University of California
Los Angeles

BOND:
Unifying Mobile Networks with Named Data

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Michael Richard Meisel

2011

Table of Contents

List of Figures

List of Tables

Abstract of the Dissertation

BOND:

Unifying Mobile Networks with Named Data

by

Michael Richard Meisel

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2011

Professor Lixia Zhang, Chair

Much research has been devoted to *freeform* networks – that is, mobile, multi-hop wireless networks where the nodes move freely and unpredictably. This includes connected mobile ad hoc networks (MANETs) as well as disconnected networks such as vehicular ad hoc networks (VANETs) and opportunistic networks. To date, the fundamental goal of such research has primarily been data delivery between node pairs as identified by their IP addresses. Despite a rich literature of solutions to this problem extending back over 30 years, no sufficiently general, agreed-upon standard exists today.

This dissertation returns to the drawing board to rethink how to align modern data communications with the highly dynamic nature of freeform wireless networks and the broadcast nature of wireless channels. This leads to an entirely different approach, which is based on Named Data Networking (NDN). NDN asserts that data, not hosts, are the central element of modern networks.

The realization of this approach and core contribution of this dissertation is

the Broadcast-Only Named Data (BOND) protocol, the first complete protocol specification based on NDN concepts. BOND has the following unique properties:

- BOND supports *all* freeform networks, connected and disconnected, without so much as a configuration change.
- BOND communication is entirely based on named data.
- BOND's performance does not depend on the level of physical mobility in the network.
- BOND uses exclusively broadcast communication for all packets, allowing for more efficient use of the wireless channel and more flexibility in the selection of MAC layer protocols.

Under simulation, BOND significantly outperforms existing protocols for both connected and disconnected networks. In all scenarios, whenever nodes are mobile, BOND consistently has the lowest roundtrip times and the highest response rate of any protocol tested, with overhead consistently ranking among the lowest observed.

CHAPTER 1

Introduction

Much research has been devoted to **freeform** wireless networks, broadly defined as mobile, multi-hop wireless networks with unpredictable mobility patterns. This includes connected mobile ad hoc networks (MANETs) as well as disconnected networks such as vehicular ad hoc networks (VANETs) and opportunistic networks. To date, the fundamental goal of freeform wireless network research has primarily been data delivery between node pairs; the network should eventually be able to deliver data to a specific destination node no matter where it is.

Even in connected ad hoc networks, data delivery is difficult due to the high degree of geographical and topological dynamics. Not only can the destination move, but the network itself can be made of mobile nodes whose locations and interconnectivity may change at any time. This defeats conventional routing protocols for wired networks whose designs are based on the notion that links and nodes are stationary and dynamics are due only to link and node failures and recoveries.

Disconnected networks[1] further complicate the issue. In these networks, there is often no path between two nodes at any given time. However, over time, it is possible to use the physical mobility of the nodes as a means of data delivery

[1]Also known as partially connected networks, intermittently connected mobile networks, and opportunistic networks, among other names. They are part of the more general category of disruption/delay-tolerant networks (DTNs).

between otherwise disconnected parts of the network.

Years of research efforts have produced a rich literature of solutions to the above mentioned problems, however it seems there has still been no convergence on a set of standard solutions, much less a single, unified one for all freeform networks. One feature often seen in existing solutions is the use of the wired Internet protocol stack. Many solutions simply swap out wired MAC and physical layers for their wireless equivalents, complete with IP address assignment to each mobile node, data delivery based on destination IP address, hop-by-hop routing, and unicast support at the MAC layer.

The choice to use unicast transmission at the MAC layer is understandable. Unicast support is a highly desirable feature for the *single-hop* wireless networks that have become ubiquitous in the modern world. Perhaps as a result, modern MAC protocols have been (and continue to be) highly optimized for unicast transmission, providing the best performance in unicast mode. However, optimizations aside, wireless channels are fundamentally broadcast in nature. Wireless unicast adds an additional layer of complexity. Furthermore, unicast transmission is not necessarily a good choice for freeform networks. In a freeform network, the destination of a packet can be many hops away, and the set of intermediate nodes that actually overhear a transmission can change constantly. If *any* of these nodes are allowed to receive a transmission, it should be possible to make freeform network communications more efficient by reducing the total number of transmissions required.

This dissertation presents the Broadcast-Only Named Data (BOND) protocol, whose design departs from existing solutions in three important ways. First, BOND follows the basic approach of Named Data Networking (NDN) [JST09, ZEB10], a recently proposed architectural overhaul for the Internet. NDN as-

serts that data, not hosts, are the central element of modern networks. Second, BOND is designed to be efficient in connected networks (or connected portions of a disconnected network), while seamlessly supporting disconnected networks. This allows BOND to provide a single solution for *all* freeform networks. Third, BOND uses exclusively broadcast communication for all packets, allowing for more efficient use of the wireless channel.

CHAPTER 2

Background and Related Work

2.1 The IP Approach to Wireless Routing

Many wireless routing protocols follow the same basic, IP-based approach to routing:

1. Each node is assigned an IP address.
2. Applications communicate by sending data to specific destination addresses.
3. The routing protocol attempts to determine a single best path to the given destination IP address, and delivers data to that IP address via that single path.
4. To cross each hop along the determined path, the sending node controls which node should receive the data. That is, despite the fact that wireless channels are broadcast in nature, IP-routing-based designs impose point-to-point delivery across each hop.

This approach is inherited from the wired protocol architecture. It has proven to be an effective design in the wired setting, but is a poor fit for the wireless environment, as the remainder of this section will show.

2.1.1 Problems with IP Addressing

A number of problems with the IP approach to wireless routing relate to IP addressing (items 1 and 2 above).

First, assigning IP addresses to moving nodes is difficult. Because IP addresses are allocated by a central authority, IP address management is tightly controlled and requires infrastructure support, such as a central DHCP server. This is in direct conflict with the infrastructure-free nature of freeform networks.

Second, technological advances have brought us an ever increasing number of computing devices, most of which are now mobile. Managing IP address assignment for all of these devices is becoming increasingly infeasible and less meaningful. In wired networks, IP addresses represent topological locations which can be aggregated into blocks. Topological aggregation has allowed IP routing to scale to an enormous number of hosts. In a network where all of the nodes may potentially move, nodes do not have fixed topological locations, so IP address aggregation is not feasible. In fact, the best way to assign IP addresses to nodes in ad hoc networks has long been recognized as an open question [Vai01]. The only clearly identifiable reason for assigning IP addresses to nodes in freeform networks, other than as an arbitrary, temporarily unique identifier for the device, is compatibility with the existing protocol stack.

Third, all data communication is meant to serve the purpose of delivering data to application processes. It is the data itself that is most meaningful to applications. However, because data is invisible in today's IP-centric architecture, data must first be mapped to a specific node, resulting in sub-optimal data delivery. When a node N receives a packet p, it forwards p, then deletes it. Yet, p may be needed again, either for retransmission due to packet loss or for applications running on other nodes. Despite this, N cannot save p – the data it is tied to its

destination node. The data cannot stand alone.

Finally, in a mobile network, there is an inherent tradeoff between the accuracy of routing state maintained at each node and the overhead to keep this state consistent. Since data is delivered over a pre-determined path, the binding between an IP address and its topological location is critical. With either high node mobility or a large network (where even when individual nodes do not move much, there is constant movement in the aggregate), one suffers from either high overhead to keep these bindings updated and/or loss of connectivity due to outdated binding information.

2.1.2 Problems with IP Routing

The remaining problems with the IP approach to wireless routing relate to the routing process (items 3 and 4 above).

At each hop along the routing path, in order for the sender to determine the next hop, it must keep an updated neighbor list. Keeping senders informed of all of their neighbors' movement and connectivity changes will necessarily require significant control overhead. To make matters worse, as the speed of the nodes increases, either the amount of overhead must increase or routing accuracy will drop.

Furthermore, letting the sender determine which node will be the receiver does not utilize the broadcast nature of the wireless channel. On a broadcast channel, there are potentially multiple receiving nodes within the sender's signal range that can hear the data transmission. Since the nodes are mobile and wireless communication is inherently prone to interference, the sender cannot know for certain which nodes will receive any particular packet. Yet, if the next hop is named by the sender, there is only one node that is allowed to accept the packet,

no matter how many nodes may be within range and able to help get the packet to its destination.

Receivers, on the other hand, have all of the relevant information about a transmission. They know when they successfully receive a packet and do not need to maintain neighbor lists to know where a packet came from – the sender can simply place this information in a header. This puts receivers in a better position to make forwarding decisions than senders.

2.2 Related Work

Research efforts on mobile wireless networking started with the DARPA Packet Radio Network (PRNet) [JT87] in the mid-1970s and became a hot research topic in the 1990s and 2000s. Over the years, many solutions have been developed. Due to the bulk of literature on the subject, this section is not an exhaustive survey. Instead, this section touches on some major, relevant categories, providing a few prominent examples from each category.

2.2.1 Complete Applications of the IP Approach

The more mature solutions generally fall into one of a few broad categories: proactive, reactive, and hybrid proactive-reactive. Though each of these protocols has a number of unique characteristics, the all share the IP-based approach to routing described in Section 2.1.

Proactive protocols, such as DSDV [PB94] and WRP [MG96], are essentially a direct adaptation of routing protocols for wired networks. Like their wired counterparts, proactive ad-hoc routing protocols establish the best path to all reachable destinations and maintain consistent, up-to-date routing information

at all of the nodes.

Reactive routing protocols, such as DSR [JM96] and AODV [PR99], aim to address the high overhead of proactive routing approaches by establishing routing paths only when needed – that is, on an on-demand basis.

Hybrid proactive-reactive protocols, such as ZRP [HPS02] and HARP [NBN01], tend to use a hierarchical scheme, combining proactive routing at certain levels of the hierarchy with reactive routing at others.

Since they apply the full IP approach, protocols in these categories generally suffer from all of the problems described in Section 2.1.

2.2.2 Other Approaches to Addressing

A number of protocols do away with IP addressing in favor of an addressing scheme with greater topological meaning.

Geographical routing schemes, such as GPSR [KK00], assume each node knows its geographical location and uses this information to find routing paths. These protocols can be extremely efficient in networks where location information is available. As they do not rely on pre-determined paths or IP address assignment, they avoid most of the downfalls of the IP approach. However, as with IP addresses, geographical locations are not necessarily meaningful to applications. This means that geographical routing requires a mapping between nodes and geographical locations (for host-based communication) or data and geographical locations (for data-based communication). These mappings must be updated either whenever a node moves, whenever data moves between nodes, or both. This process has the potential to become extremely expensive when nodes are highly mobile.

Virtual Ring Routing (VRR) [CCN06] is another example of a protocol with a novel approach to addressing. VRR assigns each node a unique numerical identifier. Inspired by distributed hash tables (DHTs), each node only maintains a path to the other nodes in the network that happen to have been assigned the numerically closest identifiers. Though VRR does not suffer from the problems associated with IP address assignment (item 1 of the IP approach), it still requires packets to be addressed to a specific destination node and makes use of predetermined paths. This means that it suffers from the same problems associated with items 2, 3, and 4 of the IP approach.

2.2.3 Other Approaches to Routing

More recently, a number of *opportunistic* protocols have been developed [BM05, YYW05, CJK07]. These protocols have the goal of improving throughput in stationary mesh networks. Assuming that the links in the network are static, these protocols perform detailed analysis of the quality of each individual link. Information about individual link quality is generally distributed through a traditional (often link state) routing protocol. Using this information, these protocols take advantage of the broadcast nature of the wireless channel, postponing forwarding decisions until packets reach the receiver.

As a result, opportunistic protocols move away from items 3 and 4 of the IP approach described above. However, they still fully adhere to items 1 and 2 of the IP approach and therefore suffer from all of the associated shortcomings. Furthermore, these protocols fundamentally rely on the static topology of stationary mesh networks. Thus, they have no support for mobility.

Epidemic routing [VB00], Spray and Wait [SPR05], and PRoPHET [LDS04] are examples of routing schemes for disconnected networks that could be applied

to freeform networks – they assume node mobility is frequent and unpredictable. These schemes tend to follow the basic approach of spreading copies of data packets throughout the network until one of the copies reaches the intended destination.

Like opportunistic protocols, data-spreading protocols do not use pre-determined paths. However, their communication model is still fundamentally *node*-based, meaning they too suffer from the shortcomings associated with items 1 and 2 of the IP routing approach. Furthermore, these schemes become too inefficient or wasteful of network resources when used connected networks. In other words, disconnected network routing protocols do not provide efficient routing in a connected network.

2.2.4 IP-Approach-Free Solutions

In the realm of sensor networks, a number of existing solutions do away with the IP approach entirely. As a result, these solutions do not suffer from the associated shortcomings. However, when viewed from the perspective of freeform networks, each of these protocols has some limitations.

Directed Diffusion [IGE00], a framework for sensor network communication, takes an overall approach similar to the Named-Data Networking (NDN) approach described in Chapter 3. Both use "Interest" packets to solicit requested data and establish a path back to the requester. However, Directed Diffusion and NDN take entirely different approaches to naming data. In NDN, data names are hierarchical and can always be aggregated. Directed Diffusion does not require this, presenting an attribute-value pair naming scheme that can only be aggregated in an application-specific manner. This has one important repercussion – in the general case, each Interest must be flooded in Directed Diffusion.

As described in Chapter 4, BOND takes advantage of the hierarchical structure of NDN names to avoid flooding requests in many cases, making it efficient for exchanging larger amounts of data.

Furthermore, Directed Diffusion uses a neighbor-based forwarding scheme that reinforces particular paths. This forwarding scheme relies on the assumption that a node's neighbor set is relatively stable. BOND nodes, on the other hand, do not keep track of their neighbors. This design decision is meant to support the potentially high mobility of freeform networks, where a node's neighbor set can be constantly changing.

Gradient Broadcast (GRAB) [YCL01, YZL05] is another data retrieval protocol for sensor networks. GRAB uses a "cost field" that is similar to BOND's notion of distance (see Section 4.1.4). However, GRAB only updates this cost field through periodic flooding. GRAB also does not address the issue of collision avoidance or forwarder prioritization. Instead, all eligible nodes forward packets at each hop. GRAB uses a hop-count-based feedback scheme to reduce forwarding redundancy. Given that GRAB was designed for static sensor networks, it has no mechanism to deal with mobility.

CHAPTER 3

Named Data Networking

In recent work [JST09, ZEB10], Jacobson, Zhang, et al. have proposed *Named-Data Networking* (NDN), a new scheme to evolve the Internet protocol architecture. As shown in Figure 3.1, NDN replaces IP with *named data* as the "waist" in the Internet protocol hourglass. This means that data, not hosts, are the first class entity in an NDN network. Instead of sending data packets to a given destination IP address, hosts in an NDN network communicate by requesting desired data.

3.1 NDN in Brief

The equivalent of an "address" in NDN is a data name. NDN data names have a hierarchical structure composed of one or more arbitrary *components*. They

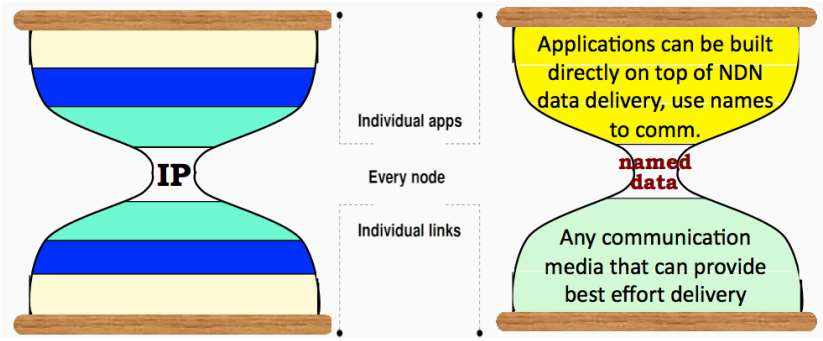

Figure 3.1: NDN shifts the focus of the Internet protocol stack

are generally written in human-readable form with the '/' character used as a delimiter. The hierarchical structure of data names allows for a notion of a data name *prefix* with the obvious definition: a data name a is a prefix of another data name b if and only if b is equal to a/c for some postfix c. For example, the data name `ucla/cs` is a prefix of `ucla/cs/home`, but not of `ucla/math` or `ucla/csdept`.

At a high level, applications in an NDN network communicate using the following three-way exchange. First, applications (running on nodes) that wish to make content available to others may announce the corresponding data names or prefixes of data names. These announcements are similar to IP routing announcements. Second, applications that are interested in a particular piece of content send out *Interest* packets which contains the names of the requested data. These Interest packets propagate along one or more paths towards potential locations of the data. Eventually, the Interest packets reach some node or nodes with the data. Third, the nodes that have the data send it back along the reverse path towards the requesting nodes. Along the way, intermediate forwarding nodes may cache the data packets to meet potential future requests for the same data. The first step may optionally be removed if Interest packets are flooded.

As Section 2.1 explained, in the traditional IP approach, a packet is intended solely for delivery to a specific destination address. Once a router forwards a packet, it is no longer meaningful and is immediately deleted from the router's memory. Named data, on the other hand, has a meaning independent from its consumers. Thus, NDN routers can cache a packet to transmit again later if they receive another matching Interest packet.

At first glance, it may seem like a burden to require each data packet to be associated with an application-level name. In fact, all data is *already* associated

with some high-level name at the application layer, these names are just not currently used in network-level data delivery. Instead, the current network requires a host to be associated with each packet, a binding which has become tenuous at best in modern networks.

3.2 NDN for Freeform Networks

In short, NDN changes the communication semantics from "where" to "what". This is particularly advantageous for freeform networks. Mobile nodes can communicate based on what data they need, instead of needing to find a path to a specific node. One no longer needs to assign IP addresses to each node; instead, applications running on mobile nodes can use data names directly to forward Interests and data packets amongst each other.

Generally speaking, application data has relatively stable structures compared to the dynamic topology of a freeform network. The IP address of a mobile node may be subject to change when its current location changes, however the application-level data names carried on the node do not necessarily change. In fact, geographical routing for mobile networks can be viewed as thinking along similar lines – routing using more stable and scalable names – except that it uses a fixed namespace with specific definitions. In some sense, it can be considered to be a special case of the more general NDN architecture.

However, freeform networks have some unique properties that have yet to be addressed by the existing NDN design. NDN expects each data packet to use the reverse path established by the corresponding Interest packet. That is, whatever path the Interest took to the responder, the data must take the same path back to the requester. In general, the highly dynamic nature of freeform networks

means that we cannot assume the return path will still be available when the data packet is sent.

Furthermore, disconnections can cause there to be no instantaneous path between a requester and any potential responder, including caches. As specified, an NDN requester cannot retrieve the data it wants unless a cached copy happens to reach the requester's locally connected partition by chance.

CHAPTER 4

Unifying Freeform Networks with BOND

The core contribution of this dissertation is the Broadcast-Only Named Data (BOND) protocol. BOND is a protocol for mobile wireless networks with the same fundamental philosophy as Named Data Networking (NDN): communication should be about data, not hosts. Instead of forwarding packets based on the destination *address* of a particular host (or node), data is forwarded based on a name that identifies the data itself. Thus, it is the data name that determines the communication endpoint. This means that the data in a data packet can stand alone, independent of the nodes involved in any particular transmission. Any node can cache data as it moves through the network and potentially become the endpoint of future requests for the same data.

However, the destination address is not the only part of the current network stack that is dependent on nodes' topological locations. The IP approach to routing dictates that, to communicate with an endpoint, we must first find a hop-by-hop path to that endpoint. Finding and maintaining a path to a particular endpoint in a network of moving nodes is already an expensive proposition, especially as the speed of node movement increases. Communicating using data names can actually compound this problem; the topological location of the data can change independently from the physical movement of the nodes. For example, a data name that is linked to a particular geographical location will have a constantly changing topological location as the nodes move through space.

Thus, BOND takes the notion of removing the dependence on topological locations a step further than the existing NDN design by doing away with explicit, pre-established routes and next hops. This change makes BOND an entirely name-based protocol – even during forwarding. As an added bonus, it also makes BOND significantly more effective in delivering packets under high mobility versus traditional wireless routing protocols. In fact, its performance is largely independent from the mobility levels in the network.

Yet, any protocol for *connected* wireless networks is still dependent on a node's temporal location. Networks of handheld or vehicle-bound devices are the most realistic scenarios for infrastructure-free, multi-hop wireless networks, such as in an urban or military setting. These types of networks are likely to be disconnected – some data exchanges can only occur when a node physically carries the data to another geographical location. However, large portions may still be dense, and therefore **locally connected** such that efficient performance in a connected setting is still desirable, if not necessary. Protocols for connected networks simply cannot handle disconnected scenarios, while protocols for disconnected networks can be extremely wasteful of network resources in the locally connected portions.

BOND also removes this final dependency by automatically detecting and handling communication between disconnected portions of the network. Node behavior in connected portions is unaffected. Removing this final dependency allows BOND to lump connected and disconnected networks together into a single category: **freeform networks**, which includes all mobile wireless networks, connected or disconnected, where node mobility is not predictable.

It is also worth noting that BOND uses exclusively broadcast transmission at the MAC layer and has its own mechanisms for collision avoidance. BOND does not (and cannot) use 802.11 unicast, ACK, or RTS/CTS support. Though

BOND cannot take advantage of the unicast optimizations in modern wireless MAC protocols, its simpler requirements can potentially allow it to work across a wider variety of wireless network technologies.

4.1 BOND Fundamentals

This section describes the basic operation of BOND at a high level, first for connected networks, then including disconnected network support. In this section, assume that the terms *sender* and *receiver* are only used in the context of single-hop transmission and reception on the wireless channel. These terms are not related to the communication endpoints.

4.1.1 The Big Picture

When an application running on a node requests a piece of named data, the node generates a request packet. BOND requesters generate a separate request packet for each data name. For each request packet, some node that has the requested data will generate a single response packet. Thus, each data name is assumed to fit in a single packet. Large files or streams must be broken up into multiple, sequential data names (see Section 4.1.3).

In order to deliver these requests and responses, nodes do not use any unicast packets nor maintain any neighbor lists. Instead, senders broadcast every packet and receivers decide whether to serve as forwarders. BOND attempts to use the best available path *on the fly*, hop-by-hop, while a data packet is moving through the network. Because it does not do any path selection ahead of time, it does not require any control packets.[1] Instead, all metadata required for forwarding is

[1]There is one exception to this: BOND has an ACK packet type that never contains application data, but it is not necessary for the correct operation of the protocol and is never

stored in the BOND header, which is added to every data packet. Data packets refresh the forwarding state at each node in transmission range.

4.1.2 BOND Names

The basic NDN design uses only data names to communicate. Requests for data are forwarded towards potential responders, leaving state behind at every hop along the way. This state can then be used to deliver data packets back to the requester. However, this approach is not reliable in a freeform network. High dynamics can invalidate the path used by the request before it can be used by the response. Furthermore, the network may be disconnected, providing no instantaneous path between requester and responder.

BOND solves this problem by defining two types of names: **data names** and **node names**. A node name is a unique identifier for each node that is used to deliver responses to that node's requests. To implement node names, BOND simply uses the unique name already present in a wireless node: its MAC address. Packets can be forwarded towards any name. Note that the name-to-location mapping is many-to-many; one location can have many names, and (with the exception of node names) one name may be found at many locations.

4.1.3 Name-Based Communication

The communication process always begins with a request. For requests, the destination is the data name being requested. At first, it is possible that the requester does not have any information about this data's location. In this case, the requester will have to flood the request. This will cause all the other reachable nodes in the network to forward a copy of the packet. When the request reaches

forwarded (see Section 4.2.2).

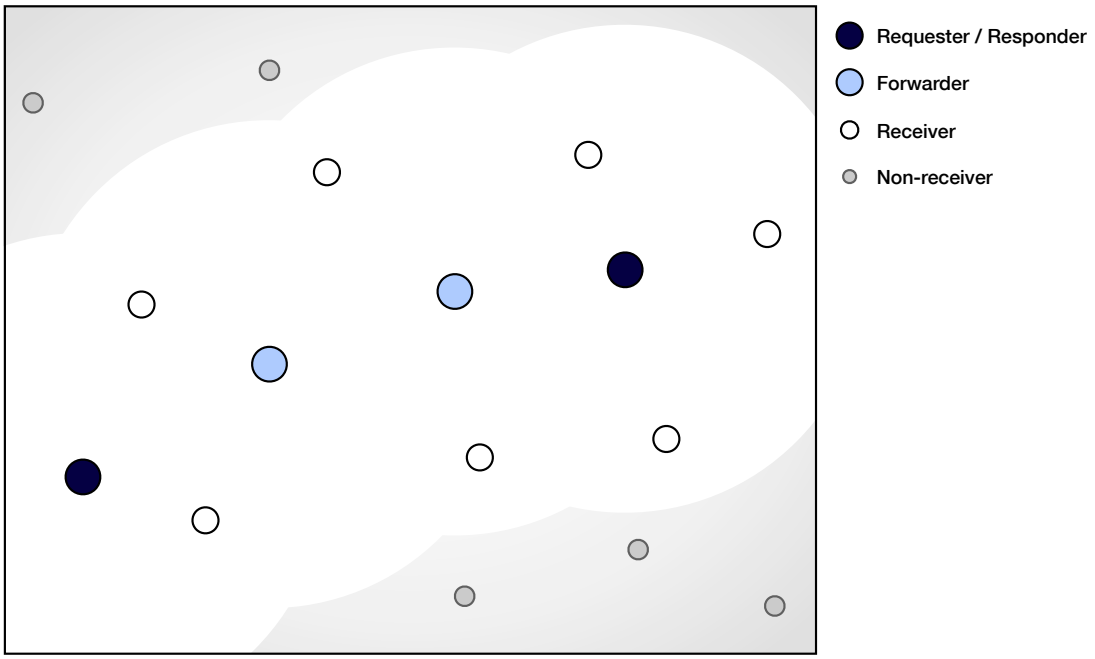

Figure 4.1: Many nodes may overhear transmitted data

a node that has the data named in the packet, that node will send a response. Using the information learned in the flooding phase, the intermediate nodes can forward the response back to the requester without flooding. Furthermore, any node in hearing distance will both (a) learn a location of the data name in the response and (b) cache the contained data so they may respond to future requests. For example, in Figure 4.1, only four nodes are actually transmitting data. Yet, all of the white nodes learn a location of the data name and can cache the data from the response.

However, there is an issue when the requester wants to request the next piece of data in the sequence. Recall that the data for each data name must fit in a single packet, so large files are broken up into multiple names, e.g., some file `data` is broken up into `data/1`, `data/2`, and so on. There must be some way for nodes to know that `data/2` is quite likely to be found at (or near) the same location as

`data/1`.

Luckily, NDN names actually are not flat but hierarchical, meaning they can be prefix matched. Thus, if a responder believes it has all of the data with a given prefix, it can indicate this in its response. Nodes can then record the location of the prefix, e.g., `data`, when receiving the response containing `data/1`. When looking for the location of a name, each node performs a longest prefix match. Thus, the request for `data/2` need not be flooded – nodes will find a prefix match (`data`) and forward the request directly to the location of the responder that provided `data/1`.

4.1.4 Forwarding

The BOND header in each data packet is used to maintain forwarding state. Forwarding state is updated at a node *every time* it receives *any* packet. This means that all nodes in transmission range of an active path (as in Figure 4.1) have up-to-date forwarding state and can potentially become forwarders at any time without any explicit coordination or path setup.

All forwarding decisions in BOND are carried out at the receiver side. Senders simply broadcast packets. Upon receipt, a receiver first decides whether it is an **eligible forwarder** for the packet. That is, whether it can help the packet make forward progress toward the destination name. If so, it waits for some amount of time, called its **listening period**, listening to the channel to see if another node "closer" to the destination forwards the packet. If not, the node forwards the packet itself.

Thus, a receiver has two important decisions to make: whether it is an eligible forwarder for a packet, and, if so, how long to wait before forwarding it.

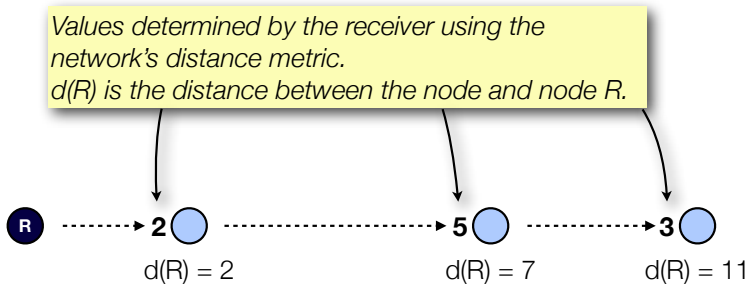

Figure 4.2: BOND distance metrics

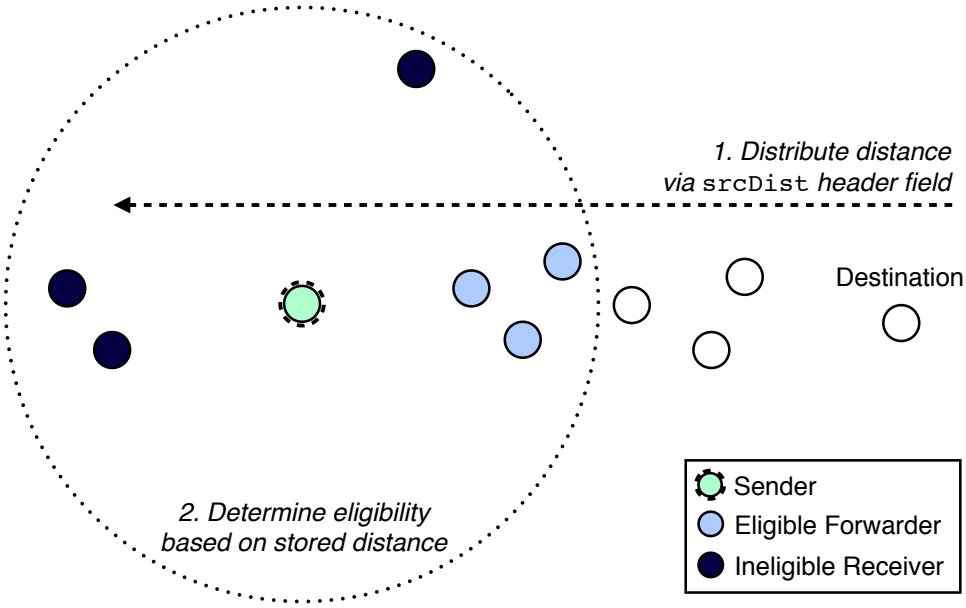

Figure 4.3: Eligible forwarders

4.1.4.1 Am I an Eligible Forwarder?

Since the goal of a forwarder is to move packets toward the destination name, a receiver is only an eligible forwarder if it is "closer" to the destination name than the sender. Nodes that are "further" from the destination name than the sender are called **ineligible forwarders**. The definitions of "closer" and "further" are determined by a **distance metric**. The distance metric is a measure of single-hop distance computed instantaneously *by the receiver* of a transmission. Our only assumptions about a distance metric are that it is (approximately) symmetrical, always positive, and that all nodes in the network use the same one. Some example distance metrics are geographical distance, the constant 1 (which will result in distances being hop counts), signal attenuation, and power consumption. The distance between a node and any location is the accumulation of distance metric values along each hop (see Figure 4.2).

The BOND header in every packet contains a `srcDist` field. At each hop, a forwarder inserts its own distance from the source name in the `srcDist` field before forwarding the packet. Nodes use the `srcDist` field to discover and update their distance from active names in the network.

Once a node knows its distance from a name, it can use this information when forwarding packets back *toward* that name. Specifically, the BOND header also contains a `dstDist` field where each forwarder places its distance to the destination name. This is the distance used by receivers to determine if they are eligible forwarders (see Figure 4.3).

4.1.4.2 How Long Should I Listen?

Since there will generally be more than one eligible forwarder for each transmission, before forwarding, each node chooses some amount of time to wait and see if other nodes perform the forwarding task. This time is called the node's listening period. It serves two important purposes: forwarder prioritization and collision avoidance.

Prioritization means assigning shorter listening periods to eligible forwarders that believe they are on a better path than their neighbors between the sender and the destination. Prioritization is determined by the network's **delay metric**. The simplest delay metric is a random value. In Section 4.3.3.2, we also define some more sophisticated, distance-based delay metrics that provide better performance.

Collision avoidance simply means reducing the chance that two eligible forwarders will choose the exact same duration for their listening period, transmit simultaneously, and cause a collision. Adding a random factor to the listening period (when the delay metric itself is non-random) is sufficient for this purpose.

4.1.4.3 Identifying Individual Packets

An eligible forwarder can only know if another node transmitted the same packet during its listening period if it has some way to identify individual packets. Otherwise, upon receiving a packet, a node will not know whether this is a new packet, a packet it is waiting to forward, or a packet it has already forwarded. A simple way to do this is to assign each packet a random nonce. Once a node forwards (or replies to) a packet with a given nonce, it will ignore any packet with that nonce for some configurable period of time (on the order of hundreds

of milliseconds).

4.1.5 Disconnected Networks

The basic property of disconnected networks is that packets may have to be "ferried" by the physical mobility of the nodes over time. Caching already gives BOND data ferrying – a node may cache some data, ferry it into a different part of the network, and then respond to a request for that data in its new portion of the network. However, ferrying cached data is not enough. If a node wants some data that resides in a disconnected portion of the network, it must either travel there itself, or wait for a node to arrive that happens to have that data in its cache. There is no guarantee either of these scenarios will ever occur, even if there are many nodes traveling back and forth between the two portions of the network.

To solve this problem, BOND takes a simple, brute-force approach: requesters may ask intermediate nodes to **replay** flooded request packets on their behalf. Requests are replayed by simply being retransmitted at some fixed interval up to some maximum number of times. The request's nonce remains unchanged, allowing nodes that are already replaying the request to identify it as a duplicate and ignore it. This allows any node to ferry requests between disconnected portions of the network. To allow BOND to operate in either a connected or a disconnected environment without requiring reconfiguration, each requester automatically detects whether replays are required (see Section 4.2.4).

4.2 Improving Efficiency

The protocol described in the previous section is functional. However, to truly have an effective and efficient protocol, BOND includes the improvements described in this section.

4.2.1 Preventing Duplicate Responses

With caching, it will be common for multiple responders to respond to the same request. If these responses all have different nonces, *all* copies of the same data will be forwarded back to the requester.

BOND solves this duplicate response problem by adjusting the scheme for assigning nonces. We need only ensure that each responder uses the same nonce. Thus, instead of each packet having a random nonce, only requests are assigned random nonces, and they must be *even*. Responders simply use the request's nonce plus 1. This way, intermediate nodes will only forward one copy of the response to any request.

4.2.2 Endpoint Acknowledgements

There is yet another way nonces can be used to reduce unnecessary transmissions. Responses can implicitly acknowledge the corresponding request. Recall that every response was elicited by a request. Upon receiving a response, a node knows the nonce of the corresponding request (the response's nonce minus 1). Obviously, if someone sent a response, the request has served its purpose, and the receiving node can ignore the request nonce. This reduces the number of extra copies of a request forwarded once a response has been sent.

However, when a requester receives a response, it does not naturally generate

any packet that can be used as an acknowledgement. When greeted with silence, not knowing the requester has already received the packet, other nearby nodes may forward the response again. To improve this situation, BOND includes an explicit acknowledgement packet (ACK). ACKs are sent by requesters when they receive a response. This stops response forwarding, preventing all of the eligible forwarders nearby from forwarding the packet. ACKs are never forwarded, they merely tell all nodes in hearing distance of the requester that the response was received. This serves the same purpose that overhearing the next forwarded copy of a packet would during the forwarding process.

Furthermore, when receiving an ACK, nodes can also ignore or cancel forwarding the corresponding request. This feature becomes important in disconnected networks, where the request may still be scheduled for replay by nodes that did not hear the response.

4.2.3 Preventing Dead-ends due to Caching

Nodes cache data opportunistically as it flows through the network. Therefore, a cache cannot guarantee that it overheard all of the packets in a given name prefix. For example, a node that cached `data/1` does not know if `data/2` exists or not. To ensure that requests are not delivered to a cache that does not actually have the data, only the originator of the data is allowed to specify a prefix that is shorter than the data name. In other words, if a node responds to a request with data from its cache, it must set the name prefix equal to the data name.

Whenever any node receives a response packet, it updates its distance to *both* the data name *and* the data prefix, if they differ. If nodes were to store distances using only the data prefix, longest prefix matching would cause caches to be preferred over the originator of the data.

Overall, this procedure ensures that a request for `data`/x (for some arbitrary x) is forwarded in a direction where `data`/x was *definitely* available at some point. The request will be flooded if no such direction is known.

4.2.4 Detecting Disconnected Networks

BOND uses two features to support disconnected networks. The first is caching by data name, which is already a fundamental part of BOND and need not be modified to support disconnected environments. The second is request replay, where all nodes that receive a flooded request will occasionally retransmit it on behalf of the original requester.

Unlike caching, request replay is *not* desirable in connected networks. BOND could simply require manual configuration to enable or disable request replays, depending on the type of network it is being deployed in. However, such a scheme (or lack thereof) for disconnected network "detection" has two problems. First, as previously mentioned, disconnected networks may be locally connected. BOND should be able to automatically handle these cases, avoiding replaying requests for data that can be found in the locally connected region. Second, on principle, a protocol meant to operate on both connected and disconnected networks should do so without any configuration changes.

BOND's basic detection method is by process of elimination – if a requester floods a request for a packet and gets no response after some number of retries, it assumes that the data must be in a disconnected portion of the network and enables request replay. Note that the "retries" performed by the initial requester are not the same thing as replays. Only the original requester performs these retires and it uses a different nonce each time.

However, this method has one major problem. Request replays could be

falsely triggered in a connected network that is highly congested. That is, the network is connected and the requested data is present, but the requester cannot retrieve it due to packet loss. In this case, enabling request replays would be a terrible idea; a large number of replays would make the problem much, much worse and almost certainly cause congestive collapse.

BOND requesters address this issue by checking the portion of time the channel was busy while waiting to retry a flooded request.[2] This portion of time is called the **busy ratio**. If a requesters finds the busy ratio to be above a configured threshold, it will not retry its request or enable replays.

4.3 BOND in Depth

This section describes the detailed operation of the BOND protocol. This section contains frequent references to the BOND header, a header placed in each data packet in lieu of separate control packets. The fields in the header, along with their descriptions, are listed in Table 4.1. Figure 4.5 summarizes the decision process taken by each node.

4.3.1 Node State

BOND nodes keep the following local data structures.

4.3.1.1 Distance Table

The BOND distance table is the primary data structure used in forwarding. It contains one entry per recently seen name or name prefix. More precisely, for

[2]Nodes already need to track when the channel is busy in order to pause send timers, see Section 4.3.3.2.

`nonce`	a random identifier for this packet
`srcDist`	The distance to the originator of this packet
`dstDist`	The expected distance to the destination name for this packet (for REQs, the `dataName`; for REPs, the `reqName`)
`FLOOD` flag	A flag indicating whether a REQ should be flooded (i.e., forwarded by all nodes)
`type`	The packet type; either REQ, REP, or ACK
`replaysLeft`	for REQs, the number of times this REQ should be replayed
`reqNameLen`	number of bytes in the `reqName`
`dataNameLen`	number of bytes in the `dataName`
`prefixLen`	for REPs, the number of bytes out of the `dataName` to be taken as the covering prefix to be used in distance tables
`reqName`	the node name of the requester
`dataName`	the data name being requested or provided

Table 4.1: Fields in the BOND header

some recently seen name n, a node stores a distance table entry containing exactly three values:

- D_n – the most recently observed distance from R to n
- V_n – the variance of this distance
- N_n – the random nonce of the packet used to update this distance

A node R may potentially update its distance table any time it overhears a packet being transmitted. Upon overhearing a packet transmitted by sender S, R first calculates its distance to the packet's source via S. This distance, $d_{src}(S)$ is simply computed as:

$$d_{src}(S) = \texttt{srcDist} + d_S$$

where `srcDist` is taken from the packet header and d_S is the instantaneous distance traversed by the packet from S to R. Though R does not know anything about S, d_S is provided by the network's distance metric (see Section 4.3.3.1).

Once R has calculated $d_{src}(S)$, it checks to see if it needs to update its distance table. The entries eligible to be updated depend on the packet type. For REQs, it is the `reqName` that names the source of the packet, so only the entry for `reqName` is eligible to be updated. For REPs, the entry for the packet's `dataName` is eligible, as well as the entry for the `dataName`'s prefix (if it differs from the `dataName`).

For each eligible name n, if R does not have an entry for n in its distance table, it simply adds one using the $d_{src}(S)$ and the `nonce` field in the received packet header. If R already has an entry for n, its behavior depends on the `nonce` field in the packet header. If the `nonce` field is equal to N_n (the nonce in its distance table), then R has already encountered this packet. R is interested in its *shortest* distance to n, so it only updates its distance table if $d_{src}(S)$ is less

than its current value for D_n. If the `nonce` field and N_n differ, R always updates its distance table, setting $D_n = d_{src}(S)$ and updating the corresponding nonce. R does this under the assumption that a packet with a new nonce carries a new, fresher distance measurement.

Whenever R updates a distance table entry with a new nonce, it also adjusts its third and final value: the distance variance V_n. Before replacing its distance table entry, it updates V_n using the same algorithm as is commonly used in TCP for estimating the variance in round-trip times [PA00]:

$$V_n = (1 - \beta)V_n + \beta|D_n - d_{src}(S)|$$

where β is a configurable weight for the rolling average, D_n is the distance value before the update, and $d_{src}(S)$ is the new value for D_n (assigned after V_n is computed).

Since distance table entries are meant to track only *active* endpoints and are expected to go stale eventually, they can expire. A distance table entry is erased after it has not been updated for a configurable amount of time.

4.3.1.2 Pending Send Index

Each node keeps track of any delayed sends in its pending send index. Pending sends are indexed by nonce, allowing a node to cancel any pending send if it hears another node send the packet first. When a pending send timer expires (without being canceled first), the node sends the packet and removes the timer from the index.

4.3.1.3 Nonce Blacklist

Whenever a node is done handling a particular packet (that is, whenever it reaches a "Stop processing" instruction in Figure 4.5), it adds the packet's nonce to its nonce blacklist. Incoming packets with a nonce found in the nonce blacklist are treated as duplicates and ignored. Entries in the nonce blacklist are timed out after the nonce has not been observed in the network for some configurable interval (on the order of hundreds of milliseconds).

4.3.1.4 Data Cache

A node stores all data from REPs that it overhears in its data cache. If a node has a particular piece of named data in its cache, it can generate a REP for any matching REQ. In order to understand the maximum potential benefits from caching, we do not consider cache replacement policies or limit cache size in any way.

4.3.2 Packet Types

The three packet types used in BOND are described below.

4.3.2.1 Requests

When an application running on node R requests a piece of named data n, R generates a request packet (REQ), placing n in the `dataName` field of the outgoing BOND header. If R has an entry for n in its distance table, it puts that distance in the `dstDist` field and forwards the packet normally, as discussed in Section 4.3.3. If R does not know its distance to n, it sets the `dstDist` field to infinity and sets the `FLOOD` flag in the BOND header, causing the REQ to be flooded.

Flooded REQs serve two purposes: ensuring that any available responder is found, and allowing other nodes in the network to learn their distance from R. This way, any node is ready to help forward the response if need be. In fact, this can allow the response packet to implicitly establish multiple, disjoint paths between the requester and the responder.

To avoid collisions and ensure accurate distance measurements during flooding, BOND uses a technique borrowed from Ye et al. [YCL01] where each node delays rebroadcasting the flooded packet for a time period relative to its distance from the neighbor that sent the packet. We make one minor modification to this algorithm: a node never rebroadcasts the same flooded packet twice, even if its distance improves. Though this may result in initially inaccurate distance measurements, it ensures minimal overhead for flooding. Any inaccuracies will quickly be corrected at any node that overhears the ensuing data exchange.

In order to support disconnected networks, requesters can put a non-zero value in the `replaysLeft` field of the BOND header of a flooded request to indicate that it should be replayed. Any node receiving the request will schedule `replaysLeft` retransmissions at (configurable) regular intervals. When replaying a REQ, a node updates the `srcDist` field with its current distance to the `reqName`. If it has no such entry in its distance table, it sets `srcDist` to infinity, indicating that the packet does not carry any distance information for `reqName`. All other fields in replayed requests are left unchanged, including the `reqName` and the `nonce`.

4.3.2.2 Responses

Once a responder receives the REQ, it produces a response packet (REP). In addition to the data corresponding to the requested data name, the REP contains the distance from the responder to the requester in the `dstDist` field of the

BOND header. This distance is used by intermediate nodes to make forwarding decisions that ultimately cause the REP to be forwarded back to the requester (see Section 4.3.3). Note that, as the REP travels, all forwarders and all of their neighbors that hear the packet will update their distance tables, learning their distance to the responder (see Section 4.3.1.1).

As a result, all of the nodes that forwarded or overheard the REP packet can serve as forwarders for future REQ packets, without the need for any more flooding. In a reasonably dense network, there is a high likelihood that this will make multiple paths available for the requester to reach the responder. Furthermore, as nodes move and bidirectional traffic continues to flow between the requester and the responder, new nodes that move into range of the current path overhear these transmissions. These new nodes also update their distance tables, providing a fresh crop of eligible forwarders.

4.3.2.3 Acknowledgements

Whenever a requester receives a REP, it generates an acknowledgement packet (ACK). ACKs are never forwarded, they merely serve to stop the REP forwarding process and cancel any REQ replays at nodes within hearing distance. The only relevant information in an ACK is the `nonce` field. ACKs have the same nonce as the REP they are acknowledging. Nodes that have already heard the REP do not need to process the ACK. Nodes that have not heard the REP cancel send timers and block nonces for the corresponding REQ and REP.

4.3.3 Forwarding in Depth

Once the distance tables in the network have been populated, normal, flood-free forwarding ensues. Receivers make forwarding decisions based on their distance

table and information in the BOND headers of received packets. Specifically, whenever a node forwards a packet, it updates the `srcDist` and `dstDist` fields in the BOND header from its distance table before transmitting the packet. Receiving nodes then compare these values to those in their own distance tables as described below.

4.3.3.1 Distance Metrics

As we briefly discussed in Section 4.1.4, an eligible forwarder is any node that both receives a transmission and is closer than the sender to the destination. Nodes determine whether they are closer to the destination by comparing the distance in their distance table to the `dstDist` field in the received packet.

This means that the distance metric is quite important. An overly optimistic distance metric – that is, one that causes too many receivers to assume that they are eligible forwarders – will cause unnecessary contention for the channel and thus extra overhead. An overly pessimistic or insufficiently granular distance metric will provide too few eligible forwarders, so that if none among this small set receive the packet, forward progress will stop and the packet will be lost. In Section 5.2.1, we evaluate different distance metrics for BOND.

Note that a node is never an eligible forwarder if it does not have an entry for a packet's destination in its distance table. In this case, the node will drop the packet.

4.3.3.2 Delay Metrics

As mentioned in Section 4.1.4, there are two reasons to set different listening periods across nodes. One is collision avoidance and the other is prioritization.

To avoid collisions, we introduce an element of randomness to a node's listening period. Additionally, we require that the MAC layer expose the state of the channel so that listening period timers can be paused when the channel is busy. This also decreases collisions and allows nodes with a clear channel to transmit first.

To prioritize closer nodes, BOND sets a threshold based on the distance from the best path. Despite the fact that BOND uses no explicit paths, a node can determine whether it is on the best path using just three pieces of information: (1) the distance provided by the sender in the `dstDist` field of the BOND header, (2) the node's distance d to the destination name according to its distance table, and (3) the length ℓ of the hop the packet just traversed, according to the network's distance metric.

The logic is as follows. If a node N is on the best path between the sender and the destination, the sender must have used N's distance to update its own distance table. In other words, the sender would have set its distance to d plus the distance from N to the sender. In a static network, if the network's distance metric is symmetric and stable, then d should be *exactly* $\texttt{dstDist} - \ell$. Obviously, it is not always the case that the network is static or the distance metric will be so reliable. However, we can reasonably assume that, the closer d is to $\texttt{dstDist} - \ell$, the closer this node is to the best path from the sender to the destination. Furthermore, if d is in fact less than or equal to $\texttt{dstDist} - \ell$, we assume that N is *on* the best path.

This information can be used to adjust the node's listening period in a number of ways. In Section 5.2.1, we compare three methods, or *delay metrics*, for prioritization of nodes' listening period:

- A purely *random* listening period, using the distance metric only to separate

eligible forwarders from ineligible ones.

- The *slotted random* metric, which divides eligible forwarders into two groups. The primary group is for those forwarders that appear to be on the best path ($d <= \texttt{dstDist} - \ell$). The rest of the eligible forwarders are placed in the secondary group. The primary group's time slot starts immediately upon reception of the packet, while the secondary group's time slot begins a fixed time period later. Within each slot, randomness is used for collision avoidance.

- The *distance + variance + random (DVR)* metric, where a node's listening period is based on its distance from the best path ($max(0,\ d - (\texttt{dstDist} - \ell))$) and the variance of its distance over time. The point of including the variance in this metric is to penalize nodes with an unstable distance value, favoring more stable paths. (The details of how the variance is calculated is explained in Section 4.3.1.1.) To these two values, we also add a random factor to break ties and avoid collisions.

Regardless of the delay metric used, a node R uses the listening period to listen for other eligible forwarders, to see if any of its neighbors forward the packet first. If R hears a neighbor forward a packet with the same `nonce` during its listening period (before it has forwarded the packet itself), it next examines the `dstDist` field. If the forwarder is closer to the destination than N, N cancels its pending forward and blacklists the `nonce`. However, if the forwarder is further away from the destination than N, N computes a new listening period and restarts its send timer, as it may still be needed to make forward progress toward the destination.

Note that both interference from other transmissions and the hidden terminal problem can result in multiple eligible forwarders forwarding a packet. On the

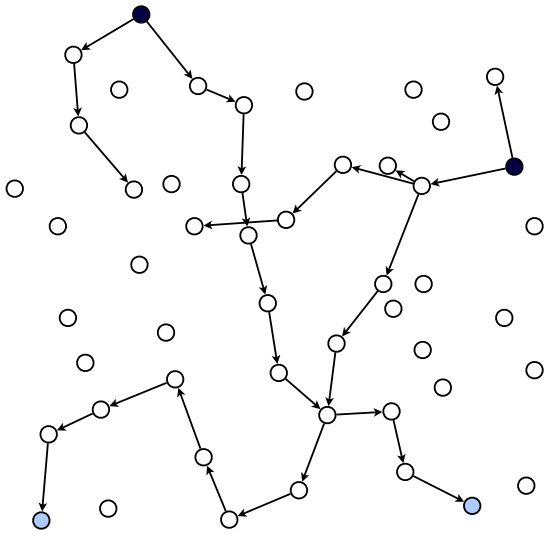

Figure 4.4: BOND forwarding paths in a connected network

plus side, this can result in the use of alternate paths, creating path diversity. On the other hand, it can create unnecessary extra transmissions. We can see an example of this effect at work in Figure 4.4. These are the actual paths traversed by BOND REP packets during a simulation of two simultaneous flows in a 50-node network. An arrow is drawn pointing from each forwarder to any other forwarder that received its transmission. In Chapter 5, we see that the plusses of path diversity and alternate path selection far outweigh any negatives.

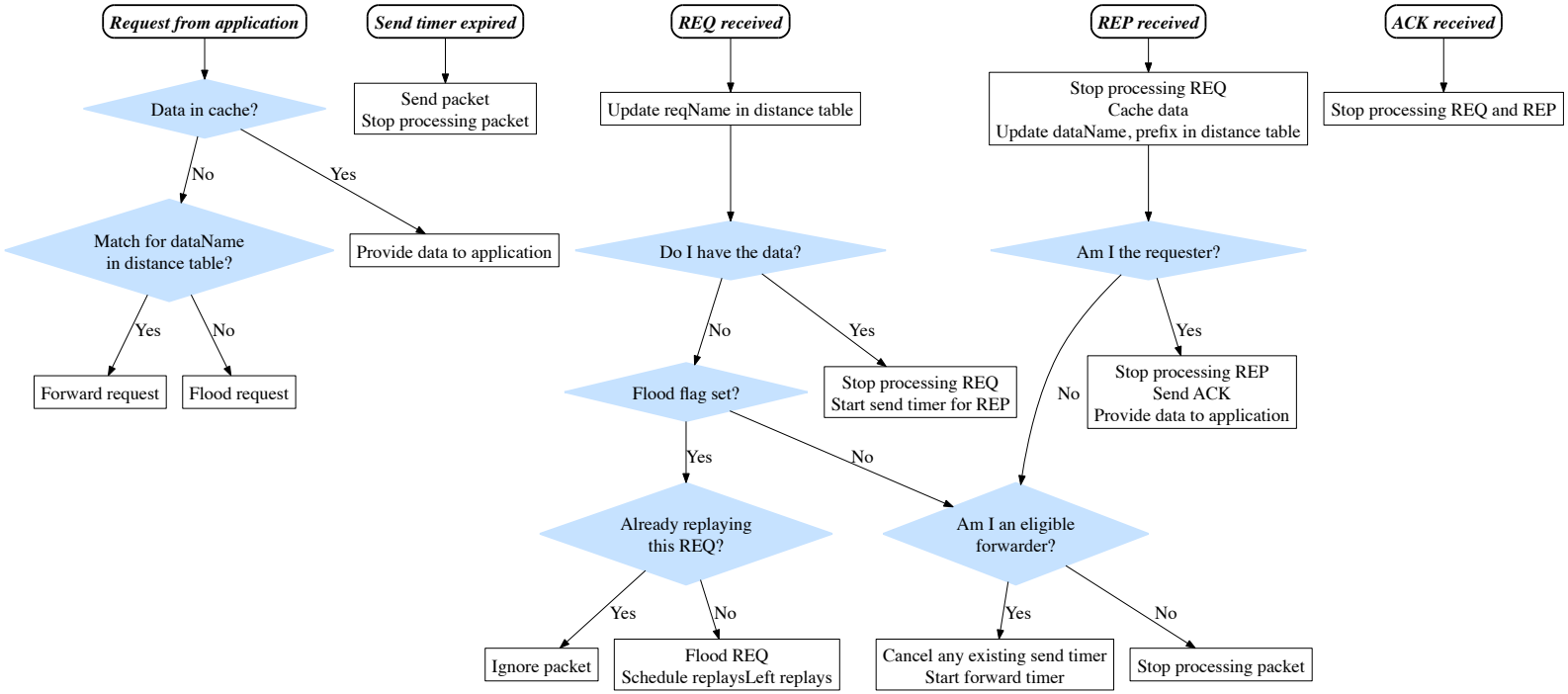

Figure 4.5: Flowchart for the BOND Protocol

4.4 BOND Walkthrough

The decision process that each nodes goes through when a given event occurs is shown in the flowchart in Figure 4.5. In order to illustrate how this decision process works in practice, let us go through a few detailed examples. At various points in the example, a reference is provided to the relevant entry point(s) in the flowchart (e.g., "REQ received").

4.4.1 Connected Data Retrieval

Imagine the following scenario. An application running on node N wants to retrieve a file called d. d is a fairly large file composed of multiple chunks named $d/0$, $d/1$, $d/2$, and so on. There is one node in the network, R, which is currently reachable from N and has the entirety of d. This is the first time d is being requested by any node in the network, so none of it has been cached by other nodes. The following steps show how the network delivers $d/0$ to N.

A1. The application starts by asking BOND to find $d/0$ ("Request from application"). This data is not in N's local data cache, nor does it have a matching entry in its distance table. Therefore, N will flood its request, putting its node name in the `reqName` field, $d/0$ in the `dataName` field, and setting the `FLOOD` flag.

A2. Some set of nodes overhears N's request – these are N's *instantaneous* neighbors. Upon hearing N's transmission ("REQ received"), each neighbor will first update their distance to N's node name. Next, assuming R is not among them, each neighbor will forward the flooded request. The neighbors will send the request only after some delay proportional to their *single-hop* distance from the sender (which in this case happens to be N). When send-

ing the packet, each node updates the `srcDist` header field to reflect its own distance from N. One other detail not mentioned in the flowchart is that each neighbor will immediately add the REQ's `nonce` to its nonce blacklist, to prevent flooding the same packet a second time.

A3. The flooding process continues in this fashion until the request reaches R ("REQ received"). As with other receivers, R first updates its distance to N's node name. However, since R has the requested data, it does not flood N's request. Instead, it adds the REQ's `nonce` to its nonce blacklist and generates a response packet containing data $d/0$. In the header, R places its own distance to N's node name in the `dstDist` field and zero in the `srcDist` field. It sets `prefixLen` to the length of d, indicating that it has all of the data with names that begin with d, not just $d/0$.

A4. Upon receiving R's response ("REP received"), all of R's instantaneous neighbors do four things: (a) add the nonce of the REQ (the REP's `nonce` minus 1) to their nonce blacklist, (b) cancel any pending send timer that would send the REQ, (c) cache $d/0$, and (d) update their distance to both the full `dataName` ($d/0$) and the prefix of `dataName` given by the `prefixLen` (d). Nodes update their distance tables using the method described in Section 4.3.1.1.

Next, assuming none of the neighbors are N, each neighbor must determine if it is an eligible forwarder, as described in Section 4.3.3. Likely having flooded (or heard another node flood) N's request, each of these neighbors should know their distance to N's node name. Neighbors that are *not* eligible forwarders immediately add the REP's `nonce` to their nonce blacklist. Neighbors that *are* eligible forwarders start a send timer to forward the REP after their computed listening period (see Section 4.3.3).

A5. Shortly thereafter, the instantaneous neighbor of R that has chosen the shortest listening period, let us call it S, will add the REP's `nonce` to its nonce blacklist ("Send timer expired") and forward the REP. S will place its own distance to $d/0$ in the `srcDist` field and its own distance to N's node name in the `dstDist` field. The instantaneous neighbors of S overhear this transmission ("REP received"). Note that, since R and S are in transmission range of each other, S's instantaneous neighbors should have a large overlap with the instantaneous neighbors of R in step A4. Each neighbor performs the same steps as the neighbors of R step A4. However, the `srcDist` and `dstDist` in the header have changed, so nodes that also heard R's transmission in step A4 may make different decisions here. Specifically, if a node was eligible to forward R's REP but not S's, it will cancel its send timer and add the REP's `nonce` to its nonce blacklist. If a node was eligible to forward both packets, it resets its send timer using a newly computed listening period.

A6. The forwarding process continues in this fashion until the response reaches N ("REP received"). N performs the same initial procedure as other nodes that overhear REPs in steps A4 and A5. Next, however, instead of determining if it is an eligible forwarder, it accepts the packet by adding the REP's `nonce` to its nonce blacklist and sending an ACK.

A7. N's ACK is received by its instantaneous neighbors ("ACK received"), who blacklist the corresponding REQ and REP nonces and cancel any pending send timers for the REP.

Suppose that, at this point (after it has received $d/0$), the same application running on N wishes to retrieve $d/1$. The following steps show how the retrieval proceeds.

B1. N receives the request from the application ("Request from application") and finds a prefix match for $d/1$ in its distance table (the distance to d added in step A6). N sends a REQ with $d/1$ in the `dataName` field, its node name in the `reqName` field, and its distance to d in the `dstDist` field. It does not set the `FLOOD` flag.

B2. Just as R and S's neighbors did in steps A4 and A5, respectively, N's instantaneous neighbors use the BOND header and their internal send timers to implicitly coordinate in forwarding the packet towards d ("REQ received", "Send timer expired"). Note that this REQ does not have to be forwarded by the same nodes that forwarded R's REP. Any node that overheard R's REP will know its distance to d and therefore may participate in forwarding N's REQ. Further note that all nodes overhearing this REQ can refresh their distance to N's node name.

B3. If little time has passed since R sent its previous REP, it will be near the same location and should receive N's REQ for $d/1$ (assume it does, "REQ received").

B4. R's REP is forwarded back to N using the standard forwarding procedure, as with the previous REP ("REP received"). Note that this also allows any nodes overhearing the REP to both (a) update their distance to d and $d/1$ and (b) cache $d/1$.

These steps can be repeated until N is able to retrieve d in its entirety. If the time gap between N's requests is too large relative to the mobility speed of the nodes in the network, it is possible that the forwarding state will become stale and N's requests will not reach any node with the requested data (in this

case, only R). If this occurs, the distance table entries for d in each node will eventually timeout, and N simply falls back to step A1.

4.4.2 Caching

Let us now discuss how BOND handles a request for data that has been cached. Suppose that, immediately after the example in Section 4.4.1 concludes, an application running on another node, M, wishes to retrieve $d/0$. Further suppose that M did not overhear the exchange between N and R, so it does not know any distance to d or $d/0$, nor does it have $d/0$ in its cache. The following steps ensue.

C1. M generates a flooded request (a REQ with the `FLOOD` flag set) for $d/0$ ("Request from application").

C2. Through the normal flooding procedure ("REQ received"), M's request eventually reaches *three* nodes that had cached $d/0$ during the previous exchange. Let us call these nodes X, Y, and Z.

C3. X, Y, and Z all generate a REP for M's REQ ("REQ received"). Recall that the `nonce` in a REP packet is always equal to the REQ's `nonce` plus one. As in the forwarding process, each node will only send their REP after waiting some time to avoid collisions and ensure no other node handles the request first. This delay is chosen in the same way as the flooding delay. (See section 4.3.3.)

Note that X, Y, and Z do one thing differently from R: they set `prefixLen` equal to `dataNameLen`, meaning the prefix is equal to the whole data name; $d/0$, *not* d. This is because X, Y, and Z cannot guarantee that they overheard all of the packets in d.

C4. Suppose X and Y are within transmission range of one another and X's send timer expires first ("Send timer expired"). Y will overhear X's transmission ("REP received"). If Y is an eligible forwarder for this packet, it will replace its REP send timer with a forwarding timer for Y's response. Otherwise, it will simply cancel its REP send timer and add the REP's `nonce` to its blacklist.

C5. X and Z's responses are both forwarded towards M. If any intermediate node between M and the two responders overhears both packets, it will only forward one copy. For example, suppose some intermediate node P first receives and forwards X's response, then receives Z's response ("REP received"). After forwarding X's response, P will add the REP's `nonce` to its nonce blacklist and ignore Z's REP (which has the same `nonce`) when it receives it. In fact, the same is true for any node that merely overhears X's response but is not an eligible forwarder.

C6. X's response eventually reaches M. M will only receive Z's (duplicate) response as well if there is a completely disjoint path between Z and M where none of the nodes are in hearing range of the path used by X's REP. Note that M did not know ahead of time which node would respond, nor does it now know which node was the responder.

4.4.3 Disconnected Data Retrieval

Now imagine a somewhat different scenario. An application on node N is once again interested in some data d. However, this time, N is in a disconnected network which is partitioned in two. R, the only node that currently has any of the data from d, is in the other half of the network. That is to say, R is not

currently reachable from N. Of course, at first, N is not aware of this fact, it merely wishes to retrieve d, starting with $d/0$. The procedure is as follows.

D1. N generates a flooded request for $d/0$, as in step A1.

D2. Though N's request is flooded to all of the nodes in its half of the network (as in step A2), none of them have the data, so N does not receive a response.

D3. After some configurable amount of time, N times out its request. Recall that nodes are notified by the MAC layer when the channel is busy. At this point, N checks its busy ratio for the time while it was waiting for a response. The busy ratio is the portion of time that the channel was busy during this period. If the busy ratio is high (above a configurable threshold), N assumes that the lack of response was due to congestion rather than a disconnected network. In this case, it would not retransmit. However, let us assume that the busy ratio is below the configured threshold. Thus, N does retransmit its request and steps D1 and D2 repeat. N repeats *this* step until it reaches its maximum number of retransmissions (also configurable). (This step is not shown in the flowchart.)

D4. N reaches its maximum number of REQ retransmissions, it has not received a response, and its busy ratio is low. Therefore, N assumes it is in a disconnected network and enables REQ replay. It does this by simply placing a number greater than zero in the `replaysLeft` field of the BOND header (this number is configurable as well) and flooding its request one more time.

D5. Upon receiving N's flooded request, each node in N's half of the network first performs its usual procedure for flooded requests ("REQ received"). However, now that the `replaysLeft` field is greater than zero, they also schedule this request to be replayed.

D6. Each node in N's half of the network sends another copy of N's request at a regular (configurable) interval. Every time a node replays N's request, it decreases `replaysLeft` by one and updates the `srcDist` field in the header with its current distance to N's node name. However, it does not change the packet's `nonce`. If its distance to N's node name has timed out, it sets `srcDist` to infinity.

D7. If some node P in N's half of the network overhears another node replay N's REQ ("REQ received"), P recognizes (using the packet's `nonce`) that it is already replaying this REQ itself, and ignores the packet. Note that P could, in theory, have the REQ's `nonce` in its blacklist and ignore it for that reason. However, an explicit check for replayed REQs is necessary because the timeout for blacklisted nonces is expected to be very short compared to the request replay interval.

D8. Eventually, some mobile node M from N's half of the network travels to R's half of the network. Let us assume it does so before it decrements `replaysLeft` to zero. M then replays N's request. The nodes in R's portion of the network, not having heard N's request before, and therefore not yet replaying it, flood the REQ and schedule `replaysLeft` replays ("REQ received"). Note that the `replaysLeft` value they receive has been decremented by M, so the total amount of time over which N's REQ will be replayed does not increase.

D9. R receives the replayed copy of N's request and responds with a REP, as in step A3. In this case, however, it is likely that M's distance table entry for N's node name has timed out, and the `srcDist` in the replayed packet is infinity. Let us assume this is the case. This causes R to set the `dstDist` in its response to infinity.

D10. All of R's instantaneous neighbors cache $d/0$, update their distance to d and $d/0$, and cancel the replay of N's REQ ("REP received"). Note that, whenever a node "stops processing" a REQ, it cancels all replays of that REQ as well. Since it is likely that none of R's instantaneous neighbors know their distance to N, they will not consider themselves eligible forwarders and will drop the REP. However, they have already cached the data.

D11. Other nodes in R's half of the network will continue to replay N's REQ, causing nodes that have cached $d/0$ to respond, just as X, Y, and Z did in step C3. Eventually, all of the nodes in R's half of the network will either (a) decrement `replaysLeft` to zero and stop replaying N's REQ or (b) cache $d/0$.

D12. After some time, suppose a mobile node S travels from R's half of the network to N's half. Further suppose that nodes in N's half of the network are still replaying N's REQ. When S hears one of these replayed REQs, since it has $d/0$ in its cache, it will respond ("REQ received").

D13. Since the `dstDist` in S's response is likely to be infinity (suppose it is), any node with *any* known, finite distance to N's node name will consider itself an eligible forwarder (any distance will be less than infinity) ("REP received"). This means that S's response can be forwarded directly back to N using the standard forwarding procedure whenever any of S's instantaneous neighbors know their distance to N's node name. If not, the data will propagate back to N (along with the other nodes in N's half of the network) in the same way it did in R's half of the network in step D11.

CHAPTER 5

Evaluation

This chapter presents a variety of simulation results for the purpose of validating and evaluating BOND's performance.

5.1 Simulation Setup

To obtain the simulation results presented in this chapter, the author implemented BOND in the QualNet network simulator[1]. All BOND simulations were run with the *exact* same protocol parameters, regardless of whether the network was connected or disconnected.

At the physical layer, all simulations use 802.11b radios operating at a fixed rate of 11 MBps. Other physical layer parameters were left at the (sensible) default values used by QualNet. We used two different MAC protocols: a simple Carrier Sense Multiple Access (CSMA) MAC, and the full 802.11 MAC. The CSMA MAC simply senses the channel, sending if it is free, or backing off for a random interval if it is busy. It uses no retransmissions, acknowledgements, RTS/CTS, etc. For BOND, only CSMA MAC results are shown; since BOND uses only broadcast transmissions, 802.11 MAC performance is similar. For other protocols, only 802.11 MAC results are shown; they perform significantly worse with the CSMA MAC.

[1] http://www.scalable-networks.com/

At the application layer, all request packets contain only a header, while all response packets contain 1400 bytes of data.

Each simulation ran four separate times with different random seeds. The values presented in the figures are the median of the four means from the different simulation runs. Error bars display the interquartile range of the four means.

5.1.1 Comparison Protocols

To evaluate the performance of BOND, this chapter contains a number of comparisons between the performance of BOND and that of other protocols. In order to maintain the utmost level of fairness, every attempt was made to ensure that the protocols were configured similarly. That is to say, the differences in the performance between the protocols cannot be accounted for by configuration parameters alone. Additionally, just as with BOND, the comparison protocol parameters were tuned for the best performance based on extensive testing.

To evaluate the performance of BOND in connected networks, results for three comparison protocols are presented. All three protocols use the same basic, IP-approach-based procedure: use controlled flooding to discover a path from source node to destination node and route all packets along the discovered path until it breaks.

The first of these protocols is the **Dynamic Source Routing (DSR)** [JM96]. In DSR, as the name implies, the source of a packet determines its route to the destination. The source discovers a route during the flooding phase, then places that route in the header of outgoing data packets. If the exact path can no longer be used, the source must discover a new route.

The next protocol is **Ad-hoc On-Demand Distance Vector (AODV)** routing [PR99]. Instead of the source placing the entire route in the packet header,

AODV uses a distance vector approach, where each node on the path learns only the immediate next hop used to reach a particular destination. Unlike DSR, the intermediate nodes in AODV know their path to the destination. Therefore, if a path is broken, the source need only find a new path to *some* node that knows its path to the destination.

The final comparison protocol for connected networks is the **Dynamic MANET On-Demand (DYMO)** routing protocol [CP10]. DYMO is the successor to AODV. It includes some incremental optimizations and improvements, including the ability to improve existing routes when new routing information is available. This is the most current of the three protocols – at the time of writing, DYMO is still under active consideration by the IETF MANET working group.

For disconnected networks, two comparison protocols are used. Both use node-based destination addressing, relying on replicating packets opportunistically until the packet reaches the destination. They are representative of the class of routing protocols for disconnected networks where node mobility is not predictable.

The first disconnected network comparison protocol is **Epidemic Routing** [VB00]. In Epidemic Routing, each node tracks its neighbors using hello packets. When a source wants to send a message, it sends a copy of the message to each of its neighbors (one at a time). Each neighbor stores the message in its local buffer. Whenever some node A encounters some node B, A sends B all of the messages it has that B doesn't have, and vice versa. This process continues until each message is replicated to all nodes or reaches a configurable timeout.

The second disconnected network comparison protocol is **Spray and Wait** [SPR05], which is essentially a limited version of Epidemic Routing. In Spray and Wait, rather than replicating messages to all nodes, the source creates a

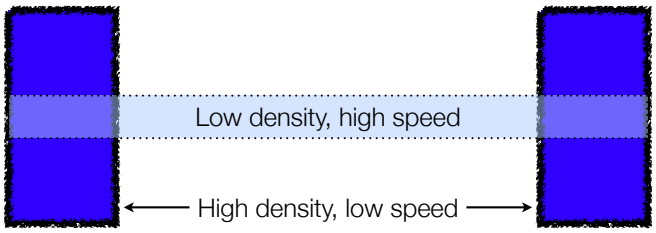

Figure 5.1: The "bridge" mobility model

fixed number of copies of the message (termed L). Starting from the source, any node that has more than one copy of a message will give half of those copies to any node it encounters. When a node has only one copy, it will not send the message to anyone except the destination node (if it ever comes within range of the destination node). The goal of this scheme is to reduce the overhead of Epidemic Routing at the cost of increased latency. Spray and Wait is designed for networks where all nodes are equally likely to encounter each other. In these types of networks, the reduced number of replications does not affect the delivery ratio.

AODV, DYMO, and DSR simulations were run using QualNet's built-in implementation of these protocols. Epidemic Routing and Spray and Wait simulations were run using an implementation graciously provided by colleagues who have previously analyzed these protocols using QualNet [WSJ09b, WSJ09a].

5.1.2 Mobility Models

All connected network scenarios use the steady-state random trip model [NC04], a random waypoint variant that is not sensitive to initial node placement. These scenarios use 100 nodes placed in a 1500-by-1500 meter area. The speed and percentage of mobile nodes vary across scenarios. Each trace is 20 minutes in duration.

For disconnected scenarios, two mobility models are used. The first is the standard Manhattan mobility model, which simulates vehicles traveling on a regular grid of roads. 100 nodes travel in a 3000-by-3000 meter grid that is 20 blocks wide. Each node travels at speed between 10 and 20 meters per second. Each trace is 30 minutes in duration.

The second disconnected mobility model is a contrived scenario designed specifically for validating BOND. In this model, let us call it the **bridge model** (see Figure 5.1), two dense areas of relatively low-speed nodes are separated by a large gap. A small, low-density group of high-speed nodes travels along a narrow "bridge" between the two areas. The density of nodes on the bridge is too low to ever create an instantaneous connection between the two outer areas. Thus, this scenario contains two locally connected networks that can only exchange data when it is ferried across the gap by the nodes on the bridge. Within each area, nodes use the random waypoint mobility model.

For the bridge-based simulations in this chapter, the outer areas are 500-by-1000 meters and contain 45 nodes. The nodes in the outer areas move at speeds between 5 and 15 meters per second. The bridge area is 3000-by-200 meters and fully overlaps the outer areas. There are 10 nodes on the bridge, each of which moves at speeds between 20 and 40 meters per second.

Random waypoint and Manhattan mobility traces were generated using the Generic Mobility Simulation Framework (GMSF) [BLS08].

5.1.3 Metrics

We use four metrics to evaluate the outcome of our simulations:

- **Roundtrip time (RTT):** the average amount of time elapsed from when

a request is sent by a requester until it receives a response.

- **Response ratio:** the total number of responses received by all requesters divided by the total number of requests sent. Since the response ratio is computed only based on requesters, this is a roundtrip metric.

- **Path length:** the average number of *roundtrip* hops taken by successfully received data packets. Note that this differs from the usual end-to-end definition. Furthermore, this metric measures the number of hops each *data* packet actually traversed, not the length of the paths chosen by the routing protocol. This definition of path length is necessary since neither BOND nor the compared disconnected network routing protocols pre-establish paths.

- **Overhead:** a measure of protocol overhead expressed as a byte ratio. This ratio is defined as B_{total}/B_{recvd}. B_{total} is the total number of bytes sent to the MAC layer for transmission by *all* nodes. This includes all bytes – headers, control packets, and data. B_{recvd} is the total number of non-duplicate data bytes received by all requesters (excluding headers).

The above definition of overhead requires some further explanation. Typical overhead metrics would only include control packets, meaning data packets would not factor into the overhead computation at all. Such a metric cannot be used for BOND, since there are no control packets. Any overhead in BOND is the result of two factors: (1) nodes that choose to forward a copy of a packet unnecessarily due to collisions or the presence of extraneous, divergent paths, and (2) the extra bytes used by the BOND header included in each packet. The inclusion of data packets in the computation takes both of these factors into account.

Furthermore, the given definition of B_{recvd} (the denominator in the overhead ratio) does not include the bytes transmitted along the forwarding path. This

means that, all else being equal, a protocol that chooses longer paths will have higher overhead. This may seem to be a flaw in the metric. On the contrary, this is an intentional choice. To understand the reasoning behind this choice, consider the alternative.

Suppose we were to redefine the overhead ratio as B_{total}/B_{path}, where B_{path} is the total number of (non-header) data bytes transmitted along the entire roundtrip path for successfully received responses. Under this definition, whatever path a protocol chooses, the overhead ratio only measures the number of bytes used to set up the path, maintain forwarding state, and/or try alternative paths. But consider what an overhead ratio of one, the absolute theoretical minimum, would mean using this metric. It would mean that the protocol transmitted *only* application data, and only along a single path. What highly efficient protocol could earn such a distinction? The answer, surprisingly, is random walk. That is, a protocol that simply forwards a single copy of each packet to a single, random neighbor until the packet happens to reach its destination. This protocol doesn't use any control packets. Every packet forwarded is on the protocols chosen path between source and destination. As long as every packet eventually reaches its destination, the overhead of this protocol would be precisely one. Clearly, random walk is not the most efficient use of network resources – this alternative definition is a poor metric to judge overhead.

The above thought exercise is particularly relevant in disconnected networks. Paths used by replication-based protocols, such as Epidemic Routing, are essentially found by random walk. This alternative metric would assign such protocols an erroneously low overhead. The definition of overhead used in this chapter, on the other hand, takes into account the number of replications made by a replication-based protocol, considering all of them to be overhead.

5.1.4 Traffic Model

In order to fairly compare BOND's name-based forwarding with the host-based system of the comparison protocols, we need a traffic model that can be applied to both. The traffic model used in this chapter is based on the concept of bidirectional flows. Each flow is composed of a single requester paired with a single responder. The requester sends a single request packet at a regular interval. The responder responds with a single data packet to any request it receives from the paired requester.

For host-based protocols, this model is implemented by applications running in a client-server configuration, where clients are the requesters and servers are the responders. In BOND, each responder originates a different name prefix. Requesters request only names from the prefix corresponding to their paired responder.

Furthermore, to prevent BOND from gaining any unfair advantage due to caching effects, the same data name is never requested twice. Instead, each request is for a different data name from the same prefix. For example, requester R's first request is for the name `a/0`, its second is for `a/1`, and so on. Only the responder paired with R will respond to any request in the prefix `a`. Similarly, requester S sends requests for `b/0`, `b/1`, etc. and only the responder paired with S responds to requests in the name prefix `b`.

Using this model, the overall data rate in a simulation is set based on two parameters: the total number of bidirectional flows and the rate at which requests are generated. Under testing, performance of all protocols was not highly sensitive to the ratio of these two values. In other words, the most important factor in setting these parameters is their product – the overall traffic rate. For most simulations, both parameters are fixed at an intermediate value that provides a

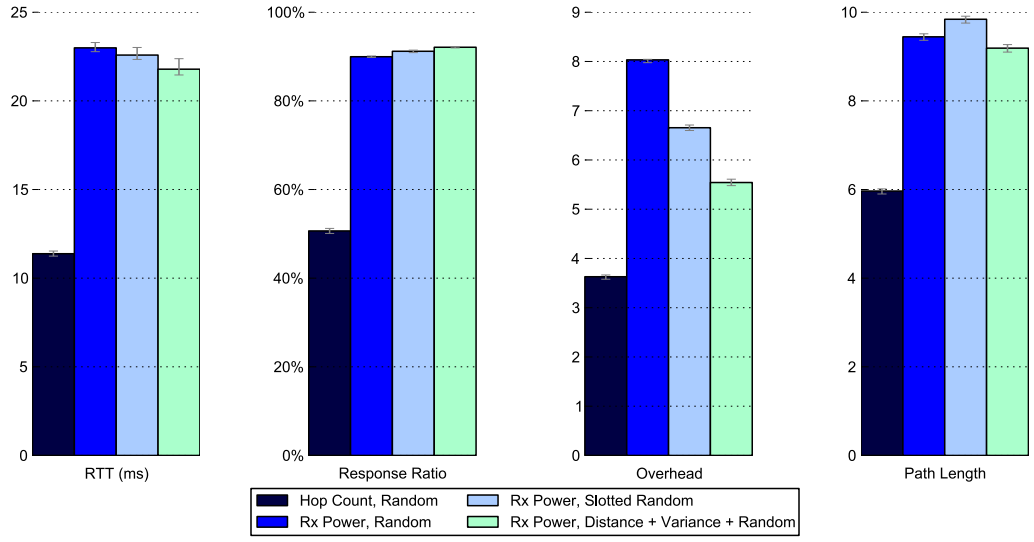

Figure 5.2: A comparison of different distance and delay metrics

reasonable amount of traffic without overburdening the network.

5.2 BOND Parameters

This section presents results that evaluate different choices for the various parameters BOND uses.

5.2.1 Distance and Delay Metrics

Though BOND does not depend on any particular distance metric, the choice of distance metric can significantly affect its performance. In these simulations, all 100 nodes move using a random waypoint model with no pause time and speeds between 10 and 30 meters per second. There are eight simultaneous, independent flows where the same data name is never requested twice.

The leftmost two bars in each group in Figure 5.2 compare BOND's perfor-

mance with two different distance metrics: hop count and received signal strength. The received signal strength metric is far more effective at delivering packets than the hop count metric, though this comes at the expense of greater delay and overhead, though the raw delay numbers (less than 25 ms) are still quite low.

Figure 5.2 also shows the effects of different choices of delay metrics. Delay metrics determine the length of time an eligible forwarder will wait before forwarding a packet. We evaluate three different delay metrics in Figure 5.2: random, slotted random, and distance + variance + random (DVR). See Section 4.3.3 for descriptions of these metrics. For the random delay metric, each eligible forwarder simply selects a random delay between zero and 5 milliseconds (ms). For the slotted random delay metric, nodes select one of two, 2.5-ms slots based on the distance metric. Within each slot, the node selects a random delay between zero and 2.5 ms. For the DVR delay metric, the node assigns itself a delay penalty based on its distance metric and its computed variance. It then adds a small random factor to break ties. The ratio of distance + variance to delay was chosen in such a way to give a median delay similar to the other metrics.

Each of these three metrics has a slightly higher response ratio than the last with significantly reduced overhead. As expected, the DVR metric has the lowest overhead since it favors more stable paths, resulting in fewer dead ends. Due to its superior performance, all other plots in this section use DVR as the delay metric.

Note that the choice of received signal strength as the distance metric results in longer paths since it prioritizes hops between nodes that have a stronger signal, and are therefore likely to be closer together geographically. This is the reason that BOND tends to have longer average path lengths than the comparison connected network protocols. When using hop count as the distance metric, BOND

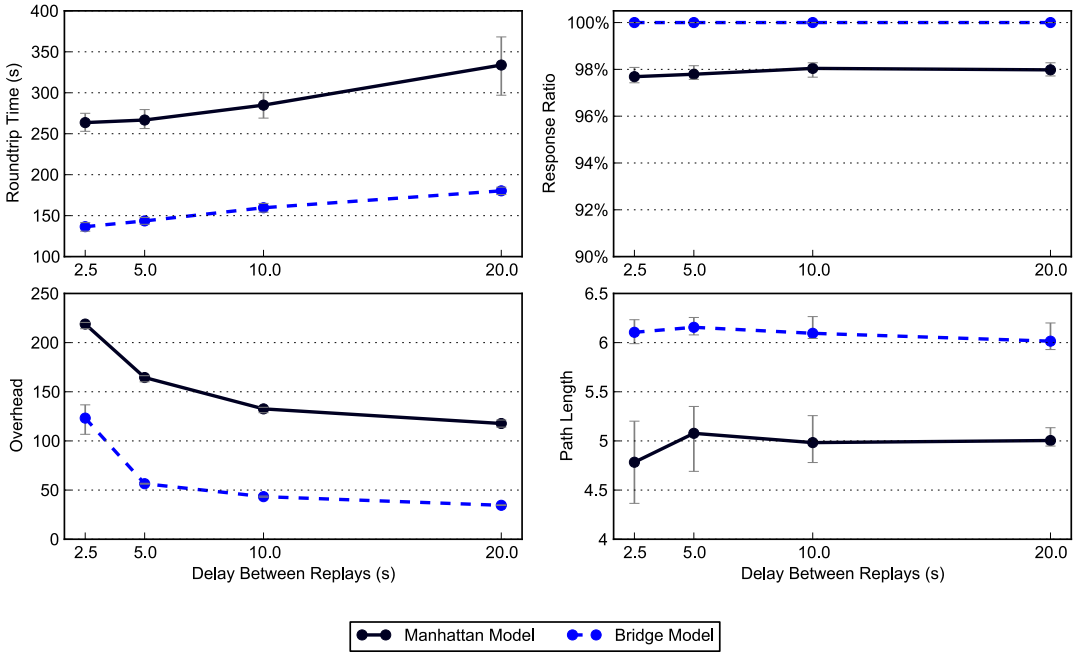

Figure 5.3: Effects of varying request replay frequency

has an average path length of about six hops, nearly identical to AODV and DYMO in other simulations using the same mobility traces.

5.2.2 Request Replays

BOND uses request replays as a simple way to support request ferrying in disconnected networks (see Section 4.1.5 for details). Requests are replayed a fixed number of times at a fixed interval. In practice, the total duration for which requests are replayed should be determined by the application – it should match the total amount of time the application is willing to wait for a response. However, it is not so clear how to tune the replay interval, especially without knowing the characteristics of the network ahead of time. Furthermore, it is worthwhile to understand what an application can expect when choosing the total replay duration.

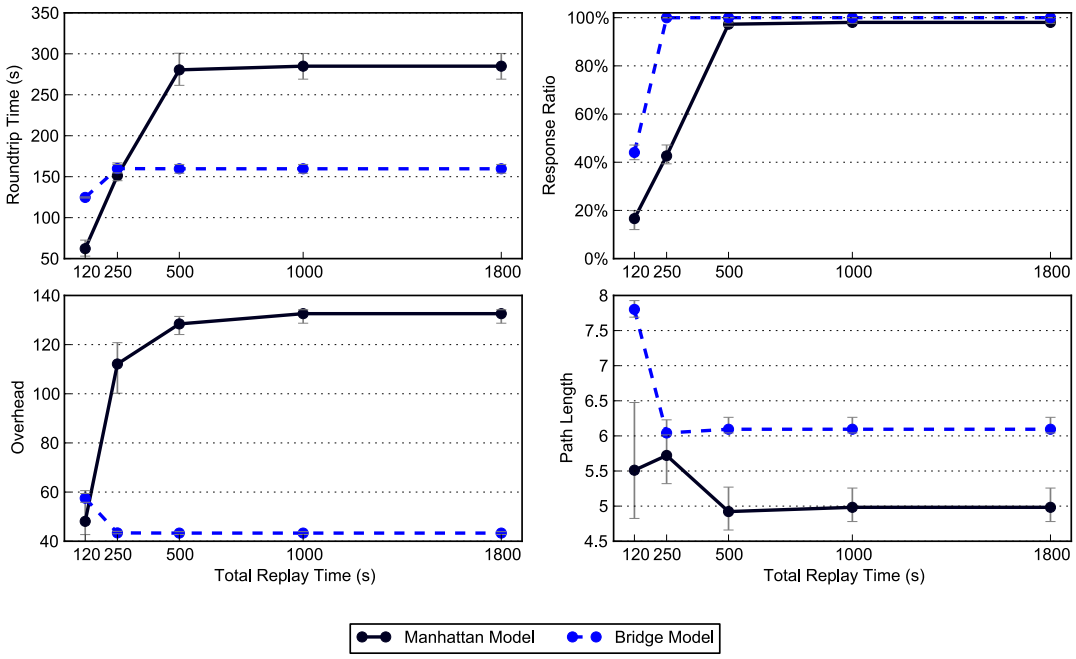

Figure 5.4: Effects of varying request replay duration

To understand the effect of varying the replay interval, Figure 5.3 shows the performance of different replay intervals for the same total replay duration. The total replay duration used was longer than the simulation run to approximate an infinite duration. Results are shown for each of the two disconnected network mobility models described in Section 5.1.2.

These results indicate that there is a simple tradeoff when choosing a replay interval – a longer replay interval result in higher latency, but lower overhead. Furthermore, the constant response ratio indicates that BOND's overall success is not sensitive to the replay interval.

To better understand the effects of differing total replay durations, Figure 5.4 shows the performance of different total replay durations using a fixed replay interval. These simulations use a replay interval of 10 seconds, which appears to be a good tradeoff between latency and overhead according to the results in

Figure 5.3.

These results show two things. First, for sufficiently long replay durations, the requested data reaches every node in the network before the maximum number of replays is exhausted. This is what causes all metrics to plateau above a certain replay duration. Second, the minimum replay duration required to achieve the maximum delivery ratio is dependent on the properties of the particular network. This makes intuitive sense, since packet transit in a disconnected network is at the mercy of the mobility of the nodes.

The remaining simulations in this section use a request replay interval of ten seconds with a maximum duration of 500 seconds. Note that requests are only replayed when the requester believes itself to be in a disconnected network, so these parameters do not affect connected scenarios.

5.3 Comparison with Connected Network Protocols

In this section, BOND's forwarding performance is compared to traditional, host-based routing protocols for connected wireless networks. It is important to note that routing protocols for mobile networks are generally evaluated using unidirectional, constant bit rate (CBR) traffic, streamed from one node to another without so much as a request or acknowledgement. As this section will show, these protocols tend to perform more poorly than their CBR performance would imply when the traffic is bidirectional.

5.3.1 Physical Mobility

Figure 5.5 shows the effect of differing amounts of physical mobility on various protocols. For these simulations, a varying percentage of nodes are mobile. The

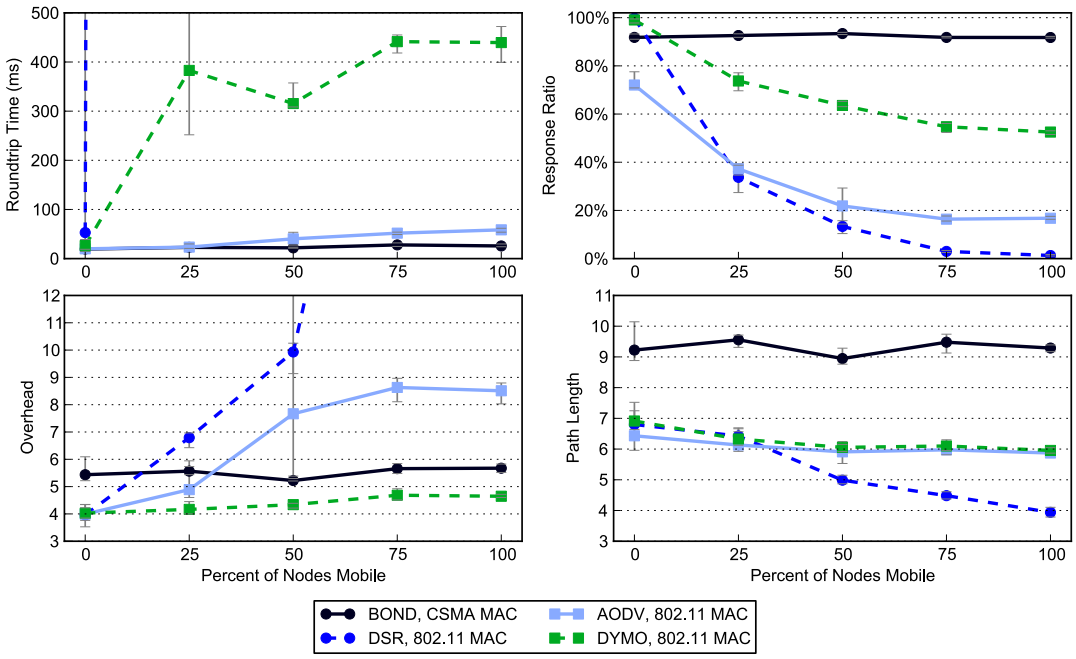

Figure 5.5: Effect of the number of mobile nodes in the network

mobile nodes move at a constant velocity of 30 meters per second using a random waypoint model. Other nodes do not move from their initial position. Eight bidirectional flows are present for this simulation with a request rate of 10 requests per second.

As intended, BOND performance is unaffected by the level of mobility in the network, maintaining a response ratio of over 90 percent. When no nodes are mobile, all other protocols perform fairly well with less overhead than BOND. This is to be expected – the cost of a single path setup is the only overhead incurred by these protocols in a static environment. However, with velocities of 30 meters per second, all other protocols perform significantly worse than BOND (in terms of latency and response ratio) as soon as any nodes become mobile.

After BOND, DYMO performs best, maintaining lower overhead than BOND, but delivering less than 60 percent of responses in the worst case. Of the three

comparison protocols, DYMO is the most flexible in its route selection, and thus most able to adjust to frequent topology changes. However, the mere fact that it must pre-establish routes means that they can still be broken and need to be re-established.

DSR performs worst – in fact, some if the DSR results for roundtrip time and overhead are not shown, as they are so high that they dwarf the differences between the other protocols. This is due to the fact that DSR, as a source routing protocol, must reconstruct the *entire* path whenever any link is broken. Note that each protocol ran on the same set of mobility traces, meaning that the drop in average path length for DSR could not reflect the presence of shorter paths in the network. Rather, as more nodes became mobile, DSR could only successfully deliver packets over shorter and shorter paths before its source routes broke.

The longer paths used by BOND are not due to any inefficiency in the protocol, but rather due to the choice of delay metric (see Section 5.2.1). Note that the overhead metric is also partially dependent on path length.

In other simulations, AODV consistently produced similar results relative to DYMO – much lower roundtrip times and much lower response ratios. DSR also produced similar results in other simulations – extremely high roundtrip times and overhead ratios, to the point that they could not be shown on the same plot. As a result, only results for BOND and DYMO are shown in the rest of this section.

5.3.2 Network Utilization

Figure 5.6 shows the effect of differing levels of network utilization on BOND and DYMO. All 100 nodes move using a random waypoint model with no pause time and speeds between 10 and 30 meters per second. There are 24 bidirectional flows

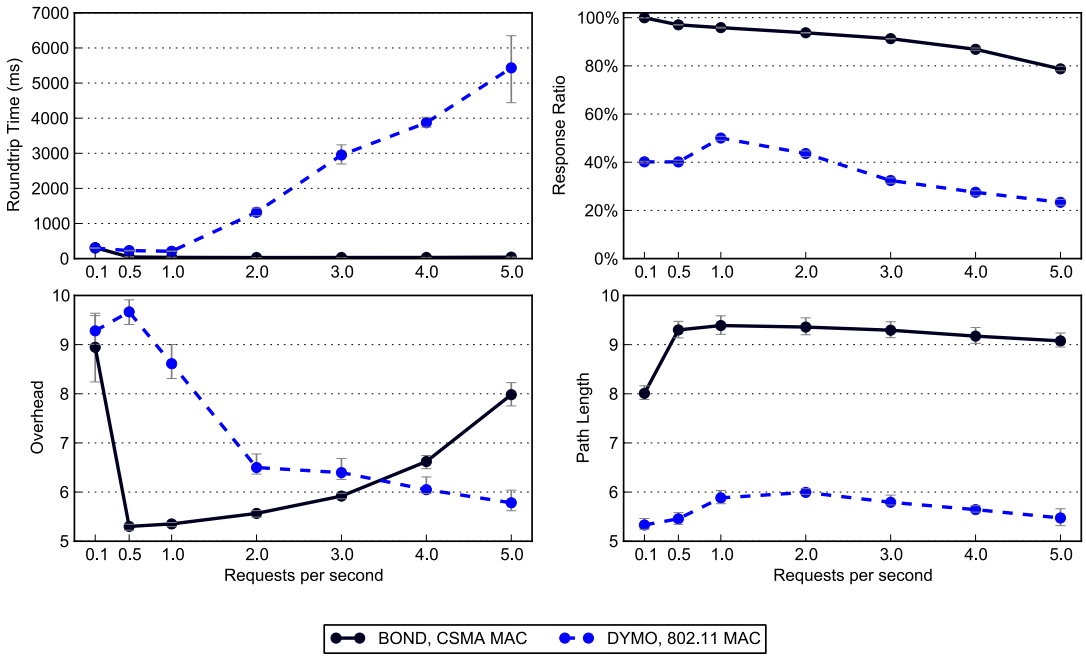

Figure 5.6: Effects of varying request rates

in the network, where no two flows share an endpoint. Thus, just short of half of the nodes in the network are actively transmitting data. The results at different request rates are shown, with the lowest being one request from each requester every 10 seconds (0.1 per second) and the highest being one request from each requester every 200 ms (5 per second).

This scenario serves two purposes. First, it shows how each protocol performs when the traffic rate is extremely low, meaning the topology of the network changes significantly between requests. Second, it shows how each protocol's performance degrades under extremely high load.

It is important to note some relevant configuration parameters for both protocols, as they are evident in these results. Recall that, in BOND, forwarding state is refreshed by data packets. Thus, when the data rate is low compared to the mobility speed of the nodes, forwarding state can become stale between data

transmissions. BOND nodes deal with this by discarding stale forwarding state after some configured time interval, falling back to flooding (see Section 4.3.1.1). For this scenario, BOND was configured to remove stale forwarding state after five seconds. Similarly, DYMO deals with stale *routes* by discarding them after a configurable time interval. For this scenario, DYMO was configured with a route timeout of three seconds.

In both protocols, a timeout occurs between every request when the request rate is 0.1 requests per second, but no timeouts occur between requests when the request rate is 0.5 requests per second or higher. In BOND, this accounts for the vastly higher overhead when the request rate is 0.1. In this case, all distance table entries at all nodes expire between requests, so each request must be flooded. Yet, the overhead in this case is still comparable to DYMO's and results in a 100% response ratio versus DYMO's 40%. In DYMO, the effect is less pronounced. The slight increase in overhead and decrease in latency when the number of requests per second increases from 0.1 to 0.5 is evidence that, in the former case, DYMO tries to find a fresh path immediately, while, in the latter case, DYMO sometimes manages to reuse an established path successfully (resulting in a decrease in average latency), but often tries to do so, fails, and must establish a new path (resulting in an increase in overhead).

BOND and DYMO's vastly different approaches to forwarding are also evident here. For BOND, when the network is near saturation, congestion causes collisions, which in turn causes more paths to be tried. This results in higher overhead, but a reasonable response ratio with low roundtrip times. For DYMO, even with low utilization, the large number of paths in the network result in a poor response ratio and high overhead. This is evidence of "hot spots" that can be created when predetermined paths cross. When the utilization is high, this

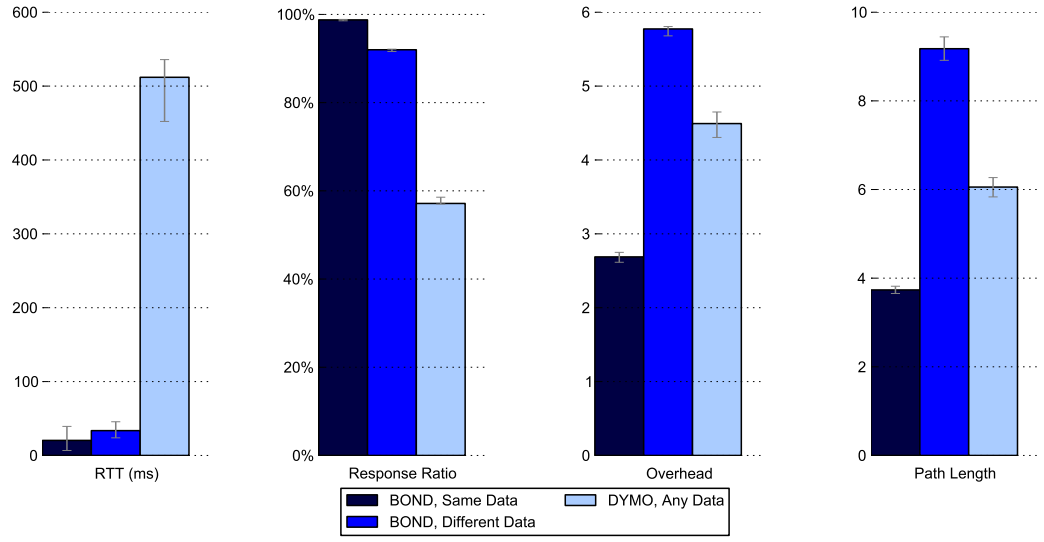

Figure 5.7: Benefits of caching

causes retransmissions in the 802.11 MAC, which in turn causes queueing. This leads to fewer packets sent (meaning lower overhead), but causes extremely high delays.

5.3.3 Benefits of Caching Named Data

Figure 5.7 shows the advantages of name-based forwarding and caching in a connected network. For this simulation, there were eight requesters and eight responders. Starting five seconds into the simulation, one requester is added per second for eight seconds. For BOND, two scenarios are shown. In the first, labeled "Same Data", all eight requesters request the same sequence of data names (starting with `a/0`). All eight responders initially have this data, advertising the prefix `a`. In the second, labeled "Different Data", BOND uses the same method described in the beginning of this section to imitate host-based forwarding. For DYMO, eight host-based flows are used.

In BOND, any node may cache data from a previous response. Thus, in the "Same Data" simulation, not only can any of the eight originators respond to any of the eight requesters, but any other nodes overhearing a response can respond to future requests for the same data name as well. In this case, approximately 45% of requests were served from a cache, with the other 55% served by one of the eight originators. The response always comes from the available responder closest to the requester. As a result, BOND performs significantly better in this scenario.

DYMO, on the other hand, is agnostic to the content of the packets it forwards. Its performance is the same regardless of what data is transmitted.

5.4 Comparison with Disconnected Network Protocols

In this section, BOND is compared with Epidemic Routing and Spray and Wait, two protocols designed for disconnected networks with unpredictable mobility. As these two protocols still use host-based addressing, BOND uses the same mechanism as in the previous section to imitate host-based transit. As this dissertation is interested in understanding the maximum potential benefits of caching, all simulations use an unrestricted cache size.

5.4.1 Data Ferrying Between Connected Networks

Figure 5.8 shows the results for the bridge mobility model described in Section 5.1.2 and pictured in Figure 5.1. To evaluate data ferrying between the two outer, connected areas, two requesters are placed in the western area, each of which have a corresponding responder in the eastern area.

This mobility model was intended as a test case for precisely the type of

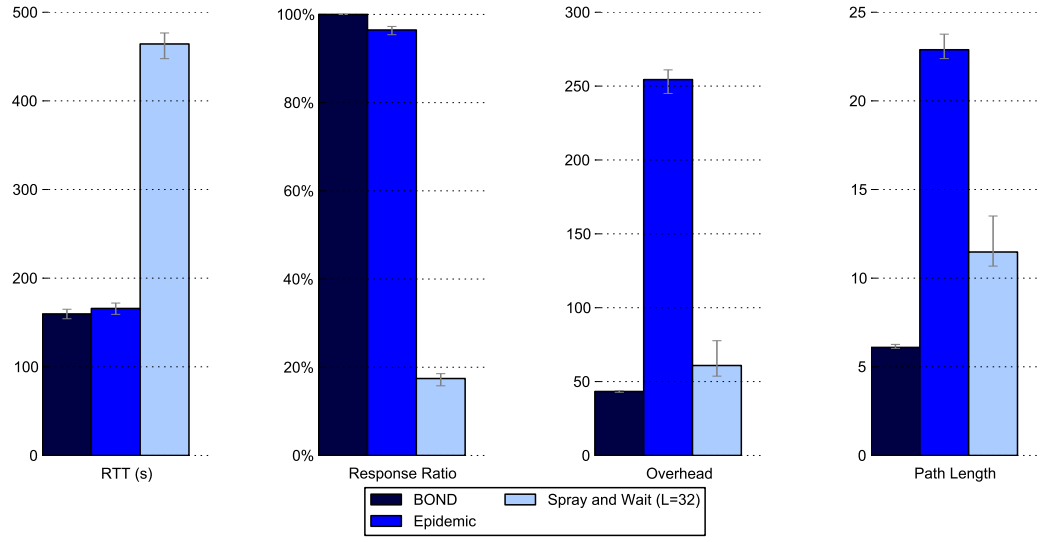

Figure 5.8: Results when ferrying data between connected networks

networks that BOND is designed for. As intended, it performs better than the competing protocols on all metrics.

Epidemic Routing has high overhead in this scenario due to its pairwise exchange of data between nodes. In the dense portions of the network, each node has many neighbors that can overhear a single data transmission. BOND takes advantage of this fact, while Epidemic Routing does not.

Spray and Wait has a serious problem with this scenario. This is due to its assumption that all node pairs are equally likely to encounter each other. As a result, nodes nearby the requester are allowed to replicate packets more times than nodes far away from the requester. In this scenario, only the bridge nodes will ever reach the network containing the responder. There is no guarantee that the bridge nodes will happen to be close to the requester when its request is sent. Thus, it is quite possible that, by the time the request reaches the bridge nodes, none of them are left with enough replications to ferry the request to the responder. Similarly, when the responder *does* receive a request, the same

problem can occur with the response. Additionally, like Epidemic Routing, Spray and Wait relies on pairwise data exchange between nodes.

It is also worth considering the meaning of the path length metric when there is no instantaneous path available. In such cases, the path length reflects the total number of hops taken by a request and its corresponding response, independent of the time *between* transmissions. Thus, when a node ferries a packet from one place to another without transmitting it, no additional hops are added to the path length.

In Figure 5.8, the path length metric tells us something about the operation of each protocol. For BOND, when in a connected portion of the network, packets are forwarded as in any connected network, resulting in short path lengths. For the other protocols, packets are not moved in any particular direction, so the packet that happens to reach the destination may have been copied many times. In comparison to Epidemic Routing, the shorter path lengths of Spray and Wait are a reflection of its design – Spray and Wait intentionally makes fewer copies of each packet, relying only on direct data ferrying once some fixed number of replications have been made (see Section 5.1.1).

5.4.2 Manhattan Mobility Model

Figure 5.9 shows the results of simulations using the Manhattan mobility model. In these simulations, the range of each wireless node was reduced in order to better approximate the limited connectivity available in an urban environment.

Using this mobility model, Spray and Wait performs comparatively better than it did in the bridge model. Its response ratio, though still poor, has increased significantly, and it has extremely low overhead compared to the other protocols.

Even without the extremely dense areas that characterize the bridge mobility

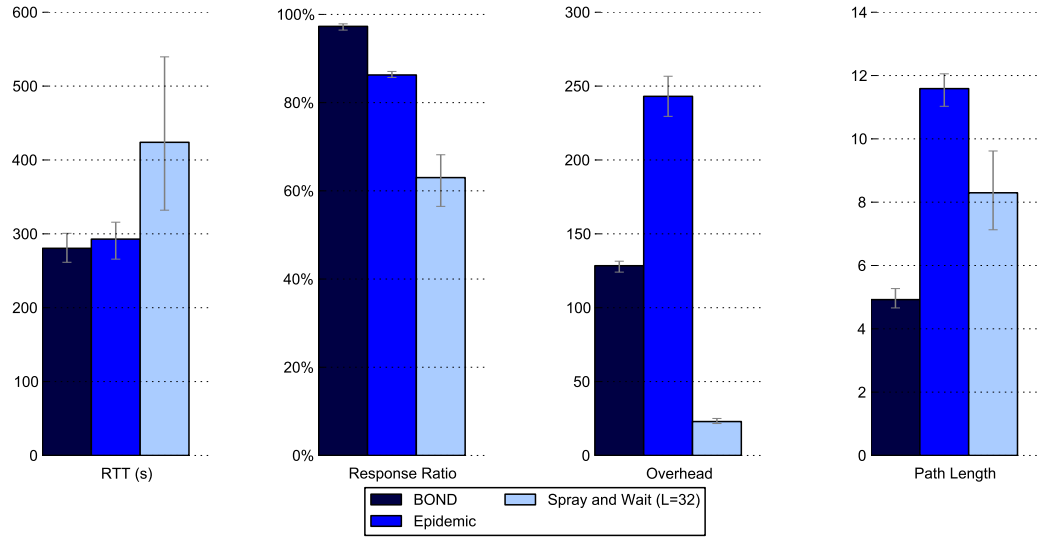

Figure 5.9: Results for the Manhattan mobility model

model, BOND has the highest response ratio and about half the overhead of Epidemic Routing. This shows that, even in a relatively sparse network, BOND's use of broadcast packets and ability to find short paths through locally connected areas has a significant effect. This is further evidenced by the fact that, on average, 10% of the data was forwarded directly from the original responder to the requester while there was a path between them. The other 90% came from cached copies retrieved via replayed requests.

CHAPTER 6

Conclusion

6.1 Suggestions for Future Work

BOND lays out a basic framework for named-based communication in freeform networks. As such, there are many opportunities to build upon it.

One such opportunity is reducing the overhead of request replays. Though BOND's request replay scheme is more efficient than the data-spreading-based protocols evaluated in Chapter 5, it is still inefficient by connected network standards. This is apparent when comparing the raw overhead numbers in connected scenarios with those in disconnected scenarios.

Luckily, BOND's request replay scheme is quite flexible. For example, one possible avenue for improvement is the fixed request replay interval. A more advanced system, such as exponential decay of the replay interval, could potentially decrease overhead without a significant impact on performance. Another approach could be a more advanced scheme for setting the replay count. For example, the replay count could be reduced based on how many replays were needed to elicit a response.

A second opportunity is the exploration of alternative distance metrics. One such metric is battery power – this would allow BOND to find the paths of least power consumption. Inter-contact time is another possible distance metric. This

would create a sort of temporal distance – the longer the time since a node was locally connected to a name, the further its distance from that name. This metric has the potential to improve performance in disconnected networks.

Third, military networks are a likely fit for BOND. However, in military networks, jamming of communications is often an important consideration. BOND's busy ratio scheme for detecting network congestion would require some adjustments in such scenarios. Otherwise, jamming could seriously affect the protocols ability to determine when data is in a disconnected portion of the network. Security features (discussed in more detail below) would also be a necessary enhancement for military networks.

Fourth, BOND may benefit from tighter integration with the physical layer. BOND's broadcast-only nature could provide many avenues for physical-layer-aware optimizations, such as collaborative transmission and physical-layer network coding.

Lastly, security considerations are not covered in this work. As defined, nothing would prevent a malicious node in BOND from responding with a virus in place of a requested piece of named data. Furthermore, nothing prevents two perfectly well-behaved responders from using the same name for different data. Name-based security has been proposed as a fundamental component of the NDN paradigm [JST09, ZEB10], including authentication of the name-data binding and support for data secrecy. However, the details are still a work in progress, and the application of name-based security in BOND has not been explored.

6.2 Final Thoughts

The number of portable devices with WiFi, Bluetooth, and/or 3G network interfaces has increased dramatically over the past few years. Yet, each of these wireless technologies is nearly always used for only a single hop, connecting the device directly to the wired infrastructure, or perhaps a peripheral. This is despite the problems with relying on 3G (and soon LTE) as the sole source of content for mobile devices. For example, at large events and festivals where crowds of people gather in a small area, cellular networks are notoriously overloaded and unusable. With the explosion of smart phones, at least one major US cellular provider has had serious capacity problems in urban areas.

As the density of short-range-wireless-capable devices continues to increase, so should the utility of forming them into a multi-hop, freeform network. A freeform network of smart phones could help to ease the pressure on cellular networks and maintain service when they fail. However, the industry has shown little interest in this technology. Could it be that the existing protocols are simply not good enough? It is probably not the only reason, but, judging from their performance under simulation, it must certainly be a factor.

BOND provides an alternative approach to freeform networking, which removes many of the barriers to deploying multi-hop wireless networks – devices do not need to be assigned IP addresses, the network can be connected or disconnected, and nearly any MAC protocol may be used. Hopefully, BOND will prove to be a step in the right direction.

References

[BLS08] Rainer Baumann, Franck Legendre, and Philipp Sommer. "Generic mobility simulation framework (GMSF)." In *MobilityModels '08: 1st ACM SIGMOBILE Workshop on Mobility Models*, pp. 49–56, 2008.

[BM05] S. Biswas and R. Morris. "ExOR: opportunistic multi-hop routing for wireless networks." *ACM SIGCOMM Computer Communication Review*, **35**(4):144, 2005.

[CCN06] Matthew Caesar, Miguel Castro, Edmund B. Nightingale, Greg O'Shea, and Antony Rowstron. "Virtual ring routing: network routing inspired by DHTs." In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pp. 351–362. ACM, 2006.

[CJK07] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. "Trading structure for randomness in wireless opportunistic routing." In *SIGCOMM '07*, pp. 169–180. ACM, 2007.

[CP10] I. Chakeres and C. Perkins. "Dynamic MANET On-demand (DYMO) Routing." Internet-Draft draft-ietf-manet-dymo-21, Internet Engineering Task Force, 2010. Work in progress.

[HPS02] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. "The Zone Routing Protocol (ZRP) for Ad Hoc Networks." Internet-Draft draft-ietf-manet-zone-zrp-04, Internet Engineering Task Force, 2002. Work in progress.

[IGE00] 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)*, pp. 56–67. ACM, 2000.

[JM96] David B. Johnson and David A. Maltz. "Dynamic Source Routing in Ad Hoc Wireless Networks." In Tomasz Imielinski and Henry F. Korth, editors, *Mobile Computing*, volume 353 of *The International Series in Engineering and Computer Science*, pp. 153–181. Springer US, 1996.

[JST09] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. "Networking

named content." In *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pp. 1–12. ACM, 2009.

[JT87] John Jubin and Janet D. Tornow. "The DARPA packet radio network protocols." In *Proceedings of the IEEE*, volume 75, pp. 21–32, Jan 1987.

[KK00] B. Karp and HT Kung. "GPSR: greedy perimeter stateless routing for wireless networks." In *MobiCom '00*, p. 254. ACM, 2000.

[LDS04] Anders Lindgren, Avri Doria, and Olov Schelén. "Probabilistic Routing in Intermittently Connected Networks." In *Service Assurance with Partial and Intermittent Resources*, volume 3126 of *Lecture Notes in Computer Science*, pp. 239–254. Springer, 2004.

[MG96] S. Murthy and J.J. Garcia-Luna-Aceves. "An Efficient Routing Protocol for Wireless Networks." In *ACM Mobile Networks and App. J., Special Issue on Routing in Mobile Communication Networks*, pp. 183–97, Oct 1996.

[NBN01] Navid Nikaein, Christian Bonnet, and Neda Nikaein. "HARP - Hybrid Ad Hoc Routing Protocol." In *International Symposium on Telecommunications*, 2001.

[NC04] W. Navidi and T. Camp. "Stationary distributions for the random waypoint mobility model." *Mobile Computing, IEEE Transactions on*, **3**(1):99 – 108, Jan 2004.

[PA00] V. Paxson and M. Allman. "Computing TCP's Retransmission Timer." RFC 2988 (Proposed Standard), November 2000.

[PB94] C. E. Perkins and P. Bhagwat. "Highly dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for mobile computers." In *ACM SIGCOMM*, 1994.

[PR99] Charles E. Perkins and Elizabeth M. Royer. "Ad-hoc On-Demand Distance Vector Routing." In *IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[SPR05] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. "Spray and wait: an efficient routing scheme for intermittently connected mobile networks." In *WDTN '05: Proceedings of the ACM SIGCOMM Workshop on Delay-Tolerant Networking*, pp. 252–259, 2005.

[Vai01] Nitin Vaidya. "Open Problems in Mobile Ad Hoc Networking." In *LCN '01: Proceedings of the 26th Annual IEEE Conference on Local Computer Networks*, p. 516, 2001.

[VB00] Amin Vahdat and David Becker. "Epidemic routing for partially-connected ad hoc networks." Technical Report CS-2000-06, Duke University, 2000.

[WSJ09a] Xin Wang, Yantai Shu, Zhigang Jin, and Huan Chen. "Weighted Epidemic Routing for disruption tolerant networks." In *Fourth International Conference on Communications and Networking in China (ChinaCOM)*, pp. 1 –5, August 2009.

[WSJ09b] Xin Wang, Yantai Shu, Zhigang Jin, Qingfen Pan, and Bu Sung Lee. "Adaptive Randomized Epidemic Routing for Disruption Tolerant Networks." In *MSN '09: 5th International Conference on Mobile Ad-hoc and Sensor Networks*, pp. 424–429. IEEE, Dec 2009.

[YCL01] F. Ye, A. Chen, S. Lu, and L. Zhang. "A scalable solution to minimum cost forwarding in large sensor networks." In *Tenth International Conference on Computer Communications and Networks*, pp. 304–309. IEEE, 2001.

[YYW05] Y. Yuan, H. Yang, S.H.Y. Wong, S. Lu, and W. Arbaugh. "ROMER: Resilient opportunistic mesh routing for wireless mesh networks." In *1st IEEE Workshop on Wireless Mesh Networks (WiMesh)*, 2005.

[YZL05] F. Ye, G. Zhong, S. Lu, and L. Zhang. "Gradient broadcast: A robust data delivery protocol for large scale sensor networks." *Wireless Networks*, **11**(3):285–298, 2005.

[ZEB10] Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D. Thornton, Diana K. Smetters, Beichuan Zhang, Gene Tsudik, kc claffy, Dmitri Krioukov, Dan Massey, Christos Papadopoulos, Tarek Abdelzaher, Lan Wang, Patrick Crowley, and Edmund Yeh. "Named Data Networking (NDN) Project." Technical Report NDN-0001, PARC, 2010.