Distance Vector Routing Protocol

In this cahpter, we introduce **ODVR** (*Ordered Distance Vector Routing*), the first routing protocol that provides loop-free routes at every instant based solely on distances to destinations maintained by nodes and reference distances included in route requests. A reference distance states the distance to a destination that a node responding to a route request is allowed to have. The motivation for the new approach to on-demand routing introduced in ODVR is twofold: (a) The four types of mechanisms mentioned above that are currently used to avoid or detect routing-table loops in MANETs can incur excessive overhead; and (b) destination-based sequence numbers have not been proven to work correctly when signaling is unreliable and nodes may lose routing state for any reason.

The design rationale for ODVR is based on the following three simple premises: (a) Ordering of nodes with respect to a destination should be based on local rather than global constraints. (b) Establishing ordering among nodes for a given destination must work correctly independently of network conditions, how long individual nodes maintain state for a given destination, or the reliability with which

signaling messages are sent. (c) A signaling message may contain multiple route requests, updates or replies to requests in order to reduce bandwidth consumption.

5.1 ODVR Operation

ODVR operates by establishing a total ordering of nodes with respect to individual destinations using distance values only. Its design centers around the simple premise that failsafe loop-free routing can be attained if nodes: (a) select as next hops to destinations only among those neighbors that are closer to destinations, and (b) accept replies to route requests only if they are created by the intended destinations or nodes that are closer to the destinations than the nodes requesting the routes.

To establish total ordering among nodes for a given destination, each route request and reply to a request states a *reference distance* to the destination. A node must forward a route request with a smaller reference distance than its current distance or reference distance and adopt that new value as its own reference distance. A node trusts only replies stating the reference distance it requires.

To enforce loop-freedom even when signaling is unreliable and nodes may lose routing state, a node that sends a route request regarding a destination for which it has no routing state must use a reference distance of 0 to force the destination to answer the request. To limit signaling overhead in this important case, a relay node with a valid routing-table entry for a destination that receives a route request with a reference distance of 0 indicates in its forwarded request the specific neighbor that should process the route request.

To forward only the first instance of a route request originated by a source and to prevent routing information for a given destination to percolate to the entire network when only a few nodes require routing-state for that destination,

a *pending request table* (PRT) is introduced and each route request states the previous hop traversed by the request. PRTs are used by nodes to process route requests and to add or update routing-table entries. A node adds a routing-table entry for a destination only when the node answers a route request or receives a reply to a pending route request. A node can update an existing routing-table entry when it receives a a reply regarding that entry. A node forwards replies to pending requests only if it has neighbors that forwarded requests stating prior hops for their requests that are different than the node itself.

To avoid deadlocks in the presence of signaling messages being lost, a node forwards all retransmissions of requests received from its neighbors, provided that they state previous hops traversed by the requests other than the node itself. It is up to the origins of request to moderate the rate of request retransmissions.

ODVR allows a signaling message to contain multiple route requests and replies to route requests, and signaling messages can be sent periodically or on an event-driven basis. ODVR provides multiple loop-free routes to a destination, and different approaches can be used for load balancing and congestion control. For simplicity, however, we describe ODVR in the rest of this section assuming that a node considers a neighbor a next hop (called successor) to a destination only if the distance to the destination through that neighbor is the shortest distance available to the node.

5.1.1 Information Exchanged

ODVR relies on the exchange of signaling messages transmitted in broadcast mode among neighboring nodes. A signaling message sent by node i at time t is denoted by $SM^i(t)$ and contains the identifier of the node (i) and one or multiple update entries. Two types of update entries can be contained in a signaling

message, namely a request for a route (REQ) or a reply to a request (REP).

$REQ^i_d[Q,\ d,\ o,\ RD^i_d, D^i_o,\ p^i_d,\ f^i_d]$ denotes a route request sent by node i for destination d originated by node o. In this tuple, Q states that the entry is a request, d is the destination identifier, o is the identifier of the origin of the request, RD^i_d is a reference distance to d used to prevent routing loops, D^i_o is the distance attained by node i to the origin of the request, p^i_d denotes the *request predecessor* (i.e., the neighbor from which node i received the request before forwarding it), and f^i_d states the neighbor(s) of node i that must process the request. If $f^i_d = 0$, the request should be processed by every node that receives the request.

$REP^i_d[R,\ d,\ o,\ RD^i_d,\ D^i_d,\ D^i_o]$ denotes a reply sent by node i for destination d. The value R states that the entry is a reply, d is the destination identifier, o is the origin of the request, RD^i_d is the reference distance of the node that originated the reply, D^i_d is the distance attained by node i to destination d, and D^i_o is the distance attained by node i to the origin of the request o.

5.1.2 Information Stored

Node i maintains four tables to operate, and a maximum lifetime (LT) is allowed for any table entry other than self entries, with the lifetime of each entry decremented from the moment it is created or updated. The set of neighbors of node i is denoted by N^i.

Link-Cost Table (LCT^i): This table is needed if link costs can have different values. If used, the table contains the cost of each link from node i to each known neighbor. The cost of link (i,k) is denoted by l^i_k. By definition, $(i,k) > 0$ for $i \neq k$.

Distance Table (DT^i): The entry for destination d of DT^i is denoted by $DT^i(d)$ and specifies: the identifier of destination d and the distance to d reported

by each known neighbor. The distance reported by node $k \in N^i$ for destination d maintained by node i is denoted by D^i_{kd}. If a known neighbor q has not reported a distance for d, then it is assumed that $D^i_{qd} = \infty$. The vector of distances reported by neighbor k and stored by node i is denoted by DT^i_k.

Routing Table (RT^i): The entry for destination d of RT^i is denoted by $RT^i(d)$ and specifies: the identifier of d, the shortest distance to d (D^i_n), the set of valid successors (S^i_d),and a lifetime for the entry (LR^i_d).

The *valid successor set* S^i_d is defined by Eq. (6.1) as follows:

$$D^i_{min}(d) = Min\{D^i_{dn} + l^i_n \mid n \in N^i\} \tag{5.1}$$

$$S^i_d = \{\ k \in N^i \mid (D^i_{dk} < D^i_d) \wedge (\ D^i_{dk} + l^i_k = D^i_{min}(d)\ )\ \}$$

The node with the smallest identifier in set S^i_d is denoted by s^i_d. By definition, $D^d_d = 0$, $S^d_d = \{d\}$, $r^d_d = 0$, and $LR^d_d = \infty$.

Pending-Request Table (PRT^i): This table keeps track of the route requests waiting for replies. The entry for destination d is denoted by $PRT^i(d)$ and states: the identifier of node d, the smallest reference distance received in a pending request from a neighbor (RD^i_d), a request list RL^i_d, and a lifetime for the entry (LP^i_d). The list RL^i_d contains of one or more tuples, with each tuple consisting of the identifier of the origin of a route request and a list of identifier-distance pairs for all neighbors from which the request was received, which we call *pending-request neighbors*. The tuple in RL^i_d corresponding to origin o is denoted by $RL^i_d(o)$ and its content is $[o, PRN^i_d(o)]$. The pair in $PRN^i_d(o)$ corresponding to neighbor k is denoted by $PRN^i_{dk}(o)$ and equals the pair $[k,\ PD^i_{ok}]$, where PD^i_{ok} denotes the distance to origin o reported by k in its route request regarding destination d.

When node i receives a signaling message from neighbor k, it processes each query and reply in the message independently of the others. We describe the operation of ODVR by focusing on a particular destination d.

5.1.3 Maintaining Routing State On Demand

ODVR maintains routes on demand for each destination d. This involves originating route requests, processing route requests from other nodes, updating routing information, and reacting to topology changes or the expiration of routing-table entries. Each of these activities is carried out while maintaining total ordering of nodes with respect to destination d by means of sufficient conditions for loop freedom based solely on the values of the distance and reference distance maintained by each node for destination d.

Originating Route Requests:

A node with data for destination d originates a request equal to $REQ_d^i[Q, d,\ o = i,\ RD_d^i,\ D_o^i = 0,\ p_d^i = i, f_d^i = 0]$ when the following condition is satisfied. The value $p_d^i = i$ given that there is no request predecessor for the route request, and the value $f_d^i = 0$ requests all the neighbors of node i to process the request.

ORC (*Originating Request Condition*): Node creates REQ_d^i if

$$(\nexists\ PRT^i(d)\ ) \wedge (\ [\ \ \nexists\ RT^i(d)\ ]\ \vee [\ (\exists\ RT^i(d)) \wedge (\ S_d^i = \emptyset\ )\ ]\ ).\ \square$$

ORC states that a node originates a route request if it is not already waiting for a reply regarding destination d and either it has no routing state for d or has routing state but has no valid successor for destination d.

Node i creates $PRT^i(d)$ with the identifier of node d, a value for RD_d^i that depends on the routing state for d, a lifetime $LP_d^i = LT$, and $RL_d^i(i) = [i, PRN_d^i(i)]$.

The list $PRN_d^i(i)$ consists of the tuple $[i, PD_{ii}^i = 0]$.

Node sets $RD_d^i = 0$ if there is no entry $RT^i(d)$ to force destination d to respond to its request. On the other hand, if $RT^i(d)$ exists but $S_d^i = \emptyset$ then node i sets $RD_d^i = D_d^i$ to allow a node with a distance smaller than D_d^i to respond. Node i adds the request REQ_d^i to its signaling message after updating PRT_d^i.

Processing Route Requests:

When node i receives a signaling message from neighbor k containing the request $REQ_d^k = [Q,\ d,\ o,\ RD_d^k, D_o^k,\ p_d^k,\ f_d^k]$ it updates DT^i with $D_{ok}^i = D_o^k$ to remember the distance to the origin of the request reported by k, and determines whether to forward, answer, ignore, or remember the request using one of the following sufficient conditions that prevent routing-table loops by ordering nodes based on their distances to destination d.

FRC (*Forwarding Request Condition*): Node i forwards REQ_d^i to all its neighbors if

$$[\ (\nexists\ PRT^i(d)\ ) \wedge (0 < RD_d^k < D_d^i)\ ]\ \vee$$

$$[\ (\exists\ PRT^i(d)\ ) \wedge (\ [0 < RD_d^k < RD_d^i] \vee [\exists\ PRN_{dk}^i(o)]\ )\ ].\ \square$$

According to FRC node i forwards a route request received from a neighbor in two cases. One case is when node i is not expecting a reply to a prior request for the same destination d and the request states a non-zero reference distance smaller than the current distance from node i to d. The other case is when node i is waiting for a reply and either the reference distance in the new request is smaller than the reference distance currently assumed by node i or the new request is a retransmission of a prior request. A request retransmission is recognized because k is listed as having sent a request regarding d from origin o.

We observe that FRC cannot be satisfied when $p_d^k = i$ because $RD_d^i \leq RD_d^k$

in that case. Hence, node i can only forward requests that are not the result of a prior request from i itself.

If FRC is satisfied, node i must create or update $PRT^i(d)$. Accordingly, node i sets $RD_d^i = Min\{RD_d^i, RD_d^k\}$, sets $LP_d^i = LT$, and updates $RL_d^i(o)$ by adding k to the list of neighbors requiring a reply, which means that $PRN_{dk}^i(o) = PRN_{dk}^i(o) \cup \{[k, PD_{ok}^i = D_o^k]\}$.

Once PRT^i is updated, node i computes $D_o^i = D_o^k + l_k^i$ and adds the request $REQ_d^i\ [Q, d, o, RD_d^i, D_o^i,\ p_d^i = k,\ f_d^i = 0]$ to its signaling message.

RFC (*Reset Forwarding Condition*): Node i forwards REQ_d^i only to $s_d^i \in S_d^i - \{k\}$ if

$$(\nexists\ PRT^i(d)\ ) \wedge (RD_d^k = 0) \wedge (S_d^i - \{k\} \neq \emptyset).\square$$

RFC states that a node with a valid routing-table entry for destination d that receives a "reset request" (a request with a zero reference distance) forwards the request directly to the neighbor other than k that has the smallest identifier among the nodes in its valid successor set.

If RFC is satisfied, node i eliminates k as a next hop to d by updating $RT^i(d)$ with $S_d^i = S_d^i - \{k\}$. Node i then creates $PRT^i(d)$ with the identifier of node d, $RD_d^i = RD_d^k = 0$, $LP_d^i = LT$, and RL_d^i consisting of the tuple $RL_d^i(o) = [o, PRN_d^i(o)]$ with $PRN_d^i(o) = \{[k, PD_{ok}^i = D_o^k]\}$. Node i computes $D_o^i = D_o^k + l_k^i$ and adds the request $REQ_d^i[Q, d, o, RD_d^i = 0,\ D_o^i,\ p_d^i = k, f_d^i = s_d^i]$ to its signaling message.

IRC (*Ignore Request Condition*): Node i ignores REQ_d^k if

$$[\ (f_d^k \neq 0) \wedge (f_d^k \neq i)\ ] \vee [(f_d^k = 0) \wedge (p_d^k = i)\ ].\square$$

IRC states that node i simply ignores a route request explicitly intended for

a different node or a request forwarded by a neighbor as a result of processing a request from node i itself.

SRC (*Store Request Condition*): Node i stores information from REQ_d^k with no other action if

$$(\ f_d^k = 0) \wedge (p_d^k \neq i) \wedge (RD_d^k \geq RD_d^i) \wedge (\exists\ PRN_d^i(o)\ ) \wedge (\nexists\ PRN_{dk}^i(o)\ ).\ \square$$

SRC states that node i can simply add neighbor k as a node that needs a route for destination d if node i has already forwarded a route request for destination d from the same origin stated by k and k is not already listed as a neighbor from which a request from origin o and with a previous hop other than i has been received.

If SRC is satisfied node i updates PRT_d^i. It sets $LP_d^i = LT$ and updates $RL_d^i(o)$ with $PRN_d^i(o) = PRN_d^i(o) \cup \{[k, PD_{ok}^i = D_o^k]\}$.

RRC (*Reply Request Condition*): Node i sends a reply to REQ_d^k if

$$[\ d = i\ ] \vee [\ (\ RD_d^k > D_d^i\ ) \wedge (\ \ \nexists\ PRT^i(d)\ )\ ] \vee [\ (f_d^k = i) \wedge (S_n^i = \emptyset)\ ].\ \square$$

According to RRC node i can answer a request in three cases. Node i can reply independently of the reference distance stated in the request if it is the intended destination. Node i can also reply if it is not waiting for a reply for a route to d and it is closer to d than the reference distance stated in the request. Node i also replies if the request is explicitly directed to itself (i.e., $f_d^k = i$) and node i does not have a valid successor to destination d (i.e., $S_d^i = \emptyset$), in which case node i informs k that $D_d^i = \infty$.

If RRC is satisfied node i can update its routing table regarding origin o. To do so node i uses Eq. (6.1) to compute $D_{min}^i(o)$ and S_o^i and updates $D_o^i = D_{min}^i(o)$.

Node i then includes the reply $REP_d^i\ [R, d, o, RD_d^i = D_d^i,\ D_d^i,\ D_o^i]$ in its signaling message.

Processing Replies:

Nodes process replies opportunistically, given that all signaling messages are sent in broadcast mode. Nodes are allowed to add new routing-table entries only after receiving replies that answer requests they have originated or forwarded. Nodes can update existing routing-table entries after receiving subsequent replies.

When node i receives the reply $REP_d^k\ [R,\ d,\ o,\ RD_d^k,\ D_d^k,\ D_o^k]$ from neighbor k, node i first updates DT^i by setting $D_{dk}^i = D_d^k$ and $D_{ok}^i = D_o^k$. Node i then uses the following condition to determine if it can use the reply from k to answer its own request or improve its distances to d or o if they are already instantiated in RT^i.

ARC (*Accept Reply Condition*): Node i accepts REP_d^k if

$$[(\nexists\ PRT^i(d)\ ) \wedge (\ D_d^k < D_d^i\ )] \vee$$

$$[(\exists\ PRT^i(d)\ ) \wedge (\ RD_d^k < RD_d^i\ )].\ \square$$

ARC states that node i can consider using neighbor k as a successor for destination d if either node i is not waiting for replies to a route request and node k is closer to d than node i is, or the reference distance reported by k is smaller than the reference distance that node i must satisfy in its own route request.

Node i does nothing else if ARC is not satisfied, because loop freedom cannot be ensured by neighbor k. There are two cases to consider when ARC is satisfied.

Case 1: ARC is satisfied and $PRT^i(d)$ does not exist.

Node i does nothing else in this case if $RT^i(d)$ and $RT^i(o)$ do not exist. Alternatively, node i uses Eq. (6.1) to compute $D_{min}^i(d)$ and S_d^i if $RT^i(d)$ exists and to

compute $D^i_{min}(o)$ and S^i_o if $RT^i(o)$ exists. If $D^i_{min}(d) \neq D^i_d$ or $D^i_{min}(o) \neq D^i_o$ node i updates $D^i_d = D^i_{min}(d)$ and $D^i_o = D^i_{min}(o)$ as needed. Node i adds the gratuitous reply REP^i_d $[R,\ d, i, RD^i_d = D^i_d,\ D^i_d, D^i_i]$ to its signaling message if $RT^i(d)$ is updated, and adds the gratuitous reply REP^i_o $[R,\ o, i, RD^i_o = D^i_o,\ D^i_o, D^i_i]$ to its signaling message if $RT^i(o)$ is updated.

Case 2: ARC is satisfied and $PRT^i(d)$ exists.

Node i uses REP^k_d as an answer to its own request in this case. If $D^k_d < \infty$, node i updates RT^i with $D^i_d = D^i_{jk} + l^i_k$ and $S^i_d = \{k\}$, else $D^k_d = \infty$ and hence node i updates RT^i with $D^i_d = \infty$ and $S^i_d = \emptyset$.

Node i updates or creates $RT^i(o)$ by computing $D^i_{min}(o)$ and S^i_o using Eq. (6.1) and sets $D^i_o = D^i_{min}(o)$. Furthermore, for each origin p for which there is an entry $RL^i_d(p)$ in in PRT^i, node i updates or creates $RT^i(p)$ by computing $D^i_{min}(p)$ and S^i_p using Eq. (6.1) and sets $D^i_p = D^i_{min}(p)$.

Node i adds the reply $REP^i_d[R,\ d, o, RD^i_d = RD^k_d,\ D^i_d, D^i_o]$ to its signaling message if either $PRN^i_{dq}(o)$ exists for $q \in N^i$ or node i updated D^i_d or D^i_o. Node i then deletes $PRT^i(d)$.

Handling Link Changes and Unreliable Transmissions:

To expedite neighbor discovery and make ODVR resilient in the presence of unreliable transmissions, each node transmits a signaling message periodically containing a *gratuitous route reply* $REP^i_i[R, i, o = i,\ RD^i_o,\ D^i_i, D^i_o]$ for itself with $RD^i_i = D^i_i = D^i_o = 0$.

To address the fact that signaling messages may be lost, node i also adds a gratuitous route reply $REP^i_d[R, d, o,\ RD^i_d,\ D^i_d, D^i_o]$ with $o = i$, $RD^i_d = D^i_d$ and $D^i_o = 0$ for each destination d in RT^i. To reduce the size of signaling messages, node i can simply include a hash value of the content of RT^i if it makes no changes to RT^i from the time it sent its previous signaling message.

Link additions are detected through the successful reception of signaling messages. Link failures are detected either through the link layer or as a result of a node not receiving any data packets or signaling messages from a neighbor for a period of time corresponding to the time needed to transmit two or three consecutive signaling messages periodically (e.g., gratuitous replies).

If node i detects a change in the cost of link (i,k), it computes $D^i_{min}(d)$ and S^i_d for each destination d for which an entry $RT^I(d)$ exists using Eq. (6.1). In addition, for each destination d for which there is an entry $RT^I(d)$ such that $S^i_d \neq \emptyset$ and $D^i_{min}(d) \neq D^i_d$ node i updates $D^i_d = D^i_{min}(d)$ and adds a gratuitous reply $REP^i_d[R, d, o = i, RD^i_d = D^i_d,\ D^i_d, D^i_o = 0]$ to its signaling message.

If node i detects that the link (i,k) to neighbor k has failed, it sets $l^i_k = \infty$ in LCT^i and treats the link failure as a link-cot change. The entry D^i_{kd} for each destination d is deleted after its lifetime expires.

Handling Soft-State and Node Reboots:

ODVR is a soft-state protocol. Each entry in DT^i, RT^i, and PRT^i has a finite lifetime to allow router i to delete entries that become obsolete as a result of topology changes (e.g., the network is partitioned or a node fails) or lack of data traffic.

Noe i renews the lifetime of $RT^i(d)$ with every data packet it processes for destination d and deletes the entry if no data traffic is received for d during the lifetime of the entry. If entry $RT^i(d)$ is deleted, node i also deletes entry $DT^i(d)$. If LP^i_d reaches 0 the entry $PRT^i(d)$ expires and node i deletes the entry and sets $S^i_d = \emptyset$ if $RT^i(d)$ exists. If node i is a source of data for destination d, node i can schedule the transmission of a new route request for d according to ORC.

If node i is asked to forward a data packet to destination d and $RT^i(d)$ does not exist or $S^i_d = \emptyset$, node i adds the gratuitous route reply $REP^i_d[R, d, o,\ RD^i_d,\ D^i_d, D^i_o]$

with $o = i$, $RD_d^i = D_d^i = \infty$ and $D_o^i = 0$ to its signaling message.

If node i initializes or reboots, it only has routing state for itself. It is assumed that $S_d^i = \emptyset$ and $D_d^i = \infty$ for any destination for which node i has no routing state.

5.2 Examples of ODVR Operation

Consider the first example discussed in Section 3 and illustrated with Fig. 2.7. The importance of using local constraints is that nodes can adapt to topology changes while incurring far less signaling overhead. If ODVR is used in the same example, nodes a and b satisfy ADC at node s and hence node s simply updates $S_d^s = \{a, b\}$ without incurring any signaling overhead.

The second example we discussed in Section 3 illustrates the importance of the total ordering of distances to destinations used in ODVR. Fig. 5.1 shows the same example of Fig. 2.8 when ODVR is used. The routing state at each node is its distance and successor set to destination d, with the distance shown next to each node.

According to the operation of ODVR the loss of its routing state for d (and any other destination) at time t_2 dictates that the route request it originates at time t_3 must use a zero reference distance as shown in Fig. 5.1(c). As we assumed for the case of AODV, the request from c is not received by b but is received by a.

Fig. 5.1(d) shows that, when node a processes the request from c stating $RD_d^c = 0$, it applies RFC and forwards the request explicitly to node b because it is its current best next hop to destination d. By the same token, as Fig. 5.1(e) shows, node b forwards the request explicitly to node c, which is its best next hop to destination d.

As Fig. 5.1(f) shows, at time t_6 node c applies RRC when it process the

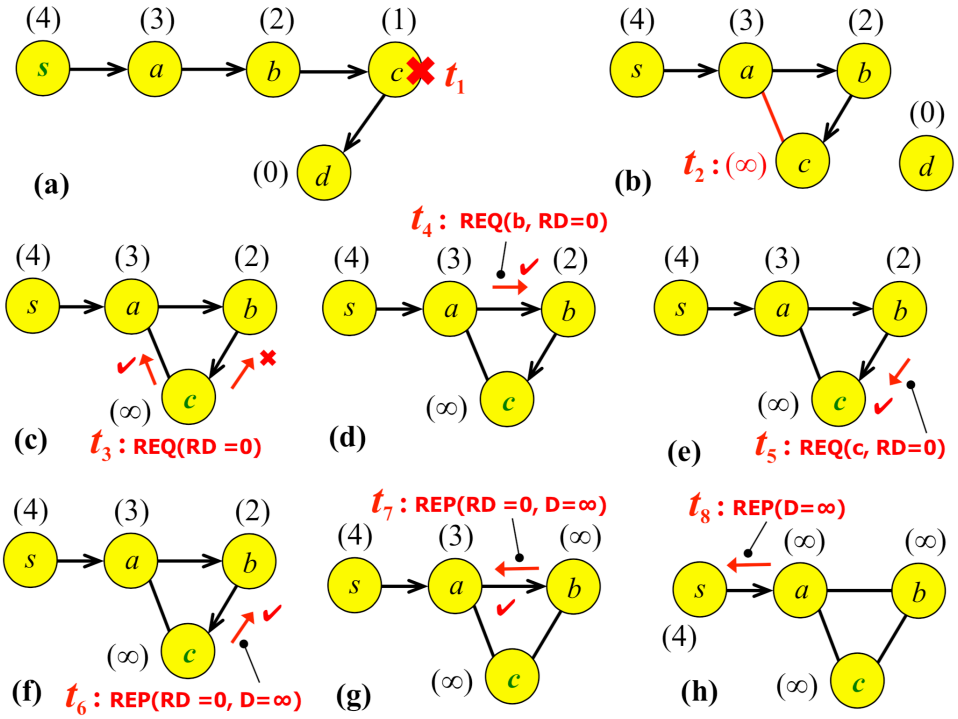

Figure 5.1: No routing-table loop occurs with ODVR even when nodes lose routing state and messages are unreliable

request from node b and states $D_d^c = \infty$. When node b processes that reply, ARC is satisfied and as Fig. 5.1(g) shows node b must send a reply to node a's query at time t_7 with $D_d^b = \infty$. By the same token, as Fig. 5.1(h) shows node a must send a gratuitous REP stating $D_d^a = \infty$ to all its neighbors. Different outcomes from the one in Fig. 5.1 would result if more messages were lost or different topology changes took place. However, the end result with any outcome is that no routing-table loops can be created given that nodes without routing state for a destination implicitly assume an infinite distance to the destination and are required to ask the destination itself to answer their route requests.

third example of the operation of ODVR illustrates the performance improvements derived from the aggregation of route requests. Fig. 5.2 shows a MANET in which ODVR is used and four different sources request routes for the same

destination d. Independently of the timing of route requests, each node forwards only a single route request for d , because either nodes attain valid routes to destination d or aggregate route requests.

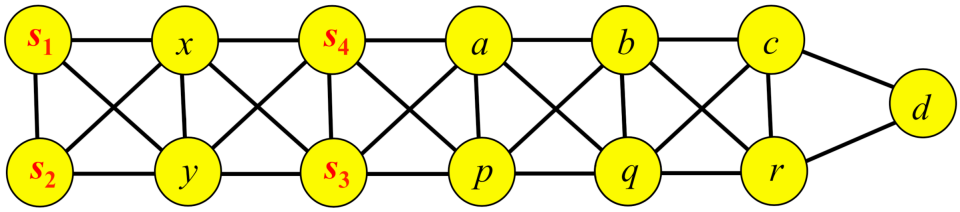

Figure 5.2: Nodes receive fewer requests when request aggregation is used

Even if routes are not established by the time subsequent requests for d are received, a node receives a number of signaling messages with route requests for d equal to its number of neighbors. By contrast, with AODV other on-demand routing protocols, unless routing state is instantiated at a node each node forwards a number of requests equal to the number of sources of requests.

5.3 Correctness of ODVR

The following theorems prove that nodes using ODVR maintain loop-free routing tables at every instant and any node with data for a destination obtains a route to the destinations in a finite time, provided that there is a physical path available. We assume the same notation introduced in Section 5. The values of D_d^i, D_{dn}^i and S_d^i at a specific time t are denoted by $D_d^i(t)$, $D_{dn}^i(t)$ and $S_d^i(t)$, respectively.

Theorem 13. *No routing-table loops can exist for destination d if $D_d^i = \infty$ or $D_{ds_d^i}^i < D_d^i < \infty$ for any network node i other than d at every instant of time.*

Proof. Consider a set of nodes $\{v_1, v_2, ..., v_h, v_1\}$ different than destination d. Assume that this set of nodes create a routing-table loop L of h hops by setting

$s_d^{v_i} = v_{i+1}$ for $1 \leq i \leq h-1$ and $s_d^{v_h} = v_1$. For the sake of contradiction, further assume that each node in loop L satisfies the ordering constraint $D_d^{v_i} = \infty$ or $D_{dv_{i+1}}^{v_i} < D_d^{v_i} < \infty$ for $1 \leq i \leq h-1$ and $D_d^{v_h} = \infty$ or $D_{dv_1}^{v_h} < D_d^{v_h} < \infty$.

Loop L cannot exist if $D_d^{v_i} = \infty$ for any i with $1 \leq i \leq h-1$ because then v_i has no next hop to d. It thus follows that $D_{dv_{i+1}}^{v_i} < D_d^{v_i} < \infty$ for $1 \leq i \leq h-1$ and $D_{dv_1}^{v_h} < D_d^{v_h} < \infty$. This, however, is a contradiction, because it implies that $D_d^{v_i} > D_d^{v_i}$ for $1 \leq i \leq h-1$ and $D_d^{v_h} > D_d^{v_h}$. Therefore, the theorem is true. □

Theorem 21 states that *any* routing protocol that satisfies the stated ordering constraint at every instant ensures loop-free routing. The proof of the following theorem demonstrates that ODVR enforces this ordering constraint at every instant.

Theorem 14. *ODVR ensures that $D_d^i = \infty$ or $D_{ds_d^i}^i < D_d^i < \infty$ for any network node i other than destination d at every instant.*

Proof. The ordering constraint $\mathcal{O}$ of the theorem can be expressed as:

$$\begin{aligned} \mathcal{O} &\equiv [\ D_d^i = \infty\ ] \vee [\ D_{ds_d^i}^i < D_d^i < \infty\ ] \\ &\equiv [\ D_d^i = \infty\ ] \vee [\ (D_{ds_d^i}^i < D_d^i) \wedge (D_d^i < \infty)\ ] \end{aligned} \tag{5.2}$$

Given that no distance can be larger than infinity, we have:

$$\begin{aligned} \mathcal{O} &\equiv [\ D_d^i = \infty\ ] \vee [\ (D_{ds_d^i}^i < D_d^i) \wedge \neg(D_d^i = \infty)\ ] \\ &\equiv ([D_d^i = \infty\ ] \vee [\ D_{ds_d^i}^i < D_d^i]) \wedge ([D_d^i = \infty] \vee [\neg(D_d^i = \infty)]) \\ &\equiv [\ D_d^i = \infty\ ] \vee [\ D_{ds_d^i}^i < D_d^i\ ] \end{aligned} \tag{5.3}$$

Assume for the sake of contradiction that every node executes ODVR correctly and $\mathcal{O}$ is not satisfied at some instant of time t. Then there must be at least one node i that executes ODVR correctly and such that $\neg\mathcal{O}$ is true at time t. From

Eq. (5.3) and DeMorgan's law our assumption means that node i updates its routing state for destination d at time t such that

$$\neg\mathcal{O} \equiv [\ D_d^i(t) < \infty\ ] \wedge [\ D_{ds_d^i}^i(t) \geq D_d^i(t)\ ] \tag{5.4}$$

From the operation of ODVR, if node i has no routing state for destination d at time t then $D_d^i(t) = \infty$. Hence, it must be true that node i has routing state for d at time t so that $D_d^i(t) < \infty$ and $\neg\mathcal{O}$ in Eq. (5.4) can be satisfied.

Let $D_d^i(t) < \infty$. According to the correct operation of ODVR, node i can select a successor for destination d at time t only if ARC is satisfied when a reply is processed. However, node i must use Eq. (6.1) to select a valid successor to destination d in this case, and this requires that $D_{ds_d^i}^i(t) < D_d^i(t)$. This implies that the correct operation of ODVR is a contradiction to $\neg\mathcal{O}$ stated in Eq. (5.4). Therefore, the theorem is true. □

Theorem 15. *ODVR provides loop-free routing at every instant.*

Proof. The proof follows from Theorems 21 and 14. □

A routing protocol can be free of routing-table loops without providing routes to destinations even if physical paths exist. The proof of the following theorem shows that ODVR provides valid routes to destinations within a finite time if physical connectivity exists. To address the ability of ODVR to provide sources with loop-free routes to their intended destinations within a finite time, we assume that no messages are lost and no topology changes or failures occur after time t_0. From that instant messages are sent reliably, the topology is static and no nodes fail or restart.

Theorem 16. *A finite time after all topology changes and message errors subside in a connected network in which ODVR is used, then every source node with data*

for a destination d must establish a valid route to d within a finite time.

Proof. Assume that ODVR is used in a network in which all topology changes and message errors subside by time t_0 and consider a connected component C of the network that includes destination d and a node i with data for d after time t_0.

Because no topology changes, link-cost changes, or message errors occur after time t_0, every node in C knows all its active neighbors and the finite cost of each adjacent link to a neighbor by some finite time t_1 with $t_0 \leq t_1 < \infty$. Assume for the sake of contradiction that at a node i has data for destination d but cannot establish a valid route to it at any arbitrary time after time t_1.

If node i does not have routing state for d at time $t_2 \geq t_1$, it must issue a route request with $RD_d^i = 0$ and $f_d^i = 0$ that only d can answer. According to FRC and RFC, the request from i must traverse at least one simple path P_{id} to d because each node receiving the request must either forward the request to all its neighbors or to a next hop along a simple path towards d given that ODVR can only establish loop-free routes (Theorem 22). Therefore, destination d must receive the request from i and sends a reply that traverses the loop-free reverse path P_{di} back to node i within a finite time because all messages are sent correctly after time t_0.

If $RT_d^i(t_2)$ exists and $S_d^i(t_2) = \emptyset$, node i must issue a request with $RD_d^i = D_d^i(t_2)$ and $f_d^i = 0$ that *any* node $p \in C$ other than node i with $D_d^p \leq RD_d^i = D_d^i(t_2)$ can answer. This must occur because $D_d^d = 0 < RD_d^i$.

Given that ODVR can only establish loop-free routes (Theorem 22), if entry $RT_d^i(t_2)$ exists and $S_d^i(t_2) \neq \emptyset$ but no neighbor of node i in $S_d^i(t_2)$ is part of a valid route, any implicit route resulting from a neighbor in $S_d^i(t_2)$ must terminate at a node with an empty successor set for d. According to the operation of ODVR,

each node $k \in C$ sends a gratuitous reply periodically for destination d if $RT^k(d)$ exists, sends a gratuitous reply if it is asked to forward a data packet for d and $RT^k(d)$ does not exist or $Sk_d = \emptyset$, and deletes $RT^k(d)$ if its lifetime expires after not receiving data traffic for d. Accordingly, within a finite time after time t_2 node i must have deleted $RT^i(d)$ before it becomes a source of data for d again, or it sends data packets towards d and receives gratuitous replies for d from all its neighbors in $S_d^i(t_2)$ stating infinite distances to d. In either case, node i must issue a route request for d after time t_2 stating either $RD_d^i = D_d^i(t_2)$ or $RD_d^i = \infty$, and it follows from the argument above that node i must receive a reply. □

5.3.1 Protocol Complexity

To compare the complexity of ODVR with other MANET routing approaches, we focus on their time and communication complexities. We assume that each transmission is reliable and reaches all neighbors of a node. The communication complexity (CC) of a routing protocol is the number of messages that must be transmitted successfully for each node to have correct routing information about all the destinations of interest after a single link failure. The time complexity (TC) of a routing protocol is the maximum time needed for all nodes to have correct routing information for all destinations of interest after a single link failure.

We assume that all nodes have a stable topology state and a stable routing state for a destination of interest. For the case of AODV this means that all nodes with active routes to the destination of interest hold the same sequence-number value. Furthermore, we only consider the complexity of the protocols after a single link failure that does not partition the network. This allows us to discuss the worst-case signaling for on-demand routing and the inherent signaling overhead of the approaches without having to worry about the possibility of routing-table

loops in AODV due to the loss of routing state. The number of nodes in the network is N and the network diameter after the link failure is h. A node n is the head of the failed link (n, d) and is also a source of data for destination d whose route is broken because of the failed link. In addition, there may be $S - 1 > 0$ other sources of data for the same destination of interest whose routes are broken because of the link failure. For simplicity, we assume S to be smaller than the maximum degree of a network node. Fig. 5.3 illustrates the scenario with $S = 3$. All the physical paths indicated in the world necessarily have lengths smaller than or equal to the network diameter after the link failure.

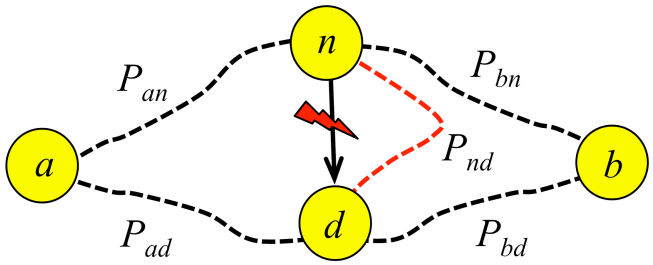

Figure 5.3: Nodes n, a and b are sources of data for d

OLSR: A link-state update (LSU) stating the loss of a link must be sent all the nodes in the network. Each LSU is disseminated using intelligent flooding in which the LSU takes the fastest path from the origin of the LSU to every other node in the network and each node transmits the LSU regarding the failed link once. Accordingly, the time and communication complexities of OLSR are:

$$TC_{OLSR} = O(h); \quad CC_{OLSR} = O(N) \tag{5.5}$$

AODV: Because node n is assumed to have data for destination d, the node is required to issue a route request with a higher sequence number. The request by node n cannot be satisfied by any node other than the destination of interest

and hence the request is disseminated to each network node, and only the nodes along a path from the destination of interest and source n need to forward the reply and such a path cannot be larger than h hops. The path between node n and the farthest source from n in the group of other sources cannot be longer than the network diameter h. Therefore, in the worst case, the route request from n may traverse no more than h hops to reach all other sources and each request generated by the other $S-1$ sources is received by at most $N-1$ other nodes, traverses at most h hops to the destination of interest and the reply traverses no more than h hops back to the source requesting the route. Accordingly, the time and communication complexities of AODV when $S > 1$ sources are affected by a link failure are:

$$TC_{AODV} = O(3h);\ CC_{AODV} = O(S(N+h)) \tag{5.6}$$

ODVR: In the worst case of the scenario we assume, none of the sources with routes affected by the link failure find a neighbor that satisfies Eq. (6.1). The time and communication complexities for ODVR are the same as in AODV, because each of the other $S-1$ sources can be up to h hops away from n and their own distances to d may be up to h hops. We therefore have:

$$TC_{ODVR} = O(3h);\ CC_{ODVR} = O(S(N+d)) \tag{5.7}$$

As Eqs. (5.5) and (5.7) show, ODVR and AODV may incur more signaling overhead than OLSR in the worst-case scenario we have presented. However, the upper bound for the complexity of ODVR is very loose, because a link failure may incur time and communication complexities equal to $\Omega(1)$. That lower bound on the complexity of ODVR would occur when n has neighbors with distances to d

that are smaller than or equal to RD_d^n or paths of length $\Omega(1)$ to d in which replies can be sent that satisfy RD_d^n, which causes the request and replies to traverse only a few hops, rather than being disseminated throughout the entire network as it must occur in AODV because only the destination can increase its own sequence number.

5.4 Performance Comparison

The performance of ODVR is compared against two other MANET routing protocols: AODV and OLSR. We implemented the ODVR protocol in the network simulator (ns3.24) and used the ns3 implementations of AODV and OLSR without modifications.

OLSR uses HELLO messages to discover and check neighbor connectivity and Topology Control (TC) messages to disseminate link-state information throughout the network. To reduce signaling overhead, OLSR takes advantage of connected dominating sets. Some nodes are elected as multipoint relays (MPRs) and only MPRs forward TC messages, and only link-state information needed to connect MPRs is advertised in the network.

In this study we considered three metrics to analyze the performance of routing protocols:Data Packet Delivery Ratio (DPDR), Signaling Overhead (overhead for short), and End-to-End Delay. DPDR indicates the number of packets received by destination routers divided by number of packets sent by the source routers. The Overhead is the total number of routing message (in bytes) for the duration of simulation. The signaling overhead in AODV includes its five types of packets: requests, replies error messages, ACKs to replies, and HELLO messages. In OLSR, the signaling overhead includes, Topology Control (TC) and HELLO messages. The signaling overhead of ODVR includes all the REQUEST and RE-

PLY messages it broadcasts. Average delay is the duration of transmitting a data packet till it reaches the destination, including all delays enforced by buffering during route discovery phase, queuing at the output queue, propagation delay and transfer times.

5.4.1 Simulation Results

The simulated network consisted of 50 nodes spread uniformly and randomly in a $300 \times 1500m$ area at the beginning of the simulation. all nodes started the experiment at a random location within the simulation area and moved as defined by the random waypoint model [87]. In this model, each node selected a random destination within the working area and moved linearly to that location at a predefined speed. After reaching its destination, it paused for a specified period (pause time) and then selected a new random location and continue the process again.

Topology Area	300 x 1500 m
Number of Nodes	50
Transmission Range	250 m
Simulation time	1000 sec
Bandwidth	2 Mbps
Hello Interval	2 sec
TC Interval (OLSR)	5 sec
Update Interval (ODVR)	2 sec
Route TimeOut	3 sec
Queue Length	64
REQ Rate	10 req/sec
Max Retries	2

Figure 5.4: Simulation parameters

The scenarios include 25 data flows from 25 different source routers to different destinations that are chosen randoomly. Traffic sources are on-off applications with on and off time of 1 second, which generate packets of size 512 bytes and

Figure 5.5: Performance comparison as a function of Node Speed.

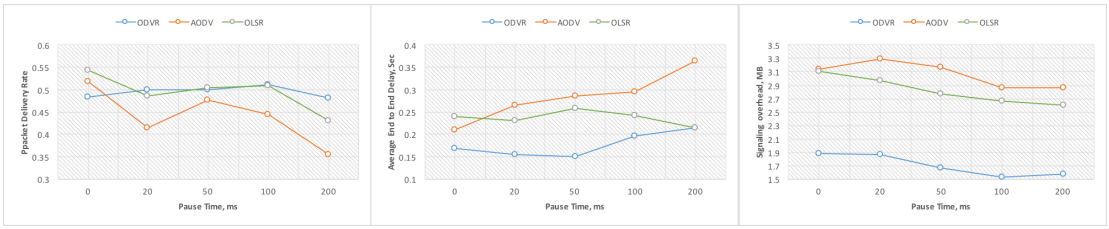

Figure 5.6: Performance comparison as a function of Pause Time.

rate of 15 packets per second. Figure 5.4 shows the summary of simulation-environment settings for the routing protocols.

We have simulated three different scenarios to study the behavior of routing protocols with respect to Mobility and number of Data Flows. These scenarios were chosen to stress all three protocols, rather than to attain good performance for either on-demand or proactive routing.

5.4.2 Effect of Mobility

We study the effect of mobility on the performance of routing algorithms by changing the pause time and speed of nodes. The longer pause time means the network is more steady and less topology changes. Speed of nodes is another parameter in mobility, and higher speed means more change in the network.

The results of measurements as a function of node speed are shown in figure 6.1. The packet delivery rate decreases as the router speed increases due to more topology changes. Routes break due to fast movement of nodes and packets drop

till the routing protocol finds a new route.

The OLSR will detect the link failure after failure in reception of HELLO messages and sends TC messages to inform all routers of the topology so that new routes can be established. Given that TC messages contains all changes take place between periodic updates, the signaling overhead increases to address topology changes in the network.

Link failures in AODV and ODVR are detected by the absence of a number of consecutive Hello messages, and a route discovery process is performed to establish new route between source and destinations. Because of the delays incurred in detecting link failures and in establishing new routes after that, as router speed increases more and more data packets traversing failed routes end up being dropped.

The ODVR has less overhead compare to AODV and OLSR, because it only sends changes in active routes, has smaller data to send (compare to AODV that sends sequence number of both source and destination), and route requests satisfies by middle nodes rather than destination itself. In OLSR, TC messages must be disseminated by MPRs throughout the network and in AODV, each RREQ is flooded throughout the network. By contrast, a Request in ODVR will be propagated in a small part of the network because the only criteria is the node satisfies the request be closer to the destination than the node originated the request. This characteristic of ODVR also causes shorter search time and results in smaller end to end delay compare to other routing algorithms.

In next scenario, we studied the affect of pause time on performance of routing protocol. Figure 6.2 shows the packet deliver, delay, and overhead as a function of node pause time. Pause times vary from 0 seconds (high mobility) to 200 seconds (low mobility), and speed of individual nodes is chosen randomly from 0 to 20

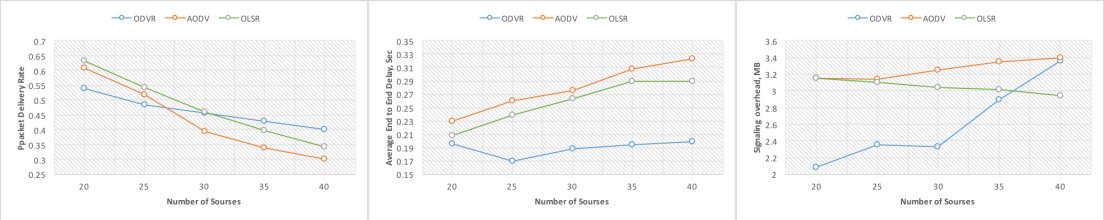

Figure 5.7: Performance comparison as a function Number of Sources.

m/s. The other parameters remain the same.

The packet delivery ratio of OLSR and ODVR are very close and are higher than AODV protocol. ODVR imposes shorter delay, compare to other two algorithms which means it converges faster and forwards packets faster than the other two protocols. Similar to the variable speed scenario, ODVR introduces less overhead compare to AODV and OLSR due to smaller messages and smaller number of nodes that forward the REQUESTs.

5.4.3 Effect of Number of Flows

In this scenario, the affect of data traffic and number of connections on performance is studies and results are reposter in figure 6.3. Sources and destinations are selected randomly and number source/destination pair varies between 20 (average data flow) to 40 (high data flow).

The delivery rate decreases and the average end-to-end delays increase for all three protools as the number of sources increases. More traffic will cause more more congestion in the network. Also, a link failure will affect more transmissions when the number of data flows increases. The ODVR has better performance when number of connections are more than 30 and offers better delay, compare to AODV and OLSR. The signaling in OLSR is independent of the number of connections in the network, hence the total signaling overhead remains the same. AODV and ODVR signaling increases as the number of connections increase,

because they have to recover more path due to topology change in the netwrok.

5.5 Summary

We introduced ODVR, the first on-demand routing protocol that eliminates routing-table loops by ordering nodes based solely on the distances they maintain and advertise to destinations, as well as reference distances they state in route requests. Routing state is established on demand by means of route requests stating the reference distance that a responder must satisfy to be allowed to reply. Intended destinations or nodes that meet the reference distances respond with replies, and nodes also send distance updates regarding the sources of requests and active destinations.

We proved that ODVR maintains loop-free routing tables at every instant and can converge to valid routes within a finite time. The correct behavior of ODVR does not depend on how long routing state is maintained for a destination or the reliability of signaling messages. We compared the performance of ODVR withAODV and OLSR, which are well-known examples of on-demand and proactive routing, taking into account the effect of mobility, number of flows, and network size on the performance of the protocols. The simulation results show that ODVR attains much better performance than the two other protocols.

Chapter 6

Adaptive Approach to Routing in Ad-Hoc Networks

An efficient routing protocol for MANETs, is the one that takes advantage of the strengths of both reactive and proactive routing algorithms and to adapts its behavior at the appropriate time and for the appropriate scope of the network. This motivates the study of hybrid MANET routing protocols. The main contribution of this work is to propose a hybrid routing protocol.

This chapter introduces **ADRP** (*Adaptive Distance-Vector Routing Protocol*), a hybrid routing algorithm for mobile ad-hoc networks that calculates loop-free routes at every node based on the distances to destination and reference distances maintained by nodes.

We describe the ADRP in details and illustrate the signaling, operation, and decision making of ADRP. We prove that ADRP is loop-free at every instant independently of the state of the topology, the amount of time a node stores routing state, or the reliability or timing with which signaling messages are exchanged among nodes.

6.1 ADRP Operation

ADRP is a node-centric hybrid routing that combines on-demand and proactive routing and calculates shortest path to destination solely based on the distance information each node maintains. ADRP adapts between reactive and proactive routing by varying the amount of routing information shared. It does so, by defining a proactive node, a hot-spot node called pop-node. Paths between pop-nodes and all other nodes are maintained proactively. The protocol uses the reactive approach to establish routes to normal-nodes.

ADRP amortizes the cost of maintaining routes to a popular destination among all the sources that communicate with that destination node. Hence, it is well suited for applications that exhibit spatial locality in their network communications. In IoT applications for example, selected nodes performing data aggregation act as hot destinations for data generated by nearby nodes, or some gateway nodes are connected to Internet and responsible for data flow from nodes in that network to a destination in the cloud. In these type of applications, the popularity of nodes follows a Zipf-distribution, making some nodes more popular than others.

ADRP maintaining total ordering of nodes with respect to a destination by means of sufficient conditions for loop freedom based solely on the values of the distance maintained by each node for each destination. Similar to *ODVR* (Section 5) , *ADRP* finds failsafe loop-free routes to destination where each node in the network (a) selects a neighbor as next hop if and only if it is closer to destination, (b) accept reply to route requests only if they are created by the destination itself or and intermediate node that is closer to the destination than the node requesting the routes. On top of these two assumptions, ADRP implies that each node, (c) process only those *gratuitous replies* that are generated by the destination itself or a node that is closer to the destination than the node itself.

To establish total ordering among nodes, request and reply messages state a *reference distance* to the destination. A node processes and forwards *gratuitous reply* messages with a smaller reference distance than its current distance and updates its routing state based on the result of processing those messages. A node trusts only those messages stating the distance smaller than its current reference distance and trust replies stating the reference distance it requires.

ADRP treats routes to pop-nodes different than regular nodes. If a node loses the routing state regarding a pop-node, it will try to find a valid route to that destination even it does not have any traffic for it. In this case if the node can not find a route, will wait if it receives a reply originated by a node closer to destination. If it does not receive such a message it will generate request. To limit signaling overhead in *ADRP* , any node with a valid routing-table entry for that destination will generate gratuitous reply whenever it detects a change in distance to that destination.

To distinguish between receiving a retransmission of a reply message by a given source from receiving multiple copies of the same message we suggest to forward only the first instance of the message originated by any source or those that cases a change in routing state. We introduce the *Reply history table* (**RHT**). *RHT* is used by a node to process reply message and to add or update routing-table entries. A node adds a routing-table entry for a destination only when the node receives a reply to a pending route request regarding either a normal- or pop-node or receives a gratuitous reply regarding a pop-destination. A node can update an existing routing-table entry when it receives a valid a reply regarding that entry.

To avoid deadlocks in the presence of signaling messages being lost, a pop-node transmits gratuitous reply messages periodically. Note that, Ad-hoc networks are dynamic and each node will send an update if it detects a change in routing table

entry for a pop-node and has a valid rout to destination or originate a route request if it can not find a valid route to any active destination. Therefore the time interval between periodic messages can be large to reduce the signaling.

6.1.1 Information Exchanged

All signalings in *ADRP* are in broadcast mode among neighboring nodes. A signaling message sent by node i at time t is denoted by $SM^i(t)$ and contains the identifier of the node (i) and one or multiple signaling entries. A signaling entry in a message transmitted by node i regarding destination d consists of a tuple (*type*, *payload*), where the type and payload of the entry varies depending on the type of entry. Similar to *ODVR*, ADRP uses (*REQ*) and the reply (*REP*) messages:

$REQ^i_d[Q,\ d,\ o,\ RD^i_d o, D^i_o,\ p^i_d,\ f^i_d]$ denotes a route request sent by node i for destination d originated by node o. In this tuple, Q states that the entry is a request, d is the destination identifier, o is the identifier of the origin of the request, RD^i_d is a reference distance to d used to prevent routing loops, D^i_o is the distance attained by node i to the origin of the request, p^i_d denotes the *request predecessor* (i.e., the neighbor from which node i received the request before forwarding it), and f^i_d states the neighbor(s) of node i that must process the request. If $f^i_d = 0$, the request should be processed by every node that receives the request.

$REP^i_d[R,\ d,\ o,\ RD^i_{do},\ D^i_d,\ D^i_o,\ f^i_d\ ]$ denotes a reply sent by node i for destination d. The value R states that the entry is a reply, d is the destination identifier, o is the origin of the request, RD^i_d is the reference distance of the node that originated the reply, D^i_d is the distance attained by node i to destination d, D^i_o is the distance attained by node i to the origin of the request o, and f^i_d is the join flag states the status of the status of the destination node. If $f^i_d = 0$, the destination is no longer a pop-node and every node that receives this update should change

the state of that node to a normal node.

6.1.2 Information Stored

Node i maintains five tables to operate, each maintain multiple entries. A maximum lifetime (LT) is associated with each entry, resetting after creation or an update to that entry.

Link-Cost Table (LCT^i): This table is needed if link costs can have different values. If used, each entry l_k^i states the cost of each link from node i to neighbor node k. For simplicity, we use the hop count as the routing metric so all link costs are the same and equal to one.

Distance Table (DT^i): Is the table that stores information reported by each neighbor. The entry for destination d of DT^i is denoted by $DT^i(d)$ and specifies: the identifier of destination d and the distance to d reported by each known neighbor. The distance reported by node $k \in N^i$ for destination d maintained by node i is denoted by D_{kd}^i. If a known neighbor q has not reported a distance for d, then it is assumed that $D_{qd}^i = \infty$. The vector of distances reported by neighbor k and stored by node i is denoted by DT_k^i.

Routing Table (RT^i): The entry for destination d of RT^i is denoted by $RT^i(d)[d, D_d^i, RD_d^i, S_d^i, LR_d^i, sf_d^i, uf_d^i]$ and specifies: the identifier of d, the shortest distance to d (D_d^i), the reference distance to destination which is the smallest value of D_d^i up to time t, the set of valid successors (S_d^i), a lifetime for the entry (LR_d^i), a state flag sf_d^i, and a change flag uf_d^i, indicating whether the entry should be send in the next signaling message or not. The state flag represents whether the destination is pop-node or not. There is an entry for node itself in the routing table: $RT^d(d)[d, 0, \{d\}, LR_d^d, 1, 0]$. The lifetime ($LR_d^d$), represents the periodic update timer and the node d will send an update once this timer is expired.

The *valid successor set* S_d^i is defined as follows:

$$D_{min}^i(d) = Min\{D_{dn}^i + l_n^i \mid n \in N^i\} \tag{6.1}$$

$$S_d^i = \{\ k \in N^i \mid (D_{dk}^i < RD_d^i) \wedge (\ D_{dk}^i + l_k^i = D_{min}^i(d)\ )\ \}$$

The node with the smallest identifier in set S_d^i is denoted by s_d^i. By definition, $D_d^d = 0$, $S_d^d = \{d\}$, $r_d^d = 0$, and $LR_d^d = \infty$.

Pending-Request Table (PRT^i): This table keeps track of the route requests waiting for replies. The entry for destination d is denoted by $PRT^i(d)$ and states: the identifier of node d, the smallest reference distance received in a pending request from a neighbor (PRD_d^i), a request list RL_d^i, and a lifetime for the entry (LP_d^i). The list RL_d^i contains of one or more tuples, with each tuple consisting of the identifier of the origin of a route request and a list of identifier-distance pairs for all neighbors from which the request was received, which we call *pending-request neighbors*. The tuple in RL_d^i corresponding to origin o is denoted by $RL_d^i(o)$ and its content is $[o, PRN_d^i(o)]$. The pair in $PRN_d^i(o)$ corresponding to neighbor k is denoted by $PRN_{dk}^i(o)$ and equals the pair $[k,\ PD_{ok}^i]$, where PD_{ok}^i denotes the distance to origin o reported by k in its route request regarding destination d.

Reply-History Table (RHT^i): plays plays a major role in $ADRP$ by keep tracking of received Replies. The content of RHT^i is simply the identifiers of destinations whose the node received update messages from and the set of neighbor nodes that forwarded that update. An entry for destination d is denoted by RHT_d^i and states d the destination, o the origin of the update, d the destination, the list of identifies for all neighbors for which the update was received, which we call it forward update neighbors ($FUN_d^i(o)$), and LT_{do}^i the lifetime associated with this

entry.

6.1.3 Maintaining Routing State

When node i receives a signaling message from neighbor k, it processes each entry in the message independently of the others. We describe the operation of *ADRP* by focusing on a particular destination d. In the following, it is assumed that a node processes all signaling messages in the same order it receives and all entries in the same order as they are in the message.

ADRP maintains routes either on demand or proactive for each destination d based on the type of the destination. By default all nodes are normal node, therefore the routes are calculated on demand. A destination can force other nodes to calculate the route proactively by sending gratuitous reply messages and sets the flag to one. Route maintenance for on-demand destinations are the same as *ODVR* we mentioned in 5.1.3. In the rest of this section, we discuss the operation of *ADRP* to maintain routes to proactive destinations (i.e. pop-destination).

ADRP relies on REQ and REP messages, to obtain and maintain valid routes to a pop-destination. This involves originating route updates as well route requests, processing route requests from other nodes, update routing information, and reacting to topology changes or the expiration of routing-table entries. Each of these activities is carried out while maintaining total ordering of nodes with respect to destination d by means of sufficient conditions for loop-freedom based solely on the values of the distance and reference distance maintained by each node for destination d.

Originating Gratuitous Reply:

Once a node becomes a pop-node it sends an Gratuitous Reply set the state flag to 1, then it sends the Gratuitous Reply periodically. Intermediate nodes,

also can initiate sending update message regarding destination d when it detects a change in its routing entry for d. A node with an update for destination d originates an update equal to $REP_d^i[R, d,\ o = i,\ RD_d^i,\ D_d^i,\ D_{od}^i, sf_d^i = 1]$ if the following condition is satisfied.

OGC (*Originating Gratuitous-Reply Condition*): Node creates the update REP_d^i if

$$(\ sf_d^d\ changed\ ) \vee ([\ sf_d^i\ ==\ 1]\ \wedge [\ (\ LR_d^i\ expired\ ) \vee (\ uf_d^i == 1\ )\ ]\ ).\ \square$$

ORC states that a node originates a Gratuitous Reply regarding a pop-destination d if the state of the destination has changed (from normal to pop-node and vice versa), or the periodic timer has been expired or there is a change in routing entry. Node d resets its timer after sending the reply. Node i adds the update REP_d^i to its signaling message and broadcast it on its next signaling advertisement. If the destination itself is the originator of then we have: $REP_d^i[Q, d,\ o = d,\ 0,\ 0,\ 0, f_d^i = 1]$

Processing Reply Message:

Nodes process Replies opportunistically, given that all signaling messages are sent in broadcast mode. Nodes are allowed to add new routing-table entries or update existing entries after receiving subsequent updates. When node i receives a reply message from neighbor k, REP_d^k, it uses the following conditions to determine if it can use it to calculate a new route or to improve its distances to d.

ARC (*Accept Reply Condition*): Node i accepts REP_d^k if

$$[(\nexists\ PRT^i(d)\ ) \wedge (\ D_d^k < D_d^i\ )] \vee [(\exists\ PRT^i(d)\ ) \wedge (\ RD_d^k < RD_d^i\ )] \vee$$

$$(\ [\nexists RT^i(d) \wedge d == o\ ]\ ) \vee\ (\nexists\ FUN_d^i(o)\ ).\ \square$$

ARC states that node i will accept, process, and may *forward* the reply message if node k is closer to d than node i is, the reference distance reported by k is smaller than the reference distance that node i must satisfy in its own route request, or it is a gratuitous reply for a pop-destination d that is new to the node i, it is from a new origin, or it is a retransmission of previous reply (periodic update). Node i will update DT^i with $D^i_d(k)$ to remember the distance to the destination reported by k.

Node i does not update routing table if ARC is not satisfied, because loop freedom cannot be ensured by neighbor k.

Assume the case that ARC is satisfied and $RT^i(d)$ does not exist. Node i accept updates regarding a new destination if the destination itself is the originator of that update. In this case, node i uses Eq. (6.1) to compute $D^i_{min}(d)$ and S^i_d. Node i updates $D^i_d = D^i_{min}(d)$ and set the update flag uf^i_d and forwards the update.

If ARC is satisfied and $RT^i(d)$ exists, the router checks whether it has received an update from the same origin before or not. If this is from a new origin then it will process the update by updating the neighbor table and update routing entry based on Eq. (6.1). Otherwise, it will check the neighbor forwarded the update. If it has received an update form that neighbor before then, it accept the update if either neighbor k is closer to d than node i is, or the reference distance reported by k is smaller than the reference distance of node i.

After processing the reply message, the node will check its pending table send reply for every request that it can satisfy. Also deleted the entry the request it generated itself, if there is any.

FRC (*Forwarding Reply Condition*): Node i forwards distance update mes-

sage REP_d^i to all its neighbors if

$$[\ (\nexists\ FUN_d^i(o)\ ) \vee (\ \exists\ FUN_d^i(o) \wedge \nexists FUN_{dk}^i(o)\ )\ ].\ \square$$

FRC states that node i will propagate the Reply to its neighbors if it is an update for a new destination, it is an update for a known pop-destination from a new origin, or a retransmission of a reply (i.e. periodic update).

SRC (*Store Reply Condition*): Node i stores information from DUP_d^k with no other action if

$$(\ sf_d^k = 1) \wedge (RD_d^k \geq RD_d^i) \wedge (\exists\ FUN_d^i(o)\ ) \wedge (\nexists\ FUN_{dk}^i(o)\ ).\ \square$$

ADRP keeps track of received updates according to this condition. Node i stores the information when it is a copy of a previous reply received via a new neighbor. If SRC is satisfied then the node will update its UHT_d^i by simply adding neighbor k to the list of neighbors forwarded the update $RTH_d^i(o) = RTH_d^i(o) \cup RTH_{dk}^i(o)$. SUC states that node i will add neighbor k as a node that forwarded an update for destination d originated by o if node i has already received an update for destination d from the same origin stated by k and k is not already listed as a neighbor from which an update from origin o and with a previous hop other than i has been received.

Topology Discovery and Handling Topology Changes:

To expedite neighbor discovery each node transmits a signaling message periodically containing an update entry $REP_i^i[U,\ i,\ o = \ i,\ RD_o^i,\ D_i^i, D_o^i, sf_i^i]$ for itself with $RD_i^i = D_i^i = D_o^i = 0$. sf_i^i is set to 1 if the node is pop-node, otherwise 0.

Link additions are detected through the reception of signaling messages. Link failures are detected either through the link layer or as a result of a node not

receiving any data packets or signaling messages from a neighbor for a period of time corresponding to the time needed to transmit two or three consecutive signaling messages periodically.

When a node detects a change in link cost or detects link breakage, it tries to find a valid successor that satisfy the 6.1 for the pop-nodes it already knows. If it finds such a successor, it will update its routing entry and send update to its neighbors. On the other hand, if a node can not find a valid successor, it will generate request with its current reference distance value. So we update the *ORC* as follows:

ORC (*Originating Request Condition*): Node creates REQ_d^i if

$$(\nexists\ PRT^i(d)\ ) \wedge (\ [\ \ \nexists\ RT^i(d)\ ] \vee$$

$$[\ (\exists\ RT^i(d)) \wedge (\ S_d^i = \emptyset\ )\ ] \wedge [\ (sf_d^i == 1\ ) \vee (\ S_d^i = \emptyset\ )\ ]\ ).\ \square$$

ADRP processes such a request based on *FRC*, *RFC*, *SRC*, or *RRC*.

FRC (*Forwarding Request Condition*): Node i forwards REQ_d^i to all its neighbors if

$$[\ (\nexists\ PRT^i(d)\ ) \wedge (0 < RD_d^k < D_d^i)\ ]\ \vee$$

$$[\ (\exists\ PRT^i(d)\ ) \wedge (\ [0 < RD_d^k < RD_d^i] \vee [\exists\ PRN_{dk}^i(o)]\ )\ ].\ \square$$

RFC (*Reset Forwarding Condition*): Node i forwards REQ_d^i only to $s_d^i \in S_d^i - \{k\}$ if

$$(\nexists\ PRT^i(d)\ ) \wedge (RD_d^k = 0) \wedge (S_d^i - \{k\} \neq \emptyset).\square$$

SRC (*Store Request Condition*): Node i stores information from REQ_d^k with

no other action if

$$(\ f_d^k = 0) \wedge (p_d^k \neq i) \wedge (RD_d^k \geq RD_d^i) \wedge (\exists\ PRN_d^i(o)\ ) \wedge (\nexists\ PRN_{dk}^i(o)\ ).\ \square$$

RRC (*Reply Request Condition*): Node i sends a reply to REQ_d^k if

$$[\ d = i\ ] \vee [\ (\ RD_d^k > D_d^i\ ) \wedge (\ \ \nexists\ PRT^i(d)\ )\ ] \vee [\ (f_d^k = i) \wedge (S_n^i = \emptyset)\ ].\ \square$$

The detail descriptions of these conditions can be found in 5.1.3. When a node accept a reply for a pop-destination, it also resets the reference distance to the current distance: $RD_d^i = D_d^i$.

Dealing with Unreliable Transmissions:

To make *ADRP* resilient in the presence of unreliable transmissions, each node transmits a signaling message periodically containing a *gratuitous route reply* $REP_i^i[R, i, o = i,\ RD_o^i,\ D_i^i, D_o^i]$ for itself with $RD_i^i = D_i^i = D_o^i = 0$.

To address the fact that signaling messages may be lost, node i always sync its NT_n^i with its neighbor. If there is any mismatch between neighbors routing state stored in neighbor table and the actual values they will exchange message till the mismatch(s) get fixed.

Handling Soft-State and Node Reboots:

ADRP is a soft-state protocol. There is a timer associated to each entry in in DT^i, RT^i, and PRT^i to delete entries that become obsolete as a result of topology changes. (e.g., the network is partitioned or a node fails). Node i renews the lifetime of $RT^i(d)$ with processing every update as well as data packets for destination d. Node i will mark pop-destination d as unreachable if if does not receive update messages for that destination and sets the distances in the vector are set to ∞, however it keeps the value of the reference distance as the previous

distance before updating entry. If node i can not find a valid next hope to a pop-destination, node i will send a gratuitous reply $Reply^i_d[R, d, o,\ RD^i_d,\ D^i_d, D^i_o]$ with $o = i$, $RD^i_d = D^i_d = \infty$ and $D^i_o = 0$ to all its neighbors and waits for a specific amount of time for an update that satisfy ARC. If it does not receive such an update it can obtain a route through the on-demand process by sending request message.

6.2 Loop Freedom in ADRP

The following theorems prove that nodes using ADRP maintain loop-free routing tables at every instant. The proof that any node obtains a route to the destinations in a finite time, provided that there is a physical path available, can be derived from this result.

All nodes coordinate using the equation 6.1 and a routing-table loop is created with respect to a specific destination after processing an update or reply message. We assume the same notation introduced in Section 6.1. The values of D^i_d, D^i_{dn}, RD^i_d, and S^i_d at a specific time t are denoted by $D^i_d(t)$, $D^i_{dn}(t)$, $RD^i_d(t)$ and $S^i_d(t)$ respectively.

Lemma 17. *Consider the case that a node has updated its routing at time $t_{[new]}$ and change its successor from old one $k[old]$ a new successor $k[new]$ or adding $k[new]$ to its successor set S^i_d after processing of a gratuitous reply $REP^i_{dk[new]}(o)$. It must be true that:*

$$RD^i_d(t_{new}) \ <= \ RD^i_d(t_{old}). \ \square$$

Proof. node i process $REP^i_{dk[new]}(o)$ if $D^i_{dk[new]} < RD^i_d$. Hence, neighbor $k[new]$ passes the first check of Eq. (6.1). The distance after adding the new successor to the set will be calculated by $D^i_{min}[new](d) = Min\{D^i_{min}[old](d), D^i_{dk[new]} +$

$l^i_{k[new]}\}$ Therefor, $D^i_{min}[new](d) <= D^i_{min}[old](d)$. By definition, the $RD^i_d(t_{new}) = Min\{RD^i_d(t_{new}), D^i_{min}[new](d)\}$, which proof the lemma. □

Lemma 17 proofs that the reference distance will be decreasing if a node process a reply message.

Lemma 18. *Consider a set of nodes $i, i+1$ such that $i+1 \in S^i_d$. Then, at anytime t it must be true that $RD^i_d(t) > RD^{i+1}_d(t)$.*

Proof. Node i will select node $i+1$ as a result of processing a route reply . If it is processing a gratuitous reply then it must be true that $RD^i_d(t) > D^{i+1}_d(t)$. Also by definition, $D^{i+1}_d(t) >= RD^{i+1}_d(t)$. Hence, $RD^i_d(t) > RD^{i+1}_d(t)$. If node i choose node $i+1$ after processing an REP then from the operation of the ADRP we have: $RD^i_d = D^i_{min}(d)$ and from 6.1 we have: $D^i_{d[i+1]} + l^i_{i+1} = D^i_{min}(d)$. Because, $l^i_{i+1} > 0$ we have: $D^i_{d[i+1]} = D^{i+1}_d < D^i_{min}(d) = RD^i_d$. Substituting $RD^{i+1}_d < D^{i+1}_d$ we have: $RD^i_d(t) > RD^{i+1}_d(t)$. □

Lemma 19. *Despite any change in the network, every node will process the reply to its latest request that reflects the smallest reference number.*

Proof. Assume that node i is waiting for replies for its request, $REQ^i_d(RD^i_d = rd1)$. Assume that node i does not process any update message, hence does not change its reference distance. According to ARC Node i simply ignores any reply, $REP^i_d(D^o_d > rd1)$. Now assume that node i update its routing entry for destination d as a result of processing of an update message. According to lemma 17 $rd2 = RD^i_d(t_{new}) <= RD^i_d(t_{old}) = rd1$. Hence, any request been sent after that will be sent by $REQ^i_d(RD^i_d = rd2 <= rd1)$ and replies to previous one will be ignored. □

Theorem 20. *No routing loop can be created as a result of processing a REP message.*

Proof. Consider a set of nodes $l = \{v_1, v_2, ..., v_h, v_1\}$ different than destination d and let $s_d^{v_i}$, $D_d^{v_i}$ and $D_{ds_d^{v_i}}^{v_i}$ denote the next hop, distance to d, and distance to d through the next hop at node v_i, respectively, with $D_{ds_i^i}^{v_i} = D_d^{s_i}$. Assume that this set of nodes create a routing-table loop L of h hops by setting $s_d^{v_i} = v_{i+1}$ for $1 \leq i \leq h-1$ and $s_d^{v_h} = v_1$.

We consider two cases:

cCase 1: processing a gratuitous reply: For the sake of contradiction, further assume that each node in loop L satisfies the ordering constraint $D_d^{v_i} = \infty$ or $D_{dv_{i+1}}^{v_i} < D_d^{v_i} < \infty$ for $1 \leq i \leq h-1$ and $D_d^{v_h} = \infty$ or $D_{dv_1}^{v_h} < D_d^{v_h} < \infty$. Assume that the loop formed at time t when v_h process its route reply $UPD_{v_1}^{v_h}$ and changes its successor to v_1. It must be true that before processing the DUP we have : $RD_d^{v_{h-1}}(t_{old}) >= RD_d^{v_h}(t_{old})$. Based on 17 we have $RD_d^{v_h}(t_{old}) > RD_d^{v_h}(t_{new})$ and reference distance of node v_{h-1} remains the same. So we have: $RD_d^{v_{h-1}}(t_{new}) = RD_d^{v_{h-1}}(t_{old}) > RD_d^{v_h}(t_{old}) >= RD_d^{v_{h-1}}(t_{new})$. Hense, $RD_d^{v_{h-1}}(t_{new}) > RD_d^{v_{h-1}}(t_{new})$. Traversing through the loop l based on lemma 18 we have:

$$RD_d^{v_1} > RD_d^{v_1}$$

which is a contradiction.

case 2: processing a reply to a request: For the sake of contradiction, further assume that each node in loop L satisfies the ordering constraint $D_d^{v_i} = \infty$ or $D_{dv_{i+1}}^{v_i} < D_d^{v_i} < \infty$ for $1 \leq i \leq h-1$ and $D_d^{v_h} = \infty$ or $D_{dv_1}^{v_h} < D_d^{v_h} < \infty$. Assume that the loop formed at time t when v_h process its route reply $REP_{v_1}^{v_h}$ and changes its successor to v_1. We argue that the origin of the reply can not be any node in the loop. By definition, node v_h can not be the originator of that reply. Assume that, node $v_i \in l \wedge v_i \neq v_h$ is the origin of the reply. Based on ARC we have:

$D_d^{v_i} < RD_d^{v_h}$ and also by definition $RD_d^{v_i} <= D_d^{v_i}$. Therefor:

$$RD_d^{v_i} < RD_d^{v_h}$$

Traversing through the loop from node v_i to node v_h and apply lemma 18 we have:

$$RD_d^{v_i} > RD_d^{v_{i+1}} > RD_d^{v_{i+2}} > ... > RD_d^{v_h}$$

Which implies:

$$RD_d^{v_i} > RD_d^{v_h}$$

and this is a contradiction. Therefore, no routing loop can be created as a result of processing of an REP message. □

Theorem 21. *No routing-table loops can exist for destination d if $D_d^i = \infty$ or $D_{ds_d^i}^i < D_d^i < \infty$ for any network node i other than d at every instant of time.*

Proof. Consider a set of nodes $\{v_1, v_2, ..., v_h, v_1\}$ different than destination d and let $s_d^{v_i}$, $D_d^{v_i}$ and $D_{ds_i^i}^{v_i}$ denote the next hop, distance to d, and distance to d through the next hop at node v_i, respectively, with $D_{ds_i^i}^{v_i} = D_d^{s_i}$. Assume that this set of nodes create a routing-table loop L of h hops by setting $s_d^{v_i} = v_{i+1}$ for $1 \leq i \leq h-1$ and $s_d^{v_h} = v_1$. For the sake of contradiction, further assume that each node in loop L satisfies the ordering constraint $D_d^{v_i} = \infty$ or $D_{dv_{i+1}}^{v_i} < D_d^{v_i} < \infty$ for $1 \leq i \leq h-1$ and $D_d^{v_h} = \infty$ or $D_{dv_1}^{v_h} < D_d^{v_h} < \infty$.

Loop L cannot exist if $D_d^{v_i} = \infty$ for any i with $1 \leq i \leq h-1$ because then v_i has no next hop to d. It thus follows that $D_{dv_{i+1}}^{v_i} < D_d^{v_i} < \infty$ for $1 \leq i \leq h-1$ and $D_{dv_1}^{v_h} < D_d^{v_h} < \infty$. This, however, is a contradiction, because it implies that $D_d^{v_i} > D_d^{v_i}$ for $1 \leq i \leq h-1$ and $D_d^{v_h} > D_d^{v_h}$. Therefore, the theorem is true. □ □

Theorem 21 states that *any* routing protocol that satisfies the stated ordering constraint at every instant ensures loop-free routing. The proof of the following theorem demonstrates that ADRP enforces this ordering constraint at every instant.

Theorem 22. *ADRP provides loop-free routing at every instant.*

Proof: The proof follows from Theorems 20 and 21. □

6.3 Performance Comparison

We compared the performance of ADRP to other MANET routing protocols: AODV and OLSR implemented in the network simulator (ns3) .

6.3.1 Simulation Results

The simulated network consisted of 50 nodes spread uniformly and randomly in a 500 × 1500m area at the beginning of the simulation. all nodes started the experiment at a random location within the simulation area and moved as defined by the random waypoint model, where each node selected a random destination within the working area and moved linearly to that location at a predefined speed. After reaching its destination, it pauses for a specified period (pause time) and then selects a new random location and continues the process again. A subset of nodes are selected as popular destination. In this simulation we assumed that 6 nodes are popular and we selected them randomly at the beginning of the simulation. The scenarios include 25 data flows from 25 different source routers to different destinations that are chosen randomly. Note that, the number of flows varies in a scenario that we measured the "Effect of Number of Flows". Traffic sources are on-off applications with on and off time of 1 second, which generate

packets of size 512 bytes and rate of 15 packets per second. The data will flow for 200 seconds and then we will generate new data flows by selecting new sources and new destinations. New data flows will be generated every 200 seconds after that.

We have two scenarios with respect to the network traffic pattern: In first one %50 of the traffic is routed toward the popular destination, and in another scenario %75 of the flows are targeted to a popular node. In each part we considered different scenarios to study the behavior of routing protocols with respect to mobility and number of data flows. The scenarios were chosen to stress all three protocols, rather than to attain good performance for either on-demand or proactive routing. The results are demonstrated in figures 6.1 though 6.3.

6.3.2 Effect of Mobility

In order of study of effect of mobility on the on the performance of routing algorithms, we changed the parameters of our mobility model: Speed and Pause time. The speed varies from 5 m/s to 30 m/s to mimic the low and high mobility respectively. The results as functions of node speed are shown in Figure 6.1, where figures 6.1 (a), (b), and (c) are for the first scenario, and (d), (e), and (f) demonstrate the results for the second one.

The packet delivery rate decreases as the router speed increases due to more topology changes. Routes break due to topology change and packets drop till the routing protocol finds a new route. Link failures ADRP are detected by the absence of a number of consecutive gratuitous Reply messages, and a route discovery process is performed to establish new route between source and destinations.

Because of the delays incurred in detecting link failures and in establishing new routes after that, as router speed increases more and more data packets traversing

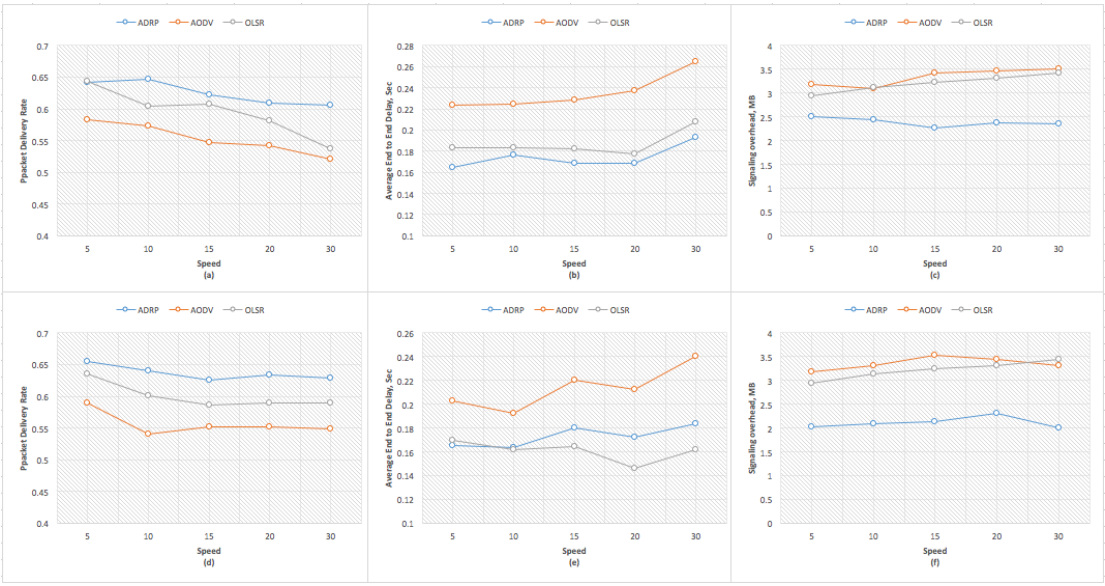

Figure 6.1: Performance comparison as a function of Node Speed.

failed routes end up being dropped. As the results in Figure 6.1 indicate, ADRP outperform all other routing algorithms.

In next scenario, we studied the effect of pause time on performance of routing protocol. Figure 6.2 shows the packet delivery, delay, and signaling overhead as a function of node pause time. The result of the first scenario is reported in figures 6.2(a), (b), and (c) and the rest show the results for . Pause times vary from 0 seconds (high mobility) to 200 seconds (low mobility), and the speed of individual nodes is chosen randomly from 0 to 20 m/s. The other parameters remain the same. In the first The packet delivery ratio of OLSR and ODVR are very close to the delivery ratio for AODV. ADRP attains shorter delays compared to other routing algorithms. The traffic is a little more than the ODVR, however it is still less than the signaling traffic of AODV and OLSR.

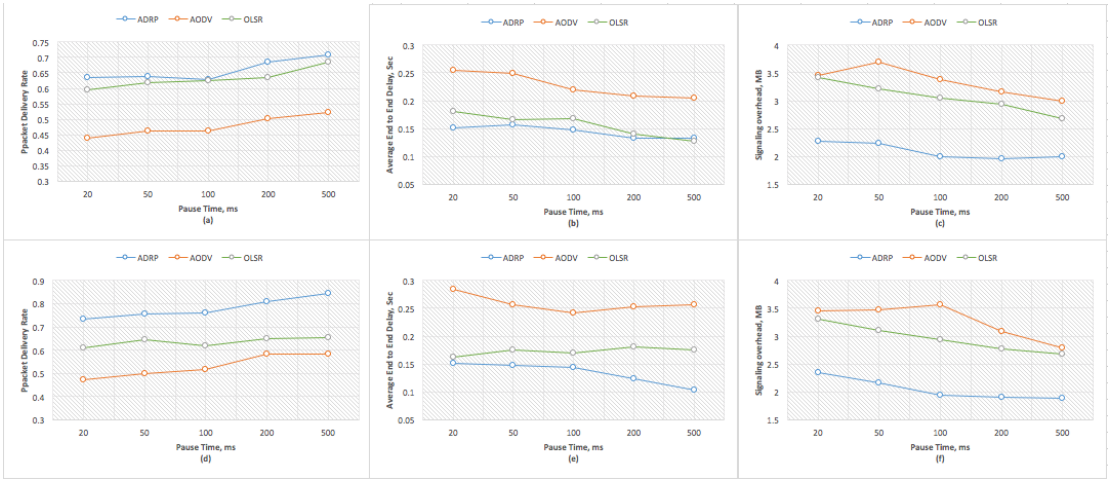

Figure 6.2: Performance comparison as a function of Pause Time.

6.3.3 Effect of Number of Flows

Figure 6.3 shows the results of the effect that data traffic and number of connections have on performance. Sources and destinations are selected randomly and the number of source-destination pairs varies between 20 (average data flow) to 40 (high data flow). The delivery rate decreases and the average end-to-end delays increase for all three protools as the number of sources increases. This is expected, as more traffic causes more congestion in the network, and a link failure affects more transmissions when the number of data flows increases. ADRP outperforms other algorithms as the number of of connections increases. The signaling in OLSR is independent of the number of connections in the network; hence, the total signaling overhead remains the same. The signaling of the ADRP is more than the signaling of the ODVR due to signaling for proactive nodes.

6.4 Summary

We introduced ADRP, an adaptive routing protocol for ad-Hoc networks that eliminates routing-table loops by ordering nodes based solely on the distances

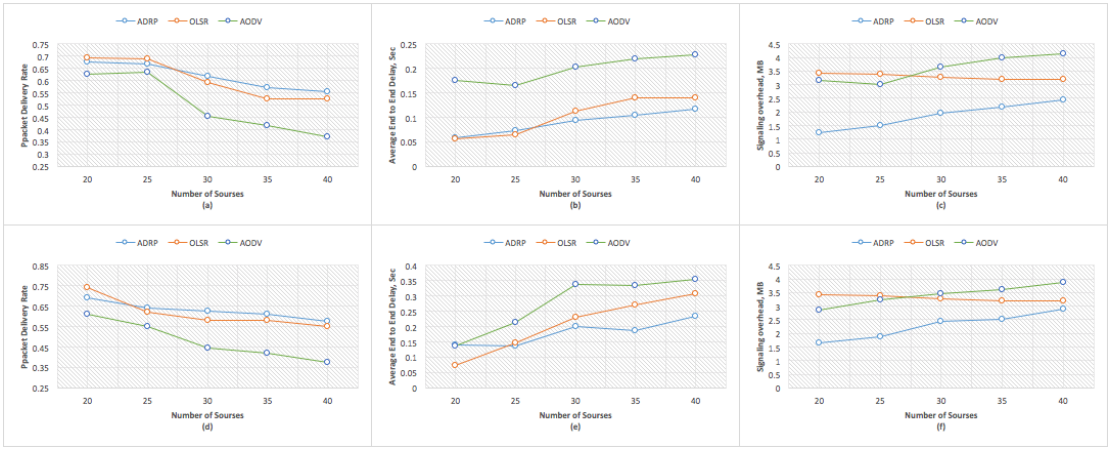

Figure 6.3: Performance comparison as a function Number of Sources.

they maintain. ADRP uses a hybrid approach to obtain and maintain routes to destination: proactive to popular nodes and on-demand to regular nodes. Both the proactive and on-demand algorithms rely on only on route request and route replies and the proactive routing can be done with minimum modification to the on-demand routing. Route requests and reply messages state the reference distance, the distance to destination reported by the originator of that message, that defines which nodes can process the message and should reply/change state based on the information of that message. Every node can respond to the request message with a reply if it meets the reference distance and can generate a reply in the absence of request, if it detects a change in a routing state for a popular destination. Every node that receives such a reply will check the reference distance and process the information if this distance meets the required reference distance of a node.

We proved that the ADRP is loop-free at every instance and have better performance than the pure on-demand routing algorithm such as its predecessor ODVR. We implemented ADRP in ns3 and compared the performance of it with both on-demand and proactive routing protocols: AODV and OLSR, considering

two different traffic patterns, where 75% and 50% of the traffic are routed to popular nodes in a network. Taking into account the effect of mobility, number of flows, and network size on the performance of the protocols. The simulation results show that with a little extra overhead, ADRP attains better performance than the other protocols.

Chapter 7

Conclusion

The Internet has changed our everyday lives and the people using the Internet has change the way it is used. The Internet has been designed as a point to point communication between two host which allowed to share limited and expensive computational resources. The internet has been evolved from very basic task of forwarding packets of data among a few number of hosts to hosting and transferring billions of user-generated contents, to allow users connect everything and almost anything and share text, image, audio, and video.

Internet of Things is a novel paradigm refers to a dynamic network infrastructure that enables every "thing" communicate anytime, anywhere, and through any media. The network design should evolve to address such a huge change in the Internet usage. Information-centric networking is becoming a promising candidate for the next generation of the Internet. Many ICN architectures have been proposed to enable access to content and services by name, independently of their location, to improve system performance and end-user experience. The core of all ICN architectures are name resolution and routing of content, and several approaches have been proposed.

Content routing algorithm is studied and a link-state and defusing computation

based routing algorithm for content-centric networks are introduced. The Link-State Content Routing (LSCR) calculates loop-free paths to the nearest replica of an NDO or name prefix. Using the full topology information, LSCR creates a directed acyclic graph for each destination. The anchor information, however, is propagated selectively and efficiently so we reduced the complexity and communication overhead. The algorithm also ranks neighbors lexicographically based on their distances and select a valid next hop from neighbors that offer smaller distance than the router itself.

The Diffusive Name-based Routing Protocol is introduced as the first name-based content routing protocol based on diffusing computations. DNRP proves that it provides loop-free multi-path routes to multi-homed name prefixes at every instant. Routers that run DNRP do not require to have knowledge about the network topology, use complete paths to content replicas, know about all the sites storing replicas of named content, or use periodic updates. DNRP has better performance compared to link-state routing protocols when topology changes occur or new prefixes are introduced to the network.

In the next step, we tackled the routing problem in Wireless networks. We introduced the first on-demand routing protocol that eliminates routing-table loops by ordering nodes based solely on the distances they maintain and advertise to destinations, as well as reference distances they state in route requests. Routing state is established on demand by means of route requests stating the reference distance that a responder must satisfy to be allowed to reply. Intended destinations or nodes that meet the reference distances respond with replies, and nodes also send distance updates regarding the sources of requests and active destinations.

And Finally, we advanced the routing in MANETs by introducing an adaptive routing protocol for Ad-Hoc networks that uses a hybrid approach to obtain and

maintain routes to destination: proactive to popular nodes and on-demand to regular nodes, with minimum modification to the on-demand routing protocol.

We introduced several routing algorithms for ICN and wireless network to make them scalable and more practical. However, more work should be done to make ICN more practical and make it a perfect candidate for the Future Internet. Possible approaches for future work include: communicating partial topology information [107], reducing the frequency with which LSAs have to be sent [108], and improving the way in which routers update anchor information after resource failures. The use of content directory for efficient routing can be explored. Also, the effect of caching on performance of each of the protocol should be investigated.

Bibliography

[1] Content centric networking project (CCN).

[2] Content mediator architecture for content-aware networks (COMET) project.

[3] Fp7 PURSUIT project.

[4] Mobility first project.

[5] Named data network project.

[6] Publish subscribe internet technology (PURSUIT) project.

[7] Scalable and adaptive internet solutions (SAIL) project.

[8] Ccnx synchronization protocol, 07 2013.

[9] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7):26–36, July 2012.

[10] Mark Ain, Dirk Trossen, Pekka Nikander, Sasu Tarkoma, Kari Visala, Ken Rimey, Trevor Burbridge, Jarno Rajahalme, Janne Tuononen, Petri Jokela, Jimmy Kjällman, Jukka Ylitalo, Janne Riihijärvi, Borislava Gajic, George Xylomenos, Petri Savolainen, and Dmitrij Lagutin. D2.3 architecture definition, component descriptions, and requirements. In *Deliverable, PSIRP 7th FP EU-funded project,*, 2009.

[11] Alexander L. Wolf Antonio Carzaniga, David S. Rosenblum. Content-Based Addressing and Routing: A General Model and its Application.

[12] Baruch Awerbuch. A new distributed Depth-First-Search algorithm. *Information Processing Letters*, 20(3):147–150, April 1985.

[13] Hitesh Ballani and Paul Francis. Towards a global IP anycast service. *ACM SIGCOMM Computer Communication Review*, 35(4):301, October 2005.

[14] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (CBT). *ACM SIGCOMM Computer Communication Review*, 23(4):85–95, October 1993.

[15] P. Baran. On Distributed Communications Networks. *IEEE Transactions on Communications*, 12(1):1–9, March 1964.

[16] Md. Bari, Shihabur Chowdhury, Reaz Ahmed, Raouf Boutaba, and Bertrand Mathieu. A survey of naming and routing in information-centric networks. *IEEE Communications Magazine*, 50(12):44–53, December 2012.

[17] J. Behrens and J. J. Garcia-Luna-Aceves. Hierarchical routing using link vectors. In *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98CH36169)*, volume 2, pages 702–710. IEEE.

[18] Dimitri Bertsekas and Robert Gallager. Data networks (2nd ed.). January 1992.

[19] A. Carzaniga, M.J. Rutherford, and A.L. Wolf. A routing scheme for content-based networking. In *IEEE INFOCOM 2004*, volume 2, pages 918–928. IEEE.

[20] Shigang Chen and Klara Nahrstedt. Distributed qos routing with imprecise state information. In *Computer Communications and Networks, 1998. Proceedings. 7th International Conference on*, pages 614–621. IEEE, 1998.

[21] C. Cheng, R. Riley, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves. A loop-free extended Bellman-Ford routing protocol without bouncing effect. *ACM SIGCOMM Computer Communication Review*, 19(4):224–236, August 1989.

[22] Jaeyoung Choi, Jinyoung Han, Eunsang Cho, Ted Kwon, and Yanghee Choi. A Survey on content-oriented networking for efficient content delivery. *IEEE Communications Magazine*, 49(3):121–127, March 2011.

[23] R. Coltun. Ospf an internet routing protocol. *ConneXions*, 3(8):19–25, 1989.

[24] M Scott Corson and Anthony Ephremides. A distributed routing algorithm for mobile wireless networks. *Wireless networks*, 1(1):61–81, 1995.

[25] S. Deering, D.L. Estrin, D. Farinacci, and V. Jacobson. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, April 1996.

[26] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[27] Andrea Detti, Nicola Blefari Melazzi, Stefano Salsano, and Matteo Pomposini. CONET. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking - ICN '11*, page 50, New York, New York, USA, August 2011. ACM Press.

[28] Andrea Detti, Nicola Blefari Melazzi, Stefano Salsano, and Matteo Pomposini. CONET. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking - ICN '11*, page 50, New York, New York, USA, August 2011. ACM Press.

[29] Edsger W Dijkstra and Carel S. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

[30] J. J. Garcia-Luna-Aceves. A distributed, loop-free, shortest-path routing algorithm. In *INFOCOM '88. Networks: Evolution or Revolution, Proceedings. Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies, IEEE*, pages 1125–1137, Mar 1988.

[31] J. J. Garcia-Luna-Aceves. A unified approach to loop-free routing using distance vectors or link states. *ACM SIGCOMM Computer Communication Review*, 19(4):212–223, August 1989.

[32] J. J. Garcia-Luna-Aceves. System and method for discovering information objects and information object repositories in computer networks, December 27 2001. US Patent App. 09/810,148.

[33] J. J. Garcia-Luna-Aceves. Name-based content routing in information centric networks using distance information. In *ACM Conference on Information-Centric Networking*, pages 24–26, Paris, France, September 2014.

[34] J. J. Garcia-Luna-Aceves. Name-based content routing in information centric networks using distance information. In *Proceedings of the 1st international conference on Information-centric networking*, pages 7–16. ACM, 2014.

[35] J. J. Garcia-Luna-Aceves. Routing to multi-instantiated destinations: Principles and applications. In *Proc. IEEE ICNP 2014*, 2014.

[36] J. J. Garcia-Luna-Aceves. A fault-tolerant forwarding strategy for interest-based information centric networks. In *IFIP Networking Conference (IFIP Networking), 2015*, pages 1–9. IEEE, 2015.

[37] J. J. Garcia-Luna-Aceves and M. Parsa. System and method for information object routing in computer networks, March 20 2003. WO Patent App. PCT/US2002/028,829.

[38] J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Transactions on Networking (TON)*, 1(1):130–141, 1993.

[39] Ali Ghodsi, Teemu Koponen, Jarno Rajahalme, Pasi Sarolahti, and Scott Shenker. Naming in content-oriented architectures. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking - ICN '11*, page 1, New York, New York, USA, August 2011. ACM Press.

[40] Ali Ghodsi, Scott Shenker, Teemu Koponen, Ankit Singla, Barath Raghavan, and James Wilcox. Information-centric networking. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks - HotNets '11*, pages 1–6, New York, New York, USA, November 2011. ACM Press.

[41] Mark Gritter and David R. Cheriton. An architecture for content routing support in the internet. page 4, March 2001.

[42] Oliver Heckmann, Michael Piringer, Jens Schmitt, and Ralf Steinmetz. On realistic network topologies for simulation. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research - MoMeTools '03*, page 28, New York, New York, USA, August 2003. ACM Press.

[43] Mahmudul Hoque, Cheng Yi, Adam Alyyan, , and Beichuan Zhang. An ospf based routing protocol for named data networking. Technical report, NDN Technical Report NDN, July 2012.

[44] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking - MobiCom '00*, pages 56–67, New York, New York, USA, August 2000. ACM Press.

[45] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies - CoNEXT '09*, page 1, New York, New York, USA, December 2009. ACM Press.

[46] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies - CoNEXT '09*, page 1, New York, New York, USA, December 2009. ACM Press.

[47] Konstantinos V. Katsaros, Nikos Fotiou, Xenofon Vasilakos, Christopher N. Ververidis, Christos Tsilopoulos, George Xylomenos, and George C. Polyzos. *On inter-domain name resolution for information-centric networks*, volume 7289 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, May 2012.

[48] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *ACM SIGCOMM Computer Communication Review*, 37(4):181, October 2007.

[49] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 181–192. ACM, 2007.

[50] Jim Kurose. Information-centric networking: The evolution from circuits to packets to content. *Computer Networks*, 66:112–120, June 2014.

[51] Vince Lehman, AKM Mahmudul Hoque, Yingdi Yu, Lan Wang, Beichuan Zhang, and Lixia Zhang. A secure link state routing protocol for ndn.

[52] A K M Mahmudul-Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. NLSR: Named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking - ICN '13*, page 15, New York, New York, USA, August 2013. ACM Press.

[53] A K M Mahmudul-Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. NLSR: Named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking - ICN '13*, page 15, New York, New York, USA, August 2013. ACM Press.

[54] James Mathewson, Maziar Barijough, Ehsan Hemmati, J. J. Garcia-Luna-Aceves, and Marc Mosko. Sconet : Simulator content networking. In *CCNx-Con*, 2015.

[55] J. Moy. Ospf version 2, rfc 1583, march 1994.

[56] Jeremy Siek. Boost graph library: dijkstra shortest paths (BOOST).

[57] Ignacio Solis and J. J. Garcia-Luna-Aceves. Robust content dissemination in disrupted environments. In *Proceedings of the third ACM workshop on Challenged networks - CHANTS '08*, page 3, New York, New York, USA, September 2008. ACM Press.

[58] Athanasios V. Vasilakos, Zhe Li, Gwendal Simon, and Wei You. Information centric network: Research challenges and opportunities. *Journal of Network and Computer Applications*, 52:1 – 10, 2015.

[59] Russ White, James Ng, Donald Slice, Steven Moore, et al. Enhanced interior gateway routing protocol. 2014.

[60] George Xylomenos, Christopher N. Ververidis, Vasilios A. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys & Tutorials*, 16(2):1024–1049, 2014.

[61] E.W. Zegura, M.H. Ammar, and S. Bhattacharjee. Application-layer anycasting: a server selection architecture and use in a replicated Web service. *IEEE/ACM Transactions on Networking*, 8(4):455–466, 2000.

[62] K. Bhargavan, D. Obradovic, and C. Gunter, "Formal Verification of Standards for Distance Vector Routing Protocols," *Journal of the ACM*, July 2002.

[63] A. Boukersche, *Handbook of Algorithms for Wireless and Mobile Computing*, Chapman and Hall, 2006.

[64] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)" *RFC 3626*, 2003.

[65] M.S. Corson and A. Ephremides, "A Distributed Routing Algorithm for Mobile Wireless Networks," *Wireless Networks*, Jan 1995.

[66] Dannewitz, C., Kutscher, D., Ohlman, B., Farrell, S., Ahlgren, B., and Karl, H. (2013). Network of information (netinf)âĂŞan information-centric networking architecture. Computer Communications, 36(7), 721-735.

[67] T. Clausen, "Comparative Study of Routing Protocols for Mobile Ad-hoc networks," Research Report RR-5135, INRIA, 2004.

[68] J.J. Garcia-Luna-Aceves, "Loop-Free Routing Using Diffusing Computations," *IEEE/ACM Trans. Networking*, 1993.

[69] J.J. Garcia-Luna-Aceves and H. Rangarajan, "A New Framework for Loop-Free On-Demand Routing Using Destination Sequence Numbers," *Proc. IEEE MASS '04*, 2004.

[70] J.J. Garcia-Luna-Aceves and S. Roy, "On-Demand Routing in Ad Hoc Networks Using Link Vectors," *IEEE JSAC*, Vol. 23, No. 3, March 2005.

[71] D. B. Johnson and D. A. Maltz, “Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Computing*, Chapter 5, Kluwer Academic Publishers, 1996.

[72] Y. Ko and N.Vaidya, “Location-Aided Routing (LAR) in Mobile Ad Hoc Networks," *Wireless Networks*, Vol. 6, No. 4, 2000.

[73] M. Marina and S. Das, “Ad Hoc On-Demand Multipath Distance vector Routing," *Wireless Commun. Mob. Comput.*, 2006.

[74] M. Mirzazad-Barijough and J.J. Garcia-Luna-Aceves, “Making On-Demand Routing Efficient with Route-Request Aggregation," *Proc. ACM MSWiM ’16*, 2016.

[75] S. Murthy and J.J. Garcia-Luna-Aceves, “An Efficient Routing Protocol for Wireless Networks." *Mobile Networks and Applications*, 1996.

[76] G. Pei et al., "A Wireless Hierarchical Routing Protocol with Group Mobility," *Proc. IEEE WCNC ‘99*, 1999.

[77] C. E. Perkins and P. Bhagwat, “Routing over Multihop Wireless Network of Mobile Computers," *Proc. ACM SIGCOMM ‘94*, 1994.

[78] C. E. Perkins and E. M. Royer, “Ad-Hoc On-Demand Distance Vector Routing," *Proc. IEEE WMCSA ‘99*, 1999.

[79] C. Perkins et al., *Ad Hoc Networkiing*, Addison-Wesley, 2008.

[80] H. Rangarajan and J.J. Garcia-Luna-Aceves, “On-demand Loop-Free Routing in Ad hoc Networks Using Source Sequence Numbers,” *Proc. IEEE MASS ‘05*, Nov. 2005.

[81] J. Raju and J.J. Garcia-Luna-Aceves, “A New Approach to On-Demand Loop-Free Multipath Routing," *Proc. IEEE IC3N ‘99*, Oct. 1999.

[82] C. Shiflet, E. M. Belding-Royer, and C.E. Perkins, “Address Aggregation in Mobile Ad Hoc Networks," *Proc. IEEE ICC ‘04*, 2004.

[83] R. Van Glabbeek et al., “Sequence Numbers Do Not Guarantee Loop Freedom–AODV Can Yield Routing Loops," *Proc. ACM MSWiM ‘13*, Nov. 2013.

[84] J. Wu and H. Li, “On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks," *Proc. ACM Int’l Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 1999.

[85] X. Wu et al., “A Unified Analysis of Routing Protocols in MANETs," *IEEE Trans. on Communications*, March 2010.

[86] M. Zhou et al., “The Proof of AODV Loop Freedom," *Proc. IEEE WCS ‘09*, 2009.

[87] T. Camp, J. Boleng, and V. Davies, 2002. A survey of mobility models for ad hoc network research. Wireless communications and mobile computing, 2(5), pp.483-502.

[88] A. Boukersche, *Handbook of Algorithms for Wireless and Mobile Computing*, Chapman and Hall, 2006.

[89] S. Murthy and J.J. Garcia-Luna-Aceves, “An Efficient Routing Protocol for Wireless Networks." *Mobile Networks and Applications*, 1996.

[90] C. E. Perkins and P. Bhagwat, “Routing over Multihop Wireless Network of Mobile Computers," *Proc. ACM SIGCOMM ‘94*, 1994.

[91] C. E. Perkins and E. M. Royer, “Ad-Hoc On-Demand Distance Vector Routing," *Proc. IEEE WMCSA ‘99*, 1999.

[92] J. Raju and J.J. Garcia-Luna-Aceves, “A New Approach to On-Demand Loop-Free Multipath Routing," *Proc. IEEE IC3N ‘99*, Oct. 1999.

[93] S. Roy and J.J. Garcia-Luna-Aceves, “Node-Centric Hybrid Routing for Ad Hoc Networks," *Proc. 10th IEEE/ACM MASCOTS ‘02*, 2002.

[94] R. Van Glabbeek et al., “Sequence Numbers Do Not Guarantee Loop Freedom–AODV Can Yield Routing Loops," *Proc. ACM MSWiM ‘13*, Nov. 2013.

[95] X. Wu et al., “A Unified Analysis of Routing Protocols in MANETs," *IEEE Trans. on Communications*, March 2010.

[96] M. Zhou et al., “The Proof of AODV Loop Freedom," *Proc. IEEE WCS ‘09*, 2009.

[97] G. Yovanof and G. Hazapis, An architectural framework and enabling wireless technologies for digital cities and intelligent urban environments, Wireless Pers. Commun., vol. 49, no. 3, pp. 445-463, May 2009

[98] Hou, Songfan, et al. "Performance Comparison of AODV and DSR in MANET Test-bed Based on Internet of Things." Vehicular Technology Conference (VTC Fall), 2015 IEEE 82nd. IEEE, 2015.

[99] Wang L, Olariu S. A two-zone hybrid routing protocol for mobile ad hoc networks. IEEE transactions on Parallel and distributed systems. 2004 Dec;15(12):1105-16.

[100] M. Steenstrup, "Cluster-Based Networks," Ad Hoc Networking, C.E. Perkins, ed., Addison-Wesley, Reading: Mass., pp. 75-138, 2000.

[101] A.B. McDonald and T. Znati, "A Dual-Hybrid Adaptive Routing Strategy for Wireless Ad Hoc Networks," Proc. IEEE Wireless and Networking Conf., pp. 1125-1130, Sept. 2000.

[102] A. Iwata, C. Chiang, G. Pei, M. Gerla, and T. Chen, "Scalable Routing Strategies for Ad Hoc Wireless Networks," IEEE J. Selected Areas in Comm., special issue on ad hoc networks, vol. 17, no. 8, pp. 1369-1379, 1999.

[103] G. Pei, M. Gerla, and X. Hong, "LANMAR: Landmark Routing for Large Scale Wireless Ad Hoc Networks with Group Mobility," Proc. MOBIHOC Conf. 2000, pp. 11-18, Nov. 2000

[104] M.R. Pearlman and Z.J. Haas, Determining the Optimal Configuration for the Zone Routing Protocol, IEEE J. Selected Areas in Comm., special issue on ad hoc networks, vol. 17, no. 8, pp. 1395- 1414, 1999.

[105] Z.J. Haas and M.R. Pearlman, The Performance of Query Control Schemes for the Zone Routing Protocol, ACM/IEEE Trans. Networking, vol. 9, no. 4, pp. 427-438, 2001

[106] V. Ramasubramanian, Z.J. Haas, and E.G. Sirer, "SHARP: A Hybrid Adaptive Routing Protocol for Mobile Ad Hoc Networks," Proc. MOBIHOC Conf. 2003, pp. 303-314, June 2003.

[107] J. Behrens, J. J. Garcia-Luna-Aceves, Hierarchical routing using link vectors, in: INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Vol. 2, IEEE, pp. 702–710.

[108] J. J. Garcia-Luna-Aceves, M. Spohn, Scalable link-state internet routing, in: Network Protocols, 1998. Proceedings. Sixth International Conference on, IEEE, 1998, pp. 52–61.

[109] Di Caro, Gianni, Frederick Ducatelle, and Luca Maria Gambardella. "AntHocNet: an adaptive natural inspired algorithm for routing in mobile ad hoc networks." European Transactions on Telecommunications 16.5 (2005): 443-455.