PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Tesis de Maestría
en Informática

RON – Redes oportunistas

Jorge Visca

2014

Jorge Visca
RON – Redes oportunistas
ISSN 0797-6410
Tesis de Maestría en Informática
Reporte Técnico RT 14-04
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.
Montevideo, Uruguay, 2014

RON - Redes Oportunistas

Jorge Visca - jvisca@fing.edu.uy

Tutor: Dr. Eng. Javier Baliosian

Tribunal:
Dr. Lisandro Zambenedetti (Revisor)
Dr. Gustavo Betarte
Dr. Marcos Viera

Facultad de Ingeniería
Universidad de la República

Noviembre de 2013

Resumen

El gran crecimiento de las redes inalámbricas ha exigido a los protocolos establecidos hasta sus límites. Características de las redes que se asumían como dadas, tales como la existencia de caminos de extremo a extremo o tiempos de propagación bajos, son cada vez más frecuentemente comprometidas. Estas situaciones pueden surgir como consecuencia de fallos en redes convencionales, o de operar redes en situaciones muy adversas o impredecibles, cuando los nodos de la red se encuentran entre ellos solo esporádicamente. Este tipo de redes son conocidas como "Redes Tolerantes a Retardos" o "Redes Oportunistas". Para operar, estas redes necesitan nuevos algoritmos de enrutamiento.

Por otro lado, cierto tipo de aplicaciones tales como redes de sensores son altamente dinámicas: nodos son agregados o retirados de la red continuamente, y nuevos flujos de datos son establecidos o modificados a medida que el propio análisis de los datos resulta en nuevos requerimientos de información. En este escenario el enrutamiento convencional, por destino, se torna engorroso debido a la necesidad de mantener y operar un inventario de los dispositivos participantes de la red. De aquí surge el interés en nuevos métodos para especificar flujos de datos, y el Enrutamiento Basado en Contenido es una tecnología prometedora en este sentido.

La combinación de Redes Oportunistas y Enrutamiento Basado en Contenido provee una plataforma altamente flexible para resolver nuevos problemas de ingeniería de redes. Este trabajo presenta un nuevo protocolo oportunista basado en contenidos, que utiliza algunos de los conceptos más exitosos de los algoritmos existentes e implementa algunas soluciones novedosas. El comportamiento y rendimiento del algoritmo es evaluado y algunos escenarios especialmente problemáticos (para este y otros algoritmos) son identificados.

Abstract

The explosive growth in use and coverage of wireless networks has pushed the established network protocols to their limits. Some fundamental assumptions usually taken for granted, such as the existence of a end-to-end path or low propagation times, are being more and more frequently violated. These violations can be caused by outages and failures in otherwise conventional networks (e.g. network partitions), or stem from the need to communicate data in extremely adverse or unpredictable situations, when network nodes meet only sporadically. Depending on the approach, these kinds of networks are called Delay Tolerant Networks (DTN) or Opportunistic Networks (ON). To support data networks in such conditions, new routing algorithms are needed.

Additionally, some network applications, such as sensor networks, are highly dynamic: nodes are added or removed at any moment, and new data flows are established and modified as new requirements are distilled from the data obtained from the network itself. In this scenario the traditional destination based routing scheme imposes and insurmountable burden on network provisioning and inventory. Other methods for specifying data flows are required, and one such method is Content Based Routing.

The combination of Opportunistic Networking and Content Based Routing offers a very flexible platform for solving novel engineering problems. This work presents a new content-based opportunistic network protocol, that builds on concepts found in existing algorithms and implements some novel ideas. This algorithm's behavior and performance is evaluated and some specially challenging scenarios are identified, that also apply to other algorithms of this class.

Contents

Chapter 1

Introduction

There is a class of networks were the usual routing algorithms do not work. These networks are called under various names, Delay Tolerant Networks, Disruption Tolerant Networks or Opportunistic Networks, but they all share a defining property: there is no guarante a path between two nodes exists when a data transmission is attempted. This means that a message has to be stored in intermediate nodes waiting for a transmission opportunity, and this might happen several times as the message travels towards its destination.

This condition appears under several scenarios, and covers a wide spectrum of network applications. On one extreme, it can be seen as a limit case for a traditional network. As Internet expanded explosively and new link technologies (like underwater communications) and extreme deployment scenarios (like interplanetary Internet) got explored, network partitions could not be considered anymore a breakage event. Not only network partitions became usual, but also other pathological conditions such as very low throughput, asymmetric links or very high propagation delays.

The other great motor of opportunistic networks was the apparition of a profusion of battery powered and wirelessly networked devices, such as smartphones, laptops, smart-home appliances, etc. Also, this profusion of low cost wireless devices created the opportunity for creating highly flexible networks in places beyond the reach of traditional Internet infrastructure. These networks could serve a very wide spectrum of tasks: emergency services, environmental data collection, low cost delivery of bulk data, vehicular fleet tracking, etc.

The classical solution for a infrastructure-less networks is the mesh (ad-hoc) network. Nevertheless, the routing algorithms used in mesh networking are direct descendants of the usual Internet algorithms, usually variations of distance-vector and link-state algorithms. Thus, mesh networks share the same assumption Internet does: the existence of a end-to-end path. So even tough a mesh network does not depend on external infrastructure, it does depend on the nodes generating a infrastructure themselves. Trouble is, mesh networks are even more vulnerable to network partitions and extreme transmission characteristics.

To implement a really infrastructure-less network, a new network paradigm is needed, were nodes are free to roam and establish connections with their pairs, and were the routing algorithm is smart enough to make intelligent decisions on what data keep, and whom it handle to when opportunity arises. Besides the routing and forwarding tasks typical of a routing algorithm, nodes have to store information between connection opportunities. Besides bandwidth, new resources have to be managed, like airtime and storage utilization. Also, as the nodes are usually battery powered, power consumption is a critical resource to manage. To make intelligent decisions, information is needed: algorithms should learn and take advantage of network patterns. There could be patterns in the movement of the nodes, the data flows, or in the availability of certain resources.

Due to having more resources to manage and more inputs for taking decisions, opportunistic routing algorithms are bound to be more complex. On the other hand, opportunistic routing is most useful on devices with limited resources, typically battery powered. This makes for a hard design balance between performance and resource consumption.

To the present day, the opportunistic routing is an open problem: new approaches, algorithms and solutions are being proposed, and new challenges are identified.

These highly dynamical networks, with nodes appearing and leaving with no announcements highlight one challenge in particular: the addressing scheme. The traditional addressing of Internet is destination based: the node at the source of the information flow specifies the destination node. This assumes the source must know the destination for a data transfer to happen, and this was found to be limiting in many applications, and a heavy burden to administer. New methods are needed to allow nodes to access the information they need. In particular, methods that allow the data flow to be specified by the data consumers, and that do not depend on the identity of nodes. Such a paradigm exists, Content Based Networking, were data consumers describe the data they need, and information of interest is routed to them, transparently, from wherever it could be.

The combination of Opportunistic routing and Content Based addressing promises a very flexible and robust network infrastructure, but there are many challenges ahead.

1.1 Objectives

The main purpose of this work is to provide a networking solution for the DEMOS project (described in more detail in Chapter 3). The DEMOS project attempts to improve the life conditions of children in environmentally vulnerable areas and neighborhoods with poor infrastructure. For this purpose it proposes deploying a wide network of low-cost environmental sensors in public areas and children's living premises. The combination of a deployment with many sensors covering a wide area and lack of infrastructure presents challenges

for the data collection: how to transmit data from these low-cost, remote sensor nodes to a collection point for processing?

The proposed method is to take advantage of a local OLPC (One Laptop Per Child) program. The existence of these numerous nodes, moving periodically trough the neighborhood as children go to and from school, create an opportunity for implementing an Opportunistic Network: the children's laptop can be used as mobile nodes to shuttle information between sensor nodes and collection points. For this purpose, a software solution must be developed, the DEMOS System (see Section 3.1). This solution is comprised of a set of tools and services that will be used to support, configure and maintain the network.

One of the components of the DEMOS System is the network protocol used to transmit data between nodes, in a Opportunistic way. This protocol is the expected result of this work. Therefore, the work is organized following three objectives, as follows.

The first objective of the present work is to perform an in-depth study of infrastructure-less wireless networks, especially the ones categorized as Delay Tolerant or Opportunistic. It is expected to lead to the deployment of a real solution, so special importance is devoted to existing proposals and their properties. In case of real deployment, it is to be made using low-cost and low-powered devices, thus the efficient use of available resource (computational and air time) is favored.

Due to the nature of the network, it was expected to adapt better to content based addressing, instead of conventional destination based (for a discussion on this issue, see Section 2.4). Therefore, Content Based Opportunistic algorithms are of special interest.

The second objective is the development and implementation of a network solution for the DEMOS project. For this purpose, a set of requirements were drawn (see Section 4.1). If no suitable technology or existing solution is found, a new solution is to be created.

Finally, the behavior of the selected solution is to be studied in depth, to prove it is adequate for the DEMOS project and get new insight on the challenges and weaknesses in the research area in general. The analysis should also provide base for future work.

1.2 Contribution

The main contribution of this work is a novel content-based opportunistic algorithm, RON. The algorithm (described in Chapter 4) uses some ideas of existing solutions, but improves over them in some areas that were not handled correctly before. The algorithm is simple, efficient and is easily modifiable. This makes it a good base platform for adopting and testing out new ideas.

The implementation is highly portable and has been extensively tested in simulated environments as well as real hardware.

A method for autonomically managing the protocol was designed and implemented (see Section 5.3). The method is very flexible and based on high-level

policies, and allows to modify the behavior of the protocol in response to environment changes.

A method for estimating the maximum delivery rate possible in an opportunistic network by an algorithm possessing perfect knowledge ("Oracle") was developed. This method is based on a Genetic Algorithm, and provides an upper bound to attainable performance.

There were identified areas that impose limitations on the performance and robustness of the protocol. These weaknesses are not exclusive of RON, and actually are shared with other ON platforms. They are described in Chapter 7. Based on the results shown in Chapter 5, RON is a good infrastructure for developing solutions for these issues, thanks to the simplicity of its architecture and flexibility of the implementation.

1.3 Structure of the Document

The description of the research area and analysis of the existing solutions is presented in Chapter 2.

A specific use case for an Opportunistic Network, the project DEMOS, is described in Chapter 3. For this application, a set of requirements are defined in 4. These requirements and the State of the Art analysis lead to the creation of the RON protocol, described in Section 4.2. The implementation is described in Section 4.3.

The solution's performance and behavior are analyzed in Chapter 5, and new work directions are presented in Chapter 7.

Chapter 2

State of the Art

The ideas behind Delay Tolerant Networks or Opportunistic Networks predate Internet. Some ideas appear in research on Packet Radio Networks by DARPA as far back as 1978[1]. Also, FidoNet operated in an environment consisting of temporary connections between nodes in the form of more or less periodic phone calls (early nineties). Similar networks were supported by the UUCP protocol since the early eighties.

Recently there has been a new surge in research in the area, driven by the widespread adoption of wireless links for the purpose of supporting Internet, and the apparition of novel network applications, such as sensor networks.

In this chapter we will present a review of the concepts and technologies used in the area, as well as implementations and solutions available.

2.1 Delay Tolerant Networks

The concept of Delay Tolerant Networks (DTNs[1]) was introduced as a specific research field in [2], in relation with IP network research. The authors point out that there are several assumptions made about the underlying links on a standard IP network, like the existence of a end-to-end path, a low round-trip time and low packet loss probability. On the other hand, the category of networks which break one or more of these assumptions ("*challenged networks*") is growing in importance, as Internet gets deployed over technologies which cannot support them.

The authors present the following characteristics of these challenged networks:

High latency, low data rate These networks provide services over low bandwidth links. For example, a low power radio as used in sensor networks could provide about 10kbps, and latencies of around 1 second can be common. Also, links can be highly asymmetric, or even unidirectional.

[1]These networks are also known as Disruption Tolerant Networks, the name under which they are referred by DARPA.

Disconnection End-to-end connectivity can be unavailable for relatively long spans of time. These disconnects can be caused by faults, or normal operations conditions. For example, nodes can be mobile and be in range only occasionally, or nodes can go on-line according to a schedule to save batteries.

Long queuing times Unlike conventional packed switched networks, queuing times can be arbitrarily long (hours or days are usual). Longer disconnected periods usually lead to longer queuing times.

Low duty cycle operation When deployed for providing communications from batteries, efforts are made to minimize the time spent transmitting and receiving data. For example sensor nodes could collect data continuously, but transmit only when enough data has been collected.

Limited resources Several of the applications envisioned (sensor networks, emergency relief) are based on small devices with limited computing resources. Limitations can be in transmitter uptime and power, raw computing power, storage, total cost, etc.

Also, several approaches to cope with challenged networks are identified. The first approach is to engineer the links as to fool upper layers into believing there is a normal IP layer underneath ("link-repair approaches"). Another approach identified is to have proxy agents at the edges of the DTN. This method does not offer a general method for providing transit. As a result, the authors argue for the use of a general purpose message oriented overlay architecture, providing a reliable asynchronous message delivery, as the appropriate approach to take full advantage of challenged networks. This came to be a very successful approach, and one of the defining architectural characteristics of DTNs.

DTN research is being done within the Internet Research Task Force umbrella, through the Delay-Tolerant Networking Research Group[3]. As such, the view of the network is Internet-centric: the architecture consists of a network of independent Internets, each supported by a typical Internet stack within. The communications between these Internets are occasional, either scheduled or random, and are sustained trough *DTN gateway* nodes.

The reference architecture of DTN is described in [4] and [5]. The end to end message oriented overlay is referred as the "*Bundle* layer". The nodes that implement the Bundle layer are the DTN nodes. To interact with a DTN, applications generate self-contained messages of arbitrary length, called ADUs (Application Data Units). These ADUs are delivered to the Bundle layer, which will split them in protocol data units called Bundles. Bundles are handled by the nodes maintaining the Bundle layer until they reach destination. Nodes have dedicated persistent storage for keeping bundles between communication opportunities.

2.1.1 DTN Bundles

The unit of information transmitted in a DTN is called a *bundle*, and is specified by the IETF in [6]. A bundle is a variable length PDU, bundling together all the information needed to complete a transaction. This is done to reduce the number of round-trips, which is of critical importance in DTN due to the very high latencies. The Bundle Protocol offers end-to-end transport for applications over a DTN, and provides such functionality as Fragmentation Control and Error Detection and is integrated with the remaining of the DTN architecture for elements such as naming and addressing.

2.1.2 Addressing

The addressing scheme for bundles is based on the concept of *Endpoints*. An endpoint is a group of nodes which can be addressed collectively. A node can participate in several endpoints, and is required to belong at least to an Endpoint of which is the only member. Endpoints are identified by an EID (Endpoint Identifier) which is a name expressed as a URI [7].

An application sends an ADU addressing it to an EID, this is, to a node or group of nodes. Conversely, if an application is interested in receiving ADUs to a certain EID, it must register that interest with the node.

The addressing flexibility of the DTN architecture is supported by the flexibility of the URI schemas. When deploying a DTN, an URI scheme must be designed to provide a naming that makes sense for the application: whether it takes into account geographical location, or follows some organizational lines, application domain groups, etc.

When a bundle is addressed to a endpoint composed of multiple nodes, it is considered as delivered when in reaches a subset of the endpoint called MRG (minimum reception group). The MRG can be single node (unicast), one of a group of nodes (anycast) or all of a group of nodes (multicast and broadcast). The MRG is an attribute associated to the endpoint.

2.1.3 Routing

The DTN specification intends to lay the ground for further research, and for this purpose provides a framework for future work. It envisions supporting operations in a very wide spectrum of condition and scenarios, from sensor networks to interplanetary Internet and deep-space communications. The involved links have vastly different behavior, and thus the routing algorithms should be different, too. To classify the different conditions the RFC4838 proposes the following classification:

Persistent Contacts Always available links. Example: a DSL link.

On-Demand Contacts Links that can be brought up when needed. A initiator node must perform some action to activate the link, which will be available until terminated. Example: a dial-up connection.

Intermittent - Scheduled Contacts Links that are known in advance will be available at some moment in time, for a given amount of time. Examples: low orbit satellite, scheduled communication sessions with a spacecraft.

Intermittent - Opportunistic Contacts Link opportunities that occur unexpectedly, without previous planning. The duration of the connection lasts an undetermined amount of time. Example: a device with a short range radio (eg. a Smart-phone with bluetooth) moves into range.

Intermittent - Predicted Contacts If it is possible to get enough information from previous story and the behavior is consistent enough, it is possible to use this information to predict future link opportunities. This can be seen as a variation of scheduled contact (with predictions instead of schedules), or a variation of opportunistic contacts (with predictions instead of completely random encounters). Examples: Wi-Fi devices on buses meeting on the road trough the day.

The routing methods themselves for these situations were proposed as an area of work in the research area of delay tolerant networking. Only a very general overview is provided, proposing maintaining the conventional separation between routing (computation of routing paths) and forwarding (delivery of bundles between nodes). The routing information is kept in the RIB (routing information base), from which the forwarding state FIB (forwarding information base) is derived.

2.1.4 Conclusion

Besides relaunching the research in the area, the DTN architecture identified some important concepts.

In first place, the recognition that it is not viable to attempt to make DTNs transparent at the IP level, and that it is needed to encapsulate the DTN behavior in a separate layer. The idea of supporting a DTN trough a message oriented asynchronous overlay has been of great importance.

Another important idea is to extend the addressing scheme beyond what is usual in IP networks. In DTN architecture is done in a very general way relying on URI expressed addresses, but it opened the door to other paradigms, like content based routing.

Finally, the importance of storage as another resource that must be managed in a carry-and-forward-network is presented.

At the same time, great effort was spent specifying a framework for Internet-like development: considerable detail was provided for the format of bundles; gateways and proxies for interacting with TCP/IP networks are described; priority (QoS) and acknowledgment flags are specified, etc. This implies a rather complex implementation even before "filling in the blanks" of routing, addressing implementation, etc., and is not very useful or can be restricting in some applications. For example, a sensor network is a single use network, with messages

that can be very small or fixed size, were message traffic flows only in one direction. A DTN like architecture could impose unneeded complexity (like bundle fragmentation) and preclude some useful behavior (like message aggregation).

2.2 Opportunistic Networks

Tough usually the terms Opportunistic Networks (ON) and DTN are used interchangeably, ON can be seen as a generalization of DTN[8]. Unlike DTN, which is based on Internet technologies, ON does not make any assumptions on the topology of the network or underlying technologies. While in DTN the points of disconnection are restricted to the gateway nodes, in ON each node that receives messages makes routing decisions.

All this makes ON a more flexible concept, that can be applied to novel areas such as Sensor Networks.

2.2.1 Taxonomy

In [8] a two-level taxonomy for Opportunistic Networks is proposed:

Infrastructure vs infrastructureless Infrastructure means that the network is asymmetric, in the sense that some nodes are more powerful and play a special role in the routing. Infrastructureless networks are flat, and all nodes are equal.

Infrastructure: Fixed vs Mobile Specific geographic locations of significance, or mobile nodes, whether predetermined trajectory or random.

Infrastructureless: Dissemination vs Context based Dissemination algorithms are forms of controlled flooding. These algorithms typically do not keep any knowledge about the network. Context based algorithms attempt to use some form of knowledge about the network behavior to select the best candidates to carry messages to a destination.

As a taxonomy, it suffers from a lack of homogeneity. The first level (whether the network has dedicated infrastructure) is based on properties of the network. But then, while the "infrastructure" case is further categorized in function of more properties of the network, the "infrastructureless" case is categorized in function of properties of the algorithms (context aware or not).

In [9] the authors propose two main classifications for DTN (ON) algorithms:

Replication versus Forwarding Replicating algorithms are those which can generate multiple copies of a message as it traverses the network. Forwarding means there is a single copy, thus at any given moment any message is located on a single mobile node.

Resource Constraints This classifies the algorithms according to the resources they take into account for making routing decisions. The two main resources are Storage and Bandwidth.

Problem	Storage	Bandwidth	Routing	Examples
P1	Unlimited	Unlimited	Replication	Epidemic, Spray and Wait
P2	Unlimited	Unlimited	Forwarding	Modified Dijkstra
P3	Finite	Unlimited	Replication	PROPHET
P4	Finite	Finite	Forwarding	
P5	Finite	Finite	Replication	Rapid, MaxProp

Table 2.1: A ON algorithm taxonomy[9]

This creates 5 main classes of algorithms, as seen in Table 2.1. The algorithms shown as examples are described in more detail in the following sections.

2.3 Opportunistic Protocols

For supporting Opportunistic Networks there is a wide range of algorithms available. It is an ongoing research field, and differing approaches can be found. Solutions vary in complexity, flexibility and robustness. A pattern can be seen where more sophisticated algorithms provide higher flexibility and higher expected performance at the cost of a more difficult setup and a explosion in the number of configuration parameters. For a overview of existing solutions, see Table 2.2.

2.3.1 Direct Delivery

It is the simplest form of ON. The source node stores the packet until it meets the destination node, when it delivers the packet and erases it from its own storage. In many realistic use cases can have extremely bad performance.

2.3.2 Epidemic Routing

One important family of protocols is Epidemic Routing [10]. Epidemic Routing is based on Epidemic algorithms, a method for synchronizing distributed databases[11]. In Epidemic Routing a node carrying a message will deliver it to each node it meets, and each node that receives the message will start replicating it in a similar way. This way, the message will eventually reach its destination. There is a compromise between resource utilization (bandwidth and storage) on one side and network performance (delivery rate and latency) on the other. To tune this behavior, the authors use two configuration parameters: a upper bound for the message hop count, and a per-node buffer space. When there are no limitations on the number and opportunity of replications, the result is flooding of the message in the network. In this case the latency is minimized, but at the cost of traffic overhead and storage waste in the nodes.

The Epidemic Routing algorithm is described as follows [10]. When two nodes A and B met, the one with the lower id initiates a session. The session

consists of three steps:

1. A sends SV_{A}
2. B requests $SV_{\mathrm{A}} + \overline{SV_{\mathrm{B}}}$
3. A sends messages unknown to B

In step 1, the node A sends a Summary Vector (SV), which is a vector containing the hashes of all messages in A's buffer. Node B intersects the incoming SV with the complement of it's own to find messages that are available in A but not in B, and requests them. Then, in step 3, the node B delivers the requested messages.

As a management strategy for the buffer, FIFO is used. Some problems are described, in particular lack of QoS and fairness with relatively small buffers. In particular, aggregated buffer space used by a host is proportional to the number of messages it emits, which can be unfair to other nodes using the network. This effect could be mitigated using some form of Fair Queuing algorithm, such as WFQ (Weighted Fair Queuing).

Behavior under resource consumption bounding is analyzed.

A weakness of this algorithm is that during a meeting messages are delivered only from the node with lower id to the one with higher id. This means that the numeration scheme for nodes must be defined in line with the expected data flows. This is not always possible.

Nevertheless, two important ideas appear implemented: the need to limit the life-cycle of messages (trough hop-count in this case), and a method for efficiently exchange information on the availability of messages.

2.3.3 Spray and Wait

Spray and Wait[12] is a method to limit Epidemic's resource utilization. Spray and Wait routing consist of two phases: first, the node replicates as in Epidemic Routing, up to L copies of a message. After that, and if the destination has not been found yet, each of the L nodes carrying a copy of the message will keep it for direct transmission (until they find the destination). In the basic version called Source Spray and Wait, the algorithm replicates to the first distinct L nodes it meets. A variation intended to reduce the overhead is to forward a copy with probability $p < 1$ (*Randomized Flooding*).

A more sophisticated variation called Binary Spray and Wait (BSW) is proved to be optimal when nodes move in IID manner. In BSW, each packet has a "copies carried" counter. The source node initializes it at L. When a node carrying a message meets a node which does not, it hands over half of the carried packets. Nodes keep handing out half of their inventory until only one packet instance is left, at which moment they switch to direct delivery.

Both Epidemic Routing and Spray and Wait are very simple methods, that do not collect any information on the network to adjust their behavior. They are an example of *infrastructureless / dissemination based* algorithms.

2.3.4 PROPHET

A widely used and studied protocol for ON is PROPHET (Probabilistic Routing Protocol [13]). This protocol is specifically proposed for DTN type networks, and as such uses the various DTN RFCs. In particular, the data units transmitted are DTN *Bundles* (see 2.1.1). This protocol is based on Epidemic Routing, with provisions made for the fact that in a real network nodes usually do not move completely randomly. To take advantage of the movement patterns of nodes, a new probabilistic metric is maintained by the nodes, the *delivery predictability*, in the form of a vector of predicted delivery chances for each destination. These probabilities $P_{(node,destination)}$ are included in the summary vector to be exchanged when nodes meet, and are updated trough a technique known as gossiping: the delivery probability for a destination in the incoming summary vector is used to update own probabilities. This is done using the following formulas:

$$P_{(a,b)} = P_{(a,b)_{old}} + (1 - P_{(a,b)_{old}}) \times P_{init} \tag{2.1}$$

$$P_{(a,c)} = P_{(a,c)_{old}} + (1 - P_{(a,c)_{old}}) \times P_{(a,b)} \times P_{(a,c)} \times \beta \tag{2.2}$$

$$P_{(a,b)} = P_{(a,b)_{old}} \times \gamma^k \tag{2.3}$$

The equation 2.1 shows the reinforcement behavior, which raises the delivery predictability of a node if it meets a given destination frequently, driven by a configuration coefficient P_{init}. The reinforcement can also be transitive, applying equation 2.2 to the entries in the predictability vector of the encountered node. In this case, the speed of the reinforcement is regulated by a configuration coefficient β.

The delivery predictability ages, slowly decreasing if a given destination is not seen. This happens following equation 2.3, where γ is the configurable aging coefficient, and k is the number of time units since last aging.

These probabilities are used to decide which messages to request from a passing node. There lays the main difference with Epidemic Routing, which requests all messages not already possessed, whatever their target.

The exact policy used to determine whether a message is going to be requested (based on it's delivery predictability) is configurable. The alternatives are[14]:

GRTR (default) A node delivers a bundle to every node with a higher delivery predictability for its destination.

GTMX As GRTR, but with a configurable maximum number of replications.

GTHR A node delivers a bundle to every node with a higher delivery predictability for its destination (as GRTR), or with said delivery predictability higher than a threshold.

GRTR+ Deliver only to nodes with delivery predictability higher than the maximum seen thus far.

GTMX+ Like GTMX, but with maximum delivery predictability tracking as in GRTR+.

GRTRSort Bundles to transmit are selected as in GRTR, but are also transmitted in a specific order. The ordering criterion is that bundles with greater differences in the predictabilities are transmitted first.

GRTRMax As GRTRSort, but the ordering criteria is that higher predictabilities of the receiving node are transmitted first.

Once enough messages are requested, the buffer becomes full. After that, in order to store a new message in the buffer some other must be dropped. As noted before, Epidemic Routing uses simple FIFO for message buffer management, which has certain limitations. To address that, PROPHET proposes several policies, selectable at configuration time [14]:

FIFO This is the default policy. Oldest bundles are discarded first.

MOFO - Evict most forwarded first The bundles that were forwarded most times are discarded first.

MOPR - most favorably forwarded first A special coefficient is defined for each destination, which is updated on each replication as $FAV = FAV_{old} + (1 - FAV_{old}) \times P$, where P is the delivery predictability of the receiver. Bundles with highest values of FAV are discarded first.

Linear MOPR - most favorably forwarded first As MOPR, but the coefficient is updated as $FAV = FAV_{old} + P$.

SHLI - Shortest life time first Bundles have an associated lifetime value. Bundles with less remaining time are discarded first.

LEPR - least probable first Between all the bundles transmitted more than a configurable MF times, discard the ones with the lowest delivery predictability first.

PROPHET is the first algorithm studied that attempts to learn the behavior of the network to make better decisions. It does so trough gossiping, a widely studied and used method for propagating information trough a network.

A criticism of the solution is that the profusion of selectable behaviors and parameters, while providing great flexibility for different scenarios, makes it very difficult to setup correctly. The difficulty is increased by the fact that the impact of the different options is not particularly obvious.

An important contribution of PROPHET is that it is one of the few DTN platforms to be actually tested in a real-life application [15]. In that application, a DTN deployment attempted to provide connectivity to Sámi reindeer herders in Laponia, northern Sweden. Besides testing out the routing, it used

a full DTN stack to provide two Internet based services, E-mail and cached Web browsing. Additionally, a fully contained ON service was implemented for debugging, NSIM (Not So Instant Message), that allowed to send text messages between nodes. Interestingly, NSIM gained popularity and became one of the most used means of communication in the DTN. At the same time, the experiment identifies a mismatch between the DTN and the e-mail service. The first (and lesser) difficulty is that the protocols used for delivering mail (SMTP, POP, IMAP) are interactive, and thus not well suited to bundling. The second, and main, difference, is architectural: the client-server nature of e-mail forced messages to go trough a gateway, even when addressed to an immediate neighbor. This is an important insight on the limitations of the DTN architecture and the need of new applications for taking full advantage of a ON.

2.3.4.1 Improving PROPHET

PROPHET serves as a basis for further improvements. For example PROPHET+ [16] proposes enhancing the delivery probability with a weighted function that takes into account several other attributes, namely Buffer Parameter (expressing available buffer space), Power Parameter (remaining power available), Bandwidth Parameter (available bandwidth in the pair communication) and Popularity Parameter (the rate of successful message exchanges for the node).

The exact coefficients used for the weighted sum are fixed at configuration time and should depend on the nature of the network. The authors propose the use of simulations for estimating optimum values.

This is a good example of increasing the flexibility of the algorithm at the cost of further increasing the amount of configurable options.

2.3.5 Timely-contact probability protocol

In [17] the authors present another variation on epidemic routing. The protocol attempts to take into account historical encounter information to predict a encounter probability in a future time slot. For this purpose, it divides the encounter history with each node in time slots, or segments. The probability of meeting with a node inside a future time slot is estimated as the proportion of slots with registered meetings to the total number of recorded slots. The meeting history must be kept in the node, and old segments get discarded to limit storage consumption and adapt to changes in contact patterns. In opposition, PROPHET keeps a single number for each destination.

During a meeting, a candidate node is selected to relay a message if its contact probability is above a threshold. This probability is computed as the sum of the probabilities of all possible paths across the nodes in the network. For this purpose, each node keeps a matrix of encounter probabilities between each pair of nodes. When two nodes meet, they exchange their matrices and update their own. This matrix is used to compute the accumulated encounter probability. This is heavier than PROPHET, which exchanges a vector of nodes with their respective encounter probabilities. Also, no explicit computation of

possible routes is done in PROPHET, instead it relies on the transitive formula 2.2.

2.3.6 RAPID

Unlike other ON algorithms, RAPID[9] attempts to explicitly calculate the effect of replication on the selected routing metric. RAPID is a utility driven algorithm: a per-packet *utility function* is used to select packets to replicate as to maximize the increase in the local utility.

For example, if the routing metric is *minimize average delay*, the utility function of a packet is the negative of the expected time to deliver said packet. This estimation is made locally, with the help of a control plane for gathering information from other nodes. RAPID will replicate packets in decreasing order of their contribution to the total utility per resource used (marginal utility).

RAPID has three core components:

Selection algorithm refers to the estimating the utility value of packets candidates to be replicated, and their sorting according to decreasing marginal utility.

Inference algorithm contains the heuristics used to estimate the utility value of a packet. These are derived from the routing metrics used (such as *minimize average delay*, *minimize missed deadlines*, *minimize maximum delay*, etc.) To help in this task, summarized information from the nodes is propagated trough the network using a control channel.

Control channel is used to disseminate meta-data on the network state, in-band using a fraction of the total bandwidth. Data shared includes acknowledgments, average size of past encounter opportunities, local estimation of delivery delay for own messages, etc. Though the information is usually old and out of sync by the time it is used, simulations show that it's use improves the performance.

The experience with RAPID lead to several interesting conclusions. The first is that a DTN routing algorithm needs information on the network to approach optimum solutions. General algorithms will perform worse than algorithms that collect data on behavior. At the same time, DTN routing is shown to be a NP-Hard problem, only approachable with heuristics. A balance must be found between amount of meta information distributed and gain obtained from it.

2.4 Content Based Networking

There is a class of networked applications that rely on rich message addressing[18]. For example, Instant Messaging can maintain many-to-many discussions in real time; personalized news services can notify users according to sophisticated selection criteria; Service Discovery allows to gain access to new service providers

according to type, location, etc. without having prior knowledge of them; monitoring tools and sensor networks that allow retrieving information from dynamic and weakly structured networks using powerful and complex criteria, like location, type and values of data, etc.

Traditional IP addressing only provides destination-routed unicast, multicast and broadcast communications, which is cumbersome to implement that kind of applications in a efficient way. Thus, a class of middleware appeared, the *content-based publish/subscribe systems* (such as CORBA Notification Service, Java Message Service, COM+ Event System, amongst others). Usually, these systems are implemented on the application layer, and only solve the application-side of the solution: they provide users means to implement rich addressing applications. Meanwhile, the underlying network support is still traditional IP addressing and the frameworks are not actually distributed, having typically Client-Server architectures. This brings problems of scalability, as all messaging must go trough one or a few nodes (*message brokers*), which lay on the critical path of all communications and thus become bottlenecks.

In [18] the concept of *content-based networking* is introduced. At the physical architecture level it is identical to a traditional network: it is modeled as a non-directed graph of nodes connected trough communication links. Nodes with multiple links are called *routers*, nodes with a single link are *hosts*. The main difference with traditional networking is at the *service model*. Nodes are not assigned unique addresses, which are not used by the source when sending datagrams. Addressing is specified by the receiver, advertising a *receiver predicate* (*r-predicate*) which specifies the messages the node is interested into. Whenever a node pushes a datagram into the network, it will be routed to all nodes that advertised a *r-predicate* that match the datagram.

Thus, if we denote the universe of datagrams with D, the universe of predicates is $P : D \to \{true, false\}$. A predicate P_n (predicate advertised by node n) is a *Content-Based Address* of node n. Any datagram d such as $P_n(d) = true$ is considered as *cb-addressed* to node n. From this can be seen than a node can have multiple *cb-addresses* and *cb-addresses* from different nodes can have identical predicates. Consequently, *cb-addresses* can not serve as node identifiers in the general case.

To define an addressing scheme, two models must be defined: a *Datagram Model* (a definition for D), and a *Predicate Model* (a definition for P). This is a very flexible method, and allows to model several usual addressing schemes. In particular, destination addressing such as used in a IP network is easily modellable: the *Datagram Model* is the definition of a IP packet, while the *Predicate Model* is the class of functions that only take the destination field as a parameter.

The concepts of *routing* and *forwarding* from traditional networking also have a related meaning in Content-Based Networking. The routing becomes the process of maintaining forwarding tables in the nodes, which contain predicates, trough the exchange of predicates and other routing information. The forwarding refers to the local decision in a node of choosing where to send a datagram, finding the matching predicates in the forwarding table.

2.4.1 Examples

A well known implementation of a content based network is CBCB (*Combined Broadcast and Content Based*) [19]. CBCB provides content based routing over an ad-hoc mesh network, and takes a two-layer approach. First, over the supporting mesh network a broadcast tree rooted at the message source is constructed. Then, this broadcast tree is pruned according to predicates leading to subscribers. Thus messages follow the branches from the broadcast tree that lead to the subscribers.

One of several methods can be used to build the broadcast trees, like a shared minimal span tree, or per source shortest path trees. Predicates are propagated upstream these trees to populate forwarding tables in the nodes.

An interesting design decision is that CBCB supports only one predicate per node. This simplifies the protocol, without losing generality, as a predicate can be the result of the combination of several predicates. A service that supports the addition and revoking of separate predicates and handling of messages to them can be implemented on top of CBCB.

We developed a content based routing platform, RNR (Router-based Notification Router) [20]. The purpose was to provide content based networking for ad-hoc mesh networks using low cost consumer grade wireless hardware. For this, RNR attempted to use as little resources as possible. It is somewhat similar to CBCB in that it combines a broadcast tree and uses predicates to direct messages, but the broadcast tree is rooted at the subscriber and is built by the same process that floods predicates. Predicates are flooded from the subscriber, and the flooding process has the side effect of building a coverage tree that leads to the subscriber along a shortest path (similar to how a *convergecast* works). This provides for a very simple and compact implementation.

2.5 Content-Based Opportunistic Protocols

There are two forces motivating the combination of content-based and opportunistic networking.

The first comes as a intent for further increase the robustness of a content-based mesh network. Just as a mesh network attempts to add support for mobility and resilience to sporadic link failure, opportunistic networking can be seen as a method to protect against frequent failures that lead to network partitions.

The other motivation happens from the need of providing more powerful primitives for applications running over a pure opportunistic network, such as a vehicular or sensor network. For example, it is frequent for a user of a sensor network to be interested on certain classes of readings from sensors, classes defined over the attributes of the readings: their values, geographical location, etc.

Content-based opportunistic network is a relatively new research branch, and there are few actual solutions described. Of these, we do not know of any

instance of real life deployment. Also, we identified some serious weaknesses in the algorithms, which were part of the motivation for developing our own system RON. This will be commented in more detail in 4.2.2.

2.5.1 Guidec, Maheo et al.

In [21] a Content-Based Opportunistic Protocol is presented. It follows the general principles behind Autonomous Gossiping[22] in the sense that it combines Gossiping message propagation with an autonomous policy for message selection.

It is optimized for reducing the consumption of resources, particularly the wireless medium. For this purpose it minimizes the global amount of data transmitted, and uses broadcast rather than unicast transmissions. The idea behind using broadcast transmissions is that they are less costly than rounds of point-to-point unicast messages. The reduced reliability of multicast is compensated by the opportunistic nature of ON: instead of relying on a reliable point-to-point protocol, a new opportunity for repeating a failed transmission can be offered by the opportunistic behavior.

The protocol is oriented to documents, and is not fully content-based in the sense that the content based behavior is applied to a document *descriptor*, a list of attributes that is associated to each document. The interest of a host on certain documents is specified by a *interest profile*, which is a predicate that selects descriptors. Documents are stored in an internal *cache*.

The protocol is based on nodes announcing its interest profile and the content of its own cache, and reacting to other node's announcements. The algorithm has three main phases:

Announcement Periodically, each node broadcasts a message containing it's own profile, and a list of descriptors of locally cached documents (or *catalog*). When a node receives such a message, it will store the profiles. This list of profiles is used to optimize the transmitted catalog: only descriptors that match some previously seen profile are included. To adapt to changes in the neighborhood, the list neighbor's profiles is reset each time the node makes an announcement.
When a node listens an announcement that contains a descriptor that matches node's profile and is not yet in the local cache, it will request the document trough a separate unicast channel (to avoid triggering answers from several nodes).

Request processing When a node receives a document request (trough a unicast message as seen above), it will broadcast said document. The document is broadcast because there can be several requests for a single document, and a broadcast can satisfy all of them in a single transmission. Each document is transmitted not more than once between announcements.

Document receptions When a node receives a document broadcast, it will check it against it's own profile. If the document is of interest, it will will

store in the local cache. An interesting property is that it is possible to overheard a document broadcast without actually requesting it, from a dialog between other nodes.

Later work by the same authors[23]added the capability of using multi-hop message exchange for better coverage of connected segments of the network that could appear. It is based on multi-point relays (MPR), and is inspired in OLSR.

In our opinion, a weakness of this protocol is that the local cache contains only documents that match the profile of the node (this property is inherited from Autonomous Gossiping). This precludes a node of being a carrier of information if it has no local interest in it, even if it's mobility pattern makes it very adequate for shuttling information between a pair of nodes. This can result in bad performance if the profiles are highly specific and do not overlap between nodes, or the proportion of nodes with a profile is small.

2.5.2 CEPMF

The Content Encounter Probability Message Forwarding scheme is presented in [24]. In the CEPMF scheme subscribers spray the predicates into the network, which will be maintained in the relay nodes with an associated utility-value. This utility parameter will indicate the priority of the relay node to drive a message towards its consumer.

In CEPMF, a message is a set of attribute/value pairs, and a predicate is a logical constraint (in DNF) over the values of said attributes. For example, a message with the content:

[class="alarm", device="node10", alarm-type="lowdisk", alarm-value=10]

would match a predicate such as:

[class="alarm" $\wedge$ *alarm-type="lowdisk"* $\wedge$ *alarm-value<15* $\vee$ *class="alarm"* $\wedge$ *alarm-type="intrusion"]*

Subscribers distribute predicates to relay nodes using Binary Spray and Wait as described in 2.3.3. Each relay node associated a message encounter probability $ep \in (0, 1)$ which will be used to evaluate the probability that the relay can transmit a matching message to the predicated creator. The ep parameters is assigned a initial value when the predicate is registered, and then is reinforced when the predicate is met in the network (equation 2.5), and ages when this does not happen (equation 2.4).

$$ep_{new} = ep_{old} \times e^{-(t-t_0)} \tag{2.4}$$

$$ep_{new} = ep_{old} + (1 - ep_{old}) \times ep_{init} \tag{2.5}$$

This allow the node to learn how good it is to relay for the different predicates, knowledge which will be used while forwarding messages. This is similar to what PROPHET does (see 2.3.4), but with an important difference: PROPHET

does not use Spray and Wait, but floods predicates limiting the propagation naturally as the encounter probability gets too low. Also, from comparing equations 2.1 and 2.5 can be seen that PROPHET takes into account the encounter probabilities of both emitter and receiver, while CEPMF does not: it applies a fixed reinforcement on each meeting instance. This means that once predicates are available in all nodes, the reinforcement stops being useful.

Special care is taken with the handling of the possible overlaps between predicates. When a node has overlapping predicates, a message that satisfies the intersection of the predicates will satisfy both predicates. Thus there is an extra advantage in carrying such a message. This behavior is implemented by splitting two overlapping predicates, P_1 and P_2 in three predicates: $P_3 = P_1 \cap P_2$ for the intersection, and $P_1 = P_1 - P_2$, $P_2 = P_2 - P_1$ for the disjoint parts. The encounter probability of the intersection is higher than of each original predicate, and is calculated as $ep_3 = ep_1 + (1 - ep_1) \times ep_2$.

Messages are tagged with the *ep* value of the predicate they match on the relay. On the message emitter, the *ep* of the message is initialized at 0, and a relay will accept a message if the message's *ep* value is less than the *ep* of a matching predicate. This way, messages follow a path along relays with increasing values of *ep*, toward the subscriber. Also, message propagation is limited with a binary TTL (similar to the one used to propagate the predicates).

Message propagation is triggered by a node (source or relay) broadcasting own content messages. This broadcast is made periodically. When a listening node receives a message that matches some predicate with a greater *ep* than the message's *ep*, it will store the message in the local cache, update its *ep*, and send a unicast acknowledgment to the source. This *ack* is used to signal the emitter that the message was accepted, so it must reduce the binary TTL. It also inhibits the relay from broadcasting the message again for a while.

The fact that nodes periodically broadcast its message inventory makes this algorithm heavier that the one described in 2.5.1 or PROPHET, which are based on broadcasting the (much smaller) list of predicates.

2.5.3 CAR

Context Aware Adaptive Routing for Delay Tolerant Networks[25] focuses on a DTN topology consisting of a set of disconnected clouds, with some mobile nodes that migrate between them. Inside each cloud a proactive mesh routing protocol is used, such as Dynamic Destination-Sequenced Distance-Vector (DSDV). Communication between clouds is done in a forwarding mode, this is, a packet is trusted to a single mobile node to be relayed to remote clouds.

CAR chooses a node to carry a message using Kalman Filter predictions and multi-criteria decision theory. To calculate a delivery probability, a node starts by calculating a utility function as a weighted sum of several *context attributes*. Basic CAR uses two attributes: a change rate of connectivity (number of connections a disconnections in a lapse of time, measures the mobility), and co-location (whether the node is the same cloud with another node). Other attributes can be added, such as battery level. The weight assigned to each

attribute is itself variable as a function of the attribute and its value. For example, the weight assigned to the battery level rises as the battery level reaches a critical value, or the co-location attribute can be ignored completely if it's value is deemed unreliable by the Kalman predictor.

Between updates of information, CAR uses Kalman Filters to predict the evolution of attributes. This allows to estimate the value of attribute when there are no updates from neighbors (like when the node is disconnected), and to reduce the frequency of exchanges of context information between nodes.

Forwarding is done to a configurable fraction of available links. Firstly, information is used on nodes visible trough DSDV. Then, if there are remaining links to explore, prediction is used. Finally, if there are links available, remaining copies are forwarded randomly.

As seen in the algorithm by Guidec, Maheo et al. (see 2.5.1), CAR also mixes a Carry and Forward with Mesh routing, where nodes besides having a buffer also maintain an ad-hoc mesh network (MPR in the former, DSDV in the later). In this case it is a natural off-shot of the view of the networks as a mesh network with occasional partitions, typical of DTN inspired work, and not as a collection of mobile nodes that only meet occasionally.

2.6 Conclusions

Amongst the context-aware opportunistic routing protocols, two families can be identified (see Table 2.2). The first group uses some method to predict the deliverability to a certain destination using information collected on the behavior of other nodes in the network. This is somewhat similar to link-state routing, and usually uses some control channel to disseminate extra routing information. Examples of this kind of algorithm are RAPID, Timely-contact Probability and CAR. These are highly complex algorithms, that resort to some sophisticated methods, such as the distributed calculation of marginal utilities or Kalman Filters. These algorithms have necessarily complex implementations, and bring doubts to their manageability, specially in the light of the difficulties with other, much simpler, algorithms.

The second group of algorithms uses a simple epidemic method, more in line with a Distance-vector routing algorithm. Each possible neighbor is assigned a coefficient which represents an abstract "quality" for it, expressing how good the node is to deliver information to it. This coefficient is adjusted following very simple rules, usually increasing as the node gets in touch with a destination (directly or trough mediators), and decreasing as time passes without encounters. Examples of algorithms in this group are PROPHET and CEPMF.

For the purpose of a real world deployment on devices with limited computing power, the second group is considered more promising. Nevertheless, all the reviewed solutions have serious shortcomings. In particular, the two Epidemic/Content Based algorithms have unsatisfactory forwarding management: the algorithm by Guidec, Maheo et al. does not support carrying information not addressed to the node, and CEPMF is based on periodical broadcasts of the

	Addressing	Routing	Forwarding	Buffer Management
Direct Delivery	Destination-based	none		
Epidemic Routing	Destination-based	Epidemic + hop count	Summary vector, along increasing node id	FIFO
Spray and Wait	Destination-based	Two phase, epidemic then direct delivery		
PROPHET	Destination-based	Delivery Predictability (reinforcement / aging)	Along increasing DP, other options	FIFO, other options selectable
Timely-contact probability	Destination-based	Matrix with possible paths and timeslots with encounters	Delivery probability over a threshold	Delivery probability over a threes hold
RAPID	Destination-based	Utility driven, control channel data dissemination	Sorted according to marginal utility of replicating	Lowest utility deleted first
CAR	Content-based	Utility + Kalman filters + DSDV	To a fraction of available links	FIFO
Guidec, Maheo et al.	Content-based	Autonomous Gossiping	Profile + Catalog announcements, binary TTL	n/a
CEPMF	Content-based	Encounter Probability (reinforcement / aging)	Broadcast inventory, messages along increasing EP	Binary TTL + aging

Table 2.2: Opportunistic Protocols

inventory, which is potentially very expensive.

This suggests the creation of a new algorithm, which is Content Based, uses a Epidemic-like routing relying on the broadcast of the profiles, and takes advantage of the broadcast medium when forwarding messages.

Chapter 3

DEMOS Project

DEMOS project (Domestic Environment Monitoring with Opportunistic Sensor networks) [26] aims to improve the environmental monitoring in areas with little infrastructure, with minimal cost. DEMOS takes advantage of the OLPC-like programs to develop a low-cost platform for environmental sensors, such as air-quality sensors, that are deployed at the living premises of children in environmentally vulnerable neighborhoods as well as at their schools, parks, etc.

The main idea is to use children's laptops to collect and carry information from widely deployed environmental sensors, as they move to and from school, in the neighborhood, etc. The fact that sensor nodes do not need to have a connection to Internet but only need a simple Wi-Fi interface allows to reduce cost, and adds flexibility to their placement.

The DEMOS project is the main driver behind the present work. As will be seen, the solution developed for DEMOS project depends on the existence of an opportunistic network, used to shuttle data between sensor nodes and collection points.

3.1 DEMOS System

To support the DEMOS project we have developed the DEMOS system. The DEMOS system is a policy-based, self-managed system that consists of a set of services deployed on many network devices or nodes. Depending on which services are deployed on a node it falls under some of the following categories:

Sensors. These are the nodes that collect environmental data. They are usually fixed to a place, and they have attached sensing hardware. Different nodes can have different sensing hardware attached.

Collectors. These nodes are usually placed at schools, and are the recipients of data generated by Sensor nodes. Data is collected and relayed to a central Management Station through the Internet.

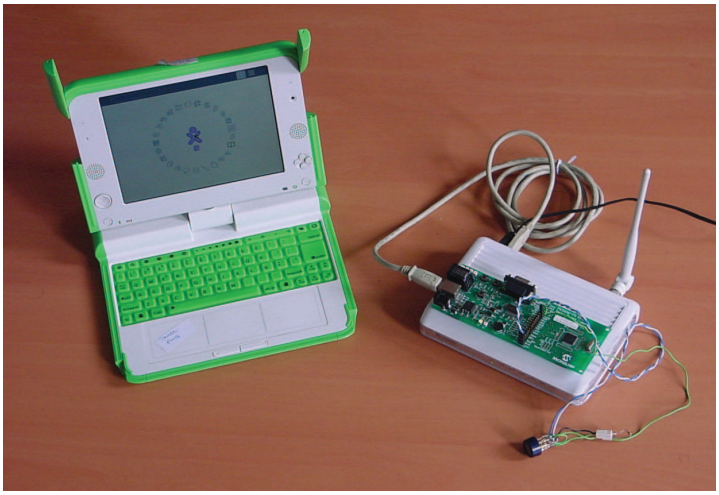


Figure 3.1: A Carrier Node and a Sensor Node

Carriers. These are the mobile nodes that relay information from the Sensors to a Collector in an opportunistic fashion. Usually these are XO laptops owned and therefore carried around by children.

Management Station. All data is summarized and analyzed here. Additionally, the DEMOS management rules are edited, compiled and deployed from this node.

3.2 Node Architecture

The general architecture of a DEMOS sensor is depicted in Figure 3.2. It is an application of the RAN Architecture [27], and possess three main services. The first is LuPA, a set of components for rules-based management. The second is Rmoon, a monitoring service. And finally RON, a common data bus which is used to exchange information between the instances of the first two services, on the same node or over the network.

3.2.1 RON

RON (RAN Opportunistic Network) is a common data bus for exchanging messages between nodes and services. It was developed for providing content-based publish-subscribe delivery in Opportunistic Networks, and can interoperate with a mesh oriented bus, RNR. In particular, this protocol is responsible for delivering sensor readings from sensor nodes to collecting points. It must be present in all nodes participating in the DEMOS network.

For a full description, see Chapter 4.

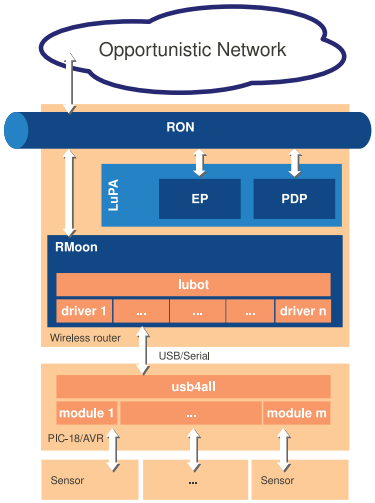


Figure 3.2: DEMOS Node architecture

3.2.2 Rmoon

Rmoon[1] is a service that monitors the state of a device, and triggers "trap" notifications when certain conditions occur. Those notifications are delivered trough the data bus to interested parties in a content based fashion. Therefore, Rmoon is the main supplier of events for the *Policy Decission Point* (PDPs) distributed over the network.

In DEMOS, the main responsibility of Rmoon is to monitor the readings of environmental sensors attached to a sensor node, and push readings of interest in the network. To setup a monitoring node, an administration station (usually a collection point) must send a "command" notification to one or more Rmoon services. The sample notification shown in figure 3.3a would configure a sensor node to trigger a message when a gas level goes over 0.7, and send out a reading at least once each 6 hours. Each such condition set up is called a *watcher*.

After setting up a watcher, a collecting point should activate a subscription that will route notifications of interest towards it. The sample subscription in Figure 3.3b will select any message of type "trap", carrying a gas sensor reading. Notice that the subscription does not specify the sensor node, so in this case the collector is subscribing to all sensor nodes, whoever set them up, as long as they generate gas level readings.

A Rmoon service will periodically monitor the readings of the attached sensors, and trigger "trap" messages as correspond. A sample trap can be seen in Figure 3.3c.

[1]The name RMoon is a pun with the well known RMON (Remote Network MONitoring) and the Portuguese meaning of Lua (moon).

```
NOTIFICATION
notification_id=123
source=coll1
target_node=sensor5
target_service=rmoon
message_type=action
command=watch_mib
watcher_id=gas_alarm
mib=gas_level
op= >
value=0.7
timeout=21600
END
```

(a) Setting up a monitor

```
SUBSCRIBE
subscription_id=2
subscriptor_id=coll1
FILTER
message_type=trap
mib=gas_level
END
```

(b) A Subscription

```
NOTIFICATION
notification_id=127
source=sensor5
service=rmoon
timestamp=1313175897
message_type=trap
watcher_id=gas_alarm
mib=gas_level
value=0.71
END
```

(c) A Sensor reading

Figure 3.3: Sample exchange with a Rmoon service

Rmoon is a generic service that is easily extensible. For the DEMOS project we extended Rmoon with support for external sensors. For this purpose we interfaced with a microprocessor software framework developed in our research group, called USB4ALL[28]. USB4ALL is a modular firmware that provides a high level communication mechanism. It allows the controller to discover installed modules, load and unload them at runtime, and query them trough an RPC-like mechanism. Originally developed for the Microchip PIC-18 series of microcontrollers, it has been also ported to AVR based Arduino platform.

The client side is implemented in a library called *lubot*, written in Lua, and thus is also highly portable. The only native code needed on the Rmoon side is a Lua - libusb binding or a small message oriented serial library (depending on how the microcontroller board is attached, trough USB or serial link).

Besides monitoring external sensors Rmoon can monitor the state of the node. For example, it could be useful to get information on battery levels or network conditions. It must be present at least in all sensor nodes, to collect data from external sensors. It also may be run in any other node which should be monitored for other purposes.

3.2.3 LuPA agent

LuPA (Lua Policy Engine) groups the policy-related services or functionalities. Following PBNM (Policy Based Network Management) architecture [29] it includes a PDP (Policy Decision Point) and an EP (Enforcement Point). Both components can be installed together or separately, according to the role of the node.

LuPA implements a general purpose decision engine. It models a management problem as a control theory problem: there is a dynamic system (the

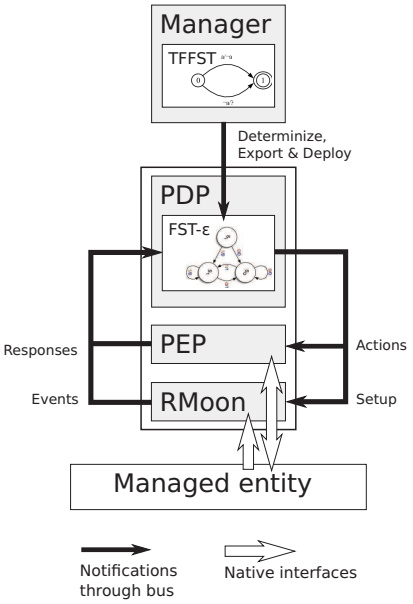


Figure 3.4: Message exchange between DEMOS components

managed entity), a set of measures that can be obtained from the system, a set of inputs (actions) available to manipulate the system, and a controller that computes these inputs from the measurements. A scheme of the information flows can be seen in Figure 3.4.

System inputs and outputs are transported trough a content based notification bus. The controller is implemented as a rule-based decision engine, allowing high level descriptions of behaviors. There can be multiple instances of the controller, potentially with different sets of rules, and the notification bus supports the exchange of messages between nodes. This means a node can take decisions based on remote information, and these decisions can impact remote nodes, so the management can be distributed.

3.2.3.1 PDP - Policy Decision Point

PDP's task is to take autonomous decisions, based on the stream of Notifications it receives trough the Notification Bus.

The rules engine is actually split in two: a central rules management service, and a multitude of rules execution services (the PDP itself). In the central rules management service rules are administered, combined, optimized, conflicts are resolved, etc. [30]. Then the resulting rule-set is transformed to a representation that can be efficiently executed in low-power nodes and deployed there. This compact representation is called a *policy file* and is pushed to the PDP agents in a command notification trough the Notification Bus.

The policy file is a Lua program that implements a Finite State Transducer with ε-transitions (FST-ε), accessible trough function calls. Both the input and output of the transducer are notifications.

When a policy file is received, the PDP executes it in a special sandboxed environment. As the policy is initialized, it can emit some notifications, for example for setting Rmoon watchers, or subscribe for some classes of messages. After the policy is set up, the PDP starts triggering it with notifications. The state is kept internally in the policy script, and the PDP must not know how it is implemented, only that call hooks are respected.

The PDP maintains a list of recent notifications, which is shared into the policy script's environment. This list contains a sliding window of the last arrived notifications. To simplify the state machine, the notifications can be ordered in the window by a fixed ordering rule. In this way, the state machine does not have to handle every possible permutation of the notifications of interest.

The size of the sliding windows can be set at configuration time. For this purpose there are two parameters: a length of time in seconds, and a maximum number of notifications in the window. Also, there is a special category of notifications: "happening notifications". Those are notifications that signal permanent change in some attribute, and thus must be kept in the window for it to be correctly processed. For example, a decision could depend on whether a given service is active or not, thus the hypothetical "going up" and "going down" notifications from that service could be marked as happening to keep them in the window. This way, the state machine could count on the presence of said notification no mater how old it is. Marking and unmarking notifications as "happening" is policy's responsibility, and the PDP will comply while maintaining the sliding window.

By expressing the state machine as Lua code, there is no intermediate representation for the state machine and thus no special parsing. All the parsing work is done in the central server, and the PDP just executes it.

When a state machine recognizes a pattern of notifications, it signals it sending one or more notifications to the common bus. Said notification can be a corrective action sent to a node (in the classical PBNM fashion), or an abstract notification signaling a complex event. Said event would be the result of correlating multiple event sources, and could be used by a higher level policy (residing in the same or a remote PDP).

This allows to implement a hierarchy of policies, where node-local events are processed directly on the node, and network-wide conditions are handled in a dedicated management node. Where and how many PDP instances there are depends on the kind of rules are expected to be used. For example, to use a PDP to recognize certain geographical patterns in sensor readings, the best place would be a collection point. If the PDP is to manage a node, like switching off power hungry sensors when power is lost and running from batteries, the PDP would reside in the sensor node itself.

3.2.3.2 EP - Enforcement Point

The EP is an ordinary enforcement point, providing functionality to configure a node. The EP subscribes in the message bus for messages with `message_type = action` that are directed to it, and implements several actions to modify device specific settings such as network configurations or starting services. For example, an EP could be used to switch a particular sensor hardware off and on to save battery. The EP is easily extensible trough shell scripts.

It also provides a `get_mib` action, which allows to query the node on its internal state. It's possible to query on, for example, the CPU load, free RAM, or a reading of an attached sensor.

Chapter 4

RON Opportunistic Network

RON (RAN Opportunistic Network) is a implementation of a Content-based Opportunistic Network algorithm[1]. The algorithm collects the best ideas of the State of the Art, and brings some novel ideas that solve shortcomings and provide new functionality.

4.1 Requirements

During the development of the DEMOS project (see Chapter 3) the need for a opportunistic network appeared. DEMOS envisions a sensor network deployed in urban or rural settings, with the objective of monitoring environmental variables. Sensors themselves have very low costs and are widely deployed, with the possibility of adding new sensor nodes as need arise. Data collection would be assured by the normal daily movements of kids carrying their XO laptops, from the country's OLPC program, Plan CEIBAL [31]. As the kids pass near sensor nodes on their way to or from school and elsewhere, data would be unloaded to the laptop and be carried around until (opportunistically) delivered to a central collection point, possibly installed in a school.

From this application a set of requirements was derived for a supporting opportunistic platform.

Content-Based Notification Bus

When dealing with an unplanned network of devices, such as the DEMOS sensor network, the maintenance of an inventory containing addresses and configurations, setups, software updates, etc. of such devices adds to the management burden. Moreover, without such an inventory, configuring and reconfiguring each sensor to adapt to changes in the interests of the operators and users of the network or to changes in its layout may be an impossible task.

[1] The sources are available at https://github.com/xopxe/Ron

In this context, content-based routing appears as an interesting solution. From previous work we had experience developing a distributed policy-driven management platform RAN [27], which originally worked over a mesh network using our mesh network content-based notification bus, RNR (see Section2.4.1).

In RAN architecture, all messages are delivered through a publish-subscribe notification bus. Subscriptions are used to route notifications to interested nodes, and notifications are used to distribute data. Notifications could be used to send commands (for example, configuration changes) and traps or alarms (for example, upon detecting some error condition).

It was envisioned using the same technology, replacing the mesh network with an opportunistic one. Ideally, the new protocol would serve as a drop-in replacement and support RAN's operation transparently. Also, the new protocol should interoperate with RNR to support mixed network deployments.

Lightweight

The platform was to be run on devices with low computing and storage power, and frequently running from battery. Sensor nodes were to be linked to consumer grade wireless routers, with USB or serial ports for interfacing the sensor hardware. Such devices run with as low as 16Mb RAM and 4Mb flash storage, with 200MHz CPUs. Even less powered sensor nodes, microcontroller based, could be expected to be used. Mobile nodes were to be XO laptops. These laptops, in their most frequent configuration at the moment, were powered by AMD Geode CPUs, with 256Mb of RAM and 1Gb of flash storage.

Efficient

Due to the low computing power and storage available, the algorithm should be very efficient in resource utilization.

Another difficulty was related to adverse radio conditions. Sometimes, the transmission windows could be very short, for example, a kid with a XO laptop on a bus driving by a sensor node installed on a utility pole. Other times there would be a very dense wireless network with a lot of active nodes in range and lots of collisions (kids near or in the school). Thus, the protocol should make the best use of the airtime, minimizing redundant transmissions and taking advantage of fleeting pairing opportunities.

4.2 RON Protocol

After reviewing existing options, it was found that none was really adequate to the envisioned use, or had some serious shortcomings. Nevertheless, some basic ideas were considered sound and it was decided to develop a protocol that used the best of the available alternatives, and implemented some of our own. The result is RON (RAN Opportunistic Network).

Algorithm 4.1 RON algorithm skeleton

```
1  N : Array[#N] of Notification
2  V : Array[#V] of {Subscription, quality}
3
4  each t seconds
5          broadcast( V )
6
7  when received_view( V' )
8          V.merge( V' )
9          N' = matching( N, V' )
10         broadcast( N' )
11
12 when received_notifications( N' )
13         N.merge( N' )
```

4.2.1 Overview

The main idea is that RON is a fully broadcast based protocol. All messages are sent without an explicit destination, and any node listening a packet makes a local decision on how to use that information. The rationale behind this decision is that as the wireless medium is naturally broadcast, unicast communications are wasteful: the medium gets occupied for all nodes in range anyway. Also, network maintenance is realized trough periodic broadcasts of a subscription-quality vector.

The algorithm is very simple (See Figure 4.1). It is patterned after a Gossiping Algorithm, with an active thread which transmits periodically, and a passive thread that processes incoming messages and generates responses.

RON manages two types of entities: Notifications and Subscriptions, which are the *datagrams* and *r-predicates* as described in Section 2.4, respectively. The implementation of these entities is described in more detail in Section 4.2.3. There are two kinds of messages being emitted: a View, which is the vector of Subscriptions with associated qualities (triggered from the active thread), and a set of Notifications (triggered from the passive thread). Special care is taken to reduce the network utilization, as will be described in Section 4.2.5.

The specific RON behavior is determined by the function MATCHING and two MERGE functions, one for Views and one for Notifications. These will be described in more detail in Section 4.2.4.

This structure is highly flexible, and allows to experiment with different solutions for routing and forwarding tasks. The routing is handled by VIEW.MERGE function. It processes the *View* vector message, and decides how to use the routing information (the quality attributes) contained. Forwarding is handled by the MATCHING and NOTIFICATIONS.MERGE pair of functions. The first selects notifications to be broadcast, and the seconds chooses which to actually keep when notifications are received. Both sets of functionality (routing and forwarding) are weakly coupled: it can be seen that only the MATCHING function

uses both Notifications and Views vectors.

The RON solution provides implementations for all of these components, as will be described in following sections, and their behavior will be analyzed in Chapter 5. Nevertheless, it is possible to provide alternative solutions, and in fact the tests of RON will point out some possible areas of improvement.

4.2.2 Previous Work

The RON protocol shares some ideas with the protocols described in Section2.5, but with some important differences.

RON shares with the protocol by Guidec, Maheo et al. (see 2.5.1) the idea of broadcast of announcements, and the ability of taking advantage of overheard traffic. But as mentioned, said protocol imposes the restriction of nodes keeping only messages that match node's own profile. This can cause poor message distribution behavior. This is somewhat alleviated by the MPR functionality, where the nodes continuously try to detect connected network segments to allow multihop delivery, but the result can be seen as unnecessarily complex and cumbersome.

RON lifts the restriction on messages having to match node's profile to be kept. A RON node makes the decision on what messages keep based on the quality of said node to *deliver* the message, whether it is targeted to it or not. This allows for a much simpler protocol.

Also, the protocol 2.5.1 uses a separate unicast channel to actually deliver the document from one to another, while RON uses broadcast messages also for this purpose (allowing for overhearing messages). But this is related to the type of data each protocol is targeted for: the former is intended for relatively big documents, the later for small data units that fit in a single packet. If RON was to be used for a similar purpose (carrying big documents), a similar scheme could be implemented.

The method used by RON to propagate the qualities and predicates has a lot in common with CEPMF (see 2.5.2). Periodically a table is broadcast, containing a list of reachable predicates with associated qualities. Said qualities are maintained in the process of listening other nodes' broadcasts: as time passes without receiving data on some predicate, the quality ages (with a exponential law), and each time a predicate is received, the associated quality is reinforced. The reinforcement is done differently: CEPMF does not take into account the incoming quality associated to the predicate (as seen in 2.5), but RON does: the higher the incoming quality associated to the predicate, the stronger the reinforcement. As mentioned in 2.5.2, there are doubts the CEPMF implementation as described is workable at all.

CEPMF distributes predicates using binary Spray and Wait (described in Section 2.3.3). This requires confirmation from the receiver that the handover was actually accepted, as the emitter must change its own state when this happens. This behavior is hard to implement in broadcast, and CEPMF uses unicast for this purpose. RON does not use Binary Spray and Wait, and uses the natural decrease in predicate quality to limit their distribution. In this, RON

```
SUBSCRIBE
subscription_id=sid23
subscriptor_id=collector1
FILTER
mib=temperature
value > 35
END
```

(a) A Subscription

```
NOTIFICATION
notification_id=notif123
source=sensor_node1
message_type=trap
watcher_id=watch_temp
mib=temperature
value=36.5
END
```

(b) A Notification

Figure 4.1: Sample messages in the Notification Bus

behavior is more in line with PROPHET, adapted for the purpose of content based routing.

In CEPMF there are two types of periodical broadcasts: the *predicate-ep* table, used to maintain the encounter probabilities as seen above, and the messages from the local buffer for initiating the message forwarding. RON is more efficient in that it only periodically broadcasts the predicate's table, with message broadcasts being triggered by listening a predicate of interest broadcast. This way, notifications are transmitted only when there is a potential interested node in reach.

As described in 2.5.2, CEPMF transmits messages following a chain of nodes with strictly increasing encounter probabilities. This can cause problems of local maximums. For example, imagine there are two internally well connected network sectors, both interconnected trough a seldom seen node. If a predicate is published in one well connected region, it will reach the other region trough the connection node. Once there, nodes can raise the quality for the predicate above the level found in the connection node, as they are in a well connected segment (see equation 2.5). Once that happens, a notification published in that segment wont be able to cross back to the predicate publisher's region. On the other hand, RON makes the decision to keep messages comparing the relative qualities of the messages in the buffer. This gives better opportunities for such "weak" messages, as the bad movement pattern of the connection node possibly affects the other messages in the buffer, too. It is also possible to tune the comparison between messages in the buffer adding weights, heuristics, etc., to better handle these cases.

4.2.3 Publish-Subscribe addressing

As mentioned in Section 2.4, the basic entities of Content-Based-Networking are *datagrams* and *r-predicates*. These are implemented in RON as Notification and Subscriptions, respectively.

Both entities are encoded as plain text multi-line strings. A Subscription (fig-

Algorithm 4.2 Views merging pseudo-code

```
1  function V.merge( V' : Set of {Subscription, quality} )
2      for s in V' do
3          if s in V then
4              V.reinforce(s)
5          else
6              V.add(s)
7          end
8      end
9      for s in V do
10         if not (s in V') then V.aging(s)
11     end
12 end
```

ure 4.1a) has two sections. The first is a header or general information section, holding a unique id for the subscription and the unique id of the subscriptions issuer node. The header is used for administrative and network maintenance tasks. Then, after the FILTER label, comes the filter section (the *cb-address*). It contains a list of conditions that describe the notifications the subscriber is interested into. Each condition is of the form attribute-operator-value, and there is an implicit *and* between all the conditions. A Notification meets the subscription if and only if it has all the attributes mentioned in the filter, and their values meet the conditions. This implies that a subscription with an empty filter is satisfied by any notification. Notice that this is less expressive than the full BNF expressions usual in publish-subscribe platforms. This decision was driven to keep the implementation as simple as possible. Nevertheless, if the need arises RON is easily modifiable to support BNF, as the notification-subscription matching functionality is orthogonal to the routing functionality.

Notifications (figure 4.1b) are the messages that actually contain the information being transmitted in the network. They consist of a set of key-value pairs. The only mandatory field is *notification_id*, a unique identifier for the message. All the rest are free to be defined by the network user. In the example, the Notification does satisfy the Subscription, and thus will be delivered to the Subscription emitter.

4.2.4 Network maintenance

As the buffer space in the node is limited, decisions should be made as to what messages to carry. These decision should be made as to maximize the performance of the network. If the nodes do not move completely at random but following certain movement patterns, some nodes will be more adequate to transport some messages. For this purpose, a quality metric $0 \leq q \leq 1$ associated to each subscription is maintained on each node.

As shown in 4.1, a node periodically broadcasts it's View, the set of Sub-

scriptions it has with their qualities. A Subscription has a fixed quality of 1 in the node that published it. When a node listens a View broadcast, it uses it to update it's own View trough a VIEW.MERGE() call (line 8 of 4.1). The VIEW.MERGE (see pseudo-code 4.2) method does two tasks: first, for every subscription in the incoming message it will either add it to the local View (if not already present there), or update the associated quality (if already there). Then for every subscription in the local View but not present in the incoming list, it will degrade it's quality ("age"). The aging is done using using expression 4.1 and the reinforcement is done using expression 4.2, where γ and P are configuration parameters driving the aging and reinforcement velocities, respectively.

$$q_{new} = q_{old} \times \gamma^{-(t-t_0)} \tag{4.1}$$

$$q_{new} = q_{old} + (1 - q_{old}) \times q_{in} \times P \tag{4.2}$$

The reception of a View broadcast, besides updating the local Views, triggers the broadcasting of all matching Notifications. This is done with the help of the MATCHING function. This functions selects from the Notifications buffer all the notifications that satisfy a given subscription predicate using *one-way matching* (for details, see Section 4.3). This way, periodical View broadcasts are answered with Notification broadcasts, which is the way Notifications are propagated in the network. The redundant Notification transmissions protect against communication failures. There are several policies in place to reduce network traffic, as will be seen in 4.2.5.

Each time a node listens a Notification being broadcast, it must decide what to do with it, and this is done by the NOTIFICATIONS.MERGE method (see pseudo-code 4.2). If the node's buffer is not full, the Notification is simply stored, as there is no penalty in carrying it. If the buffer is full, it must be decided whether ignore the incoming Notification, or drop some Notification from the buffer to make space. For this purpose, an *accumulated quality* is computed, which is the sum of the qualities for all matching Subscriptions (not counting locally registered Subscriptions, as the node will not act as a Notification carrier for those). Between Notifications with equal *accumulated qualities*, the older are considered worse. Other policies can be enforced at this point, like guarantees of minimum and maximum times being carried for Notifications, weighing the Subscriptions contribution according to their importance, etc.

The buffer can be configured to reserve a portion for locally registered notifications. This allows to guarantee local notifications will survive enough time to be handled to other nodes.

4.2.5 Network resources conservation

Several measures are taken to reduce the resources utilization, specially the air time.

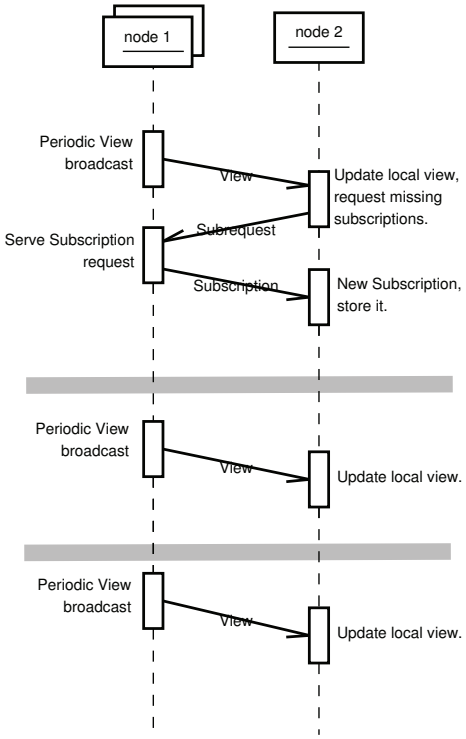


Figure 4.2: Subscription propagation and Views

Subscription propagation

The full content of a Subscriptions (i.e. full header and predicate) only has to be distributed once, as it will be then stored locally in each node. Trough the rest of the Subscription's lifetime only its unique identifier and quality are used, to maintain the Subscription qualities and trigger Notifications broadcasts, as described in Section 4.2.4. Thus, the VIEWS broadcast actually only contains the a list of identifiers and associated qualities. When a node sees a identifier for a subscription not in its local buffer, it will request it trough a separate SUBREQUEST broadcast containing said identifier. Any node that listens that broadcast and possess a copy of the Subscription will broadcast a SUBSCRIBE message containing the full subscription. This extra exchange is worthwhile as it is done only once per Subscription propagation step, and reduces the size of the periodical broadcasts considerably by omitting the predicates from then on (see Figure 4.2).

Broadcast storm control

Sometimes a single VIEW broadcast from a node can cause responses from multiple nodes at the same time, each one a broadcast. This behavior is called a broadcast storm, and can cause high losses in dense networks. For controlling this a simple policy is implemented. Each transmission is delayed a small random time, and at the same time nodes do not transmit data already seen in a small time window. This way, the first node to transmit will inhibit all others nodes in range from repeating the information. Notice that it is still possible for a node to receive several responses due the *hidden station* phenomenon.

This policy applies to Notification as response to View broadcasts and Subscribe as response to Subrequest broadcasts. For this purpose, all Subscriptions and Notifications are tagged in the nodes with a *last_seen* time stamp, updated each time they are seen in the network.

Redundant Notification broadcast reduction

When a node broadcasts a View, it can receive as answer Notifications it already has in it's buffer. To reduce this probability, nodes add to the view message notification ids they already have, as many as it fits in the message (as will be seen in section 4.3, messages have a size limit). Nodes upon receiving the View message will omit from the list of notifications to broadcast in response the ones mentioned in it.

This can be further improved using *Bloom Filters* (see Section 7.4).

Lifetime limitation

To avoid out of date entities from being a burden on performance, several measures are taken to limit their lifetime.

Notifications have a *emitted* counter associated, which counts the number of broadcasts made. It is initialize at 0 when a node accepts a notification, and is incremented by one each time it is broadcast. The Notification will be dropped when the counter reaches a *max_transmits* value.

When a Subscription quality drops bellow certain value, it is removed from the node.

4.2.6 Security considerations

Content Based networks present several security challenges when compared to destination routed networks.

The main difficulty is that in the general case it is not possible to use a tunneling schema to protect end-to-end communications. In such a scheme data is encrypted at the emitter and then decrypted at the receiver, thus traversing the network encrypted. This is not possible with RON in the general case because in a Content Based network intermediate nodes must access the content in plain form for the purpose of making routing decisions. This implies that messages must be decrypted and re-encrypted at each hop, which impacts the

performance and resource consumption. Nevertheless, a end to end encryption scheme could be supported if Subscription filters on encrypted fields were restricted to expressions testing only for equality: the expression should compare the value in the notification against a encrypted value. Only unencrypted fields would be compared for $>$, $<$, etc. If the network application could adopt these restrictions, a simple end-to-end scheme is possible, with no special support needed from RON.

Nevertheless, for full support of encryption for content based networking, routing nodes must participate.

As a proof of concept, RON implements MD5 signing on each hop, for both payload messages (Notifications), and all network maintenance messaging, like Views. The signing is done using a key specified in the configuration file.

Further analysis of alternatives for securing a DEMOS-like opportunistic network is presented in [32]. The work identifies existing approaches and their weaknesses. In particular, an unsolved problem identified is the key management in a highly dynamic and weakly structured network.

4.3 Implementation

RON is developed in Lua [33]. Lua is a compact virtual-machine based dynamic language, with great emphasis on extensibility. It is weakly typed and has a garbage collector. It is written completely in ANSI C and thus the core has minimal dependencies, and can be run in an extremely wide spectrum of platforms down to embedded microcontrollers. At the same time, it offers powerful facilities to programmers, like pattern matching, hash-based tables, functions as first class members, closures and lexical scoping.

Besides being virtual-machine based, its compiler is very compact and fast and is usually deployed compiled directly in the virtual machine. This is unlike, for example, Java development environment, where the user machines are only expected to contain the run-time environment (JRE). The compiler is only available on developer's machines (trough the JDK), and software is deployed as compiled bytecode. In Lua is usual to deploy the source code and get it compiled transparently by the runtime. This frees from the write-compile-deploy cycle, and allows for easily swapping pieces of code on the fly. As Lua has functions as first class members, which means that functions are assignable as any variable, behavior of applications can be changed even at runtime. This offers a great platform for prototyping and experimentation.

The virtual machine takes as little as 200kb of storage and 800kb of RAM (the virtual machine can be compiled with options that further reduce this footprint). This includes all the core platform-independent functionality. Additional platform dependent functionality is provided by modules, like full POSIX and Socket support.

All messages are emitted in UDP broadcast packets. This imposes a limit of about 1500 bytes minus headers per message, if we want to avoid fragmentation at the IP layer (depending on the MTU of the medium). Thus a single

Algorithm 4.3 One-way matching

```
1 function matches( Notification n, Subscription s)
2     for {key, op, value} in s.predicate
3         nvalue = n[key]
4         if not satisfies(nvalue, op, value) then
5             return False
6         end
7     end
8     return True
9 end
```

Notification can not exceed this size. This is not considered an important limitation, as the protocol main purpose is to transmit relatively small size units of information (sensor readings). To transmit bigger units of information, one possibility is to split the message in multiple notifications, send them separately and reassemble them at the destination (RON behaving as an non-reliable transport layer). Another possibility is to extend RON with the ability of handling delivery of opaque documents trough a separate unicast channel, with Notifications containing only an associated descriptor (as does the protocol described in 2.5.1).

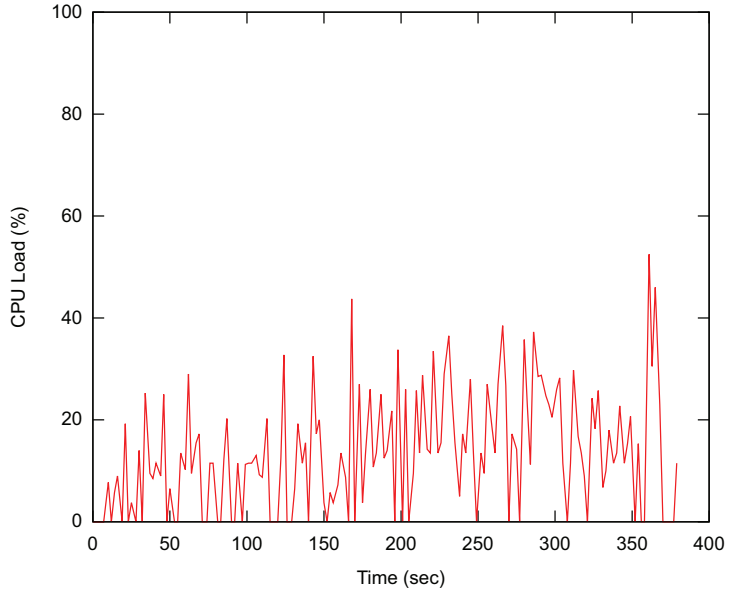
View messages are automatically split in several packets as needed. This is done splitting the quality vector in chunks of appropriate number of entries, and broadcast them sequentially. This does not affect the algorithm behavior: receivers process each View segment independently and do not have to know they come from the same node.

Algorithmically, the most expensive task is the matching of notifications against subscriptions (the forwarding decision). In RON, this is done in a simple sequential way, using one-way matching [34] (see Figure 4.3). This makes the worst case of the process $O(n \times s \times |S|)$, where n is the number of the notifications, s the number of subscriptions, and $|S|$ is the average size of a subscription (number of conditions). It must be noticed that the matching is done only once for each notification-subscription pair as a new notification or subscription is registered in the node, and the results are then cached.

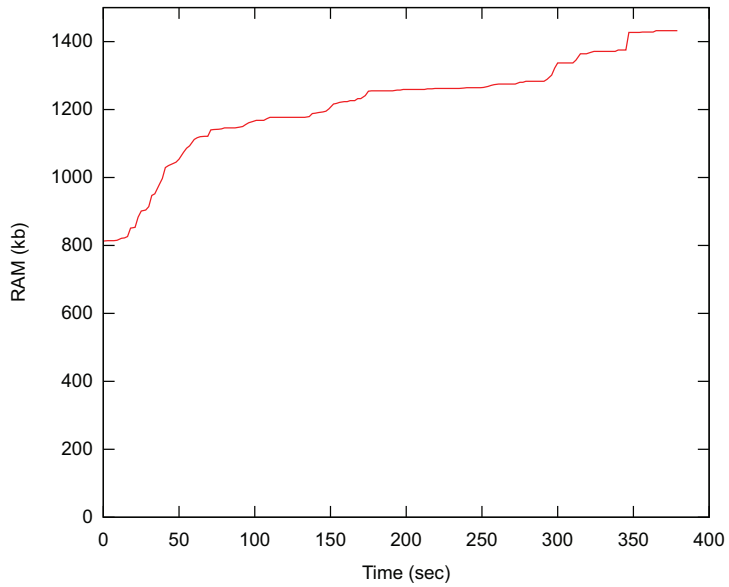
There are methods to improve the performance of this method. For example, a predicate can be strictly included in another, and thus if the more general predicate fails a match, the test for the included can be avoided.

Nevertheless, one-way matching showed adequate performance, both in memory consumption and CPU load, under a realistic traffic load and on very low end hardware [35]. In Figure 4.3 the results of running RON on a Asus 520gu, a consumer grade wireless router with 32Mb RAM, 8Mb flash storage and 200Mhz Broadcomm CPU, are shown. The router runs OpenWRT, a Linux distribution for embedded platforms.

RON is configured with a buffer of 40 messages, and the experiment runs during 380s. The message load is increased during the first 50 seconds up to 10



(a) CPU Utilization by RON



(b) RAM use by RON

Figure 4.3: Resource Consumption by RON

local data sources, that generate one notification every 5s. After second 50, a new subscription is registered each 10 seconds. Each subscription matches all the notifications arriving, and there are 33 different subscriptions at the end of the simulation. Notice that a notification matching a subscription is a worst-case scenario for one-way matching, as all the expressions in the predicate are tested.

The increase in RAM before second 50 is due to the notification buffer getting filled, and after that the memory increase corresponds to the storage of subscriptions.

One notification each 5 seconds matching over 30 subscriptions can be considered a very high load for the application. The memory consumed is under 1.5Mb, and the CPU use, while quite spiky, can be evaluated as peaking at 20-30%.

Chapter 5

RON Evaluation

For simulating the behavior of RON, we use an experimental ns3[36] branch, which is able to run native code inside the network simulator. Running the Lua virtual machine inside the Network Simulator makes possible to run the same codebase that would run in a normal deployment. This allows to verify that the application runs on realistic hardware and at the same time test big and complex scenarios in a controlled environment, that could not be realistically be maintained in the real world. An example of the simulator testing RON in a big network with 100 nodes can be seen in [37].

5.1 Subscription Quality

To test the behavior of the subscription quality control (see 4.2.4) we run simulations of a simple scenario.

The simulated terrain consists of two adjacent squares with 1000m sides (see Figure 5.1). In the extremes lay two fixed nodes, with a single subscription each, while in the center lays a single fixed node that generates notifications. Inside the squares there are mobile nodes, the same amount in both, which

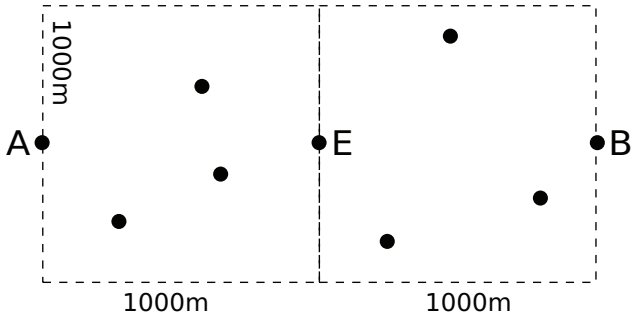


Figure 5.1: Mobility Model for Subscription Quality verification

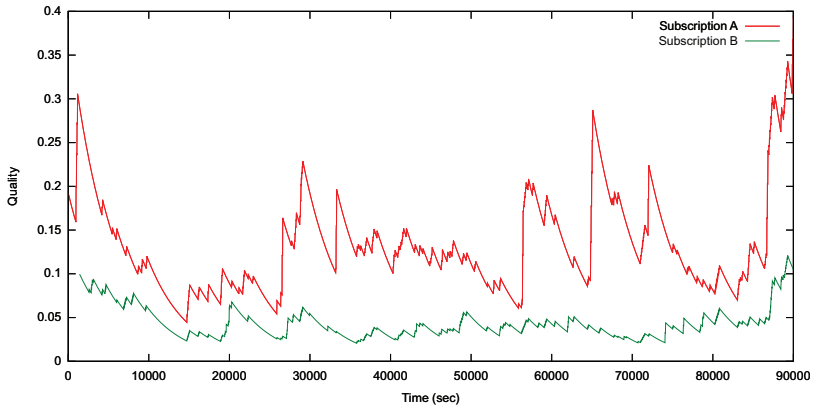
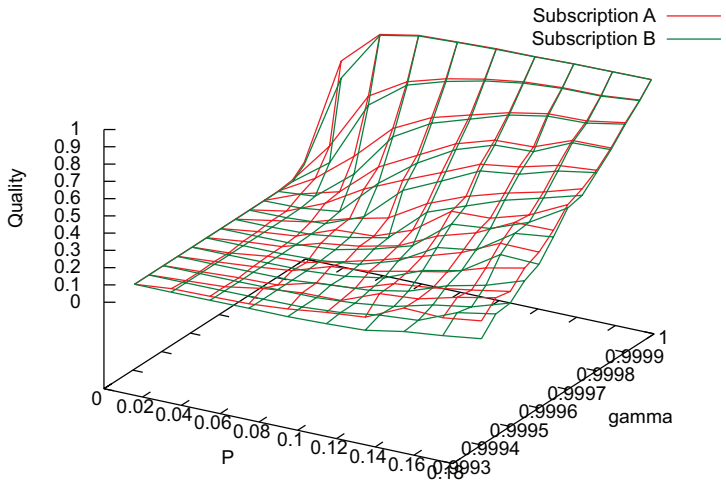


Figure 5.2: Subscription Quality adjustment, n=14, P=0.05, g=0.9998

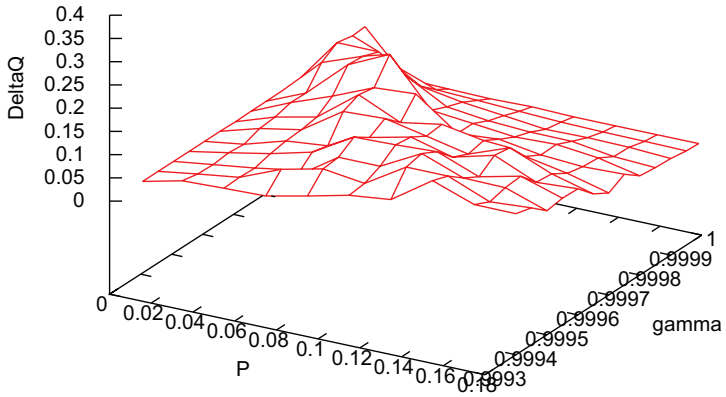
never leave their square. They move with a Random Direction mobility model, with velocities uniformly varied between 1 and 2 m/s. All the nodes have a wireless interface, with a range of 100m. VIEW messages are triggered each 60 seconds. The nodes moving in the left square randomly met the Subscriber A directly, but can receive Subscription B only trough the nodes from the right square, when meeting them in radio range near the center separation. Thus, nodes on the left square should have higher quality for Subscription A, which would make them better adapted to carry hypothetical notifications from node E that match Subscription A. Conversely, nodes on the right square would have higher quality for Subscription B. This behavior can be seen in Figure 5.2, in a network with 14 mobile nodes (7 in each half).

The values for P and γ must be determined experimentally for good results. If P is too low or γ too high, the qualities of different subscriptions will eventually get near 0 and be indistinguishable. Conversely, if P is too high or γ too low, all qualities will approach 1. For example, in Figure 5.2 we see the results of running for about 100000 second and computing the average quality for each subscription (starting at second 30000 to give time to stabilize). The figure shows the sharp rise of the quality when two nodes are in range (driven by equation 4.2), followed by a slow decay of the quality between encounters (driven by equation 4.1).

As can be seen in Figure 5.4 the optimal values for P and γ depend on the Network Density: values that work with certain network movement pattern may not be adequate in another. The graph shows the qualities for both subscriptions when P=0.05 and γ=0.9998, with 20 and 40 nodes, averaged over 5 runs. If the number of nodes in the network changes dynamically, the determination of the parameters can be a difficult task. Also, the parameters must be the same for all nodes across the network, for the qualities shared trough VIEW messages be comparable. This is another difficulty if the network is not homogeneous (some nodes are clustered in areas with different densities). This also precludes

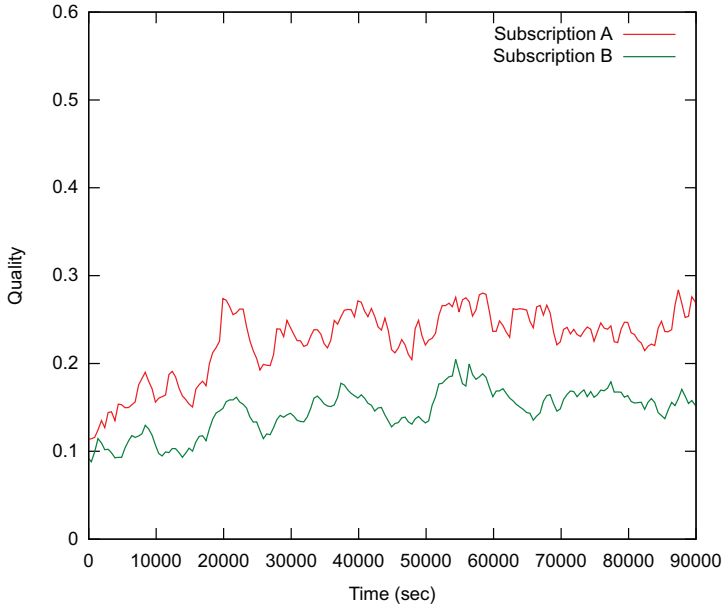


(a) Average qualities on a node over time

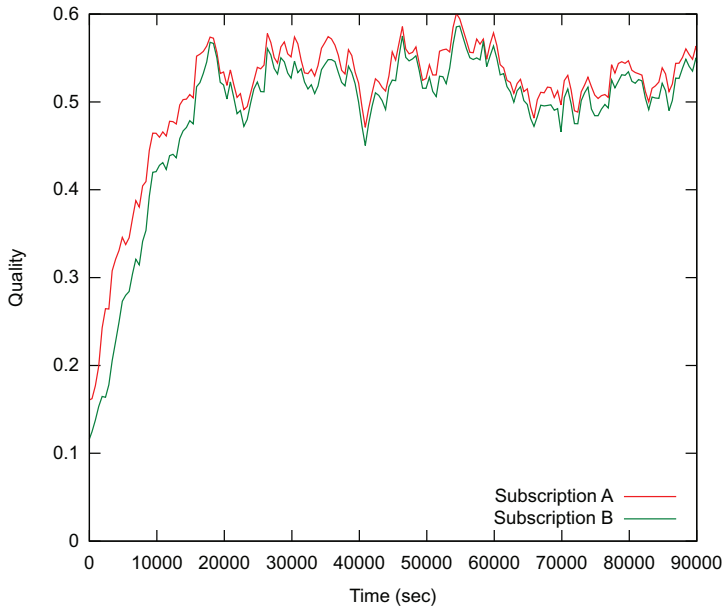


(b) Difference between average qualities

Figure 5.3: Impact of Parameters on Subscription Quality



(a) n=20, P=0.05, g=0.9998



(b) n=40, P=0.05, g=0.9998

Figure 5.4: Impact of Network Density on Subscription Quality adjustment

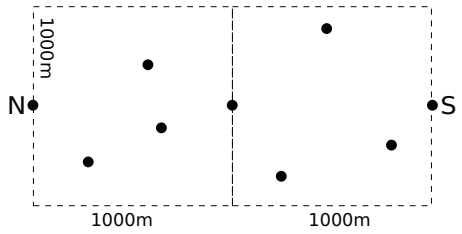


Figure 5.5: Mobility Model for Delivery Analysis

adjusting the parameters locally in the nodes from estimated network density, for example using mean times between encounters.

As a result, this method of managing qualities, while simple, cheap and widely implemented, presents some difficulties in realistic scenarios.

5.2 Delivery behavior

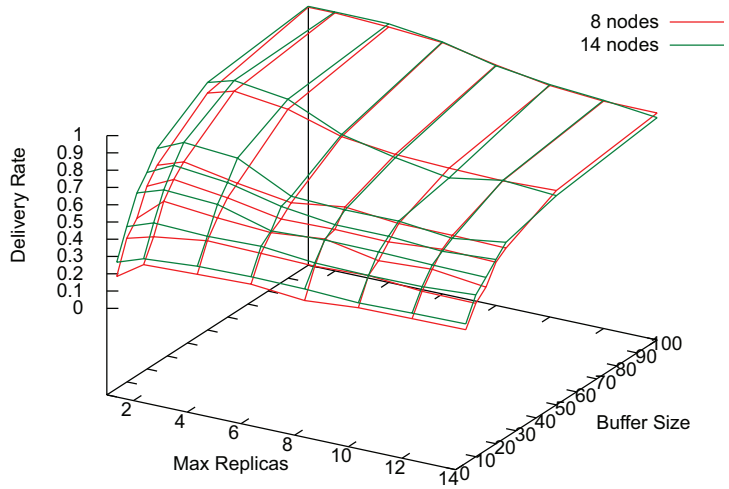
RON is a probabilistic algorithm, as the underlying problem is. RON does not make guarantees on when a message will arrive, or whether it will arrive at all. The main attribute of a Opportunistic Network routing algorithm is the Delivery Rate, which the proportion of messages emitted that actually reach destination. The Delivery Rate varies with the network characteristics (mobility patterns, node density, traffic load) and the parameters of the algorithm (size of the buffers, number of retransmits, etc.)

Another attribute of interest is the latency: the average time a message takes to reach destination. This is only defined for messages that actually arrive. This attribute must be handled with care, as a easy way of improving Latency is to reduce Delivery Rate: simply flooding a single message as to reach destination as fast as possible, and dropping everything else.

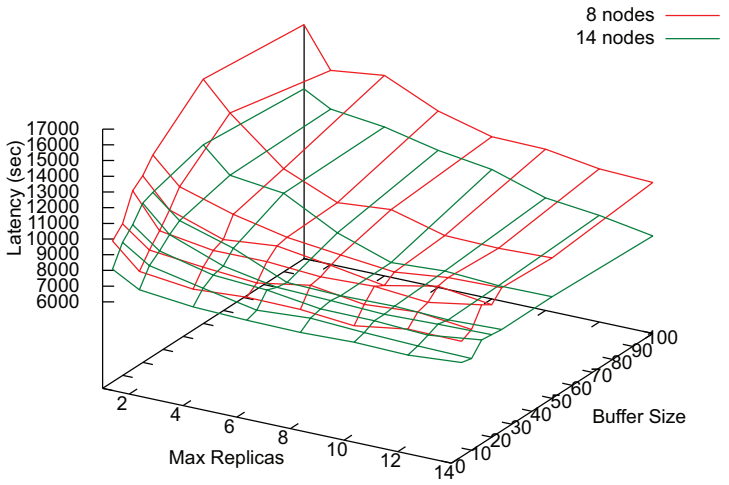
To study RON behavior related to message delivery a simulation scenario was setup (see Figure 5.5). It is similar to the one used in Section 5.1 with 8 and 14 nodes configurations, with changes to the emitter and subscriber nodes. In this scenario, the node labeled S emits a subscription, and node N will periodically emit matching notifications. These notifications are emitted each 500secs. A message to reach destination has to migrate between nodes at least once, in the separation line between both halves.

Different values for the size of notification buffer and maximum number of message replications are tried, and each scenario is run for about 1000000 seconds of simulated time. Nodes start prefetched with already full buffers, as to skip the initial transitional where the performance is better than average.

In Figure 5.6b can be seen that very low Delivery Rates come paired with low latencies, as the messages that actually reach destination are the ones that reach it quickly. Also can be seen how aggressive replicating of messages, despite having a negative impact on Delivery Rate, improves Latency.



(a) Delivery Rate of RON



(b) Latency of delivered messages

Figure 5.6: Impact of Parameters on RON Delivery

Also, denser networks have better Latencies, as they have more encounters per unit of time and this leads to faster message propagation.

A message is lost in RON when it is dropped from the buffers of all nodes that received it before reaching destination. As the buffer size grows the average lifetime of a message grows accordingly, and this leads to higher chances of reaching destination before extinction. This can be seen in Figure 5.6a. Also can be seen that it is not always worthwhile to increase the number of replications. In fact, it can lead to decrease in delivery rate. Excessive replication adds to the number of messages being transmitted, and increases the chance of a needed message missing the transmission window.

Besides internal configuration, the protocol's performance is affected by the node mobility patterns. To better understand the behavior of Opportunistic Protocols the Research Group developed a special synthetic mobility model, that allows to manipulate some of the main parameters of the network to study their impact on the protocol's behavior. The mobility model and a study of RON can be found in [37].

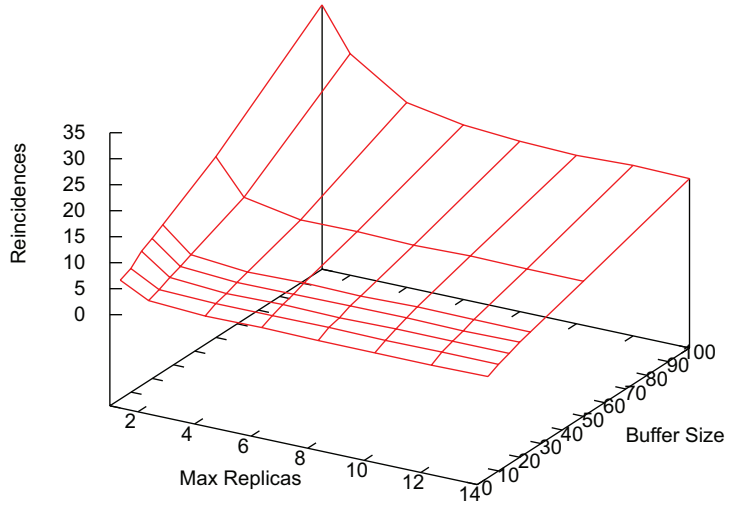
5.2.1 Notification re-incidence

Basic RON algorithm does not keep history of what notifications have been already stored in a node. This means that a given notification can be accepted again by a node after being dropped, if the node encounters the notification a second time. This is rarely a useful behavior, except when the network is configured with very low values of number of replications. In that case, the average lifetime of a notification in a node is low, as it gets dropped after a few rounds of broadcasts, but the re-acceptance can provide a greater accumulated lifetime in the node and an adequate lifetime of a notification in the network. Conversely, as the number of replicas grows, the re-incidence of notification do not contribute to the performance of the network and only occupy resources, both buffer space and airtime. This can be seen contrasting figures 5.7a and 5.6a (the number of reincidences is calculated in a network with 8 nodes).

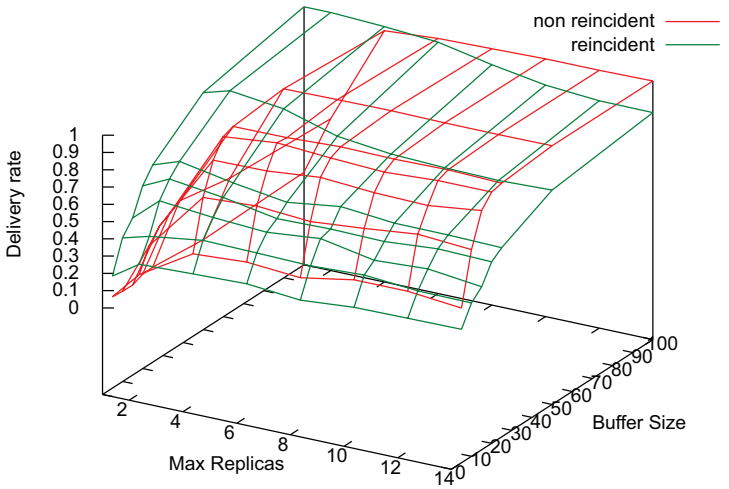
To test the impact of re-incidence control on RON performance, a modified RON was tested. The modified RON keeps a list of notifications that already were carried, and won't accept them again. The results can be seen in Figure 5.7b (again, run in a network with 8 mobile nodes). It results clear that while with very low number of replicas the re-incidence is critical to keep notifications alive long enough, usually the re-incidence control provides much better and consistent performance. This comes at the cost of a moderate increase in memory consumption, needed to keep a moving window of notification-ids.

5.2.2 Acknowledgments and destination based routing.

The re-incidence problem is related to a wider problem, the life-time control of messages. The issue is that notifications can survive in the network beyond the moment in time a copy has reached destination. This is related to the impossibility of knowing when a message has reached all potential destinations,



(a) Average number of reincidences



(b) Impact of re-incidence control on Delivery Rate

Figure 5.7: Reincidence control in RON

Algorithm 5.1 RONd ACK message pruning

```
1  function prune_for( m )
2      for n in N do
3          if n.source==m.target
4              and n.target==m.source
5              and n.seq<m.ack
6          or n.source==m.source
7              and n.target==m.target
8              and n.ack<m.ack and not n.seq
9          then
10             N.remove( n )
11         end
12     end
13 end
```

due to a fundamental difference between destination based and content based routing. In destination based routing, there exists the concept of data flow between pairs, and a ACK mechanism can be implemented. The ACK can be used to drop from buffers messages that have already reached destination. This can improve the performance of a opportunistic network [38].

This is not possible with content based networks, as there could be several recipients, and none of them have to know the existence of others nor if they received the message already or not.

A destination-routed variation of RON exists, called RONd. The algorithm is identical to RON, with the further restriction that only destination based subscriptions are allowed, of the form *target=node-id*, where a nodes subscribe to messages that have them as destination. All messages get tagged with a *source* and *target* fields. The *target* field will match the destination node's subscription. This is an example of destination routing implemented as a particular case of content-based routing as described in Section 2.4.

RONd has an ACK mechanism. An additional sequence and acknowledgment number is maintained by the nodes for each dataflow, this is a seq/ack pair for each node it sends data to or receives data from. All messages sent are tagged with it's sequence number (the *seq* field). The reverse data flow gets tagged with an *ack* field, indicating the last sequence number arrived such as $seq+1$ has not arrived yet. If there is a dataflow going back, *ack* are sent piggybacking it. Otherwise new explicit ACK messages are generated (at a relatively slow rate) with no own *seq* number.

The *ack* and *seq* fields are used by nodes to prune messages that have already reached destination. The pseudo-code of that pruning can be seen in Figure 5.1.

5.2.3 Replacement policies

The replacement policies can be critical for good network behavior. Different topologies and movement patterns stress replacement policies differently, and

the wrong policy can completely spoil the network.

Starvation by Subscriber Proximity.

Analysis shows that replacement policies that only use subscription quality to make routing decisions (like default PROPHET) show some pathological behavior. In the example shown in Figure 5.8a, suppose node A is subscribed to messages originated in B, and B is subscribed to messages from A. The expected behavior is that notifications from A flow towards B, and from B to A. Trouble is nodes closer to A will have a higher quality assigned to the subscription generated on A, and thus notifications targeted to A will always beat the notifications targeted to B (and originated in A). This means the buffers on the nodes close to A will be occupied by messages to A, and will not accept messages going for some remote node. The same happens close to B. The problem can be described as follows: if the buffer assignment is driven by subscription quality, the buffer utilization is skewed towards notification near its destination in detriment of the ones starting their journey. This can lead to death by starvation. This is bound to happen when there is a notification emitter close to a notification receiver.

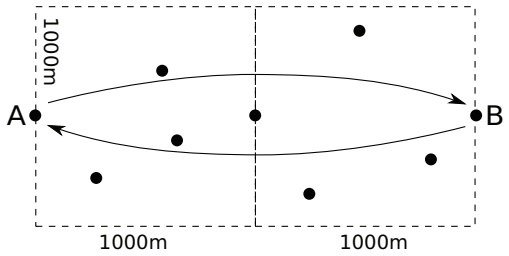
Simulating the described setup, with notifications being generated each 500 seconds, and Subscription quality adjustment as per Figure 5.2, provides the result shown in Figure 5.8b. One of the two flows (from B in this run) gets completely displaced, while the other behaves normally. Of the two flows, the one first to arrive near it's destination is the one that survives.

To handle this scenario, the replacement policy should ensure certain level of diversity where lower ranked notification have a chance of survival in the buffer, for example taking into account the age of the notifications and weighing in favor of the youngest.

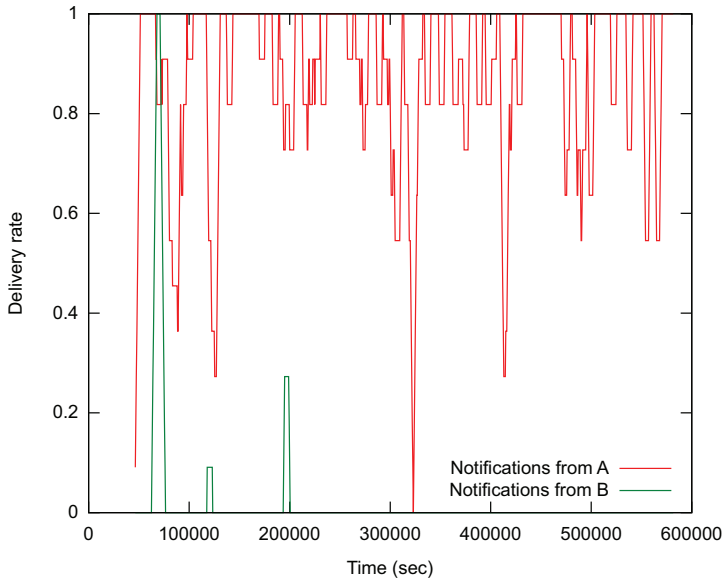
Synchronization storm

Another usual movement pattern that can lead to problems was identified while working on movement models for the DEMOS project. As mentioned in Section 4.1, the project envisions collecting data from distributed environmental sensors using the daily movement of kids going to and from school carrying their XO OLPC laptops. In that model all nodes approach a collection point (the school) near simultaneously. When that happens, all nodes come in range of each other for a window of time, and all see the same notification broadcasts. If all the nodes take the same decisions on what notifications to keep, the end result is that the buffers from all the nodes contain the same messages, bringing down the potential throughput of the network to the capacity of a single node.

The simplest case where this happens is when there is a single subscription in the network (from the data sink, ie. school), and the replacement policy mandates using FIFO for discarding messages when there is a draw in accumulated qualities for the messages in the buffer. In this case, after a single round of broadcasts per node, all the nodes remain holding the same n messages, the last n to appear in the network (where n is the buffer size of the nodes).



(a) Model with message exchange



(b) Delivery behavior when exchanging messages

Figure 5.8: Starvation with subscription quality-based replacement

5.3 Automatic parameter management

As seen in Sections 5.1 and 5.2, the algorithm is sensible to the configuration of several parameters. Correct configuration of parameters can have a critical importance on performance, and their correct values depend on the nature of the network: network density, mobility patterns, etc. The difficulty is increased by the fact that a network can be not homogeneous, or vary it's behavior as time passes. Nodes can join or abandon the network, or the mobility patterns can depend on the time of the day, etc.

As the exact behavior of the network can not be predicted, it brings the need of self-configuration, where nodes autonomously adapt to changes in the network. An example of adaptation in ON nodes can be found in [39], tough with a simple Spray-and-Wait network in a very particular mobility scenario.

When deployed in support of a DEMOS network (see section 3), there is already a generic decision-making platform running: LuPA (*Lua Policy Agent*, to see a more in depth description of LuPA see section 3.2.3). The purpose of LuPA is to allow managing a system trough high level policies. At run-time, LuPA recognizes certain patterns of events and generates actions, expressed in said policies.

As mentioned in 4.1, originally the role of RON in DEMOS was providing communications between data providers and data sinks, which in RAN architecture mean LuPA components. LuPA itself was tasked with managing sensor nodes, and the information they generated. The fact that LuPA infrastructure was deployed already and was flexible enough, created the opportunity to use it to manage the RON bus. The idea was that from LuPA's point of view, RON could be seen just as another system to be managed.

For testing the viability of using LuPA to manage RON, a scenario was created [40]. As a configurable parameter it was selected the number of replicas, which as can be seen in Section 5.2 has impact on both loss rate and latency. As the input variable network density was used, as simulation shown that it had a direct impact on optimum values for number of replications. As the network density is not directly measurable from a node, time between encounters was used as an indicator.

The process was the following: first, a simulation was used to test out different scenarios and find the optimum values for the configuration parameter in each. The experimental setup comprises between 10 and 50 nodes moving following a specially developed mobility model in a 1000m x 1000m square area. For more details on the experimental setup, refer to [40].

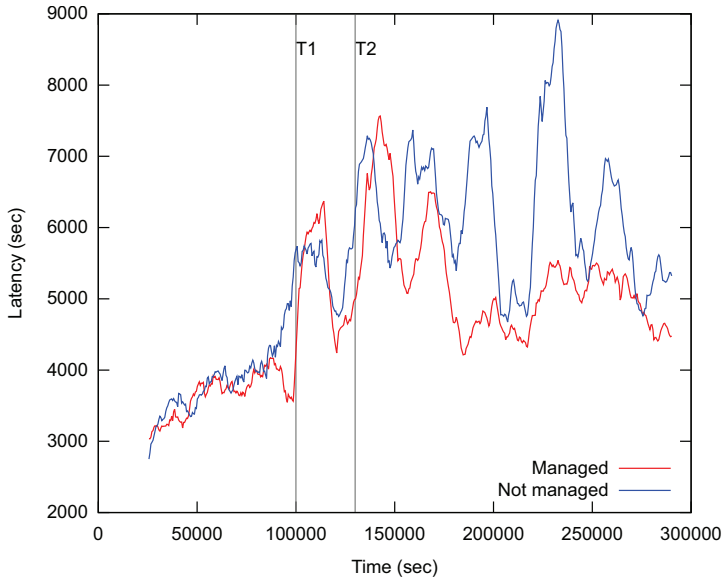
Then, from these simulations rules were derived, which would allow to select the optimum values in the different scenarios.

Rule 1 `If estimated_loss_high then reduce_max_r`

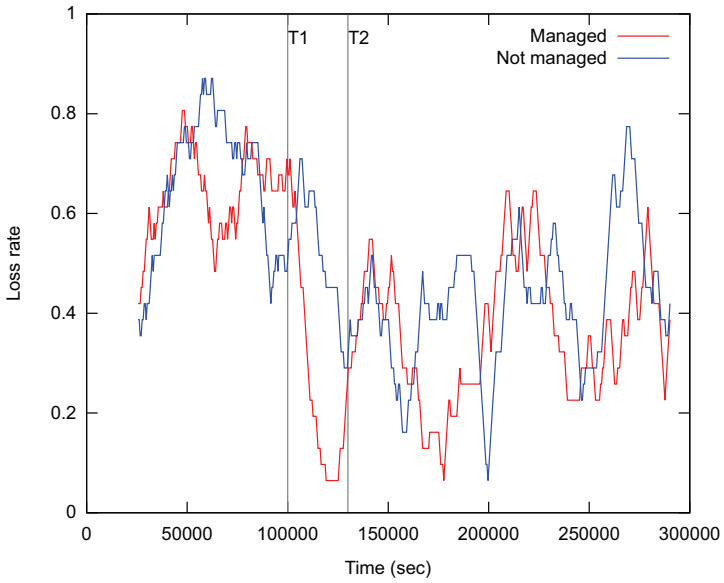
Rule 2 `If estimated_loss_low then increase_max_r`

Rule 3 `If estimated_delay_high then increase_max_r`

Rule 4 `If estimated_delay_low then reduce_max_r`



(a) Average latencies



(b) Average loss rate

Figure 5.9: Impact of parameter management for RON

The meaning of the rules is that an increase in number of replicas reduces the delivery rate but improves latency, and the reduction has the opposite effect. The conditions to increase or reduce the number of replicas are based on comparing the estimated density of the network (obtained from measuring times between encounters) with the expected optimum performance (obtained from the simulation in the first step).

Finally, these rules would be deployed to the LuPA agent residing in each node, and used to dynamically update the managed parameter.

The rules were validated installing them in the nodes, running a scenario where the number of nodes varied, and comparing the performance of the network with the management enabled and disabled. The results of the experiment are shown in Figure 5.9. At time $T1$ the number of nodes on the network drops from 50 to 10, and at $T2$ rules are applied. As can be seen, the management achieves a moderate improvement in loss rate and a important reduction in latency.

5.4 Optimal Routing estimation

RON being a probabilistic algorithm, offers no guarantee on the performance. As there is no indication what the best possible performance could be in the current scenario, it is hard to evaluate the performance of the algorithm.

Lacking a reference value to compare against, opportunistic algorithms are evaluated in the literature by comparing the performance with other algorithms. Nevertheless, not knowing the maximum performance attainable makes hard to ascertain if the differences are meaningful.

A possible solution is to compare against an Optimal Routing algorithm, that makes the optimal routing decisions (an *Oracle*). Such an algorithm is assumed to run with perfect knowledge of the movements of the nodes and data traffic trough the considered lapse of time, and has no restrictions on computing time or storage as long as it is feasible.

Such an algorithm could be applied to a recorded movement trace and compute (offline) what could have been the best routing decisions made. The outcome of this algorithm is then taken as the upper bound of what can be expected from a routing algorithm. As most practical algorithms work under a set of restrictions (limited worldview, CPU power and storage), they will deliver worse performance than Optimal Routing.

The computation of the exact solution of the Optimal Routing problem is very costly. An approach for an exact solution using Integer Linear Programming can be found in [9]. Due to scalability issues, the method is limited to scenarios with small loads.

5.4.1 Genetic Algorithm for a Opportunistic Routing Oracle

For the purpose of evaluating opportunistic algorithms we developed a non exact Optimal Routing algorithm that provides an upper bound on possible performance. It is based on Genetic Algorithms (GA), and at the cost of not being exact, provides good solutions for medium-sized networks. As an example, the algorithm applied to a trace with 20 mobile nodes and about 7000 node encounters over 13 hours, converged to a solution for the delivery of 200 messages in under 1 hour in a Intel i3 @3GHz machine.

The GA's objective is to find an upper bound solution for maximizing delivery rate, given a limited buffer in the devices. It is designed to find upper bounds for gossiping-like algorithms, this is, algorithms that exchange messages in certain communication rounds. In RON's case it corresponds to a View message and resulting Notification replies.

The GA developed makes two assumptions on the behavior of the network. In first place, it assumes that on each encounter opportunity there is enough time to exchange all the desired information. This is a reasonable expectation in DEMOS, where the messages are small (on the order of hundreds of bytes). If it wasn't so, the algorithm will have to be extended to take in account also the order in which the messages are delivered.

The second restriction is that it will not allow a message to return to a node where it has already been. Under this assumption a message can be carried by a node only once on the message's way to destination. This may lose some solutions under certain circumstances, but we expect the impact to be low (for an analysis on the impact of re-incidence of messages, see 5.2.1). This limitation allows to simplify the algorithm, as the number of hops that a message can do on it's way to destination is bound by the number of nodes in the network.

An important observation for the GA representation is that any optimal routing solution is attainable with a single instance of each message present throughout the network. The multiple copies made by opportunistic algorithms such as RON are a method to cope with uncertainty, but once a message reaches destination, we could trace-back the path followed by the message, and determine that all other copies of the message could have been omitted without loss. Thus, the GA has to find the optimal decisions to be made with a single copy of a message, this is, when a node should handle it to what other node.

A second observation is that if the optimum solution requires that a given message is not delivered, it can be safely dropped right at the emitting point.

The GA was developed using GAlib[41] version 2.4.7, a library for implementing Genetic Algorithms written in C++.

To apply Genetic Algorithms, the problem and solutions must be expressed in a suitable way. We transform the recorded trace into an internal representation, consisting of a graph that holds all the needed information for solving the optimum routing problem. In this graph, each decision point (a transmission opportunity) is represented by a node, and each decision (whether transmit or to keep a message) is represented by a directed edge connecting two nodes. The

nodes are labeled by the mobile device they belong to. A path in this graph represents the evolution of a message trough the network. Also, each node is labeled with a time-stamp from the original trace.

Messages are introduced in the network at special nodes, "Entry Points". There is a distinct Entry Point for every message. Conversely, a message reaches destination when it reaches a node labeled with the message's target device (all such nodes are "Exit Points").

When an edge connects two nodes belonging to the same device, it represents a "keep message" decision. When an edge connect nodes belonging to different mobile devices, it represents a "transmit" decision. The graph is thus represented as follows:

- $G = \{N, E\}$
- $N_I \subset N$ (entry points)
- $N_O \subset N$ (exit points)
- $N_I \cap N_O = \emptyset$
- $E = E_{keep} \cup E_{transmit}$
- Nodes have at most 2 exit edges (one "keep" and one "transmit")
- Nodes have at most 2 incoming edges (one "keep" and one "transmit")

A path a message can take will respect the following restrictions:

- Each path start at a different node from N_I (therefore there is at most $|N_I|$ paths)
- Each path ends at node from N_E(several paths can end in the same node).
- There are no loops in a path (the *no reincidence* restriction).
- Nodes are traversed by no more than Z paths, where Z is the buffer size.

The objective of the algorithm is to build as many paths as possible respecting these restrictions.

It must be noticed that in this context, the favored or "shortest" paths are not measured by the number of hops, but by the difference in the timestamps between its entry and exit points. This is due that the less time a message is kept in the network, less network resources it consumes. This heuristic also strives to minimize the latency of the network.

A Gene is represented as an array of paths. The length of the array is given by the number of entry points, this is, the number of messages transmitted. A slot in the array may be empty, representing that the given message is being dropped. Each path, in turn, is an array of fixed length equal the number of mobile devices in the network (as mentioned before, if there are no loops this is the longest possible path). Thus, a gene can be seen as a bidimensional array

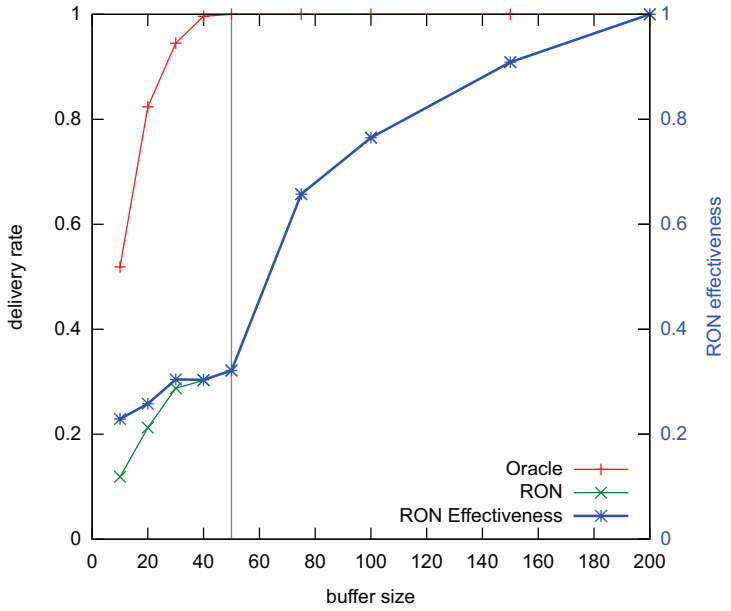


Figure 5.10: RON and Optimal Routing

sized $|N_E| \times |devices|$. Genes are designed to hold only feasible individuals (valid solutions). To keep genes feasible after applying genetic operators, a factibilization process is applied. This implies potentially modifying the gene to make it comply with the restrictions detailed above, in particular the buffer limit. The process consists of a local search step, where offending paths are attempted to be re-routed to avoid nodes with full buffers, and if this fails, the path is dropped from the gene (the message is dropped from the network).

5.4.2 RON effectiveness evaluation

In the following experiment, we compare the performance of RON with the offline solution provided by the GA Oracle. The scenario is similar to the one used in 5.1, with 10 mobile nodes. The emitter node generates 200 messages, with intervals of 2 minutes, and the network is run for 15 hours. Different buffersizes are tried, between 10 and 200. Notice that a buffer of 200 can hold all the messages present in the network, and in such conditions RON should achieve a 100% delivery rate, eventually. The graph 5.10 shows the average of 4 runs. For each run, the upper bound using the GA oracle was calculated.

RON effectiveness is calculated as the proportion of RON delivery rate to the Oracle's. It can be seen that the oracle achieves a 100% delivery rate with buffers over 50. Therefore RON effectiveness for greater values of buffer is not indicative, as the Oracle performance is already maxed out. From the delivery rate achieved by RON and the Oracle with buffers under 50, it can be claimed

that RON achieves a performance in the 20 - 30% range of the optimal, with the effectiveness improving with buffer size.

It must be noticed that in the very simple scenario used, there is no information RON can learn to make better decisions. The performance shown is achieved with the basic buffer management and forwarding schemes. When applicable, routing policies should improve over this base-level performance trough smarter buffer management. On the other hand, there is a limit on the effectiveness attainable from a distributed algorithm operating on a limited local view compared to a offline perfect-knowldege algorithm.

5.5 Test Deployment

As part of the DEMOS project, a mobility model suitable for describing the movement of kids to and from school and trough the neighborhoods was developped42. One of the inputs necessary for validating such a models is real mobility data. The data of interest includes power down and power up events, encounters with other laptops and those encounters duration, and encounters with school's Access Points. Analyzing such data it is possible to extract some metrics of interest, such as average time between encounters, average association time, time distribution of said encounters, etc.

For the purpose of collecting activity data, a small application was installed in the laptops of a selected school, with the cooperation of Plan Ceibal [31]. The application was installed on 83 laptops, and a total of 9710 events were logged over a span of 5 weeks.

The data transmission portion of collection system was handled by RON: as part of the deployed application a instance of RON was installed on each laptop. Each event detected by the data collecting application was expressed as a notification and pushed in the RON network. At the same time, a instance of RON was installed in the school's server, together with an data logging application. This application published a subscription in the RON network for all events, and stored them in a RDBMS as they arrived, for further analysis.

It must be noticed that this schema is not the described above for DEMOS. In DEMOS, notifications are generated in remote static nodes (the sensor nodes), and mobile nodes (laptops) only collect these notifications and ferry them to school. In this experiment, the notifications are generated by the mobile nodes themselves, and handled directly to the school.

For this application, a particular set of configuration options were provided (see Table 5.1).

As mentioned, in this application the mobile nodes deliver the information themselves, without being forced to relay it trough intermediate nodes: it behave as a Direct Delivery system (see 2.3.1). As the routing behavior is not used, the configuration parameters used to manage subscription quality do not apply and are not shown. Moreover, as can be seen in the configuration provided, there was a small proportion of the buffer spared for carrying notifications from other nodes (100 slots, the difference between the total inventory size and the

Parameter	Value	Description
inventory_size	700	Estimated from a worst case notification rate trough a unconnected weekend.
reserved_owns	600	Each node is mostly responsible for its own notifications.
max_notif_transmits	10	
max_ownnotif_transmits	∞	The node never drops own notifications on transmit count.
save_each	60s	The state of the RON is saved once a minute to handle power downs.
max_owning_time	3 days	Acceptable time between encounters prior to starting losing notifications.
send_views_timeout	30min	We estimate that laptops either connect for a session and are used for a period of time, or fail to establish connection.
delay_message_emit	10min	Limits the frequency at which a given message will be seen in the network.

Table 5.1: RON configuration parameters

reserved slots for own messages), but it was verified that 100% of the delivered messages were delivered by the same node that emitted them.

Therefore, this experiment is not adequate for analyzing the opportunistic routing behavior or the protocol. Nevertheless, it allowed us to verify two critical characteristics.

In first place, it proved that the application is deployable and robust enough to be run in production conditions. RON was installed automatically using the standard XO deployment mechanisms, and shown it can run reliably in the actual hardware. It handled correctly such events as forced power-offs, and the resource consumption did not register as noticeable by the laptops users.

In second place, the forwarding mechanism proved it can work in the very hostile wireless environment in the school. The school is a point where there is a high concentration of wireless nodes, actively trying to use the wireless link to the school's Access Points, all at the same time. In our experience, the environment is harsh enough to make some broadcast based protocols to fail. RON did effectively deliver data, and moreover, did so without imposing a noticeable load on the host network.

Chapter 6

Conclusion

A new protocol for Content-Based Opportunistic Networks was developed, RON. This protocol improves on previous work in the area, solving several of the weaknesses of existing algorithms.

The proposed solution is simple and easy to implement. The provided implementation is highly portable and efficient and is easily adaptable to test out new ideas.

RON is well adapted to a real use-case, the DEMOS project. Its behavior, scalability and performance has been tested extensively in a simulated environment. It is of note that the code-base that was run in the simulator is exactly the same as it is deployed on real hardware. Also a test deployment was made in the context of another project, demostrating that the implementation is able to perform on real hardware under real world conditions.

The content-based routing simplifies the deployment and management of a sensor network, allowing to add sensor nodes and new data consumers without having to change configurations nor manage a network inventory. This makes viable to deploy sensor nodes on a as-needed basis, even for short spans of time.

Several issues were identified were further work is needed. In particular, the difficulty of keeping optimal parameter configurations in the face of changing network characteristics. Also, some routines in RON have several alternatives, implementing different policies. The correct alternative depends on the nature of the network. One specially important task is lifetime management of messages in the network: if the messages are held for too short a time, they have a high chance of being dropped before reaching destination. If they are stored for too long, they overload the network increasing the chance of other messages getting dropped. The lifetime management is specially hard in content-based networks, as a single message can have multiple destinataries and is difficult to know when a message has been delivered to all of them.

Tests with RON have shown that the buffer management policy can have critical impact in the performance of the network. We have shown that under relatively common conditions, some of the most widely used policies such as FIFO or Sorting by Quality can have catastrophic results. Preliminary test

with RON and different replacement policies point out two important properties. First, a replacement policy must maintain diversity of selected messages trough all the nodes in the network. This means that it must avoid synchronization effects that lead to the same message being selected in most nodes. The second property, is that it appears that for different networks there are different optimal policies. For example, for DEMOS a special policy was developed that takes into account the periodic nature of movement in the DEMOS network. This policy performed much better than FIFO, which failed completely under same conditions.

A general solution for managing the protocol is proposed and implemented. This solution uses a general purpose policy-based decision engine, and is easily extensible to provide new functionality. It allows RON to adapt the configuration parameters autonomously in response to changes in the node or the network.

Chapter 7

Future Work

During the development and testing of RON, several problematic areas were found. From analysis of previous existing solutions, these issues are shared with other algorithms, and point out to difficulties implicit to the problem of opportunistic routing.

RON, with its simple architecture and the ease of adaptation of the implementation provides a good opportunity for exploring new solutions.

7.1 New routing metric

The tests of RON have identified difficulties with the *reinforcement/aging* pattern for estimating the usefulness of a neighbor.

In first place, the method is highly sensitive on the correct configuration of the (admittedly few) configuration parameters. As was shown in Section 5.1, the algorithm works correctly in a very narrow band of configuration parameters. The workable configuration values are dependent on encounter dynamics of the network, which in turn depends on the density, movement patterns of the nodes, etc. If the network changes, the values for which the routing behaves correctly change too. This attempts against the robustness of the network.

An extra problem stems from how these link qualities and coefficients are used: the calculated qualities are shared amongst nodes for comparison, combination, etc., so the parameters that drive the coefficient's values must be common to all participating nodes for their values be comparable. This in turn brings two difficulties: first, different areas of the network can have different dynamics and thus different correct values for the parameters. Second, it inhibits the possibility of a node autonomously adjusting the parameters adapting to the environment.

This brings the following idea: if the semantics of the quality coefficient were not absolute to the network, but only indicative of a local preference of to the node, it would be possible to adjust the quality's driving method. This could be done autonomously to provide an expected distribution of qualities for the

node, independently of the decisions taken by other nodes. Such quality, to be useful, would need also a method to be combined with the qualities received from neighbors.

A first approach could be to replace the quality coefficient with a buffer assignment coefficient, following the idea that bigger proportion of the buffer is assigned to better connected destinations. A difficulty with this approach is that it ties rigidly the buffer assignment to reachability, independently of actual traffic volumes. A low traffic but well connected destination could get disproportionate buffer space.

7.2 Replacement policy

The issue of buffer management and the policies used to assign space in the buffer for messages and how they get replaced, is little studied. Most implementations use a simple FIFO or "sorted by quality" policy (CAR, RAPID, CEPMF). PROPHET provides several policies, based on sorting along certain selectable metric, but the impact and the implications of the different alternatives is not clear.

As mentioned before, some widely used policies show patological behavior under some common scenarios. Further work is needed to develop workable alternatives, and two possible approaches were identified. One alternative is to find a policy that behaves robustly in different networks. The other possibility is to select the replacement policy autonomously, adapting to the network on the fly. This could be done using the ideas shown in Section 5.3.

Another functionality that could be easily added to a new replacement policy is prioritization of messages. Notifications could be tagged with some type of priority class tag, which would then be taken into account by the replacement routine when managing the buffer.

7.3 Data patterns

The fact that there may be different data flows with different volumes, and that these loads can change dynamically is not sufficiently explored.

In [10] fairness problems are described when using FIFO buffer management policies. Using some form of Fair Queuing algorithm is proposed, but not explored.

RAPID (see 2.3.6) creates a highly general framework that could in principle be used to take into account information volumes (defining a suitable metric), but it is not clear how that could be done.

Moreover, to the best of our knowledge none of the existing algorithms attempt to predict the future evolution of traffic loads, as it's done with the encounter opportunity patterns.

7.4 Inventory summary broadcast

As mentioned in 4.2.5, it is possible for a node to receive notifications it already has in the buffer, as a response to a View broadcast. To reduce this possibility, RON attaches a list of notifications IDs, as to allow the View listener to avoid repeating messages.

Nevertheless, this method does not scale, as it is lineal over the number of messages in the buffer. Meanwhile, there is a hard limit for a packet size of 1500 bytes (most of which is occupied by the View).

A method for including a digest of buffer content using constant space is using a *Bloom Filter*. Bloom Filters allow to build a fixed-size digest at the cost of a probability of a false positive. This means there is a probability the digest will report a element as present, when it actually was not. The digest does not report false negatives. The false positive probability increases as more elements are put in the digest. It is possible to determine the value for parameters that provide certain level of maximum false positive probability given a digest size.

A difficulty with implementing Bloom Filter digest is the implications of a false positive. It would mean that a View receiver interprets that a certain message is in the emitter's buffer when it is not. This would preclude said message from being broadcast. Moreover, the same false positive could affect all nodes, and none of them would provide the message.

7.5 Real-life deployment

At the moment, there have been only small scale deployments in real hardware, basically to verify functionality and resource utilization. The algorithm itself has only been tested for scalability in a simulated environment (up to about 200 nodes, with synthetic mobility models). The software platform itself has been deployed and used on a medium sized scenario, but the routing component has not been exercised (see 5.5).

A necessary following step would be the validation of a medium to large scale deployment for a real application, such as envisioned by DEMOS project.

List of Figures

List of Tables

Bibliography

[1] R.E. Kahn, S.A. Gronemeyer, J. Burchfiel, and R.C. Kunzelman. Advances in packet ratio technology. *Proceedings of the IEEE*, 66(11):1468 – 1496, 1978. ISSN 0018-9219. doi: 10.1109/PROC.1978.11151.

[2] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Sigcomm '03: proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications*, pages 27–34, New York, NY, USA, 2003. ACM. ISBN 1-58113-735-4. doi: 10.1145/863955.863960.

[3] Delay-tolerant networking research group (dtnrg). URL `http://www.dtnrg.org/`.

[4] K. Fall and S. Farrell. DTN: an architectural retrospective. *Selected Areas in Communications, IEEE Journal on*, 26(5):828–836, May 2008. doi: 10.1109/JSAC.2008.080609.

[5] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838 (Informational), April 2007. URL `http://www.ietf.org/rfc/rfc4838.txt`.

[6] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050 (Experimental), November 2007. URL `http://www.ietf.org/rfc/rfc5050.txt`.

[7] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), January 2005. URL `http://www.ietf.org/rfc/rfc3986.txt`.

[8] L. Pelusi, A. Passarella, and M. Conti. Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *Communications Magazine, IEEE*, 44(11):134 –141, 2006. ISSN 0163-6804. doi: 10.1109/MCOM.2006.248176.

[9] Aruna Balasubramanian, Brian Levine, and Arun Venkataramani. Dtn routing as a resource allocation problem. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 373–384, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-713-1. doi: 10.1145/1282380.1282422.

[10] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, 2000.

[11] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, PODC '87, pages 1–12, New York, NY, USA, 1987. ACM. ISBN 0-89791-239-4. doi: 10.1145/41840.41841.

[12] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, WDTN '05, pages 252–259, New York, NY, USA, 2005. ACM. ISBN 1-59593-026-4. doi: 10.1145/1080139.1080143.

[13] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. In *SAPIR*, pages 239–254, 2004.

[14] A. Lindgren, A. Doria, E. Davies, and S. Grasic. Probabilistic Routing Protocol for Intermittently Connected Networks, October 2010. URL `https://datatracker.ietf.org/doc/draft-irtf-dtnrg-prophet`.

[15] Anders Lindgren and Avri Doria. Experiences from deploying a real-life dtn system. In *Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE*, pages 217 –221, 2007. doi: 10.1109/CCNC.2007.50.

[16] Ting-Kai Huang, Chia-Keng Lee, and Ling-Jyh Chen. Prophet+: An adaptive prophet-based routing protocol for opportunistic network. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 112 –119, 20-23 2010. doi: 10.1109/AINA.2010.162.

[17] Libo Song and David F. Kotz. Evaluating opportunistic routing protocols with large realistic contact traces. In *Proceedings of the second ACM workshop on Challenged networks*, CHANTS '07, pages 35–42, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-737-7. doi: 10.1145/1287791.1287799.

[18] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, number 2538 in Lecture Notes in Computer Science, pages 59–68, Scottsdale, Arizona, October 2001. Springer-Verlag.

[19] Antonio Carzaniga, Matthew J. Rutherford, and Alexander L. Wolf. A routing scheme for content-based networking. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[20] J. Visca, J. Baliosian, and E. Grampin. A distributed notification bus for constrained devices. In *Network Operations and Management Symposium, 2009. LANOMS 2009. Latin American*, pages 1 –6, 2009. doi: 10.1109/LANOMS.2009.5338802.

[21] F. Guidec and Y. Maheo. Opportunistic content-based dissemination in disconnected mobile ad hoc networks. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2007. UBICOMM '07. International Conference on*, pages 49 –54, 2007. doi: 10.1109/UBICOMM.2007.23.

[22] Anwitaman Datta, Silvia Quarteroni, and Karl Aberer. Autonomous Gossiping: A self-organizing epidemic algorithm for selective information dissemination in mobile ad-hoc networks. In *In International Conference on Semantics of a Networked*, pages 126–143, 2004. doi: 10.1.1.58.3986.

[23] J. Haillot and F. Guidec. A protocol for content-based communication in disconnected mobile ad hoc networks. In *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, pages 188 –195, 2008. doi: 10.1109/AINA.2008.82.

[24] Liu Yazhi, Niu Jianwei, and Ma Jian. Content encounter probability based message forwarding in opportunistic networks. In *Information Science and Engineering (ICISE), 2009 1st International Conference on*, pages 2589 –2594, 2009. doi: 10.1109/ICISE.2009.426.

[25] Mirco Musolesi and Cecilia Mascolo. CAR: Context-aware Adaptive Routing for Delay Tolerant Mobile Networks. *IEEE Transactions on Mobile Computing*, 8(2):246–260, February 2009.

[26] Demos (domestic environment monitoring with opportunistic sensor networks). URL `www.laccir.org/laccir/Portals/0/RFP/20090121_Request_for_Proposals.htm`.

[27] J. Baliosian, J. Visca, E. Grampin, L. Vidal, and M. Giachino. A rule-based distributed system for self-optimization of constrained devices. In *Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on*, pages 41 –48, june 2009. doi: 10.1109/INM.2009.5188785.

[28] Andés Aguirre, Pablo Fernández, and Carlos Grossy. Usb4all - generic usb interface. URL `http://www.fing.edu.uy/inco/grupos/mina/pGrado/pgusb/material.html`.

[29] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. Terminology for Policy-Based Management. RFC 3198 (Informational), November 2001. URL `http://www.ietf.org/rfc/rfc3198.txt`.

[30] Javier Baliosian and Joan Serrat. Finite state transducers for policy evaluation and conflict resolution. In *POLICY*, pages 250–, 2004.

[31] Plan Ceibal project. URL `http://www.ceibal.org.uy/`.

[32] Leonardo Vidal, Javier Baliosian, Eduardo Grampín, Jorge Visca, Guillermo Apollonia, Matías Richart Juan Saavedra, and Martín Giachino. Seguridad en redes oportunistas (SeRO). Technical report, Facultad de Ingeniería - Universidad de la República - Uruguay, Proyecto AMPARO, 2011.

[33] Lua, the programming language. URL `http://www.lua.org`.

[34] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005. ISBN 0470095105.

[35] Jorge Visca, Guillermo Apollonia, Matias Richart, Javier Baliosian, and Eduardo Grampín. Embedded Rule-based Management for Content-based DTNs. In *1st International Workshop on Network Embedded Management and Applications, 2010. NEMA 2010.*, Niagara Falls, Canada, October 2010.

[36] ns3 network simulator. URL `http://www.nsnam.org/`.

[37] Matias Richart, Jorge Visca, Javier Baliosian, and Eduardo Grampin. A mobility model for opportunistic routing protocols validation. In *LANOMS 2011*, Quito, Ecuador, oct 2011.

[38] *Evaluating the impact of an acknowledgment strategy for APRP*, LANC '09, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-775-2. doi: 10.1145/1636682.1636695.

[39] P. U. Tournoux, J. Leguay, F. Benbadis, V. Conan, M. Dias de Amorim, and J. Whitbeck. The accordion phenomenon: Analysis, characterization, and impact on DTN routing. In *IEEE INFOCOM 2009 - The 28th Conference on Computer Communications*, pages 1116–1124. IEEE, April 2009. ISBN 978-1-4244-3512-8. doi: 10.1109/INFCOM.2009.5062024. URL `http://dx.doi.org/10.1109/INFCOM.2009.5062024`.

[40] J. Baliosian, J. Visca, M. Richart, G. Apollonia, L. Vidal, M. Giachino, and E. Grampin. Self-managed content-based routing for opportunistic networks. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 121 –128, may 2011. doi: 10.1109/INM.2011.5990682.

[41] Galib, a c++ library of genetic algorithm components. URL `http://lancet.mit.edu/ga/`.

[42] M. Giachino. Demos connectivity model. Master's thesis, Facultad de Ingeniería, Universidad de la República, 2012.