Simplified Adaptive Routing and its Impact on Quality of Service and Quality of Information

Thesis submitted for
the Degree of Doctor of Philosophy of Imperial College London

Laurence Hey
Intelligent Systems and Networks Group
Dept. of Electrical & Electronic Engineering
Imperial College London

2009
ABSTRACT

This work offers novel simplified approaches to the problem of network routing that can adapt to maintain quality of service (QoS). In traditional general-purpose wireline and wireless networks, the QoS metrics are those measures of the network’s performance the user is interested in optimising, and are suitable as optimisation targets for the routing protocol. For networks with a more focussed function, such as those in the emerging category of wireless sensor networks (WSN), the user may be interested in application-specific measures of network quality. For example, in WSNs the application’s output is information, and the user’s interest will be in the quality of information (QoI). The network’s routing protocol is responsible for the assignment of resources, the prioritisation of packets, and generally managing the QoS. A number of routing protocols and scenarios are considered here. Firstly, the Cognitive Packet Network (CPN) routing protocol’s decision making algorithm is simplified. CPN is then implemented in hardware to provide improved QoS at high speeds in wired infrastructure networks. Secondly, use of a version of the CPN protocol adapted for minimising energy use in WSNs is explored. It is discovered the Minimum Cost Forwarding Algorithm, extended with Random Re-Routing for packet prioritisation provides a combination of simplicity and flexibility to WSN QoS routing. Finally, the overlap between QoS metrics and the new concept of WSN QoI is developed to allow the routing protocol to match user requirements to WSN resources.
ACKNOWLEDGEMENTS

I would like to acknowledge a number of people for the inspiration and encouragement they have offered me throughout the research and writing of this thesis. From the Circuits and Systems group, Peter Cheung for his supervision with the FPGA aspects of the work, and from the Intelligent Systems and Networks group Erol Gelenbe for his supervision with the rest. Also Georgios, Georgia, Avgoustinos, and Stelios (the Greek Squad) for the discussions over the years. And finally, my wonderful fiancée Jennifer, for being my wonderful fiancée.


CONTENTS

1. Introduction ......................................................... 15
   1.1 Providing Network Quality ................................. 15
   1.2 Thesis Overview ......................................... 17

2. Background ....................................................... 19
   2.1 Quality of Service in Packet Switched Networks .......... 19
   2.2 The Cognitive Packet Network .............................. 22
   2.3 The Random Neural Network ................................. 25
      2.3.1 Use of the RNN in CPN ............................... 26
   2.4 Field Programmable Gate Arrays ........................... 28
      2.4.1 FPGAs in Networking Applications .................. 30
   2.5 Wireless Sensor Networks .................................. 30
   2.6 Motivations and Remarks ................................... 31

3. Alternative CPN Decision Algorithms .......................... 32
   3.1 Introduction ................................................. 32
   3.2 Autonomic Routing Algorithms in CPN .................... 34
   3.3 The Approximated RNN ...................................... 35
   3.4 Sensible Routing ............................................ 37
      3.4.1 $m$- and $\infty$-Sensible Routing ................. 39
   3.5 Experimental QoS Comparison ............................... 40
      3.5.1 Mixed Algorithm Environments ....................... 47
   3.6 Complexity Comparison ..................................... 48
      3.6.1 Software computational time .......................... 49
      3.6.2 Memory requirements .................................. 50
      3.6.3 Hardware implementations ............................ 50
   3.7 Conclusions ................................................ 59
4. FPGA Based Hardware CPN Router Implementations
   4.1 Introduction ............................................. 60
   4.2 Dedicated Hardware for Packet Switched Networks ................. 61
      4.2.1 Three Generations of Internet Routers .................. 61
   4.3 Cognitive Packet Networks in Hardware ........................ 67
      4.3.1 Field Programmable Gate Array Based Implementations .... 67
   4.4 Single Routing Engine FPGA Based CPN Router
      (1st Generation Internet Router Equivalent) ................. 67
      4.4.1 Architecture ....................................... 68
      4.4.2 Device Usage and Speed .............................. 70
   4.5 Multiple Routing Engines
      (3rd Generation Internet Router Equivalent) ................. 71
      4.5.1 Independent Interfaces and Quality of Service .......... 72
      4.5.2 Router Architectures ............................... 73
      4.5.3 Implementation ..................................... 75
   4.6 Next-hop and Mailbox Memories for Large CPN Networks .......... 89
      4.6.1 LPM in CPN ........................................ 90
   4.7 Conclusions ............................................. 94

5. Adaptive Routing in Wireless Sensor Networks .................... 95
   5.1 Introduction ............................................. 95
      5.1.1 Smart Routing in Wireless Sensor Networks ............... 96
      5.1.2 Communication Power Use ................................ 96
      5.1.3 Per-Task Packet Prioritisation .......................... 97
   5.2 Routing in Ad-hoc and Wireless Sensor Networks ............... 98
   5.3 Signal Strength and Transmission Power ........................ 102
   5.4 Implementing Smart Routing for Wireless Sensor Networks .... 105
      5.4.1 Protocol Classification ............................... 107
      5.4.2 Network Simulations .................................. 107
      5.4.3 Single Source and Destination (One-to-one) ............... 110
      5.4.4 Routing to a Common Destination (Many-to-one) .......... 113
      5.4.5 Experimental Evaluation ............................... 115
      5.4.6 Relative Overheads ................................... 118
   5.5 Packet Prioritisation by Re-Routing .......................... 120
      5.5.1 Simulation Environment ................................ 124
5.5.2 Random Re-Routing’s Performance .................................. 126
5.6 A New Approach to RRR .................................................... 130
  5.6.1 Diversion ................................................................. 132
  5.6.2 Inertia ................................................................. 136
  5.6.3 Herding ................................................................. 140
5.7 Conclusions ................................................................. 152

6. Quality of Service and Quality of Information .............................. 154
  6.1 Introduction ............................................................... 154
  6.2 Quality of Information .................................................. 155
    6.2.1 Measuring Quality of Information ............................ 157
  6.3 An Experimental Exploration of Sensor Network QoI ................ 159
    6.3.1 The Experimental Results ...................................... 161
    6.3.2 Extending the Experimental Setting with Multiple Simulta-
          neous Lights ......................................................... 167
    6.3.3 Quality of Service and Quality of Information - Some Remarks173
  6.4 Matching Requirements to Resources with
    Adaptive RRR Priorities .............................................. 174
    6.4.1 Determining Priorities Offline ................................. 174
    6.4.2 Determining Priorities Online .................................. 183
  6.5 Conclusions ............................................................. 190

7. Conclusions .................................................................. 191
  7.1 Contributions ............................................................ 191
    7.1.1 Network Quality of Service ................................... 191
    7.1.2 Wireless Sensor Network Quality of Service .............. 192
    7.1.3 Quality of Information ......................................... 193
  7.2 Future Developments ................................................... 193
  7.3 Final Remark ............................................................. 194

Appendix ........................................................................ 195

A. Single Routing Engine FPGA Based CPN Router
   (Additional Details) ......................................................... 196
  A.1 Router Processes ......................................................... 197
A.2 Memory Use .................................................. 199

B. Data Redundancy in Wireless Sensor Networks .................... 201

C. Wireless Sensor Network Testbed with Controllable Sensory Input . 204
   C.1 Testbed Architecture ........................................... 204
   C.2 Building Evacuation Simulator Wireless Sensor Network Integration 205

D. Random Re-Routing in the Wireless Context ........................ 208

Bibliography .......................................................... 210
LIST OF FIGURES

2.1 CPN packet header ........................................... 24
2.2 Representation of four neurons and the weights of their connections within an RNN ........................................... 26

3.1 Decision making algorithms in CPN routers ......................... 34
3.2 20-node testbed used for experimental comparison ................... 41
3.3 20-node testbed experimental QoS comparison: insensitive ........ 42
3.4 20-node testbed experimental QoS comparison: sensitive ........ 43
3.5 20-node testbed experimental QoS comparison: highly sensitive .... 44
3.6 20-node testbed experimental QoS comparison: route changes .... 45
3.7 46 node topology modelled after the Swiss academic Internet backbone 46
3.8 46-node testbed experimental QoS comparison ..................... 47
3.9 Average QoS of single flow within 46-node testbed for varying proportion of nodes running ∞-SR ............................. 48
3.10 Average processor ticks against number of neighbours for each algorithm ........................................... 49
3.11 A hardware design for the reduced RNN ........................... 51
3.12 Elements of the reduced RNN’s hardware implementation ........ 53
3.13 Hardware implementation of Goldschmidt division, simplified for use as an inverter .............................. 54
3.14 A hardware design for the approximated RNN ....................... 55
3.15 Hardware block for updating the weights in the approximated RNN 55
3.16 A hardware design for a sensible routing update engine .......... 56
3.17 A hardware design for generating pseudo-random values .......... 56
3.18 A hardware design for selecting the winner in SR ................ 57
3.19 Summary of hardware implementations .......................... 58

4.1 Bus-based router architectures ...................................... 62
4.2 Switch-based router architecture ...................................... 65
List of Figures

4.3 Single routing engine CPN router processes .................................. 68
4.4 Evaluation of QoS with and without independent interfaces .......... 73
4.5 CPN line card router processes .................................................. 75
4.6 Flow of a smart packet and corresponding acknowledgement through
    the router with independent interfaces ..................................... 76
4.7 Flow of a smart packet and corresponding acknowledgement through
    the router with centrally managed next-hop tables ...................... 76
4.8 Connections between input processing block and other router elements 79
4.9 State machine for the MAC-side of the input processing block .... 79
4.10 State machine for the switch-side of the input processing block .... 80
4.11 Connections between output processing block and other router ele-
    ments ............................................................................ 83
4.12 State machine for the switch-side of the output processing block ... 83
4.13 State machine for the MAC-side of the output processing block .... 84
4.14 Connections between switching/ARP processing block and other
    router elements .................................................................. 85
4.15 Receiving state machine for the switching/ARP processing block ... 85
4.16 Other state machines for the switching/ARP processing block ...... 86
4.17 Example of sub-networks for LPM in CPN ............................... 91

5.1 Current consumption corresponding to minimum transmission power
    level required in order to achieve less than 5% packet loss for differ-
    ent values of RSSI ................................................................. 104
5.2 Example simulation of the average energy required for successful
    dumb packet transmission along an 8 hop route against per trans-
    mission loss probability .......................................................... 106
5.3 The number of smart packets required for the transmission power
    used to communicate with the destination to converge, versus the
    distance of the packet source from the sink. Only one smart packet
    is active in the network at one time. ............................................ 112
5.4 Ratio between the QoS converged upon by the MCFA and tinyCPN
    protocols, versus the distance of the packet source from the sink .. 113
5.5 Ensemble average of route costs for 23 iterations of a series of smart
    packets for a source destination pair with a separation of 8 hops ... 114
5.6 Number of smart messages forwarded by nodes at differing distances from the common destination ........................................ 115
5.7 The number of smart packets required for the transmission power used to communicate with the destination to converge. Multiple smart packets are active in the network at one time. .................. 116
5.8 Ratio between the QoS converged upon by the MCFA and tinyCPN protocols, versus the distance of the packet source from the sink, in the many-to-one routing case ........................................ 117
5.9 Ensemble average of route costs 50 iterations for a series of smart packets for a source destination pair with a separation of 8 hops . 117
5.10 Experimental results comparing tinyCPN with the MCFA .......... 118
5.11 Smart and acknowledgement packet transmissions prior to convergence, in the single source-destination pair case .................. 119
5.12 Total network wide smart and acknowledgement packet transmissions prior to convergence for each source, in the many-to-one case . 119
5.13 Packet forwarding in RRR - HP packets are sent from the source \( S \) to neighbouring Node 3, as this is the best neighbour. LP packets are re-routed through either Node 1 or 4 with equal probability. . . 122
5.14 Example unmodified RRR simulation with two clusters of packet sources ................................................................. 127
5.15 Example unmodified RRR simulation with three clusters of packet sources ................................................................. 128
5.16 Representation of groupings of nodes forwarding LP packets .... 130
5.17 Ringed network topology model ........................................ 132
5.18 Packet forwarding in the modified RRR ............................... 133
5.19 Example RRR with deflection simulation with two packet sources . 134
5.20 Example RRR with deflection simulation with three packet sources 135
5.21 Example RRR with deflection and packet inertia simulation with two packet sources ....................................................... 137
5.22 Example RRR with deflection and packet inertia simulation with three packet sources ....................................................... 138
5.23 Impact of RRR on the per-hop forwarding time along one of the HP routes ................................................................. 139
5.24 Example RRR scenario with multiple LP packet sources, and the additional packet inertia and herding mechanisms 142
5.25 Example RRR scenario with multiple LP packet sources, and the additional packet inertia and herding mechanisms 143
5.26 Summary of the performance improvements for the three mechanisms 144
5.27 Ensemble per-packet energy use during the course of a simulation 145
5.28 Mean per-node energy use during three periods of the simulation 145
5.29 Maximum and total network wide energy use during the three periods of the simulation 146
5.30 The ratio of the packet latency without RRR to with RRR for low and high priority packets 147
5.31 The mean packet loss rates of high and low priority packets for the 10 rates of LP traffic, without and with RRR enabled 147
5.32 The packet loss rates of high and low priority packets over time for 10 rates of LP traffic 148
5.33 Impact of LP packet rate on the mean per-node energy use for nodes forwarding HP and LP packets 149
5.34 Example RRR scenario with multiple LP packet sources, but only one packet sink 151
5.35 The modified RRR algorithm in flow chart form 153
6.1 A basic WSN system 158
6.2 Example interpretation at the sensor network output of one or two readings 160
6.3 A time sample of the PSNR from the first experiment 162
6.4 An ensemble average of the PSNR over the light’s on and off cycles 163
6.5 An ensemble average of the pre- and post-network PSNR over the light’s on and off cycles 164
6.6 Reducing the network traffic load by selectively forwarding measurements 165
6.7 QoI resulting from only the top n readings, possible due to broadcasts 166
6.8 A sample from the experimental results with multiple simultaneous randomly placed lights 169
6.9 A sample of the results from Section 6.3.2. Number of incorrectly detected lights and the actual number of lights in the On state are shown. ................................................................. 172
6.10 Who, What, and How - The subset of the 5WH framework considered in Chapter 6 ................................................................. 175
6.11 Two examples of iteratively determining RRR priorities which satisfy the requirements of two tasks ........................................ 179
6.12 Two examples of iteratively determining RRR priorities which satisfy the latency requirements of three tasks ......................... 180
6.13 Two runs of a single HP, multiple LP scenario equivalent to that of Section 5.6 ................................................................. 181
6.14 Two runs of a single HP, multiple LP, and 1 sink scenario equivalent to that of Section 5.6 ................................................................. 182
6.15 Example simulation of RRR priority adaptation with two tasks, equivalent to the HP/LP scenario of Section 5.6 .................... 186
6.16 Example simulation of RRR priority adaptation with three tasks, equivalent to the HP/MP/LP scenario of Section 5.6 ........ 187
6.17 Example simulation of RRR priority adaptation with two tasks. Task B has a cluster of 4 sources, and Task A’s sources are distributed over the area of the network. The network has four sinks. 189
6.18 Example simulation of RRR priority adaptation with two tasks. Task B has a cluster of 4 sources, and Task A’s sources are distributed over the area of the network. The network has a single sink. ................................................................. 190

B.1 Representation of a data cluster .............................................. 202

C.1 Per-mote emitter follower LED circuitry ................................. 205
C.2 Custom control circuitry for the serial-port sensory-input LED infrastructure ................................................................. 206
LIST OF TABLES

4.1 Packet Processing Overheads ..................................... 71
4.2 Device usage summary ............................................... 89
4.3 Prefixes for example route in network from Figure 4.17 .......... 93
4.4 Prefixes for the second example route in network from Figure 4.17 . 94

5.1 Current consumption at discrete power levels by CC2420 radio ... 103
5.2 Distance ranges associated with discrete power levels in simulations 109
5.3 Probabilities assigned to each ordered neighbour for high and low
priority packets in the original RRR ................................. 122
5.4 Probabilities assigned to each ordered neighbour for packets of pri-
ority $P$ in the revised RRR ........................................ 123

A.1 Virtex-II blockRAM configurations .................................. 196
A.2 RNN in memory ..................................................... 199
A.3 Mailbox in memory .................................................. 199
A.4 RNN and Mailbox memory addresses ............................... 199

C.1 LED control byte .................................................... 204
1. INTRODUCTION

1.1 Providing Network Quality

In traditional general-purpose wireline and wireless packet switched networks, the Quality of Service (QoS) metrics are those measures of the network’s performance the user is interested in optimising. These metrics, for example the packet travel time or loss, or the link bandwidth, are inherently measurable. They impact the user’s experience in predictable ways, and are suitable as optimisation targets for the routing protocol.

For networks with a more focussed function, such as those in the emerging category of wireless sensor networks (WSN), the user may be less directly interested in the network quality expressed in terms of QoS metrics. Rather, application-specific measures of network quality are likely to be of more interest. In WSNs, the application’s output is information, and the user’s interest will be in the Quality of Information (QoI).

The network quality should therefore be expressed in terms of the user’s requirements. In a deployed network, the network’s routing protocol is responsible for the assignment of resources, the prioritisation of packets, and generally managing the QoS. The routing protocol should therefore adapt to achieve the user’s requirements, and match them to the available resources. This is true in both infrastructure networks and ad-hoc networks, although the function of the protocols and methods of providing the quality will vary.
A number of routing protocols and scenarios are considered in this work. Firstly, the Cognitive Packet Network (CPN) routing protocol’s decision making algorithm is simplified. CPN is then implemented in hardware to provide improved QoS at high speeds in wired infrastructure networks. Secondly, use of a version of the CPN protocol adapted for minimising energy use in WSNs is explored. Ultimately, it is discovered that a combination of the Minimum Cost Forwarding Algorithm (MCFA) with Random Re-Routing (RRR) provides an optimum combination of simplicity and flexibility to WSN QoS routing. Finally, the overlap between traditional network QoS metrics and the new concept of WSN QoI is developed to allow the routing protocol to match user requirements to network resources.

This work adds to the existing body of knowledge by introducing novel simplified approaches to the problem of ensuring optimal network performance on different QoS metrics which may change according to setting or task.

The principal research question driving the work is: How can low complexity adaptive routing be used to provide QoS in wired and wireless networks to improve performance according to user requirements?

Component aspects of this problem are addressed in the technical chapters:

- Can the QoS routing which CPN has been shown to provide be replicated with less complex decision making algorithms?
- Is CPN routing (using these less complex algorithms) suitable for implementation in dedicated hardware?
- Can the concepts of CPN be usefully applied in a different setting, namely large ad-hoc networks such as WSNs?
1. Introduction

- What implication does WSN QoS routing have for the ultimate aim of WSNs: quality of information?

1.2 Thesis Overview

This work is divided into a further six chapters. Chapter 2 provides the necessary background to the central concepts dealt with in detail in following chapters, including a brief overview of the prior art. This is followed by four technical chapters.

Chapter 3 proposes the use of less computationally demanding decision making algorithms in CPN than those currently available. This is motivated by the desire to overcome certain problems in implementing CPN in both dedicated hardware routers and simple battery-powered devices such as those used in WSNs. A set of algorithms, including some from the literature, and a new one introduced here, are compared with the Random Neural Network (RNN) currently used in the software router implementations of CPN. Comparisons of the QoS which CPN provides using the four algorithms are performed using network testbeds comparable in size to an Internet autonomous system.

The actual implementation of dedicated hardware routers for CPN using FPGA technology is examined in detail in Chapter 4. Two router architectures are presented, and are aimed at differing router applications. A single interface architecture is proposed for devices such as those which might be found in mass produced WSN devices. The second architecture presented is based on the designs of backbone infrastructure routers, and can forward packets at multi-gigabit rates. It has been designed in such a way that it would be compatible with a dual purpose CPN/IP router. These two architectures are presented to show that the ideas introduced by CPN are suitable for implementation in dedicated hardware routers,
and applicable to a range of tasks.

Chapter 5 also leverages the findings of Chapter 3, and starts with an adaptation of the CPN routing protocol for large WSNs. Its usefulness in such a scenario is explored. This is done both in simulation and using a real WSN testbed. It is concluded that, due to the concessions which must be made to allow for the restrictions of the WSN medium, the CPN protocol does not offer sufficient advantages over the protocols available in the literature. However, further investigation reveals that the Random Re-Routing (RRR) WSN routing protocol can more usefully be adapted to provide differentiable QoS in a manner suitable for WSNs.

The QoI generated by WSNs is likely to be the metric of most interest to the user, and the optimisation of QoI in this setting is experimentally examined in Chapter 6. The WSN testbed is used to compare an “image” of a phenomenon being monitored (in this case a map of light levels) at the input and output of the network. The results indicate that aspects of the QoI at the output of the network are influenced by network QoS metrics. Based on these insights, use of the RRR protocol to match network resources to user and task requirements is demonstrated. The protocol is also extended with an acknowledgement mechanism in order to allow it to adapt to meet requirements.

Finally, Chapter 7 concludes the work.
2. BACKGROUND

The overarching theme of this thesis is the application of packet switched network Quality of Service (QoS) in hardware and wireless sensor networks (WSNs). The focus for the hardware implementations is the Cognitive Packet Network (CPN) routing protocol, which employs a Random Neural Network (RNN) model to make autonomic routing decisions. This chapter introduces some of the concepts behind, and some implementations of, QoS mechanisms which are currently in use in networks such as the Internet. As background to the work which follows in Chapter 3, the CPN protocol, the RNN, and Field Programmable Gate Arrays (FPGA) are then described.

Finally, some prior art concerning WSNs is briefly covered.

2.1 Quality of Service in Packet Switched Networks

The current version of the Internet is best effort only. This means that although traffic is processed as quickly as possible, there is no guaranteed level of service. Network QoS is generally defined as providing a guaranteed level of performance in terms of factors such as latency and bandwidth. There are several schools of thought on QoS:

- It is a self solving problem as bandwidth will be cheap and plentiful (especially in light of upcoming fibre connections) in the future and more than cover that which is required [8].
The use of networks will always rise to fill the available capacity [9]. For example, supply induced demand has lead to new bandwidth-intensive applications, such as real-time remote surgery viewing [10], and even remote surgery itself [11].

In a world where the Internet is increasingly orientated toward multimedia uses, and with the prevalence of peer to peer applications, the latter is probably the case. This is especially true in the short to medium term, during which time bandwidth at the edges of the networks will not be cheap and plentiful, yet video on demand and other high-bandwidth applications are being introduced by internet service providers and third party broadcasters. Furthermore, while in the long run bandwidth availability may rise to be plentiful in the Internet, this may not be the case in other networks such as ad-hoc wireless networks.

With this in mind, the present section will examine some of the concepts underpinning QoS, and will also describe previous attempts to implement solutions to the challenges QoS presents.

Current approaches to QoS all emphasise the concept of classes of service, to deal with the fact that not all traffic will require the same level of performance. For example, telephony applications will require low latency and jitter, while file transfers and video require high bandwidth. More general every day Internet use, for example email, instant messaging, and web browsing, may however only require best effort. Different QoS classes will therefore cater for different requirements.

Many methods for providing QoS have been proposed by the Internet Engineering Task Force (IETF). Xiao et al. [9] and Aurrecoechea et al. [12] give an overview of some of the most important, which are the Integrated Services/RSVP model [13, 14] and the Differentiated Services (DS) model [15].
The Integrated Services (IS) model works by reserving paths and resources in the network. This is done using the RSVP protocol to first establish a path to the destination, telling the routers in the path to reserve resources and store flow state information, and then sending the data along this path. This model has three levels of service: guaranteed, predictive, and best effort. In the current Internet, all flow-related state information is stored in end systems, and IS requires the routers to be able to carry this information too. The IS model would therefore place a large strain on the routers as they would need to store amounts of information proportional to the number of flows - very high in the Internet core network. For guaranteed service, all hardware would need to support the model.

To reduce the amount of state required to be stored, RSVP can be combined with Multiprotocol Label Switching (MPLS) [16]. MPLS dates from a time when IP packet forwarding could not be performed entirely in hardware due to the routing table lookups, and the use of labels with associated forwarding rules was proposed. These labels would be assigned to packets at the ingress of an autonomous system (AS) based on their destination, and routing within the AS would then be based on a small set of these labels. Fast routing table lookups now mean that this functionality is redundant, but labelling can be used for traffic management. RSVP Traffic Engineering (RSVP-TE) [17, 18] uses RSVP to reserve a route which meets QoS requirements (in terms of, for example, bandwidth or hops), and sets up MPLS labels for packets to be forwarded using that route.

Differentiated Services (DS) use the type of service (TOS) byte in the IPv4 packet header to define the QoS class of the packet. The Internet service provider (ISP) attempts to provide a level of QoS for the customer according to the TOS byte and the Service Level Agreement (SLA) between the two. The SLA can be arranged in advance, or dynamically using a signalling protocol such as RSVP.
The ISP then uses classification\(^1\), policing\(^2\), shaping\(^3\), and scheduling\(^4\) mechanisms to provide different services. Differentiated Services are more scalable than Integrated Services. This is because the amount of state information is proportional to the number of classes (which are limited) rather than the number of flows. The operations to provide QoS are also only required at the boundaries of networks thus making DS easier to implement and deploy as it can be done in the ISP’s customer facing networks, while allowing the core routers to be as dedicated to quickly routing packets as possible. Integrated Services only has control at the edge of the network, and must therefore assume the core and networks closer to the destination to be capable of maintaining QoS.

The Differentiated and Integrated Service models are of use only during periods of high congestion in the network. This can occur as a result of high traffic load along all routes, or when there is an imbalance in the use of the network (as often results from current routing protocols which seek out the shortest path). Traffic Engineering can be used to control how traffic moves through the network, and thus balance the load over the available routes more evenly, avoiding unequal utilisation. This would work in conjunction with DS in that it would help prevent congestion, with DS providing QoS in case of congestion.

### 2.2 The Cognitive Packet Network

The CPN routing protocol introduced the concept of smart routing to provide best-effort QoS in packet switched networks [19, 20]. Previous protocols which aimed to provide QoS did so by adapting and patching the existing Internet technologies to

\(^1\) Classification: Packet sorting based on header content in conjunction with predefined rules.
\(^2\) Policing: Dealing with traffic which does not fit the policy.
\(^3\) Shaping: Delaying packets to force the traffic stream to fit a profile.
\(^4\) Scheduling: Choosing between different packets to send from multiple queues.
achieve their goals. CPN has been designed from scratch in such a way that it works within the existing OSI network stack model, yet is not encumbered by having to remain compatible with the existing Internet Protocol (IP). This allows existing low-level network infrastructure, such as Ethernet and MAC layer protocols, to be used. It also ensures that it is simple to port existing network applications.

CPN routers are able to continuously update the routes taken by packets in response to changing network conditions [21]. To do this, each router stores and maintains QoS data. For each destination and supported QoS class, routers store a combination of QoS metrics for each neighbour. These metrics are relevant to the QoS class in question, for example the number of hops to the destination for the “hop count” QoS class, and the delay to the destination for the “delay” class. These data structures are referred to as the mailboxes. These mailboxes are maintained when traffic is routed between sources and destinations in the network.

The CPN packet header can be seen in Figure 2.1. Of note are the route and the cognitive map. The route stores the addresses of the nodes, either taken, or to be taken by the packet, and is indexed by the route pointer. The cognitive map stores timestamps corresponding to the entries of the route, recorded as the packet passes each node.

Packet sources within CPN must initially discover valid routes between themselves and their destinations. In order to do this smart packets are routed based on the QoS information corresponding to the packet’s destination and QoS class stored in the mailboxes at each node. Initially the mailboxes are empty, and so smart packets must be forwarded at random. The routing decisions occur independently at each node the packet passes. Each node enters its address into the route, and a timestamp into the cognitive map. When the smart packet reaches its destination, loops are removed from the route, and an acknowledgement packet
is created with this route and cognitive map.

This acknowledgement packet is forwarded along the reverse route of the smart packet. Each node it passes can, depending of the QoS class of the packet, compare its previous timestamp from the cognitive map with the current time in order to determine the round trip delay to the destination, or compare the route length from the packet header with the current route pointer to determine the number of hops to the destination. These QoS metrics are stored in the mailbox corresponding to the acknowledgement packet’s previous hop. Once the acknowledgement packet reaches its destination (i.e. the source of the smart packet) the route is stored.

Data payload carrying dumb packets can be sent using these stored routes. Dumb packets are source-routed by setting the packet’s route equal to the stored route, which intermediate nodes use to blindly forward the packets. The arrival
of a dumb packet at its destination also results in a reply by an acknowledgement packet, which as with the smart acknowledgements results in mailbox updates.

Periodically, further smart packets are sent. Once a number of smart packets have successfully reached the destination, and smart and dumb acknowledgement packets have populated the mailboxes of the intermediate nodes with meaningful data, subsequent smart packets can be intelligently forwarded based on this data, with the aim of optimising the QoS of the traffic flow. Previous implementations have used the Random Neural Network (RNN) (described in Section 2.3 below) to make smart routing decisions. The use of other, less complex algorithms is examined in Chapter 3.

CPN has several benefits over the approaches to QoS used in IP networks.

- The use of end-to-end acknowledgement packets for both smart and dumb packets to continuously measure QoS metrics means that changes in network conditions are reflected in the routing decisions quickly.

- Distinct QoS classes mean that state is required to be stored for each destination/class pair (rather than for each flow).

- The combination of continuous measurement, source routing, dynamic routes, and user-selected QoS classes allows other applications to be built on top of or to extend the protocol. DDoS defence [22], admission control [23], and genetic algorithms [24] are examples of this.

### 2.3 The Random Neural Network

The RNN, first described in [25], is a biologically inspired algorithm which more closely matches the spiked operation of neurons than traditional artificial neural
Fig. 2.2: Representation of four neurons and the weights of their connections within an RNN

networks. Despite its non-linear structure, which is highly desirable for learning, the RNN’s state equations have been shown to provide an unique solution \[26\], which is an important characteristic for algorithmic implementations such as the one used in CPN.

2.3.1 Use of the RNN in CPN

The recurrent RNN model \[27\] used in CPN consists of one neuron per neighbour of the network node within which the RNN resides. The state of the RNN is defined by the probabilities that its neurons are excited. This is depicted in Figure 2.2, where \( q_i \) represents the excitation probability of neuron \( i \), \( w^+(i,j) \) and \( w^-(i,j) \) the rate at which excitation and inhibition spikes are sent from neuron \( i \) to \( j \) when neuron \( i \) is excited, and \( \Lambda_i \) and \( \lambda_i \) the external arrival rate of excitation and inhibition spikes to neuron \( i \). In fact, as is described by Koçak et al. \[28\], the rates at which spikes are sent to a particular neuron from each other neuron
remain identical as a result of the way the RNN is updated in CPN. Therefore only one positive and one negative weight ($w^+(i)$ and $w^-(i)$), representing the rate at which spikes are being sent from each other neuron, must be stored per neuron. The state equation of the RNN can therefore be given as follows.

\[ q_i = \frac{w^+(i) \sum_j q_j + \Lambda_i}{w^-(i) \sum_j q_j + \lambda_i + r_i} \]  

(2.1)

$r_i$ denotes the total firing rate of neuron $i$. This is the sum of all the weights of a neuron’s outgoing interconnections and is calculated by summing $w^+(i)$ and $w^-(i)$ for each neuron.

The weights are updated by reinforcement learning, a process first used with the RNN in simulating the routing of vehicles [29]. Each RNN has a reward ($R$) and threshold ($T$). The reward is related to a relevant QoS metric according to the QoS class; for example the reward may be inversely proportional to the delay. The value of the reward relative to the threshold determines the RNN update procedure. If the reward is greater than the threshold, the weights are updated in such a way that the neuron associated with the previous decision is rewarded. If the opposite is true then that neuron is punished. Before the weights are changed, the total firing rate $r_i$ is calculated. The weights are then updated as follows, where $j$ represents the previous winner (the neuron to be rewarded or punished), and $k$ all other neurons. This is repeated for all neurons $i$.

If $R \geq T_i$

- $w^+(j) = w^+(j) + (R - T_i)$
- $w^-(k) = w^-(k) + (R - T_i)/(n - 1), k \neq j$

Otherwise
2. Background

- \( w^-(j) = w^-(j) + (T_i - R) \)
- \( w^+(k) = w^+(k) + (T_i - R)/(n - 1), k \neq j \)

where \( n \) is the number of neurons in the RNN. An updated total firing rate \( r_i^* \) is calculated for each neuron, and the weights are normalised to prevent overflow by multiplying each element by \( r_i/r_i^* \). Equation 2.1 is used to iteratively update the excitation probabilities of each neuron until convergence. The neuron with the greatest excitation probability wins, and is used to select the outgoing link for the next smart packet.

The threshold is revised after each update in order to maintain an accurate representation of the network’s capabilities. This occurs according to Equation 2.2,

\[
T_{i+1} = aT_i + (1 - a)R, \tag{2.2}
\]

where \( a \) is a value between 0 and 1 (typically close to 0), which defines the rate at which the threshold updates, and \( T_{i+1} \) is the threshold used in the next update.

2.4 Field Programmable Gate Arrays

A good introduction to Field Programmable Gate Arrays (FPGA) and their architectures has been provided by Brown et al. [30]. FPGAs are reconfigurable devices made up of many repeating logic elements, and a routing fabric connecting them. These logic elements generally consist of a 4-input lookup table (LUT) and an optional register. By configuring the routing fabric and contents of the lookup tables, FPGAs can be made to perform a wide variety of tasks. Modern FPGAs also contain fast dual port on-chip RAM and have dedicated multiplier circuits, broadening their range of applicable uses.
In contrasted with software development, the advantage that FPGAs and dedicated hardware in general can bring, comes from massive parallelism. While the gigahertz clock rates of modern general purpose processors cannot be reached by FPGAs, these processors can generally only perform a limited number of operations per cycle. FPGAs on the other hand can be configured to perform any number of operations each clock cycle, and thus run applications that lend themselves to high parallelism at high rates of operations per second.

Traditionally the reconfigurability of FPGAs has made them useful for prototyping designs. Increasingly however the cost of producing dedicated ASICs rises such that only the most high volume products will be profitable using custom chip designs [31], and thus the use of FPGAs in final hardware becomes more attractive. Higher speeds may be achievable when implemented using dedicated ASIC designs, but the costs involved in producing such a device can be very high. FPGAs on the other hand can be reconfigured and designs tested relatively quickly.

There are several terms used in the following technical chapters related to FPGAs. Look up tables, mentioned above, are used to implement the logic of the design, and are the simplest measure of the device area used. The logic blocks are organised into slices, within which the interconnects are the fastest. The number of occupied slices is a further measure of device usage. Organisation of the logic within slices influences the optimality of the placement, and therefore more free slices would allow the design to grow while remaining optimised. The RocketI/O transceivers present in the Virtex-II FPGA devices allow multi-gigabit serial interfaces to be implemented.
2. Background

2.4.1 FPGAs in Networking Applications

Examples of FPGAs being used in networking applications exist in the literature. In [32] an FPGA based network application development platform called the Field Programmable Port-Extender (FPX) is described. It can be used to extend the functionality of a network switch at the input and output by operating on packets there, and has been used in a number of different applications. A frequent use of such platforms is for intrusion detection and packet filtering [33, 34, 35]. An example of an IP router on the Virtex-II Pro, which makes use of the RocketIO and PowerPCs is detailed in [36]. In this design, logic is used for common functions of the router (i.e. routing IPv4 packets) and the PowerPC processor, which unusually acts as a slave to the logic, handles less common functionality such as operations relating to IPv6 packets.

FPGAs often contain a number of dedicated cores, which implement some specific functionality more efficiently than is possible in logic. For example, most FPGAs will include a number of dedicated multiplier cores as this is a common arithmetic operation which would require considerable device area if implemented in logic. The Virtex-4 family of FPGAs [37] has dedicated the Ethernet MAC cores. In conjunction with the multi-gigabit RocketIO transceivers, these are capable of gigabit transfer rates.

2.5 Wireless Sensor Networks

Chapters 5 and 6 attempt to apply some of the findings of the chapters on CPN to large wireless sensor networks.

The model of WSNs considered in this work matches that presented previously in the literature [38, 39, 40]. The network consists of many devices, each of which
has some sensing capabilities, a small processor, memory, and a radio capable of short-range communications. These devices are battery powered, and may be distributed over large geographical areas at random to monitor some phenomenon. As the devices are not accurately placed and their measurements cannot be directly read the devices must self-organise to process and deliver measurements to the network output where they can be collated and acted upon.

In terms of packet routing, WSNs offer a number of challenges compared to wired infrastructure networks, and even compared to other types of wireless ad-hoc networks. The devices which make up the network may be densely placed in order to ensure thorough sensing coverage. This may lead to communications congestion as the transmissions of nodes will be overheard by many other neighbours. The operations of WSN devices are also typically performed on their simple processors in software. Therefore all processing dedicated to routing operations cannot be used for sensing and information processing related operations, the sensor network’s primary tasks. The routing protocols should therefore also use simple algorithms in order to preserve processing power.

2.6 Motivations and Remarks

Although some thought has been given to CPN in hardware [28], no working implementations have been produced. The use of CPN style smart routing is also untried in WSNs. The use of the RNN in CPN may however be a barrier to both. An RNN update in CPN has a number of division operations. These produce large and slow hardware due to the long carry path, and are also slow operations in software processors. The low powered processors of WSNs are especially unsuited for this. The use of simpler decision making algorithms will therefore be investigated.
3. ALTERNATIVE CPN DECISION ALGORITHMS

3.1 Introduction

Current network packet routers in the Internet backbone route Internet Protocol (IP) packets at up to 40 Gbps [41]. With the increasing use of the Internet for multimedia purposes fuelling a demand from users for higher quality video and audio, these rates only stand to increase further\(^1\). In order to achieve the per-packet processing rates required to sustain data throughput of this magnitude, dedicated network routing hardware is essential. In addition to these high data rates, applications such as real-time audio and video may require low delay, jitter, and loss in order to function in a suitable manner. Also, for financial and other types of applications, security in the infrastructure is vital. Large file transfers may however only be concerned with absolute bandwidth. Supporting these differentiable levels of service through Quality of Service (QoS) mechanisms further increases the processing required per packet.

At the other end of the bandwidth scale, ever more devices with processors of low computational ability and small memory capacities are being networked. Examples such as wireless sensor networks emphasise network lifetime as a QoS priority [42], and thus require routing protocols suited to their limited processing ability.

A number of methods to provide QoS in current networks, such as differentiated and integrated services [9], have been proposed, but they generally exhibit flaws in terms of scalability or effectiveness. An approach to the problem of providing QoS which has attracted recent interest in the research community is the use of autonomous adaptive behaviour by the network routing protocol. The Cognitive Packet Network (CPN) is an example of such an adaptive network, and has been developed in order to provide differentiable and improved best-effort QoS over the current generation of IP based networks [21]. Using neural network algorithms, routes are discovered and autonomically adapted to changing network traffic and topology conditions, so as to best utilise the available resources according to the users’ QoS goals [43].

CPN provides superior performance to IP under heavy traffic conditions [44], but development of the CPN protocol has so far predominantly focused on software PC routers, implemented as Linux kernel modules. Routing multiple flows at multi-gigabit rates is not possible in these software implementations. For that, as with IP, dedicated hardware is required. Research has so far also primarily been concerned with the Random Neural Network (RNN) [25] decision making algorithm, which does not lend itself to hardware implementation. The nature, complexity, and memory requirements of the RNN limit CPN to implementation in machines or devices with high computational resources.

This chapter attempts to establish whether a number of alternatives to the RNN, introduced in Section 3.2, which offer reduced complexity and resource requirements can be used in CPN while maintaining the proven routing quality of using CPN with the RNN. This would increase the viability of implementing CPN in dedicated hardware, and opens up the possibility of the concepts introduced by CPN being applied to other lower powered networked devices, such as wireless
sensor network Motes and personal communication devices. These are the motivating factors of this chapter, and are seen as important research goals in the development of autonomous adaptive routing.

### 3.2 Autonomic Routing Algorithms in CPN

Past work on CPN has treated the RNN as a characteristic and vital component of CPN [19], whereas here a different approach is taken. It is useful to think of CPN as the protocol; i.e. the packet headers and how they are interpreted, the packet types, the mailboxes and QoS classes, *et cetera*. The smart packet routing decisions within CPN are then made by some algorithm, *such as* the RNN. In this chapter the *approximated RNN* (aRNN), a greatly simplified algorithm which aims to mimic the routing decisions made by the RNN, is considered; as are several variations on *Sensible Routing* (SR), a probability based algorithm, in addition to the RNN.

All of the algorithms base their routing decisions, either directly or indirectly, on QoS data stored in the nodes of the network, and are therefore directly exchangeable within the framework provided by the CPN protocol.
The complexities of the algorithms are compared in a number of ways, including average software processor cycles, and device area when implemented on Field Programmable Gate Array (FPGA) [30] technology. Using both a 20-node and a 46-node test-bed, the relative QoS they provide is experimentally demonstrated.

3.3 The Approximated RNN

The function of the RNN as used in CPN is to select amongst the available neighbours when forwarding a smart packet. By independently selecting the best neighbour at each node, the RNN attempts to optimise the route each flow takes. During a reward scenario, the effect of this is to increase the excitation probability of the neuron associated with the previous decision and decrease the excitation probabilities of the other neurons. For the punishment scenario the opposite occurs: the excitation probability of the previously selected neuron is reduced, and all others are increased. This does not however happen directly. Instead, the weights are updated using reinforcement learning, and then the new excitation probabilities are calculated by iteratively running the RNN update process until they have converged.

The RNN model used in CPN involves divisions at several stages of the algorithm. In hardware designs, non-restoring division requires device area proportional to the square of the bit-width for which the design is required [45], and their long carry path results in slow operation. Iterative convergent division methods such as Newton-Raphson division [46] and Goldschmidt division (which is described later in this chapter) are faster, but can still require a number of iterations to complete. The speed restrictions are also seen in software implementations, as either the underlying hardware must implement one of these methods, or because each
stage of the operation must be performed in software. This makes the RNN, as used in CPN, unsuitable for implementation in hardware and devices with limited processing capabilities, such as those used in wireless sensor networks. The fact that the RNN model used in CPN only has a single neuron per neighbour also suggests that simplifications may be possible.

The approximated RNN (aRNN) is not a neural network model, but aims to approximate the smart routing decisions made by the RNN in CPN while only using simple arithmetic operators such as addition, subtraction, and bitwise shifts.

This is achieved by directly acting upon the “neurons” associated with the neighbours of the network node. Within an aRNN, each outgoing link of the network node has an associated weight (equivalent to the neurons of the original RNN). These are signed and initialised to zero.

In order to perform updates, the aRNN makes use of a QoS metric ($M$) and a threshold ($T$). As with the reward and threshold of the RNN, these determine reward and punishment scenarios. However, unlike the reward of the RNN, which is typically directly proportional to the quality of the link, the metric $M$ is inversely proportional to the quality of the link. For example it may be equal to the round trip delay or the packet loss rate. This avoids an additional inversion operation for many metrics.

The weights are directly updated as follows. Index $j$ is the previous winner; the weight associated with the previously selected outgoing link. Index $k$ represents all other neighbours.

\[
\begin{align*}
    w(j) &= w(j) + ((T_l - M) \gg 1) \\
    w(k) &= w(k) - ((T_l - M) \gg 1), k \neq j
\end{align*}
\]
Here $\gg 1$ represents a bitwise right-shift of one, or divide by 2.

As signed values are used, the reward and punishment scenarios are ensured without a comparison operation. In order to prevent overflow the weights are bounded to the range $[-z/2, z/2]$ where $z$ is the maximum 2’s complement positive integer value possible for the number of bits in use.

As is the case with the RNN in CPN, the weights are updated for each smart acknowledgement packet received. The behaviour of the weights is similar to that of the excitation probabilities of the RNN, which increase or decrease depending on the recorded QoS relative to the adaptive threshold. However, by acting directly on these weights and avoiding costly division operations, much of the complexity is avoided.

In practice it is observed that the weight of the currently selected neighbour’s weight increases until the threshold has converged with the measured QoS metric of the current class, or until the value $z/2$ is reached. The same occurs with the weights associated with the neighbours currently not being selected, except that of course they are decreasing. As the values of metrics such as delay are not typically constant, the difference between the weights of the “punished” neighbours and that of the currently “winning” neighbour provides a buffer which reduces the frequency of route changes. This mimics the buffering which the RNN provides.

### 3.4 Sensible Routing

CPN routers use acknowledgement packets to measure and store network QoS metrics which can then be used in order to make routing decisions which target some QoS goal. The RNN and aRNN use the stored QoS metrics indirectly, by basing their decisions on excitation probabilities and weights. A SR policy, introduced
in [47], is one that bases its routing decisions entirely on the expected QoS of the
different possible neighbours.

An example of such a policy is as follows. Assume that the expectation of a
QoS metric \( E(q_i) \), to some arbitrary destination is known for each outgoing link \( i \)
of a node. A SR policy may send packets on a particular link with a probability
proportional to the expected QoS of that link. The probability for each link would
be calculated by inverting each of the QoS metric expectations, summing these
inverted expectations, and then dividing the individual inverted expectations by
the sum, as shown below in Equation 3.3.

\[
P(j) = \frac{E(q_j)^{-1}}{\sum_i E(q_i)^{-1}} \tag{3.3}
\]

Although other definitions are possible, this particular definition is henceforth
referred to as the \textit{de facto} definition of Sensible Routing.

The most obvious benefit of using a SR policy in CPN in place of either of
the previously proposed algorithms is the memory savings it provides by using the
QoS metrics directly, rather than basing its decisions on the measured and stored
metrics indirectly. Using a neural network model as with the RNN, or using a
simpler algorithm such as the aRNN, means that state must be maintained between
updates. This requires memory for weights, excitation probabilities, rewards and
thresholds for a potentially large number of destination/QoS class combinations.

Sensible routing appears to share the RNN’s problem of costly divisions, both
from the inversions and the probability calculations. Algorithm 1 demonstrates
how the division from Equation 3.3 can be avoided, and is used in both the software
and hardware implementations discussed in this chapter.
### Algorithm 1 Avoiding division in sensible routing

1: \{There are $N$ neighbours\}
2: for $i = 1$ to $N$ do
3:   $e_i \leftarrow E(q_i)^{-1}$
4: end for
5: $S \leftarrow \sum_i e_i$
6: $R \leftarrow \text{RAND}[0, S]$
7: $x \leftarrow e_0$
8: $i \leftarrow 0$
9: while $x < R$ do
10:   $i \leftarrow i + 1$
11:   $x \leftarrow x + e_i$
12: end while
13: Select neighbour $i$ as the winner

#### 3.4.1 $m$- and $\infty$-Sensible Routing

An extension to the above involves raising the inverted expectations to a power $m$, where $m > 1$, before calculating the probability of selecting each link. This emphasises the links with the better expectation of QoS, and is called $m$-Sensible Routing ($m$-SR).

$$P(j) = \frac{E(q_j)^{-m}}{\sum_i E(q_i)^{-m}} \quad (3.4)$$

In [47] an insensitive metric measured along a route is defined as one that remains constant regardless of the traffic along that route, such as the number of hops. A sensitive metric as one which is influenced by traffic load, such as delay or packet loss. These definitions of the terms sensitive and insensitive are used extensively in the remaining sections of this chapter.

$m + 1$-SR has been theoretically shown to provide improved QoS over $m$-SR for insensitive metrics [47]. For sensitive metrics this holds true under certain conditions. Intuitively, for $m + 1$-SR to improve performance over $m$-SR for sensitive metrics, the impact on the measured metrics of increasing $m$ must be
lower than the gain which results from selecting the better links more frequently.

While varying $m$ to control the performance of a flow is a potential benefit of SR, the realities of implementation mean that anything other than 1-SR is not possible. This is because fixed point, the only realistic number format for hardware, small computational devices, and Linux kernel module implementations, does not offer the dynamic range to support high powers. $m$-SR is however a stepping stone towards the simplest approach examined here. As $m$ approaches infinity, the probability of selecting the link with the lowest expectation of some QoS metric (the best link) approaches 1, meaning that this link is always chosen. This avoids all operations except for the final comparison, and is henceforth referred to as $\infty$-SR.

CPN sends a small proportion of smart packets on random links. This was originally introduced in order to avoid the RNN becoming stuck in local minima, where a route has been found which results in successive reward scenarios, and is selected despite the existence of other undiscovered (or not recently measured) routes offering improved QoS. With $0 \leq m \ll \infty$, a proportion of packets would still be sent on links other than the best link, but as $m \to \infty$, much the same local minima problem observed with the RNN could occur. Sending a small proportion of smart packets on random links, as CPN already does, will therefore maintain up to date QoS data, even if the non-random decisions are always for the link with the best measured QoS.

### 3.5 Experimental QoS Comparison

The main aim of CPN is to maintain routing quality with respect to some user selected QoS goal. Therefore the choice of decision making algorithm must be
based foremost on the level of QoS it allows CPN to provide. One method to compare the algorithms is to do so experimentally.

The RNN and the algorithms described in Section 3.2 were compared in a number of experiments, carried out on a cluster of Linux PCs with CPN implemented as a kernel module [24]. The first set of experiments were performed using a 20-node wireline network arranged in a four by five grid topology (see Figure 3.2), with the network parameters selected in order to compare the performance of the algorithms for insensitive and sensitive metrics. As previously stated, an insensitive metric measured along a route is one which remains constant despite traffic along that route, such as the route length, and a sensitive metric is one which is affected, such as delay or loss. A further experiment was run on a larger, more realistic 46-node network topology (see Figure 3.7).

Each experiment was repeated for each of the four algorithms under test (the RNN, aRNN, SR, and $\infty$-SR), and run multiple times in order to ensure statistical reliability. Although not necessary, every node of the testbed was configured to use the same algorithm.

For the first experiment the links of the topology were set to 100 Mbps. The experiment involved five packet flows of 5 Mbps between nodes on opposite edges of the topology, and ran for 30 s. The CPN protocol was configured to minimise
Fig. 3.3: 20-node testbed experimental QoS comparison: Average over 500 iterations for a 5 Mbps flow with hop count as the metric to be minimised. 20 node topology uses 100 Mbps links, with 5 ms of artificial delay added.

the number of hops between source and destination. This is an insensitive metric. Barring losses, multiple flows within the network should therefore not influence each other. The experiment was repeated 500 times, and Figure 3.3 shows an ensemble average over the experiment’s 30 s for the flow between nodes 116 and 204. Each point is averaged over a 50 packet interval. The RNN, aRNN, and $\infty$-SR all converge upon approximately the same average number of hops; the RNN having a slightly improved performance. The aRNN shows a slower average learning process. SR results in a considerably worse average performance. This is expected as, due to the probabilistic nature of the algorithm’s link selection, a proportion of packets will always be sent along the longer routes, resulting in a greater average number of hops.

In the second experiment, the CPN protocol was set to minimise the delay
between source and destination. As the delay along a link is predominantly the queuing time spent in buffers, this is a sensitive metric: sending traffic along a link will result in increased delay for that link. However, for this experiment 5ms of artificial delay was added along each link. This, combined with the light load a 5 Mbps flow poses in a 100 Mbps link, means that the delay for a route is largely dependent on the number of hops within that route and that the measured metric is only slightly sensitive. Figure 3.4 again shows the ensemble average, this time for 1000 iterations of the second experiment. Similarly to the first experiment, the RNN, the aRNN, and ∞-SR show very comparable performance, with the aRNN adapting more slowly. Sensible routing is again the weakest algorithm.

For the third experiment the links of the network were set to the lower speed of 10 Mbps, and the artificial delay removed. The protocol was again set to minimise
delay, and the rates of the flows set to 3 Mbps. The number of flows within the network was increased from 5 to 7. The utilisation of the links of the network is therefore much higher. Switching a link from 100 Mbps to 10 Mbps is done by reducing the clock speed of the network device from 25 MHz to 2.5 MHz. This means that packets spend more time within queues in the network device; time which now, as the 5 ms of artificial link delay have been removed, forms the majority of the measured delay. The delay of a route in this experiment is therefore highly sensitive upon the traffic along that route, with changes in routing decisions of the individual flows having a greater effect upon the measured QoS metrics of other flows than in the previous experiments. The results are shown in Figure 3.5. Despite the sensitive nature of the metric, CPN using the less complex aRNN and ∞-SR maintain performance with the established RNN.
Figure 3.6 gives a more detailed example of the behaviour of a traffic flow when using the RNN and ∞-SR algorithms. The figure shows a 500 packet snapshot of the delay and corresponding enumerated routes. With all averaging removed it is possible to see that for both the RNN and ∞-SR, the delay varies from packet to packet, with occasional large changes. If the route changes are examined, it can be seen that in this interval, ∞-SR switches routes more frequently in response to delay changes, but still maintains comparably consistent delay and QoS. The RNN, aRNN, SR, and ∞-SR each have a mean (and variance) of 9.50ms (6.40), 8.71ms (1.79), 13.24ms (27.15), and 9.45ms (4.27) respectively over the second half of the experiment. The impact of frequent route changes in CPN has been studied in [48], where it was found that route switching can lead to an improvement
in overall QoS.

For the final experiment, the more complex, realistic 46-node network topology shown in Figure 3.7 was used. It mirrors the Swiss academic Internet backbone with respect to topology and link delays, and has proportionally approximated link capacities. A single flow, whose source and destination are randomly selected each iteration, is monitored. Up to ten other flows, each lasting 10 seconds, are also active between randomly selected sources and destinations at any one time. These flows provide background traffic, which (unlike the background traffic in the previous experiments) is not constant throughout the experiment.

Figure 3.8 shows an ensemble average over 300 iterations for each of the algorithms, and the results are similar to the previous experiments. In this case $\infty$-SR shows a small improvement over the RNN and aRNN. SR is again weaker. When the packet delays are averaged over the second half of the experiment the RNN, aRNN, SR, and $\infty$-SR have mean delays of 4.71 ms, 4.54 ms, 6.01 ms, and 4.36 ms
3. Alternative CPN Decision Algorithms

Fig. 3.8: 46-node testbed experimental QoS comparison: average over 300 iterations for a 5 Mbps flow with delay as the metric to be minimised.

respectively; with variances of 2.18, 1.65, 6.94, and 1.65 respectively.

These experiments have shown that the RNN, aRNN, and ∞-SR all produce comparable QoS. However sensible routing has poorer performance. The choice of parameters has produced both sensitive and insensitive metrics, for which results were similar. The results should therefore extend to other metrics which are subadditive along a route, such as the packet loss-rate.

3.5.1 Mixed Algorithm Environments

In the previous section it was shown that, from the perspective of the QoS, it is a viable option to use algorithms other than the RNN in order to make routing decisions in CPN. A question therefore arises as to the impact that a mixture of algorithms with varying performance might have on the overall network QoS.
The 46-node, single flow experiment was repeated for varying proportions of the 46 nodes running $\infty$-SR, one of the high-QoS algorithms, with the remainder running SR, the low-QoS algorithm. Figure 3.9 shows the mean delay achieved for each of the proportions. The performance varies approximately linearly with the proportion of nodes running the $\infty$-SR algorithm. This indicates that, while it is possible to use different algorithms on different nodes, the selection should be made from those that provide high QoS.

### 3.6 Complexity Comparison

In this section the several quantitative factors of each algorithm’s complexity are examined and discussed, covering hardware and software implementation.
3. Alternative CPN Decision Algorithms

3.6.1 Software computational time

Performing an RNN update in software is dominated by the iterative excitation calculation given in Equation 2.1, which, for \( n \) neighbours and a constant number of iterations is \( O(n^2) \). This is in addition to updating and normalising the weights, each of which requires \( 2n \) operations. This compares unfavourably with the other algorithms. Updating the aRNN only involves updating its \( n \) weights, and running the sensible routing algorithm means calculating \( n \) probabilities from the QoS data. Using \( \infty \)-SR reduces the required operations to a comparison only. The other three algorithms are therefore all \( O(n) \).

Figure 3.10 shows the number of processor “ticks” each algorithm took to complete plotted against the number of neighbours of the node. The data were recorded during 10 iterations of an experiment performed on the 46 node testbed shown in

![Fig. 3.10: Average processor ticks against number of neighbours for each algorithm](image)
Figure 3.7, in which 30 flows were being routed between randomly selected sources and destinations. The measured ticks for every flow were averaged for each number of neighbours. It can be seen that the RNN requires an order of magnitude greater computation time than the other algorithms (except for SR), and that the computational time of the RNN increases at a greater rate with respect to the number of neighbours than the other algorithms. It is worth noting that the random number generation used by SR constitutes most of its ticks.

3.6.2 Memory requirements

The memory requirements for all the algorithms increase proportionally to the number of neighbours. The RNN must store an excitation probability, and two weights per neighbour. The aRNN requires only one weight per node in addition to the QoS metrics to be stored, and sensible routing and its derivatives require less memory as only the QoS data must be stored to perform updates.

3.6.3 Hardware implementations

The routing algorithm comparison detailed in this paper was motivated by research into implementing CPN using dedicated hardware. Regardless of the quality of the routing they produce, the routing algorithms do not lend themselves to implementation in hardware equally. Dedicated hardware implementation allows acceleration of the algorithms if high degrees of parallelisation are possible.

For this comparison the four algorithms were implemented for a Xilinx Virtex II Pro XC2VP30 FPGA device. The number of neighbours the designs support can be selected prior to the build process. As a concession to the FPGA device, 18-bits are used for numerical representations (compared with 32-bits in the software implementations). This means that the fast 18x18-bit multipliers that the device
provides can be used. The RNN and SR designs furthermore use fast Goldschmidt dividers [49] in order to minimise area and maximise speed. Goldschmidt division is explained in the following section.

Block diagrams of these designs and their elements are shown in the series of Figures 3.14 to 3.17. Each design is pipelined in such a way to allow the designs to run at 75 MHz on the target device. The number of clock-cycles required to complete an update or select a next-hop by each of the designs is compared below. This is defined in part by the number of pipeline stages, and also by the number of iterations each design potentially requires.

\textbf{RNN implementation}

Figure 3.11 shows the overall design of the RNN, based on the reduction of the algorithm described in [28]. The design performs weight updates in parallel, normalising the weights, and finally updating the values of the excitation probabilities and selecting the largest. In the lower left of the block diagram the logic for updating the reward (from the measured metric) and new threshold (from the previous
threshold and new reward) is depicted. This logic also evaluates whether the reward or threshold is greater, and determines the absolute difference between them. The designs for the various sub-elements are shown in Figure 3.12.

The weight updating block (Figure 3.12a) calculates all four possible outcomes (reward or punishment scenario, previous winner or not) for the positive and negative weight. These are calculated in parallel and then selected using a series of multiplexors.

Normalising the weights (Figure 3.12c) is done by first calculating $r/r^*$. Due to the way in which the design is pipelined, the value of $r/r^*$ will be ready in time for when the updated weights are calculated. The value is then multiplied by the updated weights to normalise them.

The normalised positive and negative weights are then used in combination with the previous excitation probabilities. This is an iterative operation which implements Equation 2.1. Each time the equation is evaluated the output excitation probabilities are compared with the ones from the previous iteration (or, after the first iteration, the previous update). If the difference is smaller than some value “theta” for all values, the “done” signal goes high and the outputs can be used and stored.

When normalising the weights it is necessary to perform divisions. These are performed using iterative Goldschmidt dividers. Goldschmidt division iteratively multiplies both the numerator and the denominator by a factor $F_i$, in such a way that the denominator ($D_i$) converges on one, and the numerator on the quotient. The initial factor $F_0$ is estimated, and each subsequent factor can be calculated using $F_{i+1} = 2 - D_i$ [50, 51]. This operation lends itself to hardware implementation as it is simply the 2’s complement of the denominator, which is determined by inverting the bits of the denominator. This property is dependent on the accuracy
3. Alternative CPN Decision Algorithms

(a) Updating the weights

(b) Normalising the weights

(c) Updating the excitation probabilities

Fig. 3.12: Elements of the reduced RNN’s hardware implementation
Fig. 3.13: Hardware implementation of Goldschmidt division, simplified for use as an inverter

of the initial estimate for $F_0$. In the hardware implementation used in these designs, a fixed set of 16 pre-calculated “estimates” are stored in a ROM for use in the first iteration. The number of iterations is also predetermined (based on the number of iterations required to reach an error ratio threshold of 0.025), and depends on the range within which the denominator falls.

Figure 3.13a shows the design simplified for use in an inversion operation, and Figure 3.13b shows the relative error incurred in doing so for the full range of denominator for the estimates and iterations used. The inverters can be used to perform division by multiplying the output of the denominator inversion by the numerator.

**Approximated RNN implementation**

The design of the aRNN update logic is shown in Figure 3.14. The weights are updated based on the measured QoS metric and the threshold. The threshold is updated based on its previous value and the new metric. Similarly to the RNN design, updating the weights is performed in parallel, in this case with the very sim-
Fig. 3.14: A hardware design for the approximated RNN

Fig. 3.15: Hardware block for updating the weights in the approximated RNN

ple logic block shown in Figure 3.15, which either adds or subtracts the difference between the threshold and the QoS metric to the weight depending on whether the weight is associated with the previous winner or not. The updated weights are then used to determine the new winner (by selecting the largest weight). This occurs in parallel to the adjust logic block, which bounds the weight values within the range $[-z/2, z/2]$ as discussed in Section 3.3.

**Sensible Routing implementation**

The SR implementation used in the comparison is based on Algorithm 1. The SR design takes the measured QoS metrics as input. Each of these is inverted using the
Goldschmidt inverter from Figure 3.13a before being summed. The linear feedback shift register shown in Figure 3.17 is used to generate a pseudo-random number. The design is based on the primitive polynomial $x^8 + x^6 + x^5 + x + 1$ [52], and therefore provides a cycle of $2^9 - 1$ pseudo-random numbers before repeating. This random number is multiplied by the sum of the inverted QoS measurements in order to generate a random value between 0 and that sum.

The compare block shown in Figure 3.18 is then used to select the winner by placing the scaled random value in one of the ranges defined by the cumulative sums of the inverted QoS values.
Fig. 3.18: A hardware design for selecting the winner in SR. The pipelining is shown using thick vertical lines.

Summary of implementations

The number of look-up tables (LUTs) and multipliers shown in Figure 3.19 give a measure of the device area required for each implementation. One would expect the device area required to increase proportionally to the number of neighbours the hardware is designed to support, and this is borne out by the results. The divisions required by the RNN and SR result in larger designs. The aRNN and $\infty$-SR have the smallest, lowest latency designs.

The execution times of the aRNN, SR, and $\infty$-SR designs are all constant whereas the RNN has a variable number of iterations. A deterministic execution time is beneficial in high speed routers, where queuing should be avoided. The parallelism of the hardware designs results in reduced execution time compared to the software implementations, as the execution times of the software designs are all $O(n)$, or $O(n^2)$ in the case of the RNN, whereas the hardware designs for the RNN, aRNN, and $\infty$-SR are all $O(1)$ for the main stage (due to parallelism), and $O(\log_2 n)$ for the comparison stage to select the winner. The SR implementation
has an $O(1)$ main stage, and an $O(n)$ comparison stage. The number of pipeline stages shown in Figure 3.19 indicate the number of clock-cycles required for one run of the design to complete, for a number of sizes. One iteration is assumed for the RNN, making the stated number of clock-cycles required a conservative value.

The trend in high-speed hardware routers is against queuing; and the modern router must be capable of performing routing table lookups and packet processing within a set time. The use of single clock-cycle lookup tables such as content addressable memories, rather than searching algorithms, is therefore increasingly common. Similarly, in CPN, a hardware decision making algorithm should be fast and have deterministic run-time. This is a further factor in favour of the aRNN and $\infty$-SR.

Fig. 3.19: Summary of hardware implementations
3.7 Conclusions

By avoiding divisions, the approximated RNN and $\infty$-SR offer smaller and faster implementations than the RNN, both in software and hardware. They also require less memory, and scale better with respect to the number of neighbours. It has been experimentally demonstrated that QoS in adaptive networks such as CPN is maintained when using these less complex algorithms to perform routing decisions. This is beneficial as it makes CPN suitable for implementation in dedicated hardware for high speed routers. It furthermore potentially enables the smart routing concepts introduced by CPN to be used in software driven devices lacking powerful processors and large memories, such as PDAs or wireless sensor network devices.

Another conclusion which can be drawn from these results is that the adaptivity of CPN does not depend as heavily on the learning algorithm as was previously thought [21]. Instead, its ability to adapt quickly and accurately appears to stem mainly from its probing nature and the accuracy and rate at which measurements are returned. This is why the simpler algorithms, such as $\infty$-SR, provide the same level of QoS as the more complex RNN.
4. FPGA BASED HARDWARE CPN ROUTER
IMPLEMENTATIONS

4.1 Introduction

In order to realise complete communications networks it is important that hardware based approaches are available for the elements which form the backbones of the systems. This is true for the Internet as a whole, for the smaller networks which form the constituent members of its edges, for mobile phone networks, and for conventional phone networks. It is assumed that it would also be true for any future networks using adaptive protocols such as CPN.

Such networks would need to be capable of supporting the traffic loads that modern distributed applications require. This is to be expected as the CPN protocol was designed to monitor and adapt to changing network and traffic conditions, thereby enabling it to exceed the QoS delivered by existing protocols. These benefits offered by CPN over other protocols are at risk of being lost, however, if the hardware to facilitate equivalent bandwidth to the other protocols is lacking. This means that low-cost application specific microchips, which implement the levels of the network stack within which CPN operates must be viable. Furthermore, a protocol such as CPN may be expected to operate within a mixed protocol environment. It is therefore also important that the architectural approaches needed to implement a CPN router be compatible with the approaches taken for existing
protocols. The most notable alternative protocol and accompanying router design is that of the Internet, namely the Internet protocol (IP).

This chapter examines the prior art related to CPN in hardware (and to a lesser extent the RNN), before building on the work of the previous chapter to tackle the actual implementation of a functioning CPN router in hardware.

4.2 Dedicated Hardware for Packet Switched Networks

Backbone routers in the Internet today have switching fabrics capable of up to and over 1 Tbps with individual line-cards capable of 40 Gbps. It is unrealistic to emulate these speeds outside of a dedicated network hardware fabrication laboratory, but gigabit speeds such as OC48 (2.5 Gbps) are reasonable targets.

Routers are split into pipelined stages. In their paper [53], Crowley et al. compare several architecture approaches for network processors (specialised devices for high speed network trafficking). Although some of the designs have deep pipelining of up to 7 stages, there is a general approach of 3 stages as follows. The incoming packets are first stored in memory, then classified and processed, and finally forwarded to the output channel. These three operations occur concurrently. Pipelining and massive parallelism are powerful aspects of dedicated hardware design.

4.2.1 Three Generations of Internet Routers

Although network routers have shown gradual improvement over the years, both in the academic and commercial spheres, one may say that there have been three key generations of router architectures used in the Internet [54, 55]. The development of each generation has been driven by a need to increase throughput and interface
capacity, but not all current applications require the same resources. For example, network backbone devices must have the bandwidth to support thousands of simultaneous traffic flows, while the routing engine of a wireless sensor network device may demand only limited forwarding capacity, or have only a single interface (the aerial) by design. It is therefore important to examine each architectural approach in turn, as design lessons, applicable to a variety of applications, may be learnt.

**Generation I**

The first generation of packet switched network routers consisted of a number of interfaces all connected to a single software based routing engine and memory via a shared bus (Figure 4.1a). During a single routing operation on this architecture of router, a packet would enter at one of the router’s line-card interfaces, and traverse the bus (once free) to the routing engine’s memory. From there the packet would be parsed, a next hop would be selected from the routing processor’s routing table, and finally the packet would be sent to its outgoing interface, again via the shared...
bus.

The router architectures described in [56] approximately match this generation. In that paper’s first architecture, incoming packets are written to a shared memory via the bus, from where the routing processor reads the header. The processor then writes the updated header back to the shared memory, and finally the output interface reads the complete packet. This means that the shared memory is accessed via the bus a total of four times. The paper’s second architecture optimises this by enabling the processor to read the packet header at the same time as the packet is initially written to the memory, thus reducing the number of times the memory is accessed for each packet to three.

This generation of architecture has two major bottlenecks which prevent it from scaling effectively. First of all the single packet processor must handle the routing of all packets from all interfaces, including the time intensive routing table lookups. Secondly, each packet traverses the shared bus at least twice, severely limiting throughput.

**Generation II**

The routers which could be said to belong to the second generation of routers attempted to overcome the single routing engine or shared bus bottlenecks. They did this either by route caching or using multiple parallel packet processors. Both these approaches began the trend seen in later architectures of moving and distributing the processing to the interfaces.

Route caching aims to prevent the majority of packets from needing to traverse the shared bus twice. By maintaining a list of recent and common routing destination next-hops in a small fast memory locally in the line-cards, the packets with destinations resulting in cache-hits can bypass the main central routing pro-
cessor, and can be routed by simple fast routing logic in the line-cards (Figure 4.1b). Packets routed by this method are said to be on the fast-path, as opposed to those which are routed by the central processor which are on the slow-path. The fast-path has the additional benefit of reducing the load on the central processor, which must now also maintain the route caches.

This approach still fails to scale, primarily as the routing speed is limited by the shared bus, but also as the throughput becomes traffic pattern dependent. In backbone routers, the number of simultaneously active flows is very high, and temporal correlation very low. There is therefore a large cache churn rate which has associated overheads. Depending on the speed of the central software processor managing the caches, this can become a further bottleneck during bursts of traffic.

In [57], multiple routing processors are used. Incoming packets are assigned to the processors for parallel processing in a round-robin fashion, and exit the router in the same order to ensure the ordering remains consistent. As previously, some processing is moved into the line-cards. The line-cards only send the packet header to the routing engines, which performs the necessary modifications before returning the header to the original line-card. There, it is recombined with the packet’s data and sent to the outgoing interface. This again reduces the number of times the large data payload traverses the bus to one, greatly reducing contention.

In these designs the routing engine or engines remained connected to the interface line-cards by a shared bus, and this quickly became the new bottleneck. This was addressed comprehensively in the third generation.

**Generation III**

The third generation overcame the bottlenecks in two ways. In [58] the shared bus was replaced by a crossbar switching fabric, which allows packets to simultaneously
travel between different interfaces and processors. Similarly to [57], this router separates the packet headers from their bodies at the incoming line-card, and sends them on to one of the routing processors. The updated headers are then returned to the original line-card, reunited with the packet bodies, and sent over the switching fabric to the outgoing interface. The routing processors maintain a full copy of the routing table, and maintain a cache for quick lookups. The caches still have the same shortcomings as those of the 2nd generation routers, namely that performance is dependent on traffic patterns.

The logical conclusion of these advances in router architectures was to fully distribute the routing operations into the line-cards themselves and to take advantage of falling DRAM prices [59] by including redundant routing tables where every possible routing table prefix can be indexed by the destination address within two memory look-ups. An example architecture for this is described in [60]. This means that all routing operations can now be performed in the fast-path, and that the software central processor implemented slow-path can be dedicated to the less regular or time-critical operations, such as ICMP packet handling and routing
table updates (which must now be propagated to the line-cards). Furthermore, each line-card now contains its own dedicated routing engine, responsible for the fast-path routing operations. As these are extremely uniform operations the fast-path routing engines can be implemented using specialised hardware devices which are capable of forwarding traffic at the rates currently witnessed in the Internet backbone. This architecture is depicted in Figure 4.2.

As packets now traverse the switching fabric a single time and routing table lookups are bounded only by the memory access times, this architecture can scale to large numbers of interfaces and benefit from the continual speed improvements which advancing chip and memory fabrication techniques bring.

**Relevance to CPN**

The following sections of this chapter describe two CPN router designs which respectively share features of the first and third generation IP routers described above. These are currently the most relevant architectures.

Single routing engine architectures have obvious uses where there is only a single interface, such as in wireless devices, like personal mobile communicators and wireless sensor network devices. Such mass-production devices are suitable for system-on-chip approaches. These could also be used for routers where traffic loads are lower, such as in a private home, where the number of simultaneous flows will be greatly limited compared with the number closer to the core of the network.

Within the core or backbone of the network however, greater bandwidth is required. It is therefore important that a third generation equivalent router exists, capable of forwarding the aggregated flows of thousands of users.
4.3 Cognitive Packet Networks in Hardware

There has been some previous work on CPN in hardware. Koçak et al. detailed the design of a smart packet processing engine, as well as the reduction of the number of operations of the reinforcement learning update from $2n^2$ to $2n$ \cite{28}, as was detailed in Section 2.3.1. Their design was limited to simulation, and did not incorporate features such as the dumb packet switch, necessary for forwarding data packets.

4.3.1 Field Programmable Gate Array Based Implementations

The aim of this work is to realise working systems, and two approaches to the design of a CPN router have been taken. These correspond to the first and third generation of hardware router architectures described in Section 4.2.

These implementations should be seen as proof-of-concept as they are based on development boards and are therefore not suitable for use in environments beyond the laboratory, but have been used to explore the specific requirements which a CPN router has of hardware compared with equivalent IP router designs. Furthermore, both implementations are limited by the FPGA development hardware used. This is also further discussed in Sections 4.4 and 4.5.

4.4 Single Routing Engine FPGA Based CPN Router

(1st Generation Internet Router Equivalent)

The first design is targeted at the Celoxica RC300 FPGA development board which uses a Virtex-II FPGA device \cite{61}. This is a large device, with dedicated multiplier blocks. It is sufficient in size for the requirements of a single logic based routing-engine. The development board itself features several DRAM memories suitable
4. FPGA Based Hardware CPN Router Implementations

Fig. 4.3: Single routing engine CPN router processes

for the next-hop lookup tables required by CPN, but has only two Ethernet ports.

This first design motivated the development of the approximated RNN (aRNN) described previously in Section 3.3, and thus makes use of that algorithm rather than the conventional RNN used in the software implementations.

Also, unlike the 1st generation Internet routers from the literature, which use software microprocessors for routing operations, this design implements all its processes in logic. This is important if the design is to be used in a low-power device with a system-on-chip.

4.4.1 Architecture

Figure 4.3 shows the overall pipelined design of the hardware CPN router. There are five stages running concurrently, which results in low latency routing. The five stages are input (RX), classification, processing, output ‘merging’, and output (TX).

Packets are read from the MAC interface one byte per clock cycle at the input stage. Every four clock cycles a 32-bit word is written into the input port’s corresponding FIFO input buffer. Packets from the input buffers are classified into the CPN and CPN ARP types according to their Ethernet headers, and written into

---

1 Ethernet headers, which are attached to all packets, contain the source and destination MAC addresses, and define the type of packet.
that particular packet type’s input buffer. CPN packets are the smart, dumb, and acknowledgment packets described in Section 2.2. CPN ARP is an adaptation for CPN of the IP address resolution protocol [62], initially needed to learn the immediate neighbours’ 48-bit MAC and 32-bit IP or CPN addresses. These are necessary for routers to communicate at the Data-Link and Network layers of the OSI network protocol model. Other packet types are dropped.

The process ARP block parses ARP packets and fills the ARP table with the CPN addresses, MAC addresses, and the corresponding ports of the router’s neighbours. It also responds to ARP broadcasts with the router’s addresses.

The process CPN block handles all four CPN packet types. For smart packets the router will lookup the aRNN corresponding to the destination and QoS class of the packet, and as necessary create a new aRNN with a next hop randomly selected from the available neighbours, or lookup the next hop from an already created aRNN. This next hop is written to the packet’s route and the packet is forwarded. In the case of dumb packets, the next hop is looked up from the route based on the route index (recall the CPN packet header from Figure 2.1) and the packet is forwarded to that neighbour. For the two acknowledgement types the next hop is also looked up from the route, and the mailbox is updated with information relevant to the QoS class (delay based on timestamps, number of hops to destination, etc.) For smart acknowledgements the aRNN is updated and a new next hop is chosen. Should the router be the destination for smart or dumb packets, acknowledgements are sent.

The process blocks write packets to the buffers corresponding to the type and output port of the packet. The output control blocks then read from the packet type and output port buffers to the output buffers. From there the packet TX blocks write the packets to the interface.
Additional details on the implementation, including the memory configurations used to store the next-hop tables and mailboxes can be found in Appendix A.

4.4.2 Device Usage and Speed

The implementation described was built for the Virtex-II XC2V6000 device. The design uses 26% of the available 67,584 LUTs, and occupies 43% of the total 33,792 slices. It achieves a 39.477 ns register to register delay, which allows it to run at 25.175 MHz (the clock speed the development board provides).

This clock rate is low for a modern FPGA, but it is projected that through optimisation, speeds of around 50 MHz should be easily obtainable. This is borne out by the second architecture from Section 4.5 below, which runs at 75 MHz.

The process CPN block can read and write one word from the input buffer and to the output buffer per clock cycle. If a packet were to be copied from one buffer to the other, no overhead would be incurred. It is however the case that processing, such as parsing the header and updating the RNN, must be performed. Assuming route lengths of 15 and DPs carrying 1024 bytes of payload, Table 4.1 shows the overhead in clock cycles and number of 32-bit words per packet for the four packet types. Further assuming that SPs make up 20% and DPs 80% of the outgoing traffic by number, and that each packet has an equivalent ACK, the average data rate through the process CPN block for this case will be 625 Mbps if the RNN weights are updated for smart ACKs only, and 613 Mbps if they are updated for both types. This is in excess of the 400 Mbps the packet RX blocks combined can provide, and a conservative number as Ethernet packets can be up to 1500 bytes in size. This total rate is independent of the number of flows the router is handling.

Some micro-aspects of the implementation are non-optimal, and the maximum clock rate of the resulting design suffers as a result. What has been demonstrated
4. FPGA Based Hardware CPN Router Implementations

<table>
<thead>
<tr>
<th>Packet Type</th>
<th>Overhead /clock cycles</th>
<th>Data /clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>45</td>
<td>37</td>
</tr>
<tr>
<td>DP</td>
<td>37</td>
<td>293</td>
</tr>
<tr>
<td>SACK</td>
<td>51</td>
<td>37</td>
</tr>
<tr>
<td>DACK</td>
<td>42 (51)</td>
<td>37</td>
</tr>
</tbody>
</table>

Tab. 4.1: Packet Processing Overheads

however is that this approach is viable for simple single device CPN routers, equivalent to IP routers found in the home, or even used as part of wireless sensor networks used in future home automation systems or as part of larger scale environmental monitoring in any number of scenarios. The use of CPN and smart routing in wireless sensor networks is explored further in the following chapter.

First however, the second CPN router implementation is presented.

4.5 Multiple Routing Engines

(3rd Generation Internet Router Equivalent)

A single packet processor is acceptable for a moderate number of physical links, but high-speed routers must be capable of handling a large number of these connections, which would lead to contention for the single processor. An architecture for a high-speed CPN hardware router comparable to that depicted in Figure 4.2, with a routing processor per interface, is therefore proposed.

The router design detailed in the previous section uses a single processing stage to handle all packets. The modern packet routers used in the Internet infrastructure by comparison make extensive use of multiple parallel packet processors.

As previously described, a typical high-speed router consists of a routing processor on each line-card, which handles the fast-path, and a central routing processor
and routing-table memory for the slow-path and router maintenance operations. A crossbar-switching fabric efficiently links these elements. This was illustrated in Figure 4.2. The main function performed by the fast-path of an IP router is the routing table look-up. This is done using custom, highly optimised ASIC designs, and the majority of packets can be routed using the fast-path [63]. The slow-path handles packets the fast-path cannot. This includes ICMP packets, and packets that need fragmenting. It also maintains the central routing table and is responsible for propagating updates to the routing tables in the fast-path. Due to the wide range of functionalities, and smaller proportion of traffic it must handle, the slow-path can be implemented using a programmable microprocessor.

Acknowledgements from dumb and smart CPN packets result in either a QoS table update, or both next-hop and QoS table updates respectively. The number of updates performed in a CPN router would therefore far exceed those of an IP router, and as a result, create higher levels of overhead communications between interfaces and a central processor. Smart packet next-hop tables could either be maintained independently at each interface, in which case each interface’s routing processor would need to perform next-hop table updates; or in a central routing processor, and mirrored to the individual line-cards.

The “independent interfaces” approach has advantages in its simplicity and overheads. Its viability from a QoS perspective is first investigated, before discussing the resulting architectures.

4.5.1 Independent Interfaces and Quality of Service

The 20-node testbed, highly sensitive experiment from Section 3.5 was repeated in order to compare the QoS resulting from the independent interfaces and centrally managed architectural approaches. The results are shown in Figure 4.4. For this
version of the experiment the software version of the CPN router implementation was adapted to include the option of maintaining separate QoS and next-hop tables for each neighbour, and used the $\infty$-SR algorithm. The results show that the resulting QoS degrades when individual independent tables are maintained for each neighbour.

The independently maintained mailboxes are slower to update, and therefore responses to changes in traffic conditions are slower. This is therefore a negative consequence of using the simpler architecture offered by maintaining the interfaces independently.

4.5.2 Router Architectures

Both of the architectural approaches based on the 3rd generation of IP router architectures have line-cards with the overall topology shown in Figure 4.5. The
independent interfaces CPN line-card routing processor architecture’s four main parts have the following functionality:

- The *input processing* block selects a CPN packet’s next hop, either from the route stored in the packet header in the case of dumb and acknowledgement packets, or from the routing table memory in the case of smart packets. The logic block also updates CPN packet headers. ARP packets are forwarded to the ARP processing block.

- The *output processing* block forwards packets to the interface. For acknowledgement packets it also sends QoS measurements, extracted from the packet’s cognitive map, to the memory block.

- The *ARP packet processing* block adds neighbours to the memory when it receives ARP packets. It also responds to ARP requests, and broadcasts ARP messages on to the switching fabric for other line-cards to maintain their neighbour lists. The block also forwards CPN packets to the relevant output line-card via the switch, and forwards packets received from the switch to the output processing block.

- The *memory* block contains the next-hop table, the QoS tables, and a list of neighbours. It is capable of updating the next-hop table based on the information stored in the mailboxes, using the implemented decision making algorithm.

This design ensures that acknowledgements are processed in the same routing processor as their corresponding smart or dumb packet, as is shown in Figure 4.6. The centrally managed approach moves the QoS tables away from the line-cards into a central routing processor where a single copy is kept. This increases
the complexity of the router, but reduces overall memory requirements and ensures synchronisation between interfaces. Upon receipt of a smart or dumb acknowledgement packet, an internal message packet is sent to the central routing processor. This message packet contains the acknowledgement packet’s previous hop (in other words, the corresponding dumb or smart packet’s selected next hop), the destination, QoS class, and the measured delay and number of hops to the destination. From this the central QoS and, if necessary, next-hop table can be updated. If the next-hop table is updated, the new next-hop is broadcast through the switch to the line-cards, where their local next-hop tables can be maintained. The flow of packets through the architecture is shown in Figure 4.7. In order to support these message packets the switching fabric would require approximately 6% more bandwidth for a 32 neighbour router compared with the independent interfaces case.

4.5.3 Implementation

These architectures have also been implemented using FPGA technology. These proof-of-concept implementations are based around the Virtex-II Pro FPGA based XUPV2P development board. This board features a Xilinx Virtex-II Pro XC2VP30
Fig. 4.6: Flow of a smart packet and corresponding acknowledgement through the router with independent interfaces

Fig. 4.7: Flow of a smart packet and corresponding acknowledgement through the router with centrally managed next-hop tables
FPGA device and a single 10/100 Mbps Ethernet port. The design further makes use of two of its three SATA interfaces. However, it is limited in several ways by the available hardware:

- As the development board’s Ethernet port is a 10/100 Mbps port, each interface is ultimately limited to this data-rate. Commercial line-card implementations typically have eight ports. Internally however the implementation is capable of greater data rates. Both the input and output stages run at 75 MHz and are capable of forwarding 32 bits per clock cycle. This is equivalent to 2.4 Gbps of forwarding capacity.

- Rather than the space-division switching fabric used to link the interfaces of a modern router, this implementation makes use of the SATA interfaces to create a ring topology. This limits the number of interfaces the router can support, but provides sufficient bandwidth considering the 100 Mbps per interface restriction. As with the input and output processing stages, the SATA interfaces are run at 75 MHz, with a bit-width of 32 bits.

Based on the findings of Sections 3.5 and 3.6, the ∞-SR decision making algorithm has been used in this implementation (as opposed to the approximated RNN used in the single routing processor design from Section 4.4).

The build process of the design allows the choice between creating an independent interface, a dependent interface, and a central processor to be made. The central processor is implemented in logic. It would be expected that a non-proof-of-concept implementation would make use of a general purpose processor in addition to dedicated logic for performing updates at speed. This would also allow easier expansion to include additional functionalities, such as router administration and slow-path routing operations. The necessity of these additional functionalities,
combined with the drop in QoS performance resulting from independent interfaces, means that central processor architecture is the more viable option.

*Internal control messages*

Before building the design it is possible to select between building a line-card for a centralised architecture or an architecture with independent interfaces. The centralised architecture requires additional internal control messages in order to transfer information between the central processor, which maintains the mailboxes and routing tables. These messages are furthermore required to propagate the routing table updates back to the line-cards’ own routing table mirrors.

In Figure 4.7 one can see how the control messages are used in a centralised processor. A QoS table update message is sent by the input processing block to the central processor when an acknowledgement packet is received. When the QoS table update results in a change in the routing table, a routing table update control message is broadcast to all the interface cards and handled by each line-card’s output processing block. The details of these are given in the following sections.

*Input processing implementation*

The XUPV2P development board has a dedicated Ethernet PHY device. This is connected directly to the FPGA device. Xilinx provide a pure logic 10/100/1000 MAC implementation which can be configured to work on the device being used in 10/100 mode. In order to interface with the PHY for 100 Mbps operation, the MAC component must be clocked at 25 MHz. The goal with this implementation is to explore and demonstrate a high speed CPN routing architectures, so the routing logic must be clocked at a higher rate than this. Furthermore, the MAC core
4. FPGA Based Hardware CPN Router Implementations

Fig. 4.8: Connections between input processing block and other router elements

Fig. 4.9: State machine for the MAC-side of the input processing block. Some state transition labels have been omitted or simplified for clarity.
outputs bytes of data with each clock-cycle, whereas this routing logic operates on 32-bit words of data. To shift between clock domains and data sizes the MAC block contains a non-symmetric FIFO implemented using the FPGA’s dual-port blockRAM. A similar FIFO arrangement exists between the input processing block and the switching fabric.

Figure 4.8 shows the connections between the “client side” of the MAC FIFO and the input processing block, and between the input processing block and the various other connected blocks of the design. Reading and writing to the FIFOs and interactions with other router elements are controlled using state machines. The input processing block uses the two state machines shown in Figures 4.9 and 4.10 for its “receive” and “send” sides respectively.

The MAC-side state machine will sit in its “ready” state until the MAC block indicates that a packet is available by setting the “start of frame” signal. The
Ethernet header is received, and based on its content, the state machine moves to either the “ARP frame receive” or “CPN head receive” state. While in the “CPN head receive” state, the CPN header is parsed, and based on its contents the subsequent “CPN route receive”, “CPN CM receive”, and “CPN frame receive” states are traversed. Once the full packet has been received the state machine returns to the “waiting” state. When the packet is an ARP packet and the state machine enters the “ARP frame receive” state the ARP packet is parsed before returning to the “ready” state.

The MAC-side state machine is largely mirrored on the switch-side state machine which is shown in Figure 4.10. As the CPN packet headers are read from the MAC-side FIFO and parsed, the switch-side state machine will sit in the “CPN header receive” state. During this state, once the CPN packet type, QoS class, and destination have been read, and if the packet is a smart packet, the next hop is read from the next hop memory by setting the next hop memory enable signal. The next-hop address is then available in time for when the header and route are updated.

Once the header is fully received the state machine will move to the “CPN header adjust” state, where the header is adjusted, and from here the state machine moves through a series of states, appropriately writing the Ethernet header, CPN packet header, CPN route, cognitive map, and finally data to the switch-side FIFO. Adjusting the CPN header involves moving the route pointer (either forwards or backwards) when forwarding the packet. If the router is the destination, the route and CMs are truncated, and the source and destination swapped. During this state the next-hop’s MAC address is also looked-up by setting the “ARP enable” signal, and inputting the next-hop’s CPN address.

Once a complete frame has been written to the switch-side FIFO, it expects
the “good frame” signal to be set low. The “good frame” signal can only be set every 64 clock cycles, so an intermediary state (“good frame wait”) is included for the cases where a short or partial packet arrives.

After setting the good frame signal, the state machine returns either to the ‘waiting” state, or, if the line-card is for use in a centralised architecture and the CPN packet is an acknowledgement, the “send control message” state. During this state a short frame containing an Ethernet header (which the ARP/switching processor block expects, as will be seen in Section 4.5.3) with an identifying QoS update message type, the flow’s destination and CPN QoS type, the acknowledgement’s previous hop (i.e. the flows next hop), and latency read from the packets cognitive map is written to the switch-side FIFO. From here the state-machine returns to the “good frame wait” state (remember that each good frame signal must be separated by 64 clock cycles, which is longer than a control message takes to send).

When ARP packets are received on the interface they are forwarded through the input processing block unaltered for processing in the ARP processing block.

**Output processing implementation**

The basic operation of the output processing block is similar to that of the input processing block. Packets are moving in the other direction between the switching block and the MAC, but other than this the design remains governed by two state machines (which can be seen in Figures 4.12 and 4.13) controlling the reading of the packets from the switch-side FIFO, and writing to the MAC-side FIFO.

The output processing block has two unique functionalities. The first is that it updates the next-hop/mailbox memory block, which the input processing block reads from when forwarding smart packets. This occurs when a smart or dumb
Fig. 4.11: Connections between output processing block and other router elements

Fig. 4.12: State machine for the switch-side of the output processing block. Some state transition labels have been omitted or simplified for clarity.
Fig. 4.13: State machine for the MAC-side of the output processing block. Some state transition labels have been omitted or simplified for clarity.

acknowledgement packet, which did not originate at the same router, is being written to the MAC-side FIFO. The second unique functionality is that it periodically sends ARP requests in order to advertise its presence and learn its neighbours.

When an ARP reply is received from the switch-side FIFO, it is forwarded straight through to the MAC-side FIFO.

Switching and ARP packet processing implementation

As previously discussed, the bandwidth limits imposed by the XUPV2P development board’s single 10/100 Mbps Ethernet port mean it is not viable to implement the sort of crossbar switching fabric commonly seen in today’s high speed Internet routers. The development board has three SATA ports, two of which are in host configuration, and one in target configuration. By using one target and one host
Fig. 4.14: Connections between switching/ARP processing block and other router elements

Fig. 4.15: Receiving state machine for the switching/ARP processing block. Some state transition labels have been omitted or simplified for clarity.
SATA port any number of boards can be connected in a ring topology. Again, as the ports can forward data at rates greatly faster than the 10/100 Mbps Ethernet port allows a single board to receive data, this configuration is suitable for this proof-of-concept implementation.

Xilinx have published a link-layer protocol for serial data transmission called Aurora [64]. As with the MAC logic block, they provide a logic implementation of this specification which can be used to interface the FPGA device with the SATA ports.

The switching block has a number of functions in the router. Its main function is to control the flow of packets out of and into the input and output processing blocks, between the processing blocks and the ring switch, and around the ring switch. Additionally, it controls the lists of neighbours in the next-hop memory and an ARP lookup table, which is accessed by the input processing block when filling the MAC address of the next hop in the Ethernet header. To fulfil this function, ARP requests or replies received by the input processing block are processed by
the switching block, which creates ARP reply packet in response to ARP requests, and rather than any reply being sent directly to the output processing block, it is first sent round the switching ring. Each line-card on the ring then sees the packet, and can use its contents to add the neighbour to their ARP lookup table and next hop memory blocks. In a more conventional crossbar based switching fabric, this information would be broadcast to all line-cards.

Figure 4.14 shows the connections of the switching block. Data can take one of three paths through this block, in one case simultaneously:

1. from the input processing FIFO to the Aurora transmit FIFO,
2. from the Aurora receive FIFO to the output processing FIFO, or
3. from the Aurora receive FIFO to the Aurora transmit FIFO.

This is controlled by three state machines, shown in Figures 4.15 and 4.16.

The “receiving” state machine from Figure 4.15 is responsible for controlling the reading of packets from both the input processing block FIFO and the Aurora receive FIFO. When a packet is available from the input processing block, the receiving state machine moves to the “input src/dst/type” state, where it is determined whether the packet is an ARP packet. Depending on this the state machine moves to either the “input store ARP” or “input store other” state as the packet continues to be read. This distinction is important as the state machine which controls writing to the Aurora transmit FIFO (Figure 4.16b), has different behaviour for ARP than for other packet types. Based on the packet’s ARP request bit, it first determines if the packet is an ARP request from a neighbouring router attempting to learn the CPN address of this router. If so, it creates a reply packet which is written to the Aurora transmit FIFO. Otherwise the received ARP reply is written directly to the transmission FIFO. In both cases the receiving
state machine writes the MAC and CPN addresses to the ARP lookup table, and updates the next-hop memory’s neighbour list by setting \texttt{arp\_en} and \texttt{nh\_addr\_en}. Other packet types are copied directly to the Aurora transmit FIFO.

Packets incoming from the receive port of the ring switch are either destined for this node (which causes the \textit{local} signal to go high), broadcasts, or destined for another node. The receiving state machine has a similar series of states for when packets arrive from the Aurora receive FIFO as when they arrive from the input processing block FIFO. Regular CPN packets and ARP packets destined for the local line-card are handled in state “aurora store other”, whereas ARP packets which must be sent further around the ring are handled in state “aurora store ARP”. Again the ARP specific state exists so that the data from the ARP packet can be parsed and written to the ARP lookup table and next-hop memory blocks. The two output state machines respond to the received packet by copying it to either the output processing block FIFO, the Aurora transmit FIFO, or in the case of broadcast messages (i.e. control messages from the central processor) to both simultaneously.

In some cases this design is less efficient than it could be. By separating the receive state machine into two distinct state machines, it would be possible to design the switching block so that reading packets from the input processing block FIFO does not block reading packets from the Aurora receive FIFO and vice-versa. This design decision was again made as the rate at which packets can arrive from any of the input interfaces is considerably lower than the ring-switch’s bandwidth.

\textit{Next-hop memory implementation}

This implementation of the router does not have a large memory for storing the next-hops and mailboxes. It instead opts for a small logic based approach, which
simplifies the implementation, but would not be sufficient for a router routing
many flows to many destinations. The implementation of next-hop routing tables
and mailboxes in CPN routers for large networks is discussed in Section 4.6.

Device Usage and Speed

Table 4.2 summaries the device usage of the two 3rd generation equivalent architectures. For reference the 1st generation equivalent is also shown. The significant difference between the number of LUTs used by the independent and non-independent architectures is accounted for by the implementation of the mailboxes, which the non-independent approach lack, in logic. Both architectures also implement the next-hop tables in logic. Moving these off device to DRAM modules which would result in further device area savings.

The designs both synthesis to meet the 75 MHz clock constraint of the XUPV2P development board.

4.6 Next-hop and Mailbox Memories for Large CPN Networks

It is currently not viable for CPN to scale beyond single networks, because a
method for addressing large networks running CPN has not been defined. This
restricts the ability to develop a means of maintaining memories which results
in non-traffic dependent next-hop and mailbox look-up times. Increased network
sizes in CPN would also result in even larger route set-up times, as smart packets

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Independent</th>
<th>Non-independent</th>
<th>Single routing engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>23,899 (87%)</td>
<td>20,655 (75%)</td>
<td>17,572 (26%)</td>
</tr>
<tr>
<td>Slices</td>
<td>13,694 (99%)</td>
<td>13,694 (99%)</td>
<td>14,531 (43%)</td>
</tr>
</tbody>
</table>

Tab. 4.2: Device usage summary
are initially routed at random.

In IP networks, Classless Inter-Domain Routing (CIDR) [65], the Border Gateway Protocol (BGP) [66], and Longest Prefix Matching (LPM) [67] all combine to move packets within and between autonomous systems. The current implementation of CPN uses a 32-bit addressing scheme, where, as with IPv4 (without NAT), each device has a globally unique address. Unique addresses enable homogeneous end-to-end routing. The routing tables of IP routers do not however contain a unique entry for each routable location. As each autonomous system, or indeed group of autonomous systems, shares a common address prefix, the routing tables contain address prefixes. When forwarding an IP packet, the router will select the entry from its routing table with the longest prefix which matches the destination address. This is called longest prefix matching.

Current implementations of CPN, including the two routers described in this chapter, do have a unique entry for each destination and QoS class in their next-hop lookup tables and mailboxes. This is acceptable for the limited network sizes which have been used in the lab, especially as entries are only added when first used, but a network consisting of millions, or even billions of devices would make this infeasible due to the capacity required to store so many entries. By reducing the required capacity the use of content addressable memories (CAMs) [68] or redundant memories [60] for single clock lookup times becomes viable.

This section proposes a method for allowing CPN to scale to large networks by concatenating addresses in such a way to allow LPM in its lookup tables.

4.6.1 LPM in CPN

To effectively use LPM in CPN, the addressing of devices would need to be strictly hierarchical, with sub-networks containing addresses with longer prefixes than
Fig. 4.17: Example of sub-networks for LPM in CPN

those of the higher level networks, yet still matching those higher level networks’ prefixes.

The source routed nature of CPN would remain, with sources first sending smart packets to find routes to the destination, and then storing the routes of the acknowledgements for use by dumb packets. Smart packets would be routed based on the entry with longest prefix matching the destination found in the next-hop tables. Therefore if the smart packet is heading for a destination within the same sub-network as the current router, it is treated as it would be in the current implementations of CPN. If however, the destination is within another sub-network the smart packet is routed “towards” that sub-network. This means that all packets destined for a sub-network with a particular prefix share routes and memory resources within the current sub-network, and that the QoS data upon which routing decisions are based becomes more accurate towards the destination.

In order to determine the prefixes to store in the next-hop lookup tables, it would be simple to include the prefix length of the destination in the acknowledgement packet and use it to mask the destination’s address and obtain a prefix. This would however miss the opportunity to group sub-networks which share a
shorter prefix. For example, in Figure 4.17 the nodes in the X.Y.Z.0 sub-network may group all destinations in the A.A.0.0 sub-network, whereas the nodes in the U.V.W.0 sub-network would differentiate between A.A.1.0 and A.A.2.0, as that sub-network has multiple routes into the A.A.0.0 sub-network.

The prefixes within the next-hop lookup tables are determined by the routers when forwarding acknowledgement packets. These are, as previously, sent when a smart or dumb packet reaches its destination. At each hop the router will examine the acknowledgement packet’s route (recall the full route is stored in the header) between itself and the destination, by scanning through it in reverse order starting at the destination. The router will determine the portion of each hop’s address shared with the address of that hop’s preceding hop, until there is no shared part. The router will then either use that prefix, or if there is a longer matching prefix already in use in the table, the longer prefix will be used. In order to prevent too long prefixes (for example A.A.1.255 being grouped with A.A.1.254, with a prefix length of 31), the destination’s prefix length is used as a maximum. Algorithm 2 expresses this formally.

The network of sub-networks shown in Figure 4.17 can help illustrate how LPM could be used in CPN. Say the network is not initialised, that is the next-hop routing tables and mailboxes are empty. A smart packet has travelled from node X.Y.Z.2 to A.A.2.1 via nodes A.A.1.2 and A.A.2.2. The smart acknowledgement packet will therefore traverse the reverse of this route from the A.A.2.0 sub-network, through the A.A.1.0 and U.V.W.0 sub-networks, and finally through the X.Y.Z.0 sub-network. Table 4.3 shows the selected prefixes at each router the example acknowledgement packet passes. Now imagine a second smart packet travels within the same network between the same source and destination, but this time travels via nodes U.V.W.0 and A.A.2.2, directly into the A.A.2.0 sub-
Algorithm 2 Determining prefixes in CPN

1: \{ACK packet with route length \( L \) arrives at node in route position \( N \}\}
2: \( \text{prefix} \leftarrow \text{route}[L] \)
3: \( \text{prefixLength} \leftarrow \text{length}(\text{route}[L]) \)
4: \( h \leftarrow L - 1 \)
5: \( \text{while } h \neq \text{N do} \)
6: \( b \leftarrow 0 \)
7: \{Compare the bits of the prefix with the node in route position \( h \}\}
8: \( \text{while } \text{prefix}[b] = \text{route}[h][b] \text{ AND } b < \text{prefixLength do} \)
9: \( b \leftarrow b + 1 \)
10: \( \text{end while} \)
11: \( \text{if } b > 8 \text{ then} \)
12: \( \text{prefixLength} \leftarrow b \)
13: \( \text{else} \)
14: \( \text{break} \)
15: \( \text{end if} \)
16: \( \text{end while} \)

<table>
<thead>
<tr>
<th>Hop #</th>
<th>Route</th>
<th>Prefix</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>A.A.2.1</td>
<td>-</td>
<td>destination node</td>
</tr>
<tr>
<td>4</td>
<td>A.A.2.2</td>
<td>A.A.2.1</td>
<td>within same sub-network</td>
</tr>
<tr>
<td>3</td>
<td>A.A.1.2</td>
<td>A.A.2.0</td>
<td>in neighbouring sub-network</td>
</tr>
<tr>
<td>2</td>
<td>U.V.W.1</td>
<td>A.A.0.0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X.Y.Z.1</td>
<td>A.A.0.0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X.Y.Z.2</td>
<td>A.A.0.0</td>
<td>source node</td>
</tr>
</tbody>
</table>

Tab. 4.3: Prefixes for example route in network from Figure 4.17

network. Table 4.4 shows the prefixes which would be used in this case. Note that the longer prefix will now be used for all packets destined to a node in the A.A.2.0 sub-network. This reflects the discovery of the new, more direct route to this sub-network. During the discovery stage of the network running CPN (which is of course ongoing), some garbage collection algorithm would also need to run on the memories in order to remove short prefixes which have become redundant.

Note that this method of extracting the prefixes would only be effective if the nodes are addressed strictly hierarchically. That is, a packet must be routable
between any two nodes with matching prefixes without passing a node with a non-matching prefix. For example, any routers with the prefix A.A.3.0, would need to be within the A.A.0.0 network in the example from Figure 4.17, and connected to either the A.A.1.0 or A.A.2.0 sub-networks.

### 4.7 Conclusions

Two FPGA based proof-of-concept hardware implementations of dedicated CPN routers have been presented. The second design demonstrates that the traditional architectural approaches used in high speed IP routers are applicable to CPN routers. This is important because routers implementing CPN functionality would likely also require the ability to route IP traffic, as islands of QoS are created in the greater Internet. The simpler “independent interfaces” architecture was experimentally shown to negatively impact QoS. Finally a LPM based approach to CPN addressing was proposed to allow CPN to grow beyond the islands of QoS.
5. ADAPTIVE ROUTING IN WIRELESS SENSOR NETWORKS

5.1 Introduction

One application with the potential to take advantage of the QoS benefits offered by the simplified CPN is the wireless sensor network (WSN). WSNs may consist of many low-cost battery powered devices, called motes [69], each of which may have one or more on-board sensors. Motes lack powerful processors, but as they can communicate wirelessly, costs can be lower than for traditional sensing and monitoring infrastructures which may require substantial deployment investments. Additionally, as the unit cost is low, it is possible to distribute large numbers of devices over large areas by, for example, dropping them from low flying aircraft. This means that conditions in potentially difficult to reach or treacherous areas can be monitored. These areas include mountainous regions, deserts, and battlefields. WSN motes must therefore be capable of autonomously establishing and maintaining ad-hoc networks, which will need to be multi-hop if the devices are spread over a large geographical area.

The constraints which the power limitations of motes place on the protocols and algorithms which can be run in WSNs introduce new QoS requirements, such as network lifetime. Indeed, the findings of Chapter 3 suggest that CPN style smart routing, making use of a low-complexity decision making algorithm, such as ∞-Sensible Routing, may be a viable choice for QoS routing in WSNs.

This chapter investigates the viability and desirability of adaptive QoS routing
in WSNs, starting with the use of smart routing in a power aware routing protocol. The use of the Random Re-Routing protocol to load balance traffic flows over available network resources is then investigated.

5.1.1 Smart Routing in Wireless Sensor Networks

The CPN protocol introduced smart routing in order to autonomously adapt to changing network traffic and topology conditions in wired networks, while maintaining QoS [44]. In battery powered wireless networks, and especially WSNs, the QoS goals differ from those of traditional wired networks. While routing latency is still of interest, the first priority may be to ensure long network lifetime. The principles of smart routing were applied to ad-hoc wireless networks in [70], where the protocol attempted to maintain routing reliability by using absolute energy levels at nodes as a QoS metric. This chapter builds upon the previous generations of the CPN routing protocol to produce a power aware ad-hoc protocol suitable for implementation in low complexity, low power devices, such as WSN motes. In Section 5.4, this protocol, henceforth referred to as tinyCPN, is investigated and compared with other protocols presented in the literature.

5.1.2 Communication Power Use

Shortest hop routing is in common use in WSN routing protocols, but the shortest route does not necessarily lead to the lowest power use [71] or the most reliable communications path [72]. This stems from the fact that the transmission power required for reliable communication between devices is proportional to a power of the separation between the devices, with powers of 2 to 4 frequently cited in the literature [73, 74]. Depending on the power use for routing processing operations and for packet reception, a multi-hop route with more hops, but more closely
spaced devices, may require less power than an equivalent route with fewer hops. Minimising communications power use has a number of benefits:

- First and foremost, lower power use results in greater lifetime for devices using battery power. In a “set-and-forget” system, designed to monitor an environment or phenomenon over a long period of time [75, 40], system lifetime is one of the most important features; increasing the lifetime increases the fundamental effectiveness of the system.

- Secondly, less interference between communicating devices means that network wide message transfer reliability is increased due to less contention in the medium, and therefore fewer collisions and retransmissions. The resulting reduced number of retransmissions lead to further energy savings and overall greater transfer rates.

- Furthermore, less transmission power results in a smaller transmission radius, potentially meaning that fewer motes for which the message is not intended expend energy receiving the message.

These benefits are however dependent on the devices’ radio’s ability to scale their transmission power appropriately. Using the telosb mote as a case study, this is investigated in Section 5.3.

5.1.3 Per-Task Packet Prioritisation

For reasons of practicality and cost, a WSN may be deployed for multiple monitoring and data-gathering tasks. These tasks would be unlikely to have the same requirements and priority with respect to the timeliness of data delivery or with respect to their QoS.
Say for example that a large wildlife refuge, home to a number of roaming animals, is covered by a WSN capable of monitoring both vegetation growth and object motion. If the roaming animals are additionally individually and identifiably tagged, the WSN will be able to track pack movements. The WSN may therefore have two standard priority monitoring tasks; monitoring vegetation growth, and tracking pack movements. Let us say that some of the animals being tracked are commonly targeted by poachers. The WSN could therefore also track untagged movements, with the assumption that these could be the movements of poachers which need to be challenged. The tracking information would be highly time-sensitive and therefore should be treated as high priority.

A network running CPN, which monitors and maintains QoS for multiple classes of traffic, can optimise the routing of the individual classes. However each class the network supports requires a proportional increase in memory to maintain the QoS metrics. This may not be an option with simple WSN devices, which have limited memory, a proportion of which would be dedicated to their sensing tasks. Differential prioritisation of packets can however be made possible by a protocol operating in addition to the WSN’s primary routing protocol. Section 5.5 extends the Random Re-Routing (RRR) WSN routing protocol, which is designed to operate over a geographic routing protocol, and aims to provide differential prioritisation of packets by routing them according to a priority-specific policy.

5.2 Routing in Ad-hoc and Wireless Sensor Networks

Ad-hoc wireless network routing protocols can be categorised in a number of ways. Two broad categories are proactive and reactive [76]. Proactive routing protocols maintain routing information for all destinations regardless of the current rout-
Adaptive Routing in Wireless Sensor Networks

5. Adaptive Routing in Wireless Sensor Networks

Ining operations. A reactive routing protocol will determine routes on demand. Protocols may also be classified according to which source-destination pairs are possible. A general purpose routing protocol will need to be able to route packets from any node in the network to any other. However, the nodes of a data gathering WSN may only need to communicate to some common destination. Furthermore, for WSNs there is a class of protocols which treat the network as a distributed database from which data can be accessed.

The Optimised Link State Routing Protocol (OLSR) [77] is used in Mobile Ad-hoc Networks (MANETs) when a many-to-many routing topology is required. This is an example of a proactive routing protocol. Routing table updates are propagated through the network using a flooding mechanism which is optimised to reduce the total number of advertisements. A node running the OLSR protocol has knowledge of its immediate neighbours (one-hop neighbours) and of its neighbours’ neighbours (two-hop neighbours). Routing table updates are only sent to a set of one-hop neighbours referred to as the multipoint relays (MPR). These are chosen so that each of the node’s two-hop neighbours have at least one of the node’s MPRs as a one-hop neighbour. It can then be guaranteed that each of the node’s two-hop neighbours will have a MPR amongst the node’s one-hop neighbours, and that any routing table updates sent by a two-hop neighbour will be received by one of the node’s one-hop neighbours.

An example of a reactive routing protocol is the Ad-hoc On-demand Distance Vector (AODV) protocol [78]. When a packet must be sent to some destination in the network, the source floods the network with a route request message. When this message reaches a node which knows a route to the desired destination, it responds with a route reply message which returns to the source. This route can then be used to forward packets to the destination.
These protocols produce routes which minimise the number of hops between source and destination. The Minimum Cost Forwarding Algorithm (MCFA) [79] can minimise some cost function to a common destination or set of destinations. It is a proactive routing protocol. Each node maintains a list of neighbours, and a list of one-hop costs associated with each. The routing tree is constructed by nodes broadcasting messages containing a route-cost, which consists of the route-cost advertised by their “parent” summed with the one-hop cost to their parent. A node selects its parent, the neighbour to which all packets are forwarded on their route to the destination, from amongst its neighbours by choosing the neighbour with the lowest advertised cost. The common destinations all advertise a cost of zero. In order to reduce the number of advertisement messages, a node will delay its own advertisement when selecting a new parent by a time proportional to the route-cost of that node. Advertisements from lower cost neighbours therefore have time to arrive before the node’s own advertisement is made. The MCFA will function for any sub-additive cost function, and is therefore suitable for minimising cumulative transmission power along routes.

There are also a number of “query” based protocols. These include the SPIN family of protocols [80], which distribute location tagged measurements to all nodes in the network so that they can be read from any location, and Directed Diffusion [81], a protocol where a data-sink broadcasts “interests” which are broadcast through the network. Nodes which receive and can satisfy such interests, begin a process of building a routing tree to the data-sink. This tree is then used to forward data to the sink. When multiple sources can satisfy the interest, and the routing tree branches suitably, data is aggregated en route.

QoS in WSNs can be approached from the perspective of the network or the application [82]. The application may be interested in deployment specific issues,
such as sensor coverage or sensor measurement accuracy, but not necessarily factors under direct control of the routing protocol. The Sequential Assignment Routing (SAR) [83] protocol is an example of a routing protocol which uses energy levels at nodes as a QoS metric when routing towards a common sink node. Each node knows several paths to the sink, each with an associated additive QoS cost metric and energy level. Packets have credits with which they can “buy” the use of a route with low latency, but perhaps depleted energy. Important packets can be given more credits so they can buy a greater QoS.

The protocol presented in [84] considers the use of video and imaging sensors in WSNs, and therefore proposes the need for QoS suitable for real-time multimedia applications. This protocol differs from those previously presented as it clusters nodes, and routing between clusters is performed by the clusters’ gateway nodes. By providing separate queues for real-time and non-real-time packets, and scheduling between them based on the path delay, guarantees are provided for the real-time data.

When forwarding a packet, the geographical SPEED routing protocol [85] selects between its neighbours in a way which attempts to create a balanced forwarding “speed” across the network. The speed is calculated by dividing how much closer the neighbour is to the destination by the forwarding delay. Balancing the speed means that the latency between a source and destination is proportional to the distance between source and destination. Neighbours use beacons to feed back to the forwarding node an estimate of the delay. The neighbours can also inform the node to stop forward packets to themselves if they receive a sudden increase in traffic. The protocol from [86] builds on the ideas of the original SPEED protocol, but includes support for differentiated QoS. This is done by placing packets in corresponding queues, and routing them with different speed thresholds. This
differentiation must be supported by the MAC layer.

CPN is a reactive protocol which has been used in wired networks to provide QoS by minimising the number of hops between source and destination, the round trip delay, the link-loss, and combinations thereof. CPN is also suitable for minimising cumulative transmission power along routes, and the next section discusses a practical method for doing so.

### 5.3 Signal Strength and Transmission Power

How does one know the power required for communication with each neighbour in a localised power aware routing protocol? In simulation, the power requirements can be modelled, typically as proportional to some power of the separation between devices. More complex models are used in more complex simulators. The ns2 network simulator, discussed in more detail in Section 5.4.2, considers further factors in its signal propagation models, such as ground reflections. A real radio will however not know the exact distance separating its aerial from that of the receiving radio, and furthermore may not allow for fine enough distinctions between transmission power levels to treat them as continuous. A real example of the sort of radio used in WSNs is the IEEE 802.15.4 compliant Texas Instruments CC2420 [87] chip used in the commercial off-the-shelf telosb WSN mote [69]. This is used as a practical example here.

The tinyOS [88] sensor mote operating system reports the RSSI (received signal strength indicator), and the LQI (link quality indicator) to the routing protocol for every received message. The telosb mote’s radio allows the output power used to transmit a message to be varied over 8 discrete levels, shown in Table 5.1.

In order to correlate the minimum required power level with the RSSI, a simple
experiment has been conducted in which two types of broadcast message are sent from a source mote to a receiving mote:

1. The first message type is sent at the constant maximum transmission power level. The receiving mote stores the RSSI and sends it in a reply to the source. The source then records the RSSI for the reply and stores the minimum of the two values.

2. For the second packet type, the output power of the source is cycled through the 8 possible levels. The packets contain a field reporting the power level, and when the receiving mote receives a packet it will reply at that same power level. For each power level the number of packets sent and replies received by the source are recorded.

One can then determine the power level at which reliable communication between two motes occurs from the second broadcast packet type, and the corresponding RSSI (at the maximum power) from the first broadcast packet type. The results of this experiment, shown in Figure 5.1, were gathered by moving two motes around within an office space and recording the mean over 500 constant power broadcasts and 4000 variable power broadcasts for each set of locations. It was found that the

<table>
<thead>
<tr>
<th>Level</th>
<th>Current consumption (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>17.4</td>
</tr>
<tr>
<td>6</td>
<td>16.5</td>
</tr>
<tr>
<td>5</td>
<td>15.2</td>
</tr>
<tr>
<td>4</td>
<td>13.9</td>
</tr>
<tr>
<td>3</td>
<td>12.5</td>
</tr>
<tr>
<td>2</td>
<td>11.2</td>
</tr>
<tr>
<td>1</td>
<td>9.9</td>
</tr>
<tr>
<td>0</td>
<td>8.5</td>
</tr>
</tbody>
</table>

Tab. 5.1: Current consumption at discrete power levels by CC2420 radio
LQI gave little useful correlation. Reliable RSSI bounds for each power level can be used in the routing algorithm. As it was observed that even at the maximum power level, with closely placed motes there are significant losses, for this experiment reliable has been defined as less than 5% message loss. In practice, MAC level acknowledgements allow lost packets to be retransmitted on a per-hop basis.

Figure 5.1 shows the correlation between the power consumption at the distinct power levels required for reliable transmission. It is evident that the minimum consumption is only half that of the maximum. It is also evident that the minimum power level is sufficient up to a certain signal strength, after which an approximately linear relationship exists between the required current consumption and received signal strength.

These results are used in the simulations and experimental evaluation of the tinyCPN routing protocol in Section 5.4. The idea of using the minimum communications power to determine whether data should be clustered is also explored in Appendix B.
5.4 Implementing Smart Routing for Wireless Sensor Networks

This section examines the use of smart routing in WSNs through simulation of large networks and implementation on a WSN testbed consisting of 34 motes.

When implementing smart routing for WSNs consisting of many motes there are a number of practical challenges to be addressed compared to the smart routing found in wired CPN. In wired CPN, maximum packet sizes are bounded by the Ethernet MTU (minimum transmission unit) of 1500 bytes. Furthermore, as CPN runs on the same underlying MAC layer protocols as IP, the packet sizes may grow beyond this limit. (An IP packet may have a maximum size of 65,535 bytes, although not all routers support this.) In contrast, the IEEE 802.15.4 radios which motes such as the telosb use, have maximum packet sizes at the physical layer of 128 bytes. This is further reduced once MAC layer overheads are considered. Furthermore, in the testbed environment used in this section, transmission reliability becomes reduced when the packet size is increased much beyond the tinyOS default of 38 bytes. This means that it is certainly not possible to include entire routes in packets as is done in the wired CPN. This is true especially since the routes between nodes at the extremities of a large ad-hoc network can become considerably longer than those which typically arise in networks whose node degrees follow power laws, such as the Internet [89]. Source routing is therefore also not possible. Instead of storing the routes at the source, the intermediate nodes must maintain routing tables.

In tinyCPN, as in the wired CPN, smart packets are forwarded based on the cost or QoS data associated with each neighbour. Each forwarded smart packet is tracked based on its source and destination in order to forward corresponding acknowledgement packets back to the flow’s source. If a smart packet is seen for a
second time at a particular mote, its record is not updated. This ensures that no routing loops occur.

The additional overheads which wired CPN’s dumb packets’ acknowledgements create are also undesirable in WSNs as they result in a doubling of transmissions. Furthermore, a number of publications have discussed the fact that radio communication between WSN motes has a higher than expected loss rate, even under good conditions [72, 90]. This was also observed during the experimentation presented in Section 5.3. In order to achieve reliable transmission of data payloads, the MAC-layer hop-by-hop acknowledgements of the motes’ radios must therefore be relied upon. When no acknowledgement is received, a packet can be resent a number of times. The number of resends can be recorded and used as a multiplier when determining the hop-by-hop power requirements of the route for smart packets’ acknowledgements. The MAC layer hop-by-hop acknowledgements are used exclusively in place of the end-to-end acknowledgements from wired CPN, as the probability of a packet being lost or corrupted in transit is higher. The high probability of the acknowledgement itself being lost in a multi-hop route, and the
end-to-end retransmission which would result, means that hop-by-hop acknowledgements have the power saving benefits shown in Figure 5.2. The figure shows the results from an example simulation of the average energy required for successful dumb packet transmission along an 8 hop route against the per transmission loss probability. End-to-end and hop-by-hop acknowledgements are compared. An infinite number of retransmissions (i.e. the packet is always successfully transmitted) is assumed. Each dumb packet transmission requires 1 energy unit, and each acknowledgement 0.3 energy units (based on the approximate relative packet sizes).

A WSN version of the CPN protocol must therefore rely on the smart acknowledgements alone for route and QoS measurement updates.

5.4.1 Protocol Classification

According to the WSN routing protocol classifications used in [91], tinyCPN is a flat protocol as all motes are given equal routing roles, but incorporates support for QoS goal based routing.

The tinyCPN protocol is intended for applications where the motes communicate with other arbitrary motes in the network, but where the cost of proactively determining all-to-all routing tables cannot be justified.

5.4.2 Network Simulations

Here and in the following sections, a number of simulations of networks are performed which investigate the behaviour of smart packets in WSNs. These have used the ns2 network simulator [92]. The ns2 simulator is an event-driven simulation tool popular in the network research community for accurately simulating large networks. It natively supports wireless networks (the parameters of which
can be extensively adjusted), and can easily be extended to support custom routing protocols.

The network consists of 900 nodes placed in a 30 by 30 grid, spaced out by 31.25 m, with a uniformly distributed random offset in the range \([-7.8125, 7.8125]\) in both the x and y axes. Each node is configured in such a way that they have a transmission range of 40.0 m and a carrier sense range of 70.0 m. Each node therefore has an average of four neighbours. The packet queues have a size of 50 and drop the tail when full.

As part of the set-up, the simulation script informs each node of its coordinates in the plane. These are then shared with neighbours using advertisement packets, which allows nodes to calculate their Euclidean separation. Knowledge of the absolute or relative positioning of nodes in a real network would be enabled through GPS (although this may be too costly), or through triangulation or trilateration of signals from neighbours, or using other more complex techniques beyond the scope of this work [93]. This information is used by geographical routing protocols for forwarding packets (which is in fact used in this chapter in Section 5.5) or simply to associate measurements with locations (as data without context has limited utility). In this case it is used to estimate the communications power between neighbours.

One option for estimating the power required to communicate between two nodes, is to make it proportional to the square of the distance between the nodes, as is used in the radio model in [73]. In order to remain in line with the findings of Section 5.3, the cost of transmission is measured in mA at the same distinct levels from Table 5.1. The use of discrete levels, rather than a continuous curve, matches the reality of the type of radio found in sensor motes. The ranges are based on a radio model, which is also used in the ns2 simulations, which factors
5. Adaptive Routing in Wireless Sensor Networks

<table>
<thead>
<tr>
<th>Level</th>
<th>Distance&lt;sup&gt;2&lt;/sup&gt; range (m&lt;sup&gt;2&lt;/sup&gt;)</th>
<th>Current consumption (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>613 &lt; d&lt;sup&gt;2&lt;/sup&gt;</td>
<td>17.4</td>
</tr>
<tr>
<td>6</td>
<td>487 &lt; d&lt;sup&gt;2&lt;/sup&gt; ≤ 613</td>
<td>16.5</td>
</tr>
<tr>
<td>5</td>
<td>387 &lt; d&lt;sup&gt;2&lt;/sup&gt; ≤ 487</td>
<td>15.2</td>
</tr>
<tr>
<td>4</td>
<td>307 &lt; d&lt;sup&gt;2&lt;/sup&gt; ≤ 387</td>
<td>13.9</td>
</tr>
<tr>
<td>3</td>
<td>217 &lt; d&lt;sup&gt;2&lt;/sup&gt; ≤ 307</td>
<td>12.5</td>
</tr>
<tr>
<td>2</td>
<td>173 &lt; d&lt;sup&gt;2&lt;/sup&gt; ≤ 217</td>
<td>11.2</td>
</tr>
<tr>
<td>1</td>
<td>137 &lt; d&lt;sup&gt;2&lt;/sup&gt; ≤ 173</td>
<td>9.9</td>
</tr>
<tr>
<td>0</td>
<td>0 ≤ d&lt;sup&gt;2&lt;/sup&gt; ≤ 137</td>
<td>8.5</td>
</tr>
</tbody>
</table>

Tab. 5.2: Distance ranges associated with discrete power levels in simulations

ground reflection as well as free-space propagation [74]. It is described by Equation 5.1 where \( P_r \) and \( P_t \) are the received and transmitted power, \( G_r \) and \( G_t \) are the gain of the receiver and transmitter, and \( h_r \) and \( h_t \) the height of the receiver and transmitter respectively. The separation of the devices is given by \( d \).

\[
P_r = \frac{G_t \times G_R \times h_t^2 \times h_r^2 \times P_t}{d^4}
\]  

(5.1)

A reference point of \(-90\) dB at 40.0 m (the transmission range) is used to determine the distance ranges in Table 5.2, which are associated with the discrete transmission costs.

The tinyCPN protocol is a reactive protocol. As the protocol can route between arbitrary sources and destinations, route finding smart packets are only sent when a route is required, as the number of packets would otherwise overwhelm the network. Therefore in these simulations, the concept of a task is employed. A node can run multiple tasks, each of which generates data packets at some specified rate, and smart packets at a proportional (but lower) rate. The dumb data packets are labelled with their corresponding task’s identification number.

When a smart packet reaches its destination, a corresponding acknowledge-
ment packet is sent. Its cost is initialised to the cost of communicating with the destination’s neighbour, and each subsequent node increases the cost by the cost it associates with the next hop. Intermediate nodes on the reverse route can then use the packet’s cumulative cost to update the cost associated with the neighbour from which the packet has been received. The new cost is calculated as:

$$C_n^i = (1 - a) \times C_{n}^{i-1} + a \times C_{p}^{n}$$

where $C_n^i$ is the cost associated with neighbour $n$ and iteration $i$, $C_p^n$ is the cost reported by acknowledgement packet $p$ received from neighbour $n$, and $a$ is a constant. In wired CPN, a value of 0.125 was used for $a$. As there are relatively fewer cost updates in tinyCPN (as a result of data packets not producing acknowledgements), a value of 0.25 is used for $a$ to facilitate changes in the cost being reflected in the measurements more quickly, while still smoothing transient peaks.

The simulation records timestamps and the header contents for every packet transmitted and received in the network. From this, the success of packets reaching the sink, the length of the routes discovered, and the power required can be tracked over the course of the simulations.

As well as tinyCPN, a version of the MCFA has been implemented and designed to use the same cost function. It is used in the following simulations for comparison.

### 5.4.3 Single Source and Destination (One-to-one)

The first set of simulations deal with the one-to-one case, where a single source wishes to communicate with a single destination. In each case considered the network starts with no QoS information corresponding to the source-destination pair.
5. Adaptive Routing in Wireless Sensor Networks

The first smart packets are effectively forwarded at random and must use chance to find the sink node. This random forwarding recalls the diffusion model for packet travel times detailed in [94] and validated through simulation in [95]. The result of the diffusion model is given in Equation 5.3, which gives the expectation of the time (measured in units of hops) required for a packet travelling in an infinite homogeneous network to reach its destination.

\[
E[T] = -2D \frac{1 + \frac{\lambda + r}{\mu} + \frac{\lambda}{r}}{|b| - \sqrt{b^2 + 2(c(\lambda + r))}}
\]  

(5.3)

In this equation, \(D\) is the separation (in hops) of the source and the destination, \(\lambda \Delta t\) is the probability that a packet is lost in a small time interval \([t, t + \Delta t]\), \(r\) is the packet time-out (the number of hops after which the packet is dropped), and \(\mu^{-1}\) is the exponentially distributed time after which the source will retransmit a timed-out packet. The constants \(b\) and \(c\) are the average change in distance between the packet and its destination per hop, and the variance of the average change in distance between the packet and its destination per hop respectively.

It was shown in [94] that for \(b \leq 0\) where the packet either has no knowledge of the relative location of the destination \((b = 0)\), or some knowledge and on average gradually moves closer to the destination \((b < 0)\), there is an optimum time-out for packets which minimises the time it takes to find the destination, and that the packet or a subsequent retransmission will eventually reach the destination. This is important, as for the tinyCPN protocol to be useful, it must be guaranteed that the destination will be found in a finite time.

In tinyCPN the initial packets are routed as if \(b = 0\), but as messages reach the destination, acknowledgement messages fill nodes along their route with QoS data. The routing decisions therefore become more informed, \(b\) becomes negative,
Fig. 5.3: The number of smart packets required for the transmission power used to communicate with the destination to converge (reach 1.1 of the minimum power level), versus the distance of the packet source from the sink. Only one smart packet is active in the network at one time.

and the routes improve.

In the simulations, a constant time-out of 64 hops has been used, after which packets are destroyed.

Simulation Results

Figures 5.3a and 5.3b show the number of smart packets required for the paths taken by the packets to reach 1.1 of the eventual minimum. Figure 5.3a shows this plotted against the distance, whereas Figure 5.3b plots the number of packets against the number of hops on the optimal route discovered by the MCFA. The destination is placed at the centre of the 900 node network. A sample of 36 nodes from regular intervals of the central two thirds of the two diagonals of the square network were selected to generate the results. This means that all the sources had some significant separation between themselves and the nodes at the extremities of the network. The simulation was run 23 times for each of the sources. An ensemble average of the cost of the route discovered by each smart packet was
used to determine the results. Figure 5.5 shows an example of such an ensemble average.

Figure 5.3b shows an increasing trend between the hops separating nodes and the number of smart packets required for convergence. Figure 5.4, which shows the ratio between the cost achieved by the tinyCPN and MCFA protocols, shows that the effectiveness of the tinyCPN protocol decreases with the distance between source and destination.

5.4.4 Routing to a Common Destination (Many-to-one)

In a data gathering role the multiple sources of a WSN send packets to some common destination node referred to as the data sink. A further simulation was performed in which a uniformly distributed set of 38 of the 900 nodes are generating smart packets, such that each node only has a single smart packet active in the network at one time. The same 900 node network from the one-to-one simulations was used.
Fig. 5.5: Ensemble average of route costs for 23 iterations of a series of smart packets for a source destination pair with a separation of 8 hops. The cost achieved by the MCFA is shown for reference.

Simulation Results

Figure 5.6 shows that the “density” of smart packets varies throughout the network, with the density being greatest close to the sink. It is logical that in a static sensor network (where nodes are immobile, are never destroyed, and never expire) with a data rate low enough that congestion does not pose a concern, an optimal route will remain an optimal route. Therefore only a finite number of smart packets are required for each node. This was shown in Figure 5.3.

Figures 5.7, 5.8, and 5.9 show similar trends as the single source-destination pair cases. However each individual source must send fewer smart packets to achieve similar QoS levels. This is expected, due to the smart packet density closer to the sink; nodes benefit from the smart packets originating from other nodes.
5.4.5 Experimental Evaluation

In order to evaluate these protocols experimentally a 34 mote WSN testbed was constructed. The testbed has controllable sensory input. Each mote is contained within a small black box, and four white LEDs of differing brightnesses are directed at the contained mote’s light sensor and can independently be turned on and off via a central computer. Scripting light patterns over the testbed allows various environmental and situational conditions to be simulated. The testbed, further described in Appendix C, was used to run the following experiment.

The experiment compares three protocols:

1. MCFA, without the back-off mechanism intended to reduce the number of advertisements. This would not affect the performance, only the overhead.

2. tinyCPN as described in Section 5.4.

3. tinyCPN without learning. Smart packet routing decisions are always random.

Twenty of the 34 motes are generating a data packet, with the current light level,
5. Adaptive Routing in Wireless Sensor Networks

Fig. 5.7: The number of smart packets required for the transmission power used to communicate with the destination to converge (reach 1.1 of the minimum power level), versus the distance of the message source from the sink. Multiple smart packets (each from a different source) are active in the network at one time.

every two seconds. The smart and “randomised” versions of tinyCPN both generate a smart packet every five seconds.

As was discussed in Section 5.3, the telosb devices used have eight programmable transmission power levels, which are selected for each neighbour according to the measured RSSI of the advertisement packets. The smart packet goal function uses values from the set \{3, 7, 11, 15, 19, 23, 27, 31\} which correspond to the value set in the power level register to set each of the levels. Although these are in fact not proportional to the real power used at each transmission level, for the purposes of these experiments they are used to simulate the use of a greater range of transmission power than the telosb mote’s radio in fact allows.

The experiment was run 10 times for each of the protocols. The results shown in Figure 5.10 are averaged over each of the 10 runs for each of the data packets generated by each of the 20 data packet generating motes. As might be expected, the randomised version of tinyCPN performs poorly as it shows no learning. The other two protocols perform as was seen in the simulations of Section 5.4.4; tiny-
Fig. 5.8: Ratio between the QoS converged upon by the MCFA and tinyCPN protocols, versus the distance of the packet source from the sink, in the many-to-one routing case.

Fig. 5.9: Ensemble average of route costs 50 iterations for a series of smart packets for a source destination pair with a separation of 8 hops. The cost achieved by the MCFA is shown for reference.
5. Adaptive Routing in Wireless Sensor Networks

**Fig. 5.10:** Experimental results comparing tinyCPN with the MCFA

CPN converges towards a level short of the optimal level achieved by the MCFA, and takes longer to do so.

### 5.4.6 Relative Overheads

What becomes apparent from these results is that in tinyCPN the number of smart packet transmissions required to reach a cost similar to that achieved by the MCFA is significantly higher than the number of MCFA advertisement transmissions in most cases.

If the MCFA’s back-off mechanism (which reduces the total number of packet broadcasts by having a node wait a time proportional to its new cost, allowing the advertisements of more neighbours to be heard before advertising itself) is used
5. Adaptive Routing in Wireless Sensor Networks

Fig. 5.11: Smart and acknowledgement packet transmissions prior to convergence, in the single source-destination pair case

Fig. 5.12: Total network wide smart and acknowledgement packet transmissions prior to convergence for each source, in the many-to-one case
in an AODV-like protocol, and functions optimally, the number of transmissions would be equal to the number of nodes in the network, plus the length of the acknowledgement packet’s path. This is because each node will broadcast one advertisement, even after the optimal path to the destination has been found, as the nodes have no way of knowing this optimal path has been found. In the example topology used in the simulations of the previous sections this would mean that there are between approximately 900 and 920 transmissions; the number of nodes plus the length of the longest route.

In Figure 5.11 the number of smart and acknowledgement packet transmissions prior to the point of convergence (from Figure 5.3) are shown plotted against the distance (in metres and hops) from the destination. This assumes there are no loops in the smart packet’s route. For nodes at a distance greater than 3 hops from their destination, the number of transmissions is larger than the optimum for the AODV-like protocol. The tinyCPN protocol’s use of smart and acknowledgement packets is therefore considerably less efficient than a broadcast based protocol such as AODV. From Figure 5.12 it is clear that this is also the case when there are multiple sources, and that tinyCPN is not a suitable protocol for providing differentiable QoS in large wireless sensor networks.

This means that the packet prioritisation protocol which follows should be built on top of the MCFA.

5.5 Packet Prioritisation by Re-Routing

In the classic example application for WSNs [96, 97], monitoring woodland temperatures for indications of fire outbreak may be the primary task, and successful responses may be dependent on the network rapidly transferring the location in-
formation to the decision centre. If the sensor devices are also equipped with sonic or seismic sensors, they may support a secondary task by informing relevant authorities of fallen trees which need to be removed from roads and footpaths. This information is less urgent and has lower demands in terms of timeliness from the network.

Random Re-Routing (RRR) [98] is a high-level WSN protocol which operates on top of geographic packet routing. RRR provides differentiated QoS for high priority (HP) and low priority (LP) traffic in WSNs. If nodes are frequently being blocked (due to heavy traffic loads for example) reducing the transmission rates, and therefore the frequency of collisions, can lead to increased throughput [99]. To do this, the RRR protocol randomly re-routes LP packets to prioritise the optimum route for the HP packets, ensuring their timely delivery.

Packet prioritisation in network routing has historically referred to scheduling packets within or between a router’s queueing buffers [100, 101]. In multihop wireless networks, the problem may refer to the MAC layer operation of scheduling between nodes. For example, in [102] a MAC layer mechanism for prioritising the medium for high priority packets is proposed. This uses an additional channel to broadcast a “busy tone” when a high priority packet is to be sent. Such complex MAC layer prioritisations may however not be an option for WSN devices, and RRR uses the notion of packet prioritisation through spatial re-routing. The SAR protocol [83] also uses multiple routing paths for packets of different priorities, but these paths must be predetermined, which increases the overheads.

A network node running RRR sorts its neighbour list by how much closer each neighbour is to the network sink. The neighbours can then be grouped into three distinct groups:

1. The *best* neighbour: the neighbour which is closest to the sink.
Fig. 5.13: Packet forwarding in RRR - HP packets are sent from the source $S$ to neighbouring Node 3, as this is the best neighbour. LP packets are re-routed through either Node 1 or 4 with equal probability.

2. The *positive* neighbours: the neighbours which are closer to the sink than the current node.

3. The *negative* neighbours: the neighbours which are further away from the sink than the current node.

This is shown in Figure 5.13, where Node $S$ is the source of traffic which has Node $D$ as its destination. Node $S$ has the neighbouring Nodes 1 to 4, of which Node 2 is further away from $D$ than $S$ is itself, and therefore in group 3; Node 3 is closest to $D$ (group 1); and Nodes 1 and 4 are both in group 2.

The priority of traffic in the original RRR described in [98] has a binary low/high state, HP traffic being triggered by the occurrence of unusual events in the region monitored by the WSN (such as a forest fire). A network node which is receiving HP packets at a rate greater than some predetermined threshold enables RRR. When in this state, the node forwards all HP packets to its best neighbour (Node 3 in Figure 5.13) and splits the LP packets evenly between the positive
neighbours (Nodes 1 and 4). This is summarised in Table 5.3.

This work’s first extension to RRR allows greater granularity in the relative priorities assigned to sensor network tasks. Consider $P_t \in [0, 1]$, the priority of traffic generated for a Task $t$, where $P_t = 0.0$ corresponds to LP, and $P_t = 1.0$ to HP traffic of the original RRR. Packets with priority $P_t$ are then forwarded to the best neighbour with a probability of $P_t$, and to each of the remaining $n - 1$ positive neighbours with probability $(1 - P_t)/(n - 1)$, as shown in Equation 5.4 and Table 5.4. No packets are sent “backwards” to any negative neighbours, or to the previous hop.

$$
p(i) = \begin{cases} 
P_t & \text{if } i = 0 \\
\frac{1 - P_t}{n - 1} & \text{otherwise}
\end{cases} \quad (5.4)
$$

It is hoped that greater granularity in priority will translate into an ability to finely vary relative QoS, and this functionality is applied in Chapter 6. Extensive simulations are used to explore RRR’s performance in the following sections.

The second change from the original RRR relates to the way in which re-routing is triggered. In the version of RRR from the literature a node will re-route LP packets if it itself is forwarding HP packets at a rate above some threshold. As this protocol is operating in the wireless context, and the transmissions of neighbours can therefore be overheard, re-routing is triggered for a packet of priority $P$ when the neighbours are forwarding packets of priorities greater than $P$ at a rate above some threshold. This is advantageous, again because of the wireless context for

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>$\ldots$</th>
<th>$n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(i)$</td>
<td>$P_t$</td>
<td>$\frac{1 - P_t}{n - 1}$</td>
<td>$\frac{1 - P_t}{n - 1}$</td>
<td>$\frac{1 - P_t}{n - 1}$</td>
<td></td>
</tr>
</tbody>
</table>

Tab. 5.4: Probabilities assigned to each ordered neighbour for packets of priority $P$ in the revised RRR, where there are $n - 1$ positive neighbours.
the reasons which will be presented in the next sections. The transmissions of the
neighbours of a node will influence that node’s transmission rate. This is discussed
in greater depth in the following sections.

5.5.1 Simulation Environment

To support RRR in simulation a further version of the Minimum Cost Forwarding
Algorithm has been implemented for the ns2 simulator. This version uses the geo-
graphic location of the nodes to find the shortest route between each node and the
network sinks. The decision to use the MCFA was made based on the findings of
Section 5.4.6. Recall that nodes running the MCFA protocol periodically broadcast
advertisement packets which contain the originating node’s “cost” and, in
this geographic version of the protocol, the node’s (x,y)-coordinates. Other nodes
within range receive these advertisements and use them to build a list of neigh-
bours and their associated costs. Nodes determine a neighbour’s cost by summing
the neighbour’s advertised cost with the Euclidean distance between themselves
and the neighbour. From their neighbour list nodes select a “parent”: the neigh-
bour with the smallest cost; and sort their neighbours as “positive” (those with
a cost lower than the node’s own cost) and “negative” (those with a cost higher
than or equal to the node’s own cost). The data-sinks all broadcast a cost of 0,
and subsequently optimum routing information spreads outwards from the sinks
during a set-up phase. Advertisements sent during operation of the network allow
the network to adapt to node mobility and failure.

RRR operates on top of this basic routing protocol. In addition to the cost and
coordinates of each neighbour, nodes store a list of the times and priority of the six
most recently overheard packets each neighbour has sent, which allows the nodes
to dynamically adapt the operation of RRR, as will be shown in Section 5.5.2.
At the application level of the simulations which will follow, traffic is generated by nodes with an assigned RRR priority and packet rate. The interval between packets for a source with rate $r$ is uniform randomly distributed in the range $[r/2, 3r/2]$. The location of the packet sources, as well as the start and stop times, can be scripted when designing the simulation.

The simulated network topology consists of 600 nodes, placed in a grid with a uniform randomly distributed offset, so that they are on average separated by 20 m above, below, left, and right (as will be seen in Figures 5.14 onwards). There are four sink nodes all placed in close proximity, centrally in the lower half of the network’s area. As with the network used for the tinyCPN simulations, each node is configured in such a way that they have a transmission range of 40.0 m and a carrier sense range of 70.0 m. This means that each node has in the order of 12 neighbours, but is influenced by the transmissions of nodes beyond its range.

The simulations model the IEEE 802.11 MAC [103] without the RTS/CTS (request to send/clear to send) mechanism enabled as the MAC protocol used by the motes does not implement RTS/CTS. CSMA (carrier sense multiple access) is used, and if carrier is busy or a collision occurs the back-off time increases exponentially. Collisions occur when the receiving node is within range of an additional transmission and receives that transmission at a level within 10 dB of the first.

The radio signal propagation follows the same ground reflection model used in the previous simulations. In fact, at the ranges under consideration, the implementation uses the Friis free-space model [104] as this is a more accurate model at smaller distances.
5.5.2 Random Re-Routing’s Performance

In addition to the basic variable priority and listening extensions, several more changes have been made to the operation of RRR. These incorporate advantages which the wireless medium of WSNs allows (i.e. overhearing the transmissions of neighbours), while compensating for an incorrect assumption made in the original RRR theory.

Through simulation, it has been observed that in the majority of traffic cases, RRR has an overall negative impact on the latency of packets of all priorities. This occurs because the LP packets’ routes are random, and rather than following routes which avoid the best paths (which are used by the HP packets), the LP packets follow sub-optimal routes which are nevertheless, on average, geographically similar to the “best” route. This can be seen in the visualisations of two representative simulations in Figures 5.14 and 5.15; although the routes taken by the LP (and medium priority, or MP, where $P = 0.5$) packets are affected by the presence of nodes forwarding HP packets, the traffic types remain within carrier sense range of each other.

As these are wireless communications, an increase in the number of transmissions in the area of the geographical route taken by the packets results in a greater contention for the medium, more back-offs, more collisions, and ultimately less overall throughput [105]. This impacts upon the travel times of both the HP and LP packets. This eventuality was considered in the previous work on RRR [106], but it was assumed that the traffic on routes used by HP packets would be reduced. This does not take into account the impact on travel times of an increase in wireless transmissions by the neighbours of the nodes on routes used by HP packets. From Figures 5.14b and 5.15b one can in fact see that once RRR is turned on at the 330 second mark in each simulation, the latencies of all packets increase, which is not
(a) Visualisation of the traffic flows in an example simulation of the unmodified RRR. The locations of the nodes in the 500 m by 500 m area are marked by small crosses. The width of the lines between nodes is proportional to the number of packets forwarded between those nodes during the period of 330 to 345 seconds, during which RRR is enabled.

(b) Measured latencies during the same example simulation.

Fig. 5.14: Example unmodified RRR simulation with two clusters of packet sources, with \( P = 0.0 \) and \( P = 1.0 \). Each source has a rate of 20 pps.
Fig. 5.15: Example unmodified RRR simulation with three clusters of packet sources, with $P = 0.0$, $P = 0.5$, and $P = 1.0$. Each source has a rate of 20 pps.
desirable. To model this behaviour, the channel rather than the nodes themselves is considered, and queueing theory is applied, as follows.

If a single node is modelled as an M/M/1 queue, the arrival rate would be defined by the rate at which packets are being sent to that node. The service rate, however, would be dependent on the contention for the medium - the rate at which neighbouring nodes also wish to send packets. The channel itself is therefore modelled as an M/M/1 queue, where the arrival rate is the sum of the transmission rates of the nodes within carrier sense range, and the service rate is the channel bandwidth (in bytes per second) divided by the average packet size (in bytes). The forwarding time is then the sum of the node’s constant processing time and the waiting time at the node’s output interface. This is admittedly simplistic, as it fails to capture details of the collisions and exponential back-offs, but does model contention in the idealised case where the carrier sense mechanism works perfectly.

If the average transmission rate at node \( n \) is given by \( \lambda_n \), the service rate of the medium is \( \mu \), there are \( N \) nodes total within carrier sense range, and \( \rho \) is a constant representing the time to process a packet, then the expected forwarding time \( E(\tau) \) of a node can be expressed as shown in Equation 5.5.

\[
E(\tau) = \frac{1}{\mu - (\sum_{n=1}^{N} \lambda_n)} + \rho
\]  

(5.5)

From this it is clear that to decrease the average forwarding time and subsequently the average packet travel time, the average packet arrival rate within the carrier sense range of the nodes on the “best” paths must be reduced. The aim of RRR must therefore be altered, rather than to simply “randomly” re-route LP packets, to encourage the LP packets to take greater spatial detours from the routes taken by the HP packets. This would minimise the traffic both on that route and in its
5.6 A New Approach to RRR

Nodes forwarding LP packets in the vicinity of a node $N$ can be sorted into four groups, illustrated with the help of Figures 5.16a and 5.16b:

1. The neighbours of node $N$ (nodes 1, 2, 3, and 4) and node $N$ itself.

2. Nodes which are outside of the transmission range of node $N$, but within its carrier sense range (nodes $A$, $B$, $C$, and $D$).

3. Nodes outside of node $N$’s transmission and carrier sense ranges, but within those of other nodes on the path.

4. Nodes outside of the transmission and carrier sense ranges of all nodes on the path.

Packets forwarded by nodes in groups 1 and 2 impact the transmission rate of a particular node on the path (as Equation 5.5 has shown), while packets forwarded by nodes in group 3 affect the overall packet travel time. The goal of RRR must
therefore be to ensure that the majority of the LP packets, which would have been forwarded by nodes in groups 1 to 3, are re-routed to group 4 nodes for as large a length of the geographical HP route as possible.

Consider the following view of a WSN based on a model from [107]. The sink or destination node is at the centre and is surrounded by other nodes which are placed on approximately concentric rings centred about the destination node, $D$. Packets travelling within the network from source node $S$ therefore have three possible options:

- To move one hop closer to the destination, the packets must move *forward* to a node on the next concentric circle closer to $D$.

- To remain at a constant distance from the destination, the packets must move *sideways* to a node on the *same* concentric circle.

- To move one hop further away from the destination, the packets must move *away* to a node on the next concentric circle further away from $D$.

As re-routed packets in RRR do not move further away from the destination, the third option can be disregarded when modelling the travels of a packet. Figure 5.17a depicts an example re-routed packet’s path, and (in grey) the best route between $S$ and $D$. The example re-routed LP packet travels sideways along the circles a total of three times, and forward six times to reach $D$, travelling a total of nine hops. The HP packet simply travels forward six times. In Figure 5.17b the LP packet also covers nine hops, yet travels sideways in such a way as to spatially avoid the path taken by the HP packet for a large proportion of its route.

To achieve this forwarding behaviour in RRR three mechanisms are used, packet *diversion*, *inertia*, and *herding*. 
5. Adaptive Routing in Wireless Sensor Networks

Fig. 5.17: Ringed network topology model: the nodes of the WSN are arranged along approximate concentric circles centred about the sink. A HP packet (path shown in grey) moves forward with each hop by moving to the next inner circle. A LP packet (path shown in black) has a probability of moving along a circle with each hop it takes, therefore staying at the same distance from the destination $D$.

5.6.1 Diversion

The packet diversion mechanism ensures that LP packets forwarded by group 1 nodes are re-routed away from this group, by redirecting packets away from nodes which forward higher priority packets.

To enable diversion, nodes make use of the data known from listening to the transmissions of their neighbours and monitoring the rate at which they are sending or forwarding packets (by storing the timestamps of the last six overheard packets), as well as the priority of these packets. As RRR is intended to operate over geographical routing protocols, the assumption that nodes know their neighbours’ absolute or relative coordinates, and therefore the angles between themselves and their neighbours on a two dimensional plane can also be made.

If RRR is enabled, nodes average the coordinates of the neighbours which are forwarding HP packets and attempt to forward their LP packets in the direction opposite to that point. LP packets are only sent to neighbours which satisfy three
conditions:

- They are closer to the destination than the current node (as was previously the case).
- They are not sending HP packets above a certain rate.
- They are within an angular range opposite in direction to the mean location of the HP traffic forwarding neighbours.

This is illustrated in Figure 5.18. A neighbour is selected at random from those which satisfy these conditions. If no neighbours satisfy all three conditions, the angular range is increased in steps until one or more do. If no neighbour can satisfy all three conditions, the conditions are relaxed in the order “angle”, “rate”, and finally “direction”.

From the visualisations of two simulations in Figures 5.19a and 5.20a the effect of “diverting” the packets away from the HP forwarding neighbours can be seen. In Figures 5.19b and 5.20b one sees the impact this has on the packet travel times.
Fig. 5.19: Example RRR with deflection simulation with two packet sources
Fig. 5.20: Example RRR with deflection simulation with three packet sources

(a) Location of packet sinks

(b) Mean packet travel time (seconds)

- LP traffic starts
- MP traffic starts; latency of LP traffic increases
- HP traffic starts; latencies of LP & MP traffic increase
- RRR starts; latencies of all traffic decrease slightly
- LP & MP traffic stop; latency of remaining traffic falls
There is some improvement compared to RRR without diversion, but it does not provide sufficient improvement compared to not using RRR.

From visual inspection of Figure 5.19a it is clear that this is due to the fact that while the LP flow is beyond the transmission range of the HP forwarding nodes, it remains within their carrier-sense range. As these transmissions are not received, they can not be used to further re-route the LP packets, but still potentially cause MAC layer back-offs and therefore greater queuing. To address this shortcoming, packets are given *directional inertia*.

### 5.6.2 Inertia

The above diversion mechanism has the effect of sending the LP traffic away from the HP traffic. However, once a LP packet is out of range of the HP traffic in group 1 it will no longer be re-routed, and the packet begins to follow a new best path. This will either have the effect of returning the LP packet back towards the HP traffic (at which point it is re-routed again and the cycle repeats), or as was seen in Figure 5.19a, the LP will remain within carrier-sense range of the HP packet forwarding nodes.

Giving the diverted LP packets “inertia” forces them to continue to be re-routed in the same direction even after they are out of transmission range of the HP traffic, with the aim of moving their paths beyond groups 2 and 3. This increases the geographical disparity between the HP and LP traffic routes beyond the carrier sense range of the devices’ radios. For this implementation, packets are given an integer inertia value which decrements with each hop once out of range of the HP packets. The initial inertia of re-routed packets is scaled with the difference between their priority and the average priority of packets the forwarding node has overheard.
Fig. 5.21: Example RRR with deflection and packet inertia simulation with two packet sources
Fig. 5.22: Example RRR with deflection and packet inertia simulation with three packet sources
The simulation results in Figures 5.21 and 5.22 illustrate the effectiveness of the combination of diversion and directional inertia in re-routing LP traffic away from the paths taken by HP and medium priority (MP) packets. In Figure 5.21a it is seen how when RRR is enabled (at the 330th second mark), LP traffic is routed away from the HP traffic. In this example the LP and HP packets have RRR priorities of 0.0 and 1.0 respectively. The impact this re-routing has on the packet travel times can be seen in Figure 5.21b. The packet latencies drop for the HP traffic and rise for the LP traffic once RRR is enabled after the 330th second. In Figure 5.22a the third set of traffic sources with a medium RRR priority of 0.5 is also routed away from the HP traffic and has some impact on the routes taken by the LP traffic. In Figure 5.22b it is seen that, similarly to the previous case, enabling RRR has a beneficial impact of the travel times on the MP and HP packets.

Figure 5.23 helps illustrate why RRR with the deflection and inertia mechanism has the positive benefits on the route taken by the HP packets compared to the original unmodified RRR. Figure 5.23a shows that enabling the unmodified RRR has little impact on the per-hop forwarding time for the majority of the route,
but has a highly negative effect at the second hop. In Figure 5.23b one sees that enabling the modified RRR reduces the per-hop forwarding time for all nodes but the sink.

5.6.3 Herding

Returning to the poacher targeted wildlife refuge of this chapter’s introduction allows the scenario to be expanded to include greater numbers of LP traffic sources. In the example, the sensor network was supporting multiple data gathering tasks with low priority, including wildlife movement tracking and vegetation growth. The tracking of intrusions into the area of the sensor network by poachers is however of high priority. In this scenario there may therefore be many distributed LP traffic sources generating packets and a greatly smaller number of (perhaps clustered) HP sources.

Figure 5.24 illustrates an example simulation in which there is a single cluster of four HP sources (RRR priority of 1.0), each generating 20 pps (packets per second), and 24 LP sources (RRR priority of 0.0) each generating 4 pps. The topology and same set of four data sinks from the previous sections are used. Figure 5.24a shows a representation of the number of transmissions between particular nodes for LP and HP packets for a 15 second interval of a simulation in which RRR does not employ either the diversion or inertia mechanisms. Figure 5.24b depicts the equivalent for the case where the diversion mechanism is enabled. In Figure 5.24c the ensemble averages of the packet travel times of 50 iterations of 45 seconds of the same simulations are plotted.

In each case the HP traffic is started at the 300th second mark, LP packet sources begin transmitting after 315 seconds, and RRR is enabled after 330 seconds. As was previously observed, RRR without diversion and inertia has no positive
impact on the travel times of the HP packets, and the addition of the diversion mechanism has some small positive impact. The addition of the inertia mechanism has an additional positive impact. However, none of the mechanism can affect LP packets which are not forwarded by nodes in group 1, but whose paths encompass many group 2 and 3 nodes. A third mechanism, packet herding, is introduced in order to address this issue.

Collective animal behaviours such as herding occur where the individuals of a group act in a manner based on some aggregated function of their peers’ behaviours, resulting in an emergent group behaviour. Individual birds or fish may act in such a way to avoid predators [108]. Individual people may base their actions on those of others due to a perception that others have knowledge they lack [109]. In order to influence the routing of LP packets beyond the transmission range of HP forwarding nodes, a herding mechanism can be applied.

In order to divert LP packets, nodes running the modified RRR listen to the transmissions of their neighbours for packets of high priority. By additionally noting the inertia of packets forwarded by neighbours, and calculating their trajectory (based on the forwarding neighbour, and the next intermediary hop, if also a neighbour), an average “direction” of flow of re-routed packets can be determined. If a node must then forward a LP packet which has no inertia, and would not normally be re-routed based on the presence of HP traffic, it can be “herded” in the same direction as the average flow, and given an inertia based on the average overheard inertia. Re-routed LP packets therefore result in the re-routing of other LP packets within and beyond the carrier sense range of a HP forwarding node.

The simulation with multiple LP sources has been repeated with a version of RRR with the herding mechanism. The results are shown in Figure 5.25 together with those of a simulation with all the mechanisms up to inertia enabled. Although
5. Adaptive Routing in Wireless Sensor Networks

(a) Packet routes visualisation with no additional mechanisms
(b) Packet routes visualisation with diversion

(c) Ensemble average of latencies for 50 iterations

Fig. 5.24: Example RRR scenario with multiple LP packet sources, and the additional packet inertia and herding mechanisms
5. Adaptive Routing in Wireless Sensor Networks

Fig. 5.25: Example RRR scenario with multiple LP packet sources, and the additional packet inertia and herding mechanisms.
on inspection the differences between the routes taken by LP packets in Figures 5.25a and 5.25b are not great, the impact that herding has on the resulting latencies is significant. In Figure 5.25c it can be seen that, in this example, the modified RRR results in a latency improvement of over 50% of the difference in latency between the times when there is no LP traffic and when RRR is not enabled. A summary of the improvements which each of the mechanisms cumulatively provides in this scenario is shown in Figure 5.26.

Figures 5.27, 5.28, and 5.29 illustrate both the cost and the benefit of RRR in terms of the energy use trade-off. The energy use is measured from the simulations, and includes the energy used to send and receive packets of all types. Enabling RRR reduces the per-packet and per-node energy use along the HP route. However, the longer routes taken by the LP packets result in a higher per-packet and per-
Fig. 5.27: Ensemble per-packet energy use during the course of a simulation

Fig. 5.28: Mean per-node energy use during three periods of the simulation
node energy use for the LP packets and the nodes forwarding the LP packets respectively. With RRR, the energy use is distributed more evenly across the network, and the maximum per-node energy use is reduced, which would extend the network lifetime. This comes at a cost of the total network wide energy use, which is increased.

**The Impact of Network Load**

It stands to reason that an increase in the network traffic will result in overall greater latencies and losses. The plots of Figure 5.30 and 5.31 show the impact which the LP traffic rate has both with and without RRR enabled. The same low and high priority traffic sources as the previous simulation were used, where each of the four HP sources generates an average of 20 pps, but the number of packets each of the 24 LP sources generates on average per second is varied between 1 and 10. Recall that in previous simulations (Figures 5.24 and 5.25) the number was 4 pps.

In this scenario, increasing the rate of LP packets increases the positive impact of RRR on the latencies (Figure 5.30). This occurs for the HP traffic, as is the goal
Fig. 5.30: The ratio of the packet latency without RRR to with RRR for low and high priority packets

Fig. 5.31: The mean packet loss rates of high and low priority packets for the 10 rates of LP traffic, without and with RRR enabled
Fig. 5.32: The packet loss rates of high and low priority packets over time for 10 rates of LP traffic. As previously, the LP traffic starts at the 315 second mark, and RRR is enabled after 330 seconds.
Fig. 5.33: Impact of LP packet rate on the mean per-node energy use for nodes forwarding HP and LP packets.

of RRR, but additionally also does so for the LP packets to the extent that at 6 pps RRR is beneficial for both the high and low priority traffic. This increase peaks between 7 pps and 8 pps, at which point the rate of increase drops sharply for the HP traffic, and the benefit is lost for LP traffic. Inspection of Figures 5.31 and 5.32, which illustrate the losses and therefore the load on the network in relation to its capacity, reveals what is happening. For LP packet rates lower than 7 pps there are no significant losses either prior to RRR being enabled or after. An exception are the transient LP packet losses at the 315th and 330th second marks, at which points the LP traffic starts and RRR is enabled respectively. For 7 LP packets per second and over, there are increasing losses for both the HP and LP packets prior to enabling RRR. Enabling RRR removes the losses for the HP traffic (as the LP traffic is re-routed), but increasing the LP packet rates also increases its loss rates.

As would be expected, the mean energy use rises on both the nodes forwarding HP and LP packets as the LP packet rate is increased. This is seen in Figure 5.33
which shows the mean energy use for 10 iterations during each of the 3 phases of the simulations. The rate at which the energy use increases on the HP nodes is decreased by enabling RRR. For a LP packet rate of 7 pps and above, the rate of energy use increase decreases for the HP nodes when RRR is not enabled. This is explained by the increasing packet losses at those packet rates when RRR is not enabled.

**Single Sink Scenario**

The previous simulations have all involved four nodes in close proximity acting as sinks. Some of the benefit of modified RRR in these cases results from a larger proportion of the LP packets being routed to an alternative sink than the HP traffic. In the example of Figure 5.34 only a single sink is used, together with packet rates of 20 pps and 4 pps for the high and low priority packet sources respectively. The impact this has is twofold. Once both the LP and HP packet sources are started, but prior to enabling RRR, the latency of the HP packets rises to an average level of approximately 0.16 s, an order of magnitude higher than seen in simulations with four sinks and equivalent packet rates, but only slightly higher than those where the LP packet rate was 5 pps. Furthermore, once RRR is enabled at the 330th second mark, the latencies of both the LP and HP packets fall. The considerably higher latencies with no RRR result from the greater congestion in the nodes close to the sink where the routes are converging. Enabling RRR has the effect of distributing the paths in the vicinity of the sink. This load balancing property of RRR should be investigated further in future work.
Fig. 5.34: Example RRR scenario with multiple LP packet sources, but only one packet sink
5.7 Conclusions

Simulations demonstrating the behaviour of smart packets in the tinyCPN wireless sensor network routing protocol indicated that the overheads are substantial compared with broadcast based routing protocols available from the literature. This combined with the fact that the benefits of the wired CPN (source routing and continuous measurement of network QoS) are not present or substantially reduced, means that smart routing is not suitable for efficiently providing differentiable QoS in large sensor networks. The protocol was however implemented and experimentally demonstrated to provide reduced power use compared with an equivalent case in which smart messages are randomly forwarded.

The unsuitability of smart routing meant that another protocol was considered. Rather than simply settle on the use of the MCFA, use of the Random Re-Routing protocol to extend the MCFA was investigated. Several extensions to the RRR protocol which balance assumptions of the original theory were proposed and demonstrated. The protocol’s operation in a realistic wireless simulation of 600 nodes indicates that the modified RRR can effectively provide differentiated QoS to packets sharing the resources, and there are initial results demonstrating that the modified RRR may have uses as a load balancing protocol when a high network traffic load is unevenly distributed.

The steps involved in routing packets in RRR are shown in Figure 5.35.
Fig. 5.35: The modified RRR algorithm in flow chart form
6. QUALITY OF SERVICE AND QUALITY OF INFORMATION

6.1 Introduction

The previous chapter explored the use of adaptive routing protocols in large distributed wireless sensor networks and came to the conclusion that the overheads of the CPN protocol made it unsuitable for use in WSNs, but that the RRR protocol can effectively be combined with a simple lower level protocol to provide differentiated QoS. QoS in a sensor network is not necessarily an end unto itself, as it may be considered in traditional networks. For example, telephony applications require low latency, and streaming video services require adequate bandwidth for good user experiences. Instead, as sensor networks are ultimately intended to generate information regarding some phenomenon, QoS is required for the network to generate data suitable for producing information of high quality.

In the field of WSNs, Quality of Information (QoI) is a new concept, which focusses on information rather than the raw data. Sensors, networked or not, produce data in the form of measurements, such as 5 m, 10°C, or 100,000 Pa, which without context have little or no utility. However, within context, perhaps provided in the form of additional data on the location and the time a measurement was taken, the data collectively becomes information. The quality of that information is then related to the accuracy of the individual pieces of data, and the manner in which they are fused to produce the information.

From a study of the literature (Section 6.2) it becomes clear that there is a heavy
overlap between QoS and QoI, and that in particular the QoI issues of timeliness and reliability are tightly coupled with the latency and reliability dimensions of network QoS. This is true especially as they relate to WSNs, where link quality may be poor owing to short range radios and limited device lifetimes. Other resource constraints such as bandwidth may also impact the QoI delivered. Latency and reliability are influenced by the packet routing protocol used in the network, which also defines the network's ability to support differential QoS/QoI, which is necessary to support multiple parallel tasks using finite resources.

In order to explore how QoS can be measure and optimised, this chapter first examines the work on QoI from the literature, before providing an image processing inspired metric for measuring QoI, and an experimental exploration of that metric. The use of the sensor network routing protocol, specifically the Random Re-Routing (RRR) protocol, to meet task specific QoS/QoI requirements is then investigated. A method for predetermining the packet priorities required to meet task requirements is proposed, before the protocol is extended to also adapt its routing priorities once deployed.

6.2 Quality of Information

As QoI in sensor networks is a new area of study, the body of work directly referring to QoI in sensor networks is rather limited. The key early pieces of work have made effort to enumerate the attributes which comprise sensor network QoI [110, 111], many of which are found in the literature on organisational information quality [112]. The attributes can be divided into low and high level attributes. The low level attributes of timeliness, accuracy, reliability, throughput, and cost can be applied to the data coming directly from the sensors as well as the infor-
mation obtained from this data, and overlap heavily with traditional network QoS metrics. Thus they are influenced by the network and are suitable for targeting for optimisation and balancing by the routing protocol in similar ways to those discussed in the previous chapters.

The higher level information quality attributes of completeness, relevancy, and utility, are applicable once the data has been fused, and are therefore less directly linked to the routing. That said, information completeness may in some way be a function of network losses. It will however also be a function of sensor placement, sampling rate, and other properties beyond the control of the routing operations such as the sensor types in use (relative to the nature of the phenomenon being sensed). Information relevancy and utility are dependent on whether the agent requesting sensor data has requested the correct data and whether that data is presented suitably to the agent and are thus also beyond the influence of the network routing.

In addition to these attribute classifications, previous work has explored the role of QoI in hypothesis testing frameworks. In [110] Neyman-Pearson hypothesis testing is used to demonstrate the intuitive results that a stronger signal and faster sampling rate both increase the detection probability for transient events with decaying signals. In [111] the same system is investigated using Bayesian hypothesis testing, where the a priori probabilities of the event having occurred or not having occurred must be known, a perhaps unrealistic assumption. Similar relationships are demonstrated, with the addition of the result showing that the probability of detection increases with the probability of the event occurring. These methods are placed within an analysis framework for planning and designing a sensor network in a QoI aware manner in [113, 114].

In Section 6.4.1, aspects of Bisdikian et al.’s 5WH (Why, Who, What, Where,
When, and How) letter soup framework [115] are employed. This is a further framework, intended to organise the elements of a human-readable WSN task statement into each of the 5WH categories, so that the information quality requirements of tasks and the capabilities of the WSN resources can be matched.

By way of example, consider the following task statement: To protect its fragile ecosystem, the Brazilian authorities need to monitor a region (ρ) of the Amazon rainforest for illegal felling. In this example, the why (ecosystem protection) and who (Brazilian authorities) have an impact on the higher level priority considerations. What covers the more specific goals of the task (monitoring for illegal felling). How may be the deployment of a WSN capable of monitoring for the sounds of heavy machinery, or involve a network of human informants. The spacial and temporal where (region ρ) and when (undefined, but is presumably dependent on the urgency of the why and what) relate to the how as they define the location and times during which the WSN’s resources must be dedicated to the task.

The abstract scenario which will be described in Section 6.4.1 is similar in nature to the scenarios used in the RRR simulations of Chapter 5, and focuses on a subset of the 5WH framework, namely the interactions between the who, what, and how elements. The framework is used to illustrate a heuristic method for assigning RRR routing priorities (as described in Chapter 5) to tasks running simultaneously in the network, in such a way as to meet task requirements. Due to the complex interplay between the traffic of multiple tasks, this is non-trivial.

6.2.1 Measuring Quality of Information

Little work has gone into quantifying QoI in a manner specifically designed for sensor networks. One approach to this is considered here.

Consider a view of an area being monitored by sensors in terms of the phe-
nomenon the sensors are measuring, for example the brightness of the light being cast on the area’s surface, or the intensity of some other form of radiation at each point of the area’s surface. This view, when captured through a suitable camera with a bird’s eye view of the area would be the *ground truth* and would consist of a series of frames $G_t$.

The sensor devices are of course sampling the phenomenon in the area and individually sending their measurements through the network’s ad-hoc infrastructure to its output. Suppose that these readings are input to some fusion algorithm in an attempt to recreate the frames of the ground truth. These recreated frames $V_t$ will differ from those of the ground truth due to temporally and spatially non-continuous sampling (as the sensors have distinct locations and take readings at some interval), quantisation errors, and also influences of the network routing. The network will affect the output in two main ways. Measurements will be delayed by the packet travel times resulting from queueing at intermediate nodes, collisions and subsequent retransmissions, and indeed by packet loss en route to the network output. This system is shown in Figure 6.1.

Now consider the two frames $G_t$ and $V_t$ at the same time instant $t$; $G_t$ can be
thought of as the signal, while $V_t$ is the “signal plus the noise”. Thus the mean square error or noise $M_t$, and the peak signal to noise ratio (PSNR) $Q_t$, at time $t$ can be defined as:

$$M_t = \frac{\sum_{i=1}^{I} \sum_{j=1}^{J} [G_t(i, j) - V_t(i, j)]^2}{I \times J} \quad (6.1)$$

$$Q_t = \frac{\max\{G_t(i, j) : 1 \leq i \leq I, 1 \leq j \leq J\}}{M_t} \quad (6.2)$$

where $G_t(i, j)$ and $V_t(i, j)$ are the pixel values of $G_t$ and $V_t$ respectively, and $I \times J$ is the size of the frame. Note that while the PSNR is often expressed in decibels, a pure ratio is used for this definition. For illustrative purposes a third frame $N_t$ is also considered in the examples which follow. $N_t$ is a reconstruction of the signal using measurements directly from the sensors, that is, before the packets have travelled through the network and have been subjected to its properties.

The MSE and PSNR are metrics commonly used to measure quality in the field of image processing [116]. The following section explores the use of PSNR as a measure of QoI.

### 6.3 An Experimental Exploration of Sensor Network QoI

A number of experiments which attempt to capture a range of properties and influences that the sensor network has on the outputs are performed in the following sections. Each of the experiments uses the real wireless sensor network, consisting of 34 motes, each of which is connected through a wired network to a central computer, but communicate with other motes using their radios. The location of the motes is simulated, and the ground truth frames $G_t$ are generated on the central computer. The corresponding sensory data are sent to each mote over their wired connection as the changes occur in the frames. The motes periodically
sample this sensory data and simultaneously send their measurements wirelessly over multiple hops to a mote acting as the network output sink, and directly over the wired infrastructure to the central computer. There the measurements are gathered and fused, again by the central computer, to create the frames $V_t$ and $N_t$, which can be used to determine the PSNR.

The ground truth frames in these experiments consist of a dark area in which points of light turn on and off. The motes’ sensors are assumed to be omnidirectional light sensors, and so the light levels sent to the motes are the brightnesses at the points at which the motes are placed, and are proportional to the sum of the inverse of the square of the distance from the origin of each light point. Figure

Fig. 6.2: Example interpretation at the sensor network output of one or two readings. The location of the motes reporting measurements is shown by small crosses.
As the brightness fades from the point of origin of the light in a known way, and as the lights are assumed to be of fixed intensity, a brightness measurement at the output of the network is effectively a distance measure. This means that a single reading places the origin of the light on a circle centred about the mote, which can be interpreted as shown in Figure 6.2b. A second measurement from a second mote places the origin of the light in one of two locations defined by the intersection of the two circles centred about each of the motes. This is shown in Figure 6.2c. A third reading allows the light to be accurately placed, and further readings could be used to reduce the error. This trilateration is an example of a simple fusion algorithm which allows the series of frames $V_t$ to be generated, and in turn the PSNR to be calculated as a measure of QoI.

As motes also forward measurements when there is no light to detect, light readings of zero are also received at the sensor network output. These readings do not reveal the location of a light, but can be used to determine the absence of light. This is especially useful when only two motes have returned readings indicating the presence of light. As two readings place the light in one of two locations at the intersection of the circles created by the readings, a reading of zero from a third mote will help to determine which of the two intersections is not the location of the light. This increases the QoI.

6.3.1 The Experimental Results

In the first experiment, the setting includes 20 wireless sensing motes, placed at regular intervals of 100 cm, in a $4 \times 5$ m rectangular grid. A single light turns on and off every two and a half seconds in a random location within the area of the sensor network. The light has a brightness which can be sensed by a mote up to
a distance of 125 cm, but not beyond. The motes themselves take a measurement every one second and transmit the reading over multiple hops to the sink. Readings arriving at the sink are kept for one second after they are received and are used to create the view $V_t$. The “pre-network” readings sent directly to the central computer over the testbed’s wired connection are treated in the same manner and used to create the view $N_t$.

Figure 6.3 shows a sample of the time varying PSNR resulting from this approach. The “light” and “dark” areas of the graph illustrate when the light is on and off. The two lines show the instantaneous PSNR of the frame $V_t$ (in black) and $N_t$ (in grey). In both cases there is a delay between the light turning on and the arrival of measurements at the sensor network output (and the subsequent increase in the PSNR or QoI). Lost or delayed packets also mean that there are not always three readings to faithfully reproduce the ground truth.
When the light is off, the absence of light is treated as the signal by inverting the frame $G_t$. As readings of zero arrive at the sink, or as existing readings time-out, the frame can be recreated perfectly so the PSNR rises to an arbitrarily applied limit, 100.

When three readings are being used to position the source of the light, the offset between the actual location and the sensed location could be used as a QoI metric. When the output of the sensor network has produced enough data to place the light in one of two locations the sum of the separation between the real position and the two sensed positions could be used as the measure of QoI. However with a single reading producing a continuous circle, this method ceases to function. This is where the advantages of the PSNR are apparent as a measure of the QoI. By providing a continuous measure of the QoI, applicable in applications where radiation the intensity of radiation is measured over some area, the PSNR is a versatile metric.

From Figure 6.3 one can observe a further measure which can be applied to
the QoI. The latency, i.e. the difference in times at which the event occurs in the environment and is reflected in the measurements. In this scenario there are three main sources of error which impact the QoI: the latency between a change in the ground truth and the sensors sampling, the quantisation error which results in offsets between the final placement and the ground truth, and network packet losses. The network latencies are in the case of this small network two orders of magnitude lower than the sampling period and are therefore negligible in the figure. Figure 6.4 shows ensemble averages (over one “on” and “off” period) of the PSNR, for four sampling periods, for both the $V_t$ and $N_t$ cases. It is observed that
as the sampling interval decreases, the *latency* between the event (i.e. the light turning on or off) and the subsequent rise in the QoI also decreases. However, for a sampling interval of 500 ms, this benefit is reduced, and additionally the QoI achieved is on average lower than in the other cases. The extra load this sampling rate places on the network results in a significantly higher packet loss rate of 0.48, compared to 0.07 with a 2000 ms sampling interval.

In order to further illustrate the sensor network’s effect on the interpretation of the readings, additional experiments using a different approach to packet transmission have been run. In the first experiment, all motes were sampling the light level at 1 s intervals and sending their readings to the sink. For this second approach the aim is to reduce the overall network traffic. When a mote senses the presence of light, it broadcasts its measurement so that other motes in the area, which may have also sensed the same light, receive it. This allows the motes to intelligently determine whether to transmit their measurements to the network output. For example, only the mote with the strongest measurement (logically the mote closest to the origin of the light) may transmit, as shown in Figure 6.6, or if the computational abilities of the motes allow, individual motes may attempt
Fig. 6.7: QoI resulting from only the top \( n \) readings, possible due to broadcasts. The mean number of readings used for each value of \( n \) is shown above the PSNR values.

to estimate the QoI from the overheard readings in order to intelligently decide whether their readings are required to meet some QoI threshold. These approaches should reduce the number of packets travelling on the multi-hop route to the sensor network output as well as the overall power consumption.

Figure 6.7 shows the results from experiments where the motes with the top \( n \) readings are allowed to send their readings. The PSNR values shown are the mean of those achieved by the time just before the light turns off. The figure also shows the mean number of readings received at the sink at that point in time. The results from the previous experiment are shown for reference. It can be seen that, when using broadcasts, there is an approximately linear relationship between \( n \) and the QoI. Also evident is that the number of readings received at the sink is slightly higher than \( n \). This occurs as some motes may not receive all broadcasts, and therefore transmit their readings despite not having one of the \( n \) strongest readings. However, the numbers are still considerably lower than the
mean of approximately 14 readings for the case without broadcasts, indicating a substantial power saving.

**Interpretation of PSNR in this Context**

One issue yet to be addressed is the interpretation of a particular PSNR value with respect to the QoI. For a single reading the PSNR is a non-linear measure of the distance between the sensor and the origin of the light. The PSNR ranges between close to zero at 125 cm, and 10 at a distance of 5 cm, and as the distance approaches zero the PSNR approaches infinity. When the sensed interpretation places the light in one of two locations, and when the two locations are significantly separated, the PSNR is dominated by the “incorrect” location. Assuming the “correct” location matches perfectly, the PSNR is just below 60. As the two locations begin to converge this value rises. A third reading places the light at a single point, and the PSNR is a measure of the offset between the real location and the sensed location. In this case one expects a PSNR between 30, where the distance approaches the sensing range, and 100 (the arbitrary limit). One may therefore heuristically state that, when monitoring a single light, a PSNR above 60 is excellent as it indicates an accurate location of the origin of the light, and a PSNR above 30 is good.

### 6.3.2 Extending the Experimental Setting with Multiple Simultaneous Lights

In Section 6.3.1, the experimental setting was limited to a single light or event occurring within the monitored area. This was an aid to the introduction of some of the concepts of QoI used in the previous experiments. A more realistic scenario would extend to the more challenging case where the sensor network is used to monitor a less constrained environment in which a number of events may be occurring simultaneously. This section explores several such scenarios, and in each
case proposes an algorithm with which to process the outputs.

**Lights in Random Locations**

Here, multiple lights are incorporated, which may be on simultaneously. It is ensured that the lights are sufficiently far apart, so that each mote only senses one particular light. Therefore, before performing the trilaterations, the readings from different motes must be grouped into sets associated with individual lights. As each mote reading provides only data on distance and not direction, and as readings may originate from lights which are no longer on due to network and sampling delays, this grouping is non-trivial.

In this experiment, as previously, the motes are sampling the light at one second intervals, and sending their readings to the sink. The algorithm which is employed at the sink has several steps. Initially, a simple spatial clustering algorithm is applied in order to separate groups of readings which cannot be related due to distance. For each of these spacial clusters, a “pair-matrix” is then generated. Recall that a single reading places the origin of the light on a circle centred around the mote. The pair-matrix determines which motes’ reading-circles intersect, indicating that they may be readings of the same light.

An estimate for the number of lights within each spacial cluster is calculated by recursively determining the largest number of “unpaired” readings; in other words, readings which must be of different lights. These readings are dubbed “anchor” readings and simplify the next step of the algorithm. For each anchor reading, the largest “pair-set” is determined. Each reading paired with an anchor reading results in two intersect points. A “pair-set” is a set of readings paired with the anchor which all match a particular intersect point. If the anchor has no paired readings then the pair-set size is one. If there is only one paired reading or each
paired reading has no other matching readings then the pair-set has a size of two. Each intersect point is checked for “zero”-readings from nearby motes which make the point unlikely.

The anchors are sorted by the size of their largest pair-set. The size of the pair-set reinforces the probability that that point is the location of the light, and thus a cluster is created with the readings from the largest pair-set. If there is more than one anchor with that size of pair-set, the anchor with the smaller number of pair-sets takes precedence. The motes are removed from the spacial cluster, and the largest pair-sets for each anchor recalculated. This is repeated until there are either no more motes in the spacial cluster, or all the anchors are used.

This algorithm allows the successive stages of trilateration to then be performed on each cluster of readings to recreate the frame. As previously, to evaluate the use of the PSNR as a QoI metric, experiments have been run on the sensor network
testbed. Figure 6.8 illustrates a sample of the time-varying PSNR as three lights turn on and off. Other than the number of lights, the setting matches that of the previous experiments. From the figure it is clear that the average PSNR is lower than for the single light example. This is expected as with multiple lights there are multiple locations for error or noise in the recreated frame. This is an issue that arises in quantifying the QoI from multiple events in this manner.

*Lights in Fixed Locations*

The setting for this experiment envisions a scenario where there are multiple lights with fixed locations placed over an area being monitored by motes, whose number is smaller than the number of lights. This scenario may arise if, for example for cost reasons, the number of motes with wireless radios and processing capabilities is limited, and events are detected using Triggers which communicate with the motes via lights or other forms of radiation or signalling. By using the sensed light reported by the motes to calculate which Triggers are *on* and *off*, one can determine where in the monitored area the activity is taking place.

The readings reported by each mote are the sum of the intensities of the light from each Trigger at that mote. As both motes and Triggers have fixed locations, and the light intensity decreases over distance in a predictable manner, the intensity which each mote should sense from each Trigger which is *on* is known.

Let \( r_m \) be the reading at mote \( m \), and let \( i_{m,\tau} \) be the contribution from Trigger \( \tau \) to mote \( m \), assuming \( \tau \) is On. Let \( x_\tau \) be a binary value determining if the Trigger \( \tau \) is On or Off. Let \( M \) be the number of motes, and \( T \) the number of Triggers. This then forms a system in the form \( r = Ix \), where \( r \) is a vector of the readings \( r_m \), \( I \) is the matrix of known light values \( i_{m,\tau} \), and \( x \) is the binary vector of the
Triggers’ states $x_\tau$ to be calculated. This is given in (6.3).

$$
\begin{pmatrix}
 r_1 \\
 r_2 \\
 \vdots \\
 r_M
\end{pmatrix} =
\begin{pmatrix}
 i_{1,1} & i_{1,2} & \cdots & i_{1,T} \\
 i_{2,1} & i_{2,2} & \cdots & i_{2,T} \\
 \vdots & \vdots & \ddots & \vdots \\
 i_{M,1} & i_{M,2} & \cdots & i_{M,T}
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 \vdots \\
 x_T
\end{pmatrix}
$$

(6.3)

As there are more unknowns than equations in this linear system, it cannot be solved uniquely. Furthermore, as the network impacts the accuracy of the values in $r$, as packets may be delayed or lost, there may be no solution which satisfies all the equations at any one point in time. A common method for solving such underdetermined systems is to apply the Moore-Penrose pseudoinverse to matrix $I$ to calculate a least squares solution [117].

Before using the pseudoinverse to determine which Triggers are On or Off, the number of unknowns is minimised to increase the performance. Let $L(\tau)$ be the set of all (other) Triggers except $\tau$. Then, for each mote $m$, if there is a recent reading:

- if $r_m < i_{m,\tau}$ then Trigger $\tau$ is obviously Off,
- if $r_m \geq i_{m,\tau}$ and $i_{m,\tau} > \sum_{l \in L(\tau)} i_{m,l}$ then $\tau$ is On.

The Triggers with known states are then removed from the system in (6.3), and the Triggers determined to be On have their known light contributions $i_{m,\tau}$ subtracted from the motes’ readings. These adjusted readings are then used in the pseudoinverse calculation to determine the state of the remaining Triggers.

This example uses six motes arranged in two columns of three, and twelve

---

1 A neural network based solution using the RNN [118] was also attempted, but considering the large training sets and time required for satisfactory results, was not fully explored.
Fig. 6.9: A sample of the results from Section 6.3.2. Number of incorrectly detected lights and the actual number of lights in the On state are shown.

Triggers arranged in three columns of four between the motes. Every 2.5 seconds each Trigger has an independent probability of 0.1 of switching On if Off, and 0.75 of switching Off if On. A sample window from the results is shown in Figure 6.9. In this example the number of incorrectly determined Triggers is used as a measure of the QoI. As in the previous examples, the PSNR could be employed, yet would simply be proportional to the number of correctly interpreted Triggers, while obfuscating the information. It therefore makes sense to use a scenario specific metric.

As in the previous experiments, there is latency between the occurrence of events and the observed improvement of the QoI at the sink. As updated measurements arrive at the sensor network output, applying the pseudoinverse proves to be an effective means of determining the state of the Triggers.
6.3.3 Quality of Service and Quality of Information - Some Remarks

These series of scenarios have highlighted that it is difficult to determine a single measure suitable for the concept of QoI, as it is highly dependent on the scenario. Another crucial issue is that the methods of quantifying the QoI from the previous sections are only suitable when the ground truth is known, which typically is not the case. This means that such metrics are useful during the “calibration” stage, for example in order to state that the network is configured to provide some level of QoI. This is different from where the QoI is expressed in probabilities (which was the direction evident from the approaches in some of the literature), which either the network itself or the data fusion algorithm at the network output determines and attaches to data or information as meta-data. This can be used by the agent who is ultimately acting on the information to inform the decision.

The examples from the previous sections do however highlight the overlap between QoS and QoI. Increasing the sampling rate is the simplest, most direct method for increasing the QoI from the application standpoint, as was also shown in [110]. From the routing protocol’s point of view, packet loss was the main aspect of the above scenarios. The RRR protocol, modified as shown in Chapter 6, has been shown to target network losses in highly congested networks, and can also differentiate or prioritise the latency of traffics. In larger networks the latency would be expected to play a larger role on the QoI than was observed here. In the examples in Section 5.6 of Chapter 5 where the sampling rates were higher (at 20 pps) and the packet latencies in the same order, this was indeed the case. The following section therefore deals with matching the QoS/QoI requirements of specific tasks with the resources the network can provide by adapting their RRR priorities suitably.
6.4 Matching Requirements to Resources with Adaptive RRR Priorities

An assumption often made in the literature when considering the use of WSNs in defensive military detection roles is that modern warfare is highly asymmetric and that the latency between detection and reaction is zero. This can not be the case, as even in a fully automated defence system there will still be communication and processing delays.

RRR provides best-effort differentiated QoS in a multi-hop wireless network. However, it does this indirectly, by assigning user or task specified priorities to packets, and routing them using a relevant routing policy, rather than by directly specifying a QoS/QoI (or quality) requirement. The protocol cannot therefore guarantee a quality requirement or goal, only that one traffic flow will have a higher quality than another, assuming their flows interact in the network. Two mechanisms are proposed to tackle this problem, both offline and online.

6.4.1 Determining Priorities Offline

In this section, an abstract scenario is presented, where \( M \) tasks simultaneously wish to use a WSN resource \( R \) for some monitoring purpose. Suppose that each task has a quality requirement which can be expressed in terms of a utility function \( f_{U,m} \), which takes some combination of quality metrics as input, and a utility threshold \( \lambda_m \). Each task may also provide a “desired” set of those quality metrics \( X \).

As there are multiple tasks and the WSN has fixed total resources distributed both temporally and spatially, the initial desired quality metrics of all tasks may not be achievable. Configuring the network in such a way that the utility threshold
of each task is met may be non-trivial. Furthermore, existing tasks running in the WSN deployment may also have their own quality requirements, and the additional load of new tasks on the resources may cause these requirements to no longer be met.

A subset of the 5WH framework (described previously in Section 6.2), consisting of the who, what, and how elements, where each of the elements is a black box with interactions between them is shown in Figure 6.10. The idea is for the interactions between the boxes to be defined so that the system determines a set of RRR priorities to meet the tasks’ requirements. The tasks reside within the who-box and issue their requests (in terms of \(f_{U,m}, \lambda_m\), and \(X\)) to the what-box. The what-box begins an iterative process with the how-box to determine the final network parameters \(Z\). Note that the interaction between the who- and what-boxes is one-
way, assuming it is always possible to satisfy the requirements of all tasks. A more
general solution would incorporate admission control, and reject tasks from the
system when it is not possible to satisfy the requirements of all tasks.

During this process, the how-box determines a set of intermediate parameters
\( T_i \), which are first used in a simulation of the WSN running the tasks. This
approach allows the actual quality (\( Y_i \)) these network parameters will provide to
be accurately determined, without testing on the real network and interfering with
already deployed tasks. As it is however difficult to model the relationship between
\( T_i \) and \( Y_i \) (hence the WSN simulation), it is unlikely that the initial values \( T_i \) will
satisfy all tasks. Therefore the iterative process allows the how-box to adjust the
network parameters \( T_i \) based on the utility threshold and achieved quality (from
the simulations) for each task, until the what-box determines that the achieved
quality satisfies all tasks’ utility thresholds.

Once the what-box does so, the final network parameters \( Z \) can be passed to
the real network, and the tasks deployed. A realised system should also consider
feedback from the real WSN to the simulation, as the simulator must be aware of
the current state of the resources it is modelling, for example node failures and
existing tasks.

**Iterative Translation of Latency Requirements into RRR Priorities**

This section is interested in tasks which must rapidly and automatically respond to
detected events, and approaches QoI from the perspective of the network routing
protocol, with specific interest in the travel time of readings in the network. As
the RRR protocol does not guarantee specific latencies, rather that if traffic \( A \)’s
priority is greater than \( B \)’s the latency of traffic \( A \) will be lower than that of \( B \), the
tasks’ latency requirements must be translated into RRR packet priorities. The
complex interplay between different flows in the network also means that this is
not a simple task.

Say there are $M$ tasks and a set of latency requirements $\Lambda = \lambda_0, \lambda_1, \lambda_2, ..., \lambda_M$. The goal is to determine an equivalent set of RRR priorities $Z = p_{i,0}, p_{i,1}, p_{i,2}, ..., p_{i,M}$, where $p_{i,m} \in [0, 1]$. After initialising the priorities to random values, the simulation is run with these priorities. Then, based on the achieved set of latencies $Y_i = y_{i,0}, y_{i,1}, y_{i,2}, ..., y_{i,M}$, the priorities are updated and a further simulation is performed. This is repeated until the requirements $\Lambda$ are met, as shown in Algorithm 3.

The constant term $\alpha$ in Algorithm 3 influences the convergence rate. As the algorithm is heuristic in nature, the value of $\alpha$ will be dependent on the scenario in question, but experimentation has found that for the cases presented here values between 1 and 7.5 are effective.

**Algorithm 3** Iteratively determining RRR priorities

1: $i \leftarrow 0$

2: for $m = 1$ to $M$ do

3: $p(i,m) \leftarrow $ RANDOM

4: end for

5: repeat

6: run WSN simulation on priorities $P$ to determine latencies $Y$

7: for $m = 1$ to $M$ do

8: if $y(i,m) > \lambda(m)$ then

9: $p(i + 1,m) \leftarrow p(i,m) + \alpha \times (y(i,m) - \lambda(m))$

10: for all $n \in [1...M]$ where $n \neq m$ do

11: $p(i + 1,n) \leftarrow p(i,n) - \alpha \times (y(i,m) - \lambda(m))/M$

12: end for

13: end if

14: end for

15: $i \leftarrow i + 1$

16: until $y(i,m) \leq \lambda(m)$ for all $1 \leq m \leq M$
Simulation Results

The first scenario considered involves two tasks which both share the same four packet sources, each generating traffic at the same rate of 20 pps per task. Except for the placement of the packet sources, the fact that the simulation tasks are only active for 20 s, and that RRR is always enabled, these parameters mirror those of the HP/LP scenario from Chapter 5.

Figure 6.11 shows two examples of this scenario. It can be seen that over a series of iterations the RRR packet priorities adapt in both cases from their initial random values to levels which result in the latency requirements of both tasks being met.

A further two simulations, this time with three tasks each with a group of traffic sources in the locations previously seen in the HP/MP/LP example of Chapter 5 has also been performed. Each packet source is generating a packet every 0.075 seconds. The results are shown in Figure 6.12.

The preceding two examples both have equivalents in the Chapter 5. In those cases the priority was pre-determined in order to demonstrate the differential QoS RRR could provide. In these examples however, the latency requirements have been chosen and the iterative algorithm has been able to determine routing priorities which achieve these latencies. Each scenario was run twice, and in each case has arrived at a different set of RRR priorities, and has taken a different number of iterations to converge. This illustrates that there are no unique solutions to each problem and that the final results are dependent on the initial selected priorities.

The remaining figures (6.13 and 6.14) illustrate two runs of the algorithm for scenarios equivalent to the remaining scenarios of Chapter 5. Again, the priorities are seen to converge until the specified latency requirements are met.
(a) Determining RRR priorities to achieve latencies of 0.019 and 0.022 seconds

(b) Determining RRR priorities to achieve latencies of 0.02 and 0.021 seconds

Fig. 6.11: Two examples of iteratively determining RRR priorities which satisfy the requirements of two tasks. Each figure displays two sets of priorities (scaled to the left y-axis), and achieved and required latencies (scaled to the right y-axis).
Fig. 6.12: Two examples of iteratively determining RRR priorities which satisfy the latency requirements (0.022, 0.015, and 0.0105 seconds) of three tasks
Fig. 6.13: Two runs of a single HP, multiple LP scenario equivalent to that of Section 5.6
Fig. 6.14: Two runs of a single HP, multiple LP, and 1 sink scenario equivalent to that of Section 5.6
Determining Priorities Online

Once deployed, RRR currently has no way of adjusting its priorities in order to adapt to changing network and topology conditions, as was possible with smart routing. The limited lifetimes of the battery powered devices, of which WSNs consist, mean that such functionality would however be important. Even a network in which the devices are stationary is likely to have changes in topology over its lifetime which mean that the initially assigned routing priorities will not continuously maintain the same required quality. In this section, a method for adjusting the RRR priorities online is proposed.

Assuming the nodes in the network have synchronised clocks, as is possible using methods such as those discussed in [119], the data sinks can determine the network travel time of each packet. If the packets are not meeting their QoS requirements (or indeed if the network is over-achieving, and there is room to allow other tasks’ packets greater priority), the sinks may tell the sources to adjust their RRR priorities.

Suppose that each node in the network maintains a list of recent packets it has forwarded, including their identifying details, such as source, task affiliation, and sequence number, and also their previous hops. This is already typically the case as routing protocols will wish to suppress routing loops by dropping packets they see more than once. It would be possible to update the RRR priorities dynamically if this list were long enough to keep this information for the time required for the packet to reach one of the data sinks, have that sink evaluate the quality performance of the packet, and subsequently reply with an acknowledgement packet which would use the previous hops from the list to traverse the reverse route back to the source. Indeed, this method was used for forwarding the acknowledgement packets in the tinyCPN routing protocol of Chapter 5.
In the tinyCPN protocol there are however specialist non data-carrying *smart* packets which triggered the sinks to respond with acknowledgement packets. As this implementation of RRR uses a geographic version of the MCFA protocol, which avoids such overheads, there is no equivalent. Therefore the concept of a *tracer packet* is introduced. A tracer packet carries data in the standard manner, but additionally leaves a trail in the form of previous hops stored at the nodes it passes. Once a tracer packet reaches the sink, the sink may decide to send an acknowledgement if required.

In terms of overheads this approach is relatively small. The proportion of data packets which need to also be tracers would be a function of the frequency with which each packet source is generating data packets and the required time between the routing priority needing to be updated and the update occurring.

**Centrally Managed Priorities**

The routing in the MCFA with RRR is distributed. There is no global coordination in the network, the nodes each making their routing decisions based on the information they have available from their neighbours. This decentralised approach makes the network robust to failures. The quality requirements and achieved quality of the traffic flows are however only known by the data sinks, and therefore the decisions regarding the prioritisation of the different tasks are appropriately made in a centralised manner at the sinks.

For the results which follow below, tracer and acknowledgement packets have been incorporated into the simulation. Each data packet has a probability of 0.03 of being a tracer packet. Each sink maintains a list of tasks, their target latencies, average latencies the tasks’ packets are achieving, and their target priorities. In order to remain current, the rolling averages are updated for every data packet the
sink receives.

When a tracer packet arrives, the receiving sink applies a basic algorithm (similar to that used in the offline case) to the target priorities and sends an acknowledgement if the target priority of the tracer’s task is altered. To improve the latency of a task, its priority must be increased relative to the priorities of tasks whose packets’ paths may overlap. This is estimated by clustering the tasks according to the angle between their sources and the sink. If the tracer’s task’s latency requirement is not being met, its priority is increased by an amount proportional to the difference between the requirement and the measured latency. The priorities of all other tasks in the cluster are decreased, also by a proportional amount.

When there are multiple sinks (as there are in these simulation scenarios), these sinks will not be coordinated. As there is a randomised element to re-routed packets’ paths, packets from the same task/source pair may arrive at separate destinations. However, this slows down the adaptation process. No solution to this is proposed here, although the data sink nodes in a real WSN deployment could have a network connection beyond the WSN itself, thus making synchronisation a simpler matter.

Simulation Results

Here again each of the RRR scenarios is repeated with incorrectly assigned initial routing priorities and a target latency for each task. The tracer packet acknowledgement mechanism is allowed to update the priorities to meet the latency targets. For example, based on the latencies achieved in the example from Figure 5.21, where RRR packet priorities of 0.0 and 1.0 achieved a latency between approximately 0.012 s and 0.014 s for the HP task, the equivalent task is given a target latency of 0.012 s at the 315th second mark.
(a) The packet priority of sources with a specified latency requirement adapt over time. The dark points show when an acknowledgement packet with a specific priority is sent, and the lighter points when a data packet is received. Logically, data packets with specific priorities follow acknowledgements specifying those priorities. At the 315th second, a requirement of 0.012 s is set for the second task.

(b) The changing of the priority is reflected in the achieved latencies of the two packet sources. The straight line indicates the target latency.

Fig. 6.15: Example simulation of RRR priority adaptation with two tasks, equivalent to the HP/LP scenario of Section 5.6
(a) Adapting priorities in a three-task scenario. At the 360th second, requirements of 0.015 s and 0.01 s are set for Task B and Task C.

(b) The changing of the priority is reflected in the achieved latencies of the three packet sources. The straight lines indicate the target latencies.

Fig. 6.16: Example simulation of RRR priority adaptation with three tasks, equivalent to the HP/MP/LP scenario of Section 5.6
Once the target latency is assigned, the acknowledgement mechanism iterates the priority of the task’s packets up from 0.0 (where both tasks started) to the range of 0.9 to 1.0 as can been seen in Figure 6.15a. It is possible to converge on a range as there are multiple sinks managing the priorities independently, each task has four sources, and there is randomness to the re-routing of packets. As each source’s packets can arrive at different sinks, each source may end up assigning different priorities to its traffic. The changing priorities are reflected in the packet travel times, which diverge until the target of 0.012 s is reached.

When a third task is added, the adaptation performs less predictably, as the second simulation in Figure 6.16 illustrates (which mirrors the HP/MP/LP scenario from Chapter 5). Similarly to the two-task scenario, target latencies of 0.015 s have been assigned to Task B, and 0.01 s to Task C. These are based on the latencies achieved by the example in Figure 5.22, and are assigned at 360 s. It is observed that, over time, the acknowledgement mechanism will assign priorities to the different tasks which result in packet travel times comparable to the target latencies.

The mechanism has been less successful here, as the final priorities are considerably different from the base three-task example. In fact Task B (the MP task in the base example) ends with priorities at its sources which equal and exceed those of Task C (the HP task in base example). This means that while Task B meets and exceeds its target, Task C falls short.

As in Section 6.4.1, the remaining figures (6.17 and 6.18) illustrate results for scenarios equivalent to the remaining scenarios of Chapter 5. Figure 6.17 demonstrates the case where there are multiple LP sources. The simulation is repeated, in each case with a different initial priorities. Again, because the sinks are not synchronised, the success of the adaptation is dependent on these initial
6. Quality of Service and Quality of Information

(a) At the 315th second a requirement of 0.0235 s is set for Task B

(b) The changing of the priority is reflected in the achieved latencies of the Task B’s packet sources. The straight line indicates the target latency.

(c) At the 315th second a requirement of 0.0235 s is set for Task B

(d) The changing of the priority is reflected in the achieved latencies of the Task B’s packet sources. The straight line indicates the target latency.

Fig. 6.17: Example simulation of RRR priority adaptation with two tasks. Task B has a cluster of 4 sources, and Task A’s sources are distributed over the area of the network. The network has four sinks.
6. Quality of Service and Quality of Information

(a) At the 315th second a requirement of 0.0235 s is set for Task B

(b) The changing of the priority is reflected in the achieved latencies of the Task B’s packet sources. The straight line indicates the target latency.

Fig. 6.18: Example simulation of RRR priority adaptation with two tasks. Task B has a cluster of 4 sources, and Task A’s sources are distributed over the area of the network. The network has a single sink.

priorities. By contrast, in Figure 6.18, which presents results equivalent to the single sink scenario, the process is reliably successful.

6.5 Conclusions

This chapter has experimentally explored the concept of sensor network Quality of Information and concluded that its overlap with network Quality of Service means that it is suitable for targeting for optimisation by the network routing protocol. Based on the results of Chapter 5, the modified sensor network specific version of Random Re-Routing was used to demonstrate methods for matching user or task QoS/QoI requirements to the network resources. In order to do this, the RRR protocol was extended to include a sensor network suitable tracer/acknowledgement mechanism, similar to that used in smart routing, allowing messages and instructions to be transferred to the sources where the tasks reside.
7. CONCLUSIONS

7.1 Contributions

The contributions of this work fall under three categories; wireline network QoS, WSN QoS, and QoI. They are summarised below.

Several novel approaches to the problem of ensuring optimal network performance on different Quality of Service metrics which change according to user requirements have been proposed and experimentally evaluated in this research. The work attempts to show how the advantages of low complexity adaptive routing protocols can be leveraged in both wired and wireless network contexts to improve performance along multiple QoS dimensions.

7.1.1 Network Quality of Service

Existing routing algorithms used in CPN (such as the RNN) have been shown to provide QoS routing but at considerable cost in terms of processing requirements. In order to investigate the possibility of reducing these costs without compromising QoS, a study of the use of decision making algorithms other than the RNN in CPN was performed in Chapter 3. The complexity of these algorithms was compared, and it was found that those of lower complexity (in terms of run-time, memory requirements, and hardware size) provided equal QoS performance to the established RNN. The conclusion which can be drawn from this is that the QoS which CPN provides stems from its continuous measurements of the QoS along
links, and the ability this gives the protocol to adapt continuously.

Chapter 4 describes the development of the first working CPN hardware routers implemented using FPGA technology, to investigate whether it is feasible to use the CPN routing protocol in dedicated hardware. These used the approximated RNN and $\infty$-SR algorithms described in Chapter 3. It was shown that CPN is compatible with the architectures used in IP routers, and that therefore the realisation of dual purpose CPN/IP routers is a viable option. This would be necessary in order to create “islands of QoS” in the broader Internet. Furthermore, in order to enable the use of CPN in larger networks, while restricting the growth of the next-hop memories and mailboxes, a method for using longest prefix matching was proposed.

7.1.2 Wireless Sensor Network Quality of Service

The next stage of the research attempted to discover whether the advantages of the CPN principles of smart routing could be beneficially applied to wireless sensor networks, work which was described in Chapter 5. It was hypothesised that use of CPN in WSN may help to minimise the energy use along routes in the network. However, due to the restrictions which WSNs introduce, some of the advantages of smart routing in the wired CPN were lost. These included the continuous measurement along routes, and source routing. These issues, combined with the fact that through simulation it was found that the overheads of smart routing where considerable, leads to the conclusion that the tinyCPN protocol is unsuitable for providing QoS routing in WSNs.

Random Re-Routing was therefore used to build on the underlying QoS provided by the Minimum Cost Forwarding Algorithm, the protocol found to share CPN’s QoS optimising characteristics, while having less overhead in the WSN.
context. By providing a method of load balancing packet flows over the spatial
distribution of the nodes, RRR is able to differentiate the priority of packets and
provide differential QoS. In Chapter 5 the version of RRR described in the liter-
ature was found to not sufficiently re-route low priority packets to prioritise the
high priority packets due to the nature of the wireless medium. To overcome this,
three mechanisms were introduced, which distribute traffic of differing priorities
over the spatial area of the network.

7.1.3 Quality of Information

In WSNs it is likely that the Quality of Information produced by the contextuali-
sation of data emerging from the network will be a high priority for users. Chapter
6’s main contribution to the area of WSN QoI was to highlight the overlap between
the attributes of QoI and the measurable network metrics of QoS, through a survey
of the literature, and experimentation. The chapter also proposed the use of the
PSNR as a measure of QoI for scenarios where radiation, distributed over an area,
is being measured by the WSN.

This knowledge was applied to use the RRR protocol to assign network re-
sources to task requirements, expressed in terms of the latency.

7.2 Future Developments

There is scope for further exploration of the herding mechanism in RRR. This is
a new concept in network routing, introduced in this work, which takes advantage
of the wireless medium and dense node placements envisioned for wireless sensor
networks. Future developments should extend the simulation work performed in
Chapters 5 and 6 with analytical work to define exactly when this mechanism
should be enabled to provide QoS. The networks used for the simulations in this work are large, and thus provide the “space” for packets to be re-routed. This need not always be the case, and the network should determine when it is appropriate and when not.

Chapter 4 demonstrated a router architecture suitable for use by single interface devices, such as WSN motes. An interesting future development would be an investigation of the viability of motes designed around FPGA devices. This would allow the well studied use of FPGAs as fast digital signal processors to be utilised. If the unit cost of FPGA devices is too high for motes, which may be lost after deployment, the FPGA based motes could be used for prototyping, before mass-producing a finished design.

7.3 Final Remark

While much prior work has addressed the issue of QoS routing in wireline networks, there is substantially less work in the field of wireless sensor networks. This work’s contributions to sensor network QoS routing for large dense mission critical WSN deployments and the overlapping concept of WSN QoI should therefore be useful going forward.
APPENDIX
A. SINGLE ROUTING ENGINE FPGA BASED CPN ROUTER

(ADDITIONAL DETAILS)

The block RAMs of the FPGA are used as FIFO buffers between the five stages of the design. The memories can be used in the configurations shown in Table A.1.

<table>
<thead>
<tr>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>16,000 x 1 bit</td>
</tr>
<tr>
<td>8,000 x 2 bits</td>
</tr>
<tr>
<td>4,000 x 4 bits</td>
</tr>
<tr>
<td>2,000 x 9 bits</td>
</tr>
<tr>
<td>1,000 x 18 bits</td>
</tr>
<tr>
<td>512 x 36 bits</td>
</tr>
</tbody>
</table>

*Tab. A.1: Virtex-II blockRAM configurations*

The 9, 18, and 36 bit configurations expose an extra bit per byte, intended for use as a simple checksum. To maximise the throughput, this design works with 32 bits of data per clock-cycle, and therefore the memories are used in the 36 bits per address location configuration. The first four “checksum” bits are used to indicate the start and end of packets in the buffers and the remaining 32 bits to store four bytes of the packet. The start of a packet is marked with the binary ‘code’ 0101, and the end of a packet using the four bits as a validity mask for the four bytes of the final word in memory. Everywhere else the first four bits are zero.
A.1 Router Processes

A CPN packet, like the majority of network packets, consists of a header followed by data. As can be seen in Figure 2.1, the header of a CPN packet has three main components: The header prefix, the route, and the cognitive map (CM).

The header prefix contains the CPN packet type (i.e. smart, dumb, etc.), the QoS class, the source and destination of the packet, and information necessary to make use of the route and cognitive map. The route is a list of all the hops from the source of the original dumb or smart packet to the destination. The contents of the CM depends on the QoS class of the packet. For example, when delay is the QoS metric the CM is filled with timestamps for each node on the route, while when path length is the metric, there is no CM as all necessary information is available from the route.

The process CPN block reads the packet from the CPN input buffer one word at a time. Once it has read the header prefix, it buffers the route into registers. The process of choosing the next hop then varies between the different packet types. The types which follow a predefined route (DP, ACK) extract the next hop from the route according to the route index, and then look up the MAC address of that neighbour from the ARP table. SPs read the previous hop from the route, and then proceed to read the aRNN for the relevant combination of QoS class and destination from memory (see Section A.2). If the aRNN is valid (has been accessed previously and initialised), the next hop is read from the aRNN. Otherwise, a neighbour (which is not the previous hop) is randomly chosen from the ARP table. This is then written to the aRNN, and the MAC address retrieved from the ARP table.

At this point there is enough information to start building the packet to be
sent to the output. The Ethernet header is written to the output, followed by a relevant CPN header prefix. The route is then written from the registers to the output. In the case of a smart packet the next hop is added to the route at the route index.

How the CM is handled depends on the QoS class and packet type. Only the delay and hop count classes have been implemented. The software implementations also include a packet loss QoS class, although this has not been used in any published work; and several hybrid/variations on the other classes. For the delay class the CM is copied from the input to the output. For smart and dumb packets the current timestamp is written to the CM at the position indicated by the route index. For acknowledgement packets the timestamp indicated by the route index is read and the round trip delay is calculated using the current timestamp of the router. This delay is then written back to the CM in the timestamp’s location. The hop count class has no CM and so it is skipped.

The time used in the timestamps is independently maintained by individual routers and need not be synchronised as the round trip travel time recorded from acknowledgement packets compares the time at the specific router. The unit of measurement must however be the same, as the destination router will remove loops from the route of smart packets it receives and adjusts the CM accordingly.

At this stage data is copied from the input to the output. Acknowledgement packets update the mailbox of the QoS class and destination with either the delay or the number of hops. If the packet is a smart acknowledgement the weights of the RNN are then updated, and the next hop for the next smart packet is chosen accordingly.
A.2 Memory Use

In order to read and write the aRNNs and MBs used in CPN, memory lookups must be made. In the arithmetically simple aRNN update algorithm these memory lookups dominate the clock cycles required to complete. It is therefore important to keep them to a minimum. The SRAM modules of the RC300 development board are used to store the aRNNs' weights and MBs' QoS values. There are four 8MB SRAM chips on the development board which each have a 21-bit address space and can be accessed in parallel. The aRNNs and MBs can therefore be stored in the manner shown in Tables A.2 and A.3. The average QoS value stored in the MB varies depending on the QoS metric of the class.

<table>
<thead>
<tr>
<th>Offset</th>
<th>SRAM0</th>
<th>SRAM1</th>
<th>SRAM2</th>
<th>SRAM3</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>valid</td>
<td>reward</td>
<td>threshold</td>
<td>decision</td>
</tr>
<tr>
<td>01</td>
<td>$Q_0$</td>
<td>$Q_1$</td>
<td>$Q_2$</td>
<td>$Q_3$</td>
</tr>
</tbody>
</table>

Tab. A.2: RNN in memory

<table>
<thead>
<tr>
<th>Offset</th>
<th>SRAM0</th>
<th>SRAM1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>valid</td>
<td>average QoS value</td>
</tr>
</tbody>
</table>

Tab. A.3: Mailbox in memory

<table>
<thead>
<tr>
<th>bits</th>
<th>20</th>
<th>19 - 18</th>
<th>17 - 10</th>
<th>9 - 2</th>
<th>1 - 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN</td>
<td>0</td>
<td>0</td>
<td>destination</td>
<td>QoS class</td>
<td>offset</td>
</tr>
<tr>
<td>MB</td>
<td>1</td>
<td>neuron</td>
<td>destination</td>
<td>QoS class</td>
<td>offset</td>
</tr>
</tbody>
</table>

Tab. A.4: RNN and Mailbox memory addresses

The SRAMs would be addressed as shown in Table A.4. In this configuration there is sufficient address space to allow the majority of combinations of RNN and MB currently in use to be stored with no collisions. This means that lookups
and writes will always take the same number of clock cycles as no searching need be performed. As can be inferred from these tables, this design is limited to 4 neighbours. Furthermore, currently only the last byte of the 32-bit CPN address format varies between the nodes of the CPN test-bed. Should this change the addressing would need to be adapted. Although there are currently only a small number of QoS classes defined (most being variations on delay, number of hops, and link loss rates) the CPN packet header has space for $2^8$ classes. CPN addresses also take the same 32-bit form as IPv4 addresses. In the software testbed, only the last 8 bits are used, so these are used in the memory lookup in this hardware implementation.

This approach allows one C-class network (255 nodes) to be stored with no searching and would not be sufficient for any large scale deployment of CPN. Section 4.6 at the conclusion of Chapter 4 discussed some possible solutions to this problem, and how CPN addresses would fit within the existing Internet architecture.
This section examines how locality can be used to minimise the redundancy of information transmission in the network, thereby further reducing power consumption.

When many motes are randomly distributed over a large geographical area, there is a great potential for data redundancy. Many sensors, clustered within a small area, may be expected to see correlations in their data. For example, closely spaced motes may be monitoring:

- Landslides or avalanches on a mountain side.
- Seismic activity on a volcano or tectonic boundary.
- Meteorological or environmental variables such as light and temperature in both open and closed spaces.

It can therefore be beneficial if some form of data aggregation, or averaging, is done in the network before messages are sent to the data sink, and that this is enabled by the routing protocol. A method for data clustering, based on the locality of motes, is proposed here.

The findings of Section 5.3 can be used to make a routing protocol power aware, but can also be used to determine an approximated knowledge of the relative distance between motes. Two motes which can communicate on the minimum power level or have a high RSSI are likely to be closely placed, with few or no obstacles obstructing radio propagation. They are therefore also likely to be sensing
Combining data from closely placed motes has the effect of reducing the number of messages, and therefore results in further power savings, as is shown in the simple Theorem 1 below.

**Theorem 1.** Assuming no loss, if the energy $m$ required to communicate between motes in close proximity is less than the energy $K$ needed for one of these motes to send a packet to the sink (see Figure B.1), sending a packet between $n$ motes before the sink requires less energy than sending $n$ packets directly.

**Proof.** The energy needed to send and receive a packet between motes within the data cluster is $m$. The energy required for any of the motes within the data cluster to send a packet to the destination (possibly via multiple hops) is constant, with value $K$.

The total energy required to send $n$ packets without aggregation is $nK$.

The total energy required to send one packet to each mote within the data cluster before sending it to the destination is $(n - 1)m + K$.

The inequality $(n - 1)m + K < nK$ reduces to $m < K$, and the theorem follows. 

A protocol implementation can enable local data clustering by providing the application layer with a list of the $n$ neighbours with the greatest RSSI above a
certain threshold (within the minimum power level’s range). When the application layer generates a data packet, it can mark the packet as “within cluster”, examine the list of local motes and forward the packet to one of these. This neighbour can then intercept the packet from the routing layer and add its own data. This process can be repeated until a maximum cluster size is met, or there are no more new neighbours within the cluster. At this stage the packet can be marked as having left the data cluster and forwarded to its ultimate destination.

This is currently not applied in the implementations described in Chapter 5, but was used by the author in [4].
C. WIRELESS SENSOR NETWORK TESTBED WITH
CONTROLLABLE SENSORY INPUT

C.1 Testbed Architecture

Twenty of the motes in the WSN testbed have controllable sensory inputs, which can be scripted from a central computer. The control circuitry is based around four PIC16F877A microcontrollers [120] running at a clock frequency of 3.6864 MHz. The board, the circuitry of which is shown in Figure C.2, is connected to the central computer via an RS232 serial interface, which communicates with the PICs through a ADM202E RS232 line driver/receiver device.

Each of the four microcontrollers has 20 output pins which correspond to an LED on each of the 20 motes’ boxes. Each box has a circuit board (Figure C.1) with 4 LEDs, connected in series with their transistors in a grounded emitter configuration. Each LED is also in series with a resistor, which controls the current through and therefore brightness of the LED. Resistor values have been chosen so that each LED produces a different brightness of light, and therefore in theory 16 unique light levels can be directed at the motes’ light sensors.

A Java class, or bash script running on the central computer sends single bytes

<table>
<thead>
<tr>
<th>LED#</th>
<th>mote#</th>
<th>on/off</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6</td>
<td>5.1</td>
<td>0</td>
</tr>
</tbody>
</table>

Tab. C.1: LED control byte
C. Wireless Sensor Network Testbed with Controllable Sensory Input

Fig. C.1: Per-mote emitter follower LED circuitry

(Table C.1) over the serial interface. The microcontroller corresponding to the LED# receives this byte (all others filter it), and turns the output pin corresponding to the mote# either on or off. The pin is connected to the relevant transistor’s base, and therefore controls whether the LED is lit or not.

C.2 Building Evacuation Simulator Wireless Sensor Network Integration

An important part of an intelligent building environment is its ability to monitor for the presence of any hazards. Rudimentary versions of these abilities are familiar from domestic smoke detectors, or the more advanced forms present in office buildings, where the location of the source detector is reported to some central location. Future buildings will however be monitored by distributed WSNs, capable of processing the data returned by their sensors, communicating independently of the building infrastructure, and playing an integral role in evacuation procedures in the presence of hazards.

Imagine a building in which, as is currently the case in commercial airliners, low-
Fig. C.2: Custom control circuitry for the serial-port sensory-input LED infrastructure
level lighting directs the occupants to the nearest exit when there is a fire or some other hazard. Furthermore, such a system could direct building occupants to the nearest exit along a safe evacuation path considering the location and development of the hazard (be it fire or gas leak or similar), and also the human congestion in the building. A WSN would enable such a system by continually monitoring the changing environment, and processing its data (in a distributed manner if necessary) to determine safe current and future evacuation routes.

In order to aid the researching and development of such systems, the WSN testbed has been tightly integrated into the BES. In addition to their wireless capabilities, the networks’ devices are connected to the wired network, which allows them to be programmed and also facilitates two-way communication between the devices and the simulator.

Each sensor device has an agent presence within the simulator. These agents relay the intensity of the hazards in the location of the sensor to real sensor, which can then “sample” these values (before performing in-network processing and potentially communicating them to a common data-sink). The data-sink also has a simulator presence, in that each floor of the building receives updates on the state of the hazard via the network.

The agents also enable other integrations between the real sensors and the BES, including mobility, and allow the impact of network properties (such as packet delays and losses) on the evacuation success to be studied. The author contributed this work on the WSN integrated in [5].
D. RANDOM RE-ROUTING IN THE WIRELESS CONTEXT

The following is the algorithm of the modified RRR described in Chapter 5.

1: {Each node maintains two sets of rolling registers, indexed by $i$ and $j$, with
details of overheard packet transmissions. Packets with priority $P > 0.0$ have the
time $P_t$, their priority $P_p$, and their associated mission $P_M$ stored. Packets with
inertia $I > 0$ have the time $I_t$, their priority $I_p$, their inertia $I_i$, and their angle
of travel $I_\alpha$ stored. After storing overheard details, the index is incremented.
In the implementations of this chapter the information of the 6 most recent
packets is stored.}

2: {When a packet $Q$ with priority $Q_p$, inertia $Q_i$, angle $Q_\alpha$, and mission $Q_M$ and
is to be forwarded at time $NOW$...}

3: $DIVERSION \leftarrow FALSE$
4: $INERTIA \leftarrow FALSE$
5: $HERDING \leftarrow FALSE$
6: $PRIO \leftarrow 0$
7: $INERT \leftarrow 0$
8: $ANGLE \leftarrow 0$

9: \textbf{if} $NOW - P_t(i) \leq T$ \textbf{then}

10: \quad \{Is the mean overheard priority greater than this packets priority?\}

11: \quad $TOT \leftarrow 0$
12: \quad $DIV \leftarrow 0$
13: \quad \textbf{for} $n = 0$ to $5$ \textbf{do}
14: \quad \quad \textbf{if} $Q_M \neq P_M(n)$ \textbf{then}
15: \quad \quad \quad $TOT \leftarrow TOT + P_p(n)$
16: \quad \quad \quad $DIV \leftarrow DIV + 1$
17: \quad \quad \end{if}
18: \quad \textbf{end for}
19: $PRIO \leftarrow TOT/\text{DIV}$
20: if $Q_\rho < PRIO$ then
21: $DIVERSION \leftarrow \text{TRUE}$
22: end if
23: end if
24: if $Q_i > 0$ then
25: $\text{INERTIA} \leftarrow \text{TRUE}$
26: else
27: {Have 6 packets with inertia been overheard in the last $T$ seconds?}
28: if $NOW - I_t(i) \leq T$ then
29: {What is the mean overheard inertia of packets with priority less than or equal to this packet?}
30: $DIV \leftarrow 0$
31: for $n = 0$ to 5 do
32: if $Q_\rho \geq I_\rho(n)$ then
33: $INERT \leftarrow INERT + I_t(n)$
34: $ANGLE \leftarrow ANGLE + I_\alpha(n)$
35: $DIV \leftarrow DIV + 1$
36: end if
37: end for
38: $INERT \leftarrow INERT/\text{DIV}$
39: $ANGLE \leftarrow ANGLE/\text{DIV}$
40: if $DIV \neq 0$ && $INERT \neq 0$ then
41: $\text{HERDING} \leftarrow \text{TRUE}$
42: end if
43: end if
44: end if
45: if $DIVERSION$ then
46: if $Q_\rho \geq RAND$ then
47: {Set $Q_i$ equal to a value proportional to $PRIO - Q_\rho$}
48: {Divert packet $Q$}
49: else
50: {Forward packet $Q$ to the best neighbour}
51: end if
52: else if \texttt{INERTIA} then
53: \hspace{1em} \texttt{\(Q_i \leftarrow Q_i - 1\)}
54: \hspace{1em} \{Forward packet \(Q\) to neighbour at within angular range of \(Q_\alpha\)\}
55: else if \texttt{HERDING} then
56: \hspace{1em} \texttt{\(Q_i \leftarrow \texttt{INERT} - 1\)}
57: \hspace{1em} \{Forward packet \(Q\) to neighbour at within angular range of \texttt{ANGLE}\}
58: else
59: \hspace{1em} \{Forward packet \(Q\) to the best neighbour\}
60: end if
BIBLIOGRAPHY


[18] A. Farrel, A. Ayyangar, and JP. Vasseur. Inter-domain mpls and gmpls traffic engineering – resource reservation protocol-traffic engineering (rsvp-te)


