Implementation of SAMPLE Protocol

by

Kevin Nash

Dissertation

Presented to the

University of Dublin, Trinity College

in partial fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2005

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Kevin Nash

September 12, 2005

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Kevin Nash

September 12, 2005

Acknowledgments

I would like to thank my supervisor Jim Dowling for his patience and providing a very interesting and challenging project.

I would like to thank Andronikos Nedos, Barbara Hughes, Kulpreet Singh and Stefan Weber for their many answers and not being tempted to ban NDS students from the Llyod institute.

Thanks to my friends and family for their great help and support through out the year.

Finally I would like to thank the NDS class for a very memorable year.

Kevin Nash

University of Dublin, Trinity College
September 2005

Implementation of SAMPLE Protocol

Kevin Nash
University of Dublin, Trinity College, 2005

Supervisor: Jim Dowling

Wireless networks are pervasive in our society. With an increased interest shown by the general public in wireless technologies, protocols and hardware are being actively developed by academic and industrial groups alike. Moblie ad hoc networks (MANET) are an important subset of wireless networks and have particular routing needs which are not supported by 802.11, the most widely deployed wireless protocol. They must operate in the presence of interference, contention and cope with challenges such as limited energy resources mobility and lack of global state.

Numerous ad hoc protocols have been developed to support communication in MANETs but take a discrete approach. To consider a link as discrete does not reflect the reality of congestion and signal degradation. SAMPLE approaches these problems using statistical analysis, reinforcement and collaborative learning. Network events are observed over period of time and allow SAMPLE to determine the quality and stability of routes. This differs from a discrete approach which can trigger unnecessary and inefficient route updates while consuming bandwidth. SAMPLE uses promiscuous

listening to integrate data from overheard traffic to enhance its current view of the network.

This dissertation presents an implementation of the SAMPLE protocol on the Linux 2.4 operating system. Increased portability and maintainability are obtained by integrating the SAMPLE protocol with the more stable netfilter architecture rather than the kernel core.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter gives an overview of the different technologies involved in wireless communication. The rest of this thesis will be structured as follows:

- *Chapter 2* introduces MANET protocols, using three different protocols as sample of the current state of the art. These are Zone Routing Protcol (ZRP), Dynamic Source Routing (DSR), Ad Hoc On-Demand Distance Vector Routing (AODV). It finishes by walking through the Linux protocol stack.

- *Chapter 3* describes in more detail the SAMPLE routing protocol giving the reasons why it offers better performance in networks where signal degradation, interference and congestion are problems. It concludes by giving an example of how a node would join and integrate itself into a SAMPLE network.

- *Chapter 4* presents the implementation of SAMPLE, describing the reasons behind the choice of architecture and highlighting some of the features in the design which are not directly supported by Linux such as forwarding of packets.

- *Chapter 5* describes the results of experiments using VMware to create a virtual network and concludes by evaluating the design.

- *Chapter 6* concludes the thesis and presents possible future work.

1.1 Wirless Networks

Wireless networks use radio waves to transfer data without using wires. Radio waves are created when electrically charged particles accelerate with a frequency that lies in the radio frequency portion of the electromagnetic spectrum. In 1970, Norm Abramson et al. developed a radio-based communication system know as ALOHANET[1]. This was the first wireless packet switch network and was the result of an experiment of a group of researchers in Hawaii trying to find the most effective means to share data between four university sites on 4 separate Islands. Since then wireless technologies have evolved and can be seen in many different forms today. Some of these include [2][3],

- Wi-Fi used generically when referring to any 802.11 network, 802.11a,802.11b or 802.11g. Operating in both the 2.4 and 5 GHz band with a theoretical data transmission rate of 54Mb/s.
- GPS receivers communicate with a series of satellites to provide highly accurate location and time references.
- Bluetooth is an industrial specification(802.15.1) for wireless personal area networks (PANs). The communication range, link speed, and transmit power level for Bluetooth were chosen to support low-cost power efficient single chip implementations. Bluetooth also operates in the Industrial, Scientific and Medical Band (ISM) 2.4GHz with a data rate of 1Mbp/s.
- SensorNets are based on the 802.15.4 IEEE specification and operate both in the ISM band and 915Mhz with data transfer rates of 20 to 250Kb/s.

1.1.1 Operating Modes

As described in [4], As well as the existence of many different types of wireless networks there a number of ways these networks can be be organised. The two main operating modes for wireless networks are:

1. First, an example of infrastructure mode is an 802.11 networking framework in which devices communicate with each other by first going through an Access Point (AP). In infrastructure mode, wireless devices can communicate with each other or can communicate with a wired network. When one AP is connected to wired network and a set of wireless stations it is referred to as a Basic Service Set (BSS). An Extended Service Set (ESS) is a set of two or more BSSs that form a single subnetwork. Most corporate wireless LANs operate in infrastructure mode because they require access to the wired LAN in order to use services such as file servers or printers.

2. Aternatively wireless networks can operate in Ad hoc mode. These are networks with no pre-existing infrastructure. There are distributed because nodes which operate in ad hoc mode tend to be highly mobile and therefore it is necessary to ensure the communication does not collapse when one of the nodes powers down or is moved out of range. In distributed networks all nodes have to operate in the same frequency band since there is no special node to translate from one frequecy to another. An example of an ad hoc network is Wireless Ad Hoc Network for Dublin (WAND) which is an 802.11 networking framework in which devices or stations communicate directly with each other, without the use of an access point (AP). Ad-hoc mode is also referred to as peer-to-peer mode or an Independent Basic Service Set (IBSS).

1.2 MANETS

A collection of nodes which operate in distributed mode are called Mobile Ad hoc Networks (MANETS). SAMPLE is a protocol which has been designed to enable nodes to communicate in MANETs because existing wireless protocol do not directly support ad hoc communication.

A communication protocol is a sets of rules which define the way blocks of data, which are called Protocol Data Units (PDU) can be sent though a network. Each PDU consists of a packet header and a packet payload. A protocol specification defines the operation of a protocol and should consider three areas:

1. Definition of the Protocol Control Information (PCI) which forms part of the

PDU header

2. The procedures for transmitting and receive PDU's.

3. Definition of a the set of services provided by the protocol layers.

Chapter 2 describes in more detail the properties of MANET protocols.

As described in [5] MANETS are a collection of potentially highly mobile nodes which can take various forms for example:

- Rovers exploring Mars.
- A group of sensor nets monitoring the habitats of wildlife [6].
- Laptops communicating at a conference.

These networks may operate in isolation or have gateways to fixed networks. The later scenario is envisaged for SAMPLE in [7]. Here a core group of nodes with low mobility and high quality links create stable paths for another set of highly mobile nodes which wish to communicate with Internet gateways. MANETSs have several characteristics which differentiate them from wired networks

1.2.1 Dynamic Topologies

Nodes are free to move at will and therefore the network topologies are constantly evolving in ways which are impossible to predict in advance. This may result in both bidirectional and unidirectional links. MANETs therefore have to keep the amount of prior configuration to a minimum. As an exception to this are each node should have a unique IP address and all nodes should operate on the same radio band.

1.2.2 Bandwidth constrained

Due to the effects of contention, degradation of signal, interference the realised throughput of network links is often much less then the theoretical maximum rate. This is in addition to the fact the wireless networks have significantly lower capacity to begin

with compared to wired networks due limitations on the size, power and range of their transmitters. The resources of MANETs will continue to be strained as end users will expect the same level of service when using wireless networks as compared to wired networks.

1.2.3 Energy Constraints

A significant percentage of nodes in MANETs rely on battery power. If some cases such as in the deployment of certain sensor networks the life of the node is equal to the life of its energy reserves. An example for this are sensors placed internally in the structure of bridges to monitor the curing of concrete. In this case it is impossible to retrieve the sensors and if it was they might be too numerous to make it practical. Therefore energy constraints can have a huge impact on the behaviour of MANETs.

1.2.4 Limited security

Mobile networks are more prone to physical attack. In particular networks which are deployed in hostile territory:

- may be under treat from physical tampering if the network is a military application.
- or these threats might environmental, for example extreme temperatures in the case of a sensor net been deployed in a burning building by a disaster management team.

In addition the systems which have zero prior configuration, a distributed structure and in some cases limited processing power may have difficulties using existing security techniques used in wired networks.

1.3 802.11

Each MANET protocol must integrate with a Medium Access Control[4] (MAC) protocol at some level. Since one of the steps to fully evaluate SAMPLE will be to compare its

performance to AODV on WAND which uses 802.11 this section will give an overview of the 802.11 MAC protocol. The 802.11 standard specifies a common medium access control (MAC) Layer, which provides a variety of functions that support the operation of 802.11 based wireless LANs. The 802.11 MAC Layer uses an 802.11 Physical Layer, such as 802.11b, 802.11a or 802.11g, to perform the tasks of transmission, carrier sensing, and receiving of 802.11 frames. The main job of the MAC protocol is to regulate the usage of the medium, and this is done through a channel access mechanism.

The 802.11 family used Carries Sense Multiple Access/Collision Avoidance) CSMA/CA. It is a distributed protocol in that there is no need for any controlling node. If a node wishes to send a packet using 802.11 is performs the following steps:

- listen on a desired channel
- if the channel is idle send the packet
- otherwise if the channel is busy, wait until the transmission stops plus an additional contention period. This contention period is a defined as a random interval determined to allow all nodes to have statistically equal access to the media. The random back off period significantly reduces the number of collisions and corresponding retransmissions, especially when the number of active users increases.
- If the channel is idle at the end of the contention period the node can transmit its packet other wise it repeats the random back off above.

Since most nodes cannot listen for collisions while sending data, because the transmitter can't have its receiver turned on while transmitting the frame, there is a need for the receiving station to send an acknowledge (ACK) if it detects no errors in the received frame. If the sender does not receive an ACK it will assume the packet was corrupted or lost and will attempt to retransmit the packet. It should be noted that only unicast packets above a certain size are acknowledged.

This is the physical interpretation of the notation of link cost described in Chapter 3. The fact that it may take a number of transmissions , up to a maximum of seven, before a 802.11 considers a packet has been successfully sent or not, is considered by SAMPLE. SAMPLE uses historical data to estimate the cost of a link.

802.11 also has the option to uses Request To Send (RTS) and Clear To Send (CTS) to avoid the hidden node problem. This occurs when two nodes (**A**,**C**) see Fig. 1.1

which are out of wireless range attempt to send to an intermediate node (**B**).Both **A** and **C** consider the network channel idle and transmit simultaneously causing Node **B** to receive corrupted data. The solution to this is to use a four-way handshake: RTS-CTS-DATA-ACK where the RTS and CTS packets are significantly smaller than the average data packet. **A** would first send a RTS packet and would not transmit until a CTS packet has been received from **B**. Likewise **C** holds off from transmitting until its receives a CTS from **B**.

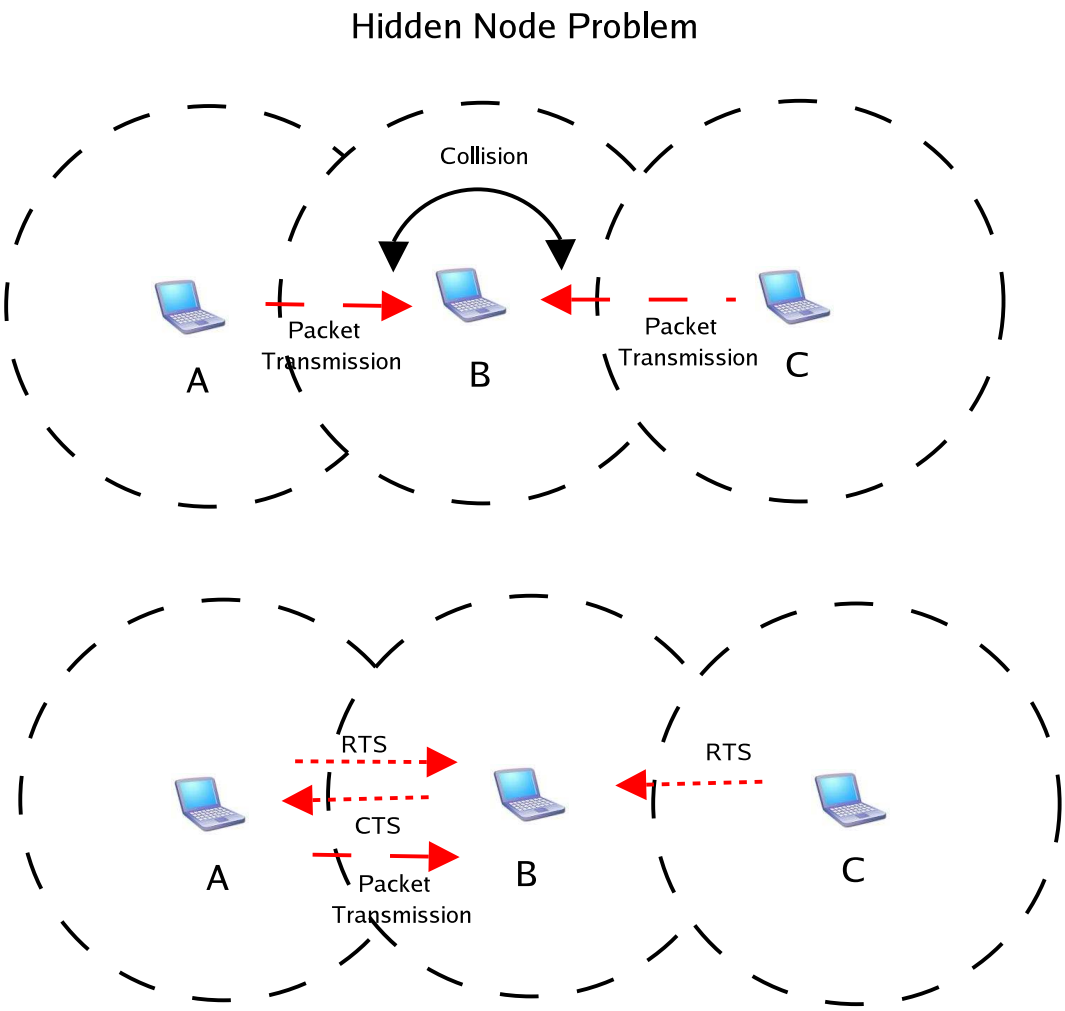

Figure 1.1: *802.11 Hidden node problem*

802.11 also supports fragmentation which enables a node to divide data packets into smaller frames. This is done to avoid needing to retransmit large frames in the presence of interference. The bits errors resulting from interference are likely to affect a single frame, and it requires less overhead to retransmit a smaller frame rather than a larger one.

1.4 Linux

SAMPLE is implemented on the Linux 2.4 operating system. The WAND network also runs on Linux 2.4 which provides an existing MANET for the evaluation of SAMPLE. Linux was originally developed as a hobby by Linus Torvalds while attending the University of Helsinki. The first version was released in September 1991 and grow out of all portion since then. It has four major parts:

- Kernel, a low level operating system.
- Shell, a user interface for executing commands.
- X, a graphical system which provide a GUI
- Thousands of userspace programs for file editing, audio, mathematics etc.

It is one of the most prominent examples of free software and of open-source development, unlike other major 'closed-source' operating systems (such as Windows or Mac OS) and programs, all of its underlying source code is available to the public and anyone can freely use, modify, and redistribute it.

1.5 Netfilter

Netfilter [8] and iptables are building blocks of a framework inside the Linux 2.4.x and 2.6.x kernel. This framework enables, network address translation packet filtering, and other packet mangling. Netfilter is a set of hooks 1.2 inside the Linux kernel that allows kernel modules,such as the SAMPLE kernel module, to register callback functions with the network stack. A registered callback function is then called back for every packet

that traverses the respective hook within the network stack. Netfilter has five hooks in the kernel architecture.

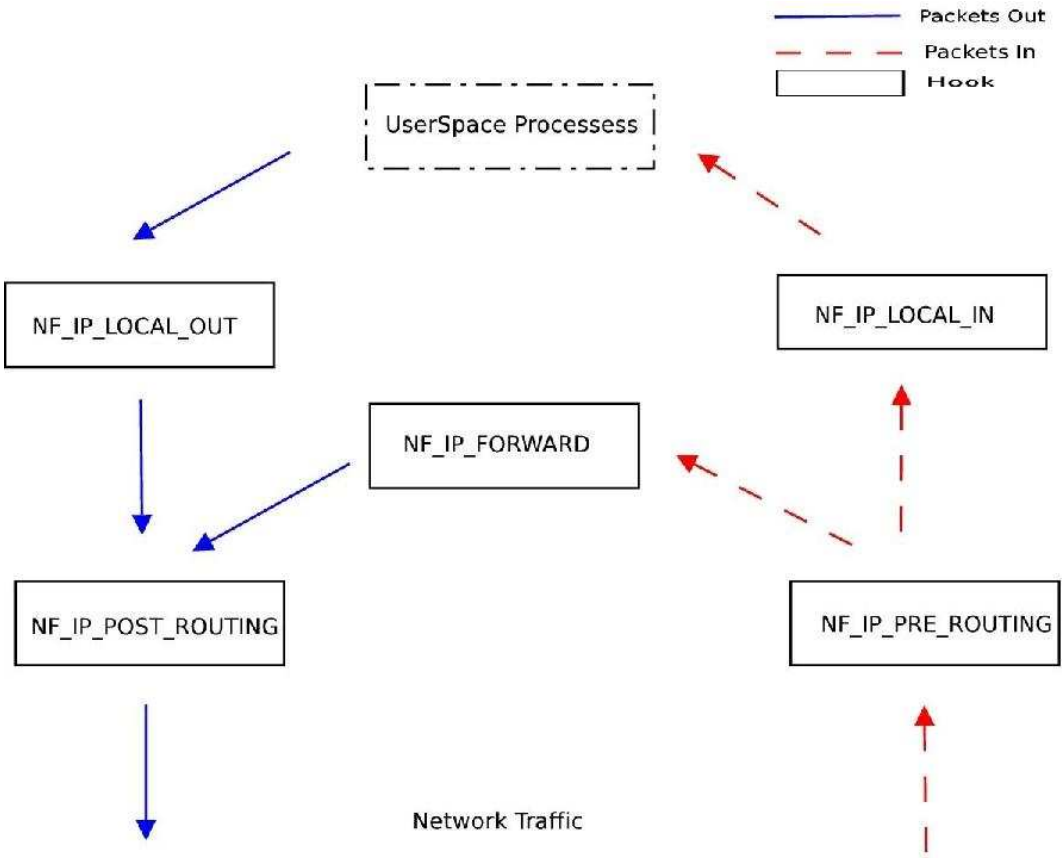

Figure 1.2: *Netfilter Hooks*

1.6 Collaborative Learning

Finally a brief word on collaborative learning in SAMPLE, see reference [9] for more details. SAMPLE uses collaborative learning in order to solve a problem,locating routes, with no prior solution. SAMPLE interacts with an environment, the network, through trial and error. Positive feedback occurs when routes of high quality are advertised, which causes more nodes to use this route and will in turn advertise it.

This causes a group of routing agents to converge on optimal routing decisions. On the other hand negative feedback is provided by congestion and the notion that routing information decays overtime. This allows SAMPLE to be operative in a dynamic environment and respond to unpredictable changes.

Chapter 2

Ad hoc Protocols

This chapter opens with a description of the properties found in MANET protocols. Following a brief overview is given of three protocols which have been studied during the SAMPLE[7] implementation.

- Dynamic Source Routing (DSR)[10], a reactive protocol
- Zone Routing Protocol (ZRP) [11] , a hybrid protocol whose architecture most closely resembles that of SAMPLE
- Ad Hoc On-Demand Distance Vector Routing (AODV) [12] , an improvement on the Highly Dynamic Destination-Sequenced Distance-vector Routing (DSDV) [13] protocol.AODV has been installed on the Wireless Ad Hoc Network for Dublin (WAND).

The chapter is closed by tracing the path of a packet through the Linux kernel.

2.1 Routing

Routing consists of moving information through a network of interconnected nodes. This is usually driven by routing tables which provide the next hop for the packet. In wired networks these routing tables are generally reasonably static where as in MANETs protocols update existing tables or create their own dynamically. Routing in all networks,wired or wireless involves two main components:

- Determining the optimal routing paths in a network. This is not a trivial task.
- Transporting information along these paths

Routing protocols in general should provide mechanisms to ensure packets have been successfully delivered and notification of failures. Protocols should adapt to network changes, for example if a network link goes down alternate routes should be located.

2.2 Properties of MANET protocols

MANET protocols have to provide similar functionality to wired protocols but in addition they must have certain properties to deal with the particular challenges of MANETs described in section 1.2. The following is a brief overview of these properties.

2.2.1 Distributed

The protocol should be distributed as MANETs are distributed by their vary nature.

2.2.2 Loop-freedom

A number of techniques can be used to stop packets from hopping continuously around a network. A TTL field can be decremented until zero is reached at which time it can be dropped. Sequence numbers can be used to detect stale packets. Where packets with sequence numbers less than the current value of the node are dropped.

2.2.3 On Demand

On demand protocols are reactive, that is they do not initiate route discovery until it is required. This allows the protocol to adapt to traffic patterns on a need basis. Proactive route discovery causes an initial delay on route discovery but when implemented intelligently can allow bandwidth and energy to be conserved.

2.2.4 Proactive

Proactive protocols may be used if the delay in, *On demand* based route discovery is unacceptable. Protocols which maintain a route table for all destinations in the network have the advantage that communication with arbitrary destinations have minimal delay in sending packets. Such protocols have the disadvantage of having to *maintain* routing entries before they are needed and therefore require additional traffic to update stale routing entries. Due to the fact that by their nature nodes in MANETs are mobile, links can be broken frequently. Repairing such links can be costly, consuming scare bandwidth and causing congestion at intermediate network points as nodes occupy valuable queue space. A compromise between proactive and reactive protocols could be to maintain several routes between source and destination though not necessarily all routes. This approach would require a mechanism to purge stale routes otherwise applications might experience even longer delays as each route in tried in turn.

2.2.5 Hybrid

Hybrid protocols delay the trade off between proactive and reactive protocols from design to run time.This allows the protocol to observe current network characteristics and then decide on the optimal routing policy.

2.2.6 Unidirectional link support

Some routing protocols make the assumption of the existence of symmetric communication links although unidirectional links are more general in radio networks. Such an assumption might be justified in cases where it is considered that unidirectional links may be on the verge of failure and therefore causing less stable routes to be discovered. This places an extra strain on network resources as route discovery is initiated due to early failure of less robust links.

2.2.7 Sleep

In order to conserve energy supplies nodes may adopt a sleeping strategy. Protocols should be able incorporate these sleep periods without unfavourable consequences.

2.2.8 Bootstrapping

When a network is first created or a node's routing information has gone stale the protocol must have the ability to operate correctly and discover new routing information. This can be achieved by broadcasting an initial route requests.

2.3 DSR- Dynamic source routing

DSR is a reactive source routing protocol which is able to manage routing in a MANET without using periodic table-update messages. By placing a list of network addresses in each packet it reduces the need for forwarding nodes to have routing tables which need to be kept up to date. This helps to reduce the amount of network traffic.

The DSR protocol is composed of two components which work together to allow routing in a MANET. The first of these is the path finding process.

2.3.1 Route discovery

Route discovery is initiated only when the source node wishes to send a packet to the destination node and doesn't already know a route to the destination.

In Fig 2.1 **A** wishes to send a packet to **D**. If **A** already has a suitable source route in its cache it places the source route in the header of the packet. A source route is a sequence of hops a packet should follow from the source to destination node. However if **A** has no source route to **D** it initiates a route request. A route request is broadcast to all nodes in wireless range and contains the

- identity of the initiator **A**
- identity of the target **D**

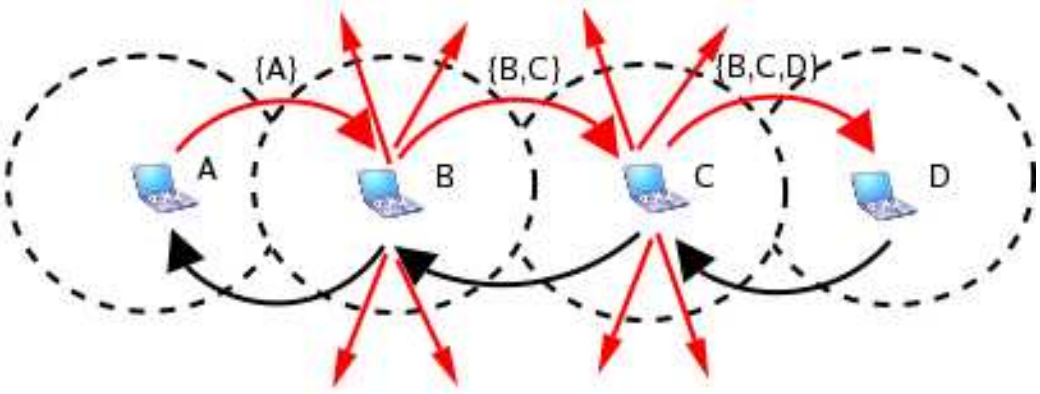

Figure 2.1: *Route Discovery in DSR Networks*

- unique request ID
- route record

The route record is a list of all the intermediate nodes through which this copy of the route request message has been through.

When **B** receives the Route Request messages it checks to see if is the target of the route discovery. In this case **B** returns a route reply message to the route discovery initiator. If node B has recently seen another Route Request from node A or the address of node B is already in the route record node B discards the request. Otherwise node B appends it address to the route record and broadcasts this message in turn. When node A receives a route reply, it caches this route for use in sending subsequent packets to this destination.

2.3.2 Route maintenance

When a node forwards a packet it is responsible for confirming that the packet has been received by the next hop along the source route. A packet can be retransmitted up to a specified maximum number of attempts. In Fig 2.2 **B** would be responsible for receipt at **C** and likewise **C** is responsible for receipt at **D**.

Confirmation of packet reception can be achieved in a number ways.

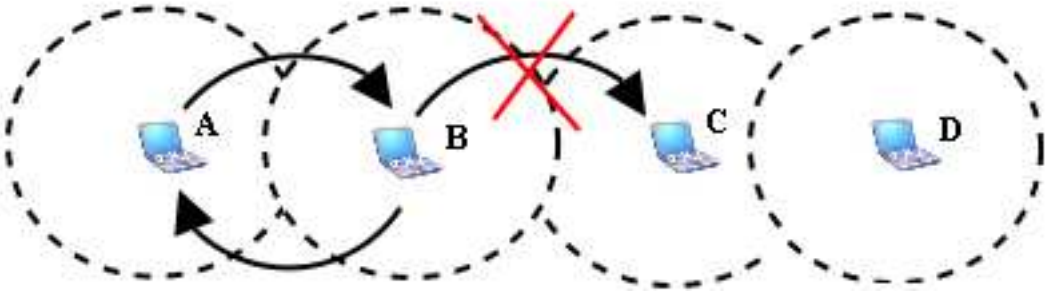

Figure 2.2: *Route maintenance in DSR Networks*

- The acknowledgement may come as part of the MAC protocol in use where in wireless networks neighbouring nodes acknowledge the receipt of packets from a transmitter.
- Passive acknowledge may occur when B overhears C forwarding the packet to D.
- Alternatively the forwarding node can set a bit in the header of the packet specifically requesting acknowledgement.

In Fig 2.2 if a packet is transmitted by **B** the maximum number of times, a route error is returned to **A** stating the link from **B** to **C** has been broken. A then removes this route from its route cache. It can attempt to retransmit the packet if it has another suitable source route in its cache due to either additional responses to its initial route request or from overheard traffic. If no other route is available a route request packet is broadcast.

DSR has the advantage of not creating additional network traffic while updating routing tables or by maintaining a list of neighbouring nodes. DSR is capable of routing correctly over networks that have unidirectional links since a route reply need not be sent over the reverse path of a route request. Likewise acknowledgements may follow a different multihop path to the previous node in the source.

A disadvantage of this protocol is that as the source route of the packet increases so does the size of the actual packet. Problems occur in MANETs which contain a large number of nodes. In a network of 100 nodes simulations have shown that DSR [14] can deliver 5-50% fewer packets compared to AODV.

2.4 ZRP - Zone Routing Protocol

While DSR is a purely reactive protocol, ZRP combines the advantages of both reactive and proactive protocols deciding which is the optimal solution depending on the circumstances. ZRP provides support for unidirectional links and actively maintains a list of neighbours so broken links can be tracked. This section introduces how packets are forwarded in ZRP and where proactive and reactive decisions are made.

2.4.1 IARP - Intrazone Routing Protocol (IARP)

IARP [15] is a proactive table driven protocol. Routing zones are constructed using the *Neighbour Discovery Protocol (NDP)*. A node maintains its list of neighbours using Hello beacons. Neighbours may be selected based on strength of signal or frequency. If no beacon was received from neighbour during a defined period the neighbour is considered lost. Conversely if a beacon is received from a neighbour that was previously unrecorded it is considered found. The *IARP* is notified of any change in status. Although *IARP* provides very efficient routing within a zone, in order to conserve bandwidth this proactive approach is not used for routing between different zones.

2.4.2 IERP - InterZone Routing Protocol (IERP)

IERP [16] in contrast to *IARP* is reactive, that is it is only used on request. If a node encounters a packet whose destination is outside of its own zone, it forwards this packet to a peripheral node instead of flooding its own zone. Peripheral nodes maintain routing information about neighbouring zones. Thus a decision on where to forward a packet can be made using a *Bordercast Resolution Protocol (BRP)* [17] . *BRP* is used to direct requests made by the global reactive *IERP*. It maximises efficiency by removing redundant queries. *BRP* keeps track of which nodes a query has been delivered to and so it can prune the bordercast tree of nodes that already have received a query. When a node A receives a query packet from a node not in its local zone it constructs a bordercast tree and forwards this to it neighbours. Its neighbours will then reconstruct the bordercast tree and check whether or not it belongs to the bordercast tree of the sending node. If it doesn't it will determine whether the destination node lies within its

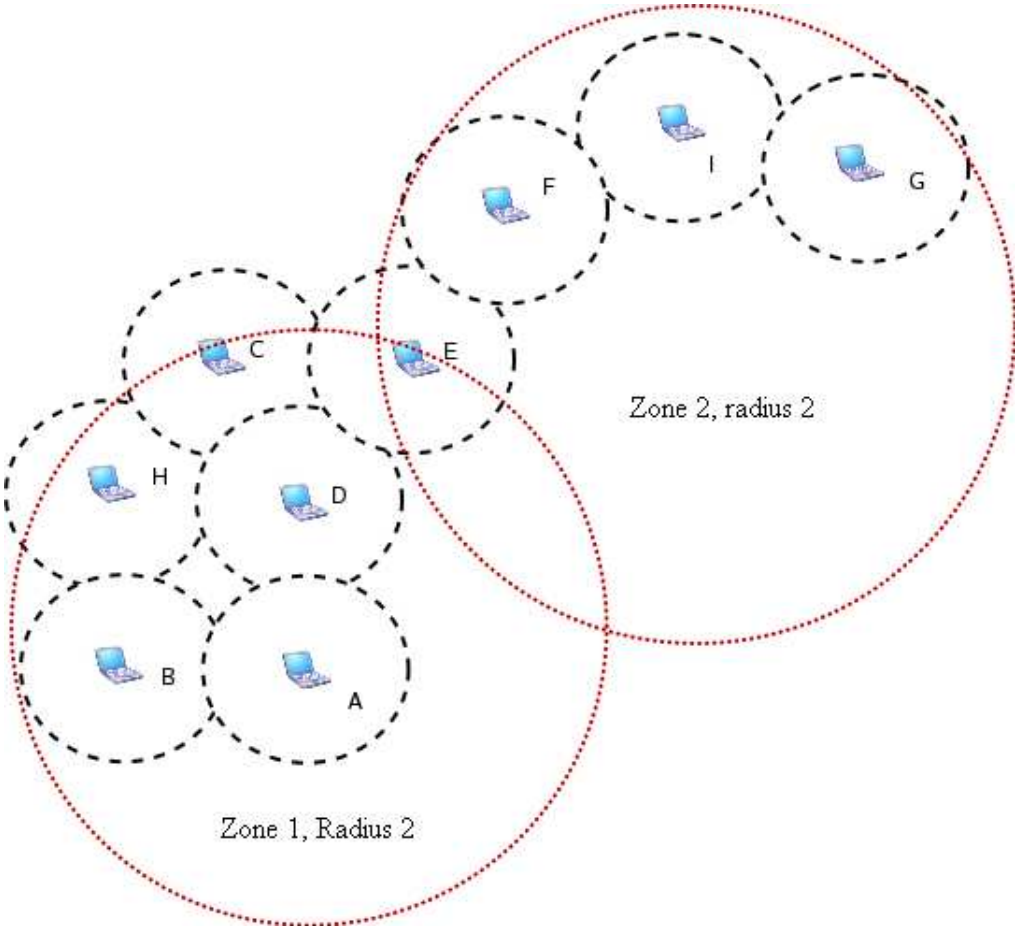

Figure 2.3: *Two ZRP zones each with a radius of 2 hops. H,C and E act as peripheral nodes*

zone and which neighbours the packet has been sent to. Two levels of query detection exist:

- QD1: As nodes relay queries, the note which zones have already been queried
- QD2: Zones which have a single broadcast and can overhear queries been sent.

Fig 2.3 gives an example of *IERP*. **A** wished to send a packet to **G** and therefore constructs a bordercast tree spanning the peripheral nodes **H**,**C** and **E**. It forwards this to its neighbours **B** and **D** who verify that node **G** is not within their zone and so construct another bordercast tree. **B** and **D** only include nodes which have node

been covered previously, which means **B** will node not include node **D**. **D** forwards the request to **E** who realises **G** is within its zone and therefore replies with the correct route.

As well as detecting whether or not nodes have been covered they must drop packets which would be forwarded to nodes which have already been covered. This is carried out through *Early Termination*[18] and *Loopback Termination*[18], in order to avoid queries been sent back to their originating node. In addition *BRP* uses selective Bordercasting, see Fig 2.4 to further reduce redundant bordercasting.

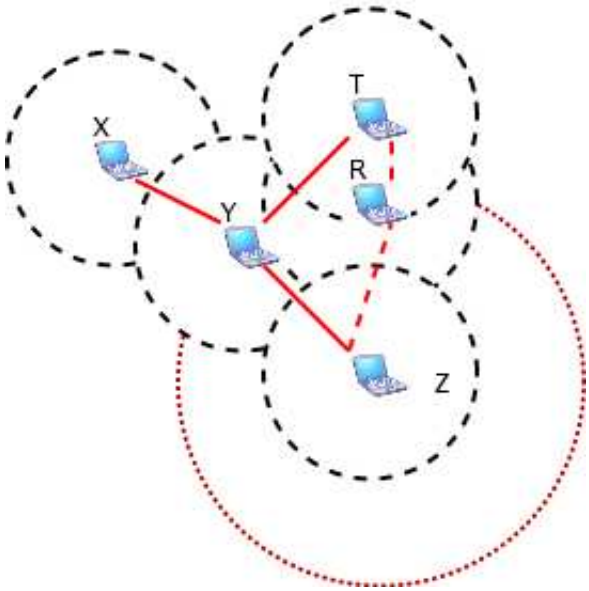

Figure 2.4: *Selective Bordercasting, Messages to R are Redundant*

2.5 AODV - Ad Hoc On-Demand Distance Vector Routing

AODV is a reactive routing protocol introduced in 1997 based on DSDV. AODV improves on DSDV because it minimises the number of required broadcasts by creating routes on a demand basis. It was designed for networks with a large number of mobile nodes. Broken links are reported using RERR messages and loop freedom is provided by the use of Sequence numbers. AODV is of interest because its the protocol which the Wireless Ad Hoc Network for Dublin (WAND) uses and where SAMPLE will eventually be evaluated. This section gives an overview of how messages are routed through an AODV network.

AODV contains five message types, Fig. 2.5 (adapted from [19]):

- *RREQ*, Route Request
- *RREP*, Route Reply
- *RERR*, Route Error
- *HELLO*
- *DATA*

AODV uses *HELLO* messages to detect and to maintain active links to its current neighbours. A node maintains its list of current neighbours by listening to periodic broadcasts of *HELLO* messages. In this manner if a node fails receive several *HELLO* messages from its neighbour it can detect broken links. On detection of a broken link a node will check for all nodes which use this neighbour as the next hop, mark them as invalid and add them to a *RERR* message. Every node which receives a *RERR* message will remove all routes from its routing tables containing the invalid next hop. A node will also broadcast a *RERR* messages if it receives a data message for which it has no route.

When a source node desires to send a packet to a destination and does not already have a valid route to the destination, it initiates a route discovery process. A route re-

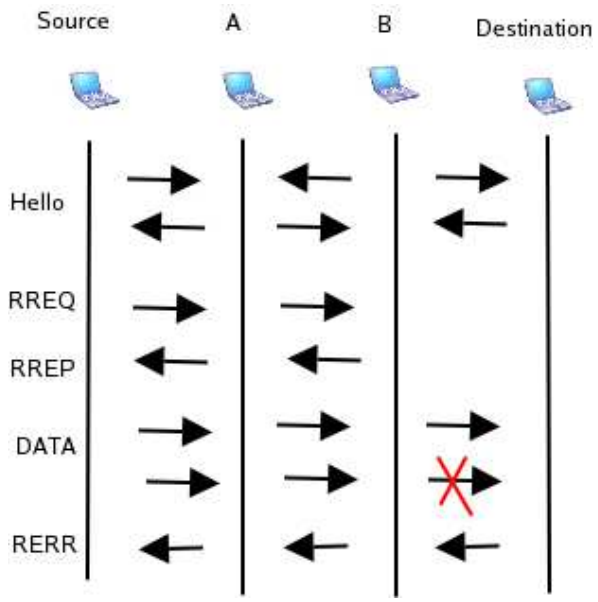

Figure 2.5: *AODV uses 5 messages types.The arrows represent message flow between source and destination*

quest packet, *RREQ* is broadcast to its neighbours. Each *RREQ* contains the following fields:

- Hop Count
- RREQ ID
- Destination IP Address
- Destination Sequence Number
- Originator IP Address
- Originator Sequence Number

A broadcast ID is incremented every time a node sends a RREQ. Together the nodes IP address and Broadcast ID are combined to create a unique RREQ ID. The source node attaches the most recent sequence number for the destination of which it is aware. AODV utilises sequence numbers to eliminate routing loops and to ensure freshness of routes. A receiving node may send a *RREP* if it is the destination or its sequence

number for the destination is greater than the one contained in the RREQ. If neither of these is the case the receiving node decrements the TTL field and rebroadcasts the RREQ message. Every node maintains a list containing the *RREQ IDs* it has already received and discards any duplicate messages. As *RREQ* propagates back through the network each intermediate node updates its routing information for the destination node if:

- The RREQ contains a greater sequence number
- Or the same sequence number with a reduced hop count

AODV has the advantage that its distance vector routing approach is simple and does not require much processing or memory.

2.6 Life of a Linux Packet

Every protocol has to be integrated into an operating system. The SAMPLE protocol was implemented on the linux 2.4 kernel. This section takes a look at the journey of a packet through the kernel.

Each network card is hard wired with a particular MAC address and listens for packets on its interface. When the interface sees a packet whose MAC address is equal to its own address or the link layer broadcast address it starts reading it into memory. Once a packet reception has been completed the card generates an interrupt request. The interrupt service routine in the card driver handles the request and carries out the following operations.

- Allocates a new sk_buff packet structure, which is how the kernel sees the packet. The sk_buff structure is defined in $include/linux/skbuff.h$
- The packet data is read from the card buffer and places into the newly created sk_buff structure
- The sk_buff gets timestamped if the driver has not already done so
- The generic network reception handler $net/core/dev.c : netif\_rx()$ is called.

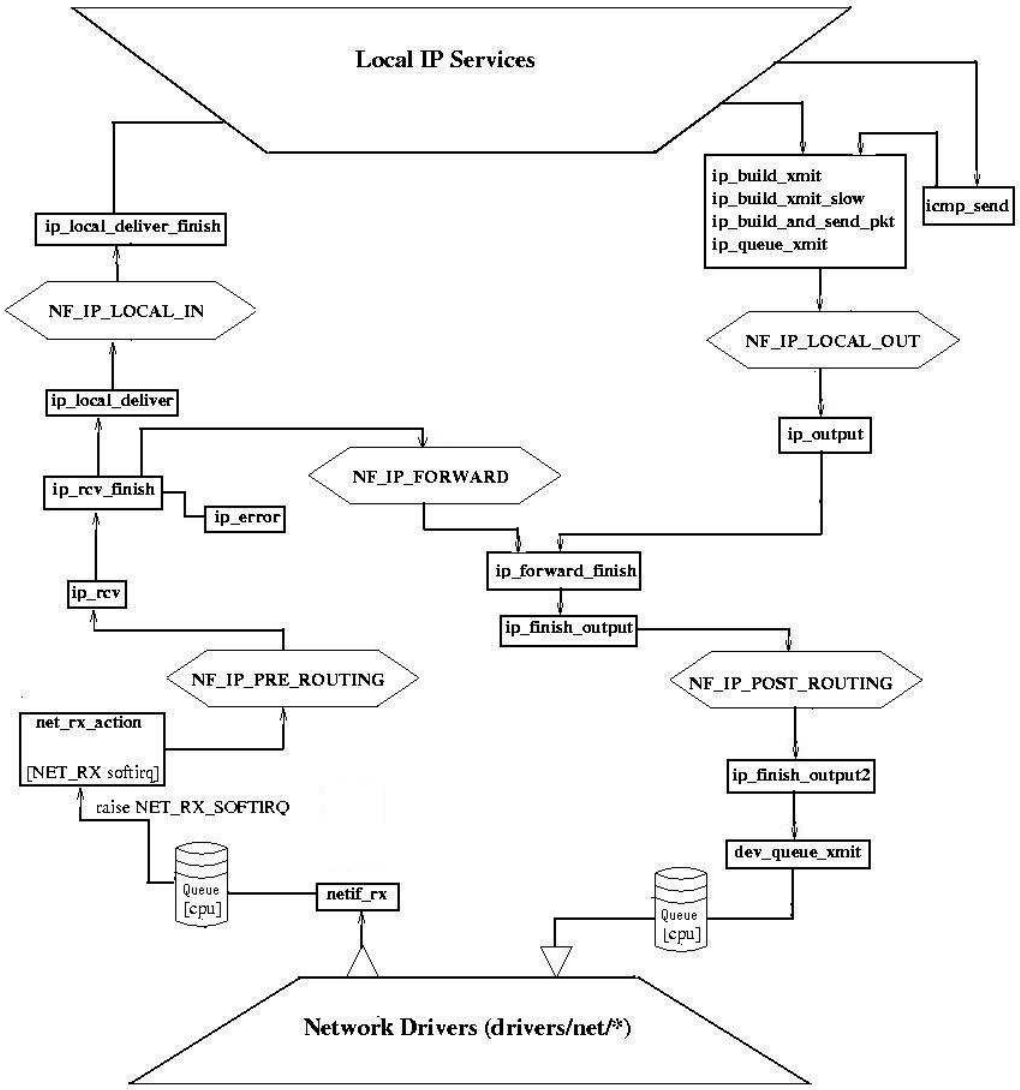

Figure 2.6: *The flow of IP packets though layer 2 and 3 in the Linux Kernel*

- On return from $netif\_rx()$ interrupts are re-enabled and the service routine is terminated

The $netif\_rx()$ function has the task of receiving the sk_buff from the network driver. It then places the sk_buff onto the appropriate queue for the processor handling which will handle it.Due to the fact that this function runs in interrupt context with other interrupts disabled the amount of work it has to do is minimised.Prior to placing packets on the congestion levels of the queue is checked and if necessary the packet it dropped due to critical congestion.

Further handling of the packet is done in the network receive *softirq*, $NET\_RX\_SOFTIRQ$, which is called from the $kernel/softipq.c : do\_softirq()$. This represents a major difference between the linux 2.2 and 2.4 architectures. *softirq* have the advantage that they can run simultaneously on more that one CPU. A *softirq* can get called in three main ways inside the kernel.

- $arch/i386/kernel/do\_IRQ()$, the generic interrupt handler
- $arch/i386/kernel/entry.S$, when a system call is invoked by an application
- $kernel/sched.c : schedule()$, when a new process is scheduled for execution

So if execution passes one of these points, $do\_softirq()$ is called. This function verifies a bit mask and if the correct bit is set, the corresponding handling routine is called, which in this case is the $net\_rx\_action()$. The sk_buff is dequeued and the packet type verified. Each protocol handler function registered for this particular packet type is called in turn.

So for example ARP packets are sent to $net/ipv4/arp.c : arp\_rcv()$ and IP packets are sent to $net/ipv4/ip\_input.c : ip\_rcv()$. New protocol handlers can be added using the $net/core/dev.c : dev\_add\_pack()$ function. This adds a packet type structure defined in $include/linux/netdevice.h$ containing a pointer to the function that will be called when a packet of that type received. It is possible to define custom protocols and their handlers with the help of this function.

Once the sk_buff has been received the $ip\_rcv()$ function it performs sanity checks on the IP header such as checksum verification. If the sk_buff looks correct it is passed on to the through the first netfilter hook, $NF\_IP\_PRE\_ROUTING$, and on

to $net/ipv4/ip\_rcv\_finish()$. This is the last IP layer function involving sk_buffs destined for the local host. After this point it carries on past the $NF\_IP\_LOCAL\_OUT$ hook through the upper layers.

If a packet is destined for another host if the $net/ipv4/ip\_route()$ function is called. Again various sanity checks are preformed. Packets will be dropped if ip header destination address is a broadcast address or the source address is equal to that of the local host. The packet is then passed on to the $net/ipv4/ip\_forward.c : ip\_forward()$ function which decrements the TTL value and drops the sk_buff if necessary. The $sk\_buff's$ header is checked to see if there is enough tailroom for the destination device's link layer header and expanded if necessary. If the *don't fragment* bit is set in the IP header and the size of the sk_buff payload is bigger than the MTU the skb_buff is dropped. A ICMP message is returned to the sender telling it a fragmentation is needed.

The packet then passes the $NF\_IP\_FORWARD$ hook and assuming it is not dropped by any userspace application or kernel module it will be sent to the $net/ipv4/ip\_forward\_finish()$ function which set any additional options in the IP header. The $net/ipv4/ip\_forward : ip\_finish\_output()$ function checks to set if the sk_buff needs to be fragmented otherwise the it will join local outgoing packets at the $NF\_IP\_POST\_ROUTING$ hook in $net/ipv4/ip\_forward : ip\_finish\_output()$.

Transport functions call $net/ipv4/ip\_output.c : ip\_build\_xmit()$ in order to build a packet. An sk_buff is allocated using $sock\_alloc\_send\_skb()$ and space is reserved for the hardware header. The packet then passes the $NF\_IP\_LOCAL\_OUT$ hook and if it isn't stolen it will carry on to $net/ipv4/ip\_output/ip\_output()$. $ip\_output()$ checks to see if fragmentation is necessary after which the sk_buff passes the final netfilter hook $NF\_IP\_POST\_ROUTING$. $net/ipv4/ip\_finish\_output2$ checks to see if the destination entry has a hardware cache. Finally the sk_buff is queued for driver output. The driver is then woken up and the actual transmission occurs.

Chapter 3

SAMPLE Protocol

3.1 Overview

SAMPLE is an on demand statistical protocol which does not separate control and data messages. This approach is different from other statistical protocols like AntNet[20] which is built upon the ant colony metaphor. AntNet uses routing agents, called ants, to adaptively estimate the quality of each local routing choice. These agents are routed probabilistically[1] whereas data packets are routed using links that give the highest probability of success. This is in contrast to SAMPLE where control information is added to the packet and all packets are routed probabilistically.

This approach exploits that fact that the cost of sending additional packets in a wireless network is more expensive then increasing the size of each packet by a small amount. One of the advantages of including routing information in every packet is that the more popular destinations receive a higher volume of routing traffic and therefore the quality of routing information to these destinations is higher. In simulations SAMPLE has been show to exploit this increased quality of routing information by detecting stable paths[2] to popular destinations such as internet gateways.

[1] With the possibility of sub optimal decisions occurring

[2] Stable paths occur when high quality links between less mobile nodes allow other more mobile nodes to use these stable routes to form traffic corridors to popular destinations such as Internet gateways

3.1.1 Route Costs

Link quality is defined as the probability of a successful transmission of a unicast packet to a given neighbouring node. In order to estimate such a probability it is necessary to track the rate of certain network events. SAMPLE observes the following network events:

- Attempted Unicast transmission N_A
- Failed Unicast transmission NF
- Unicast Receive N_U
- Broadcast Receive N_B
- Promiscuous received N_B

Once the rate of these network events in the current window[3] is known it is possible to estimate the likelyhood of a sucessful transmission in the following manner:

$N_T = N_U + N_B + N_P$: *Total number of packets received by the routing agent*
$N_S = N_A - N_F$: *Number of Successful Unicasts*
α : *The belief about a transmission been successful, 0.5*
β : *The significance of received packets compared to transmitted packets, 0.2*

$$P(success) \quad = \quad \frac{N_S \ + \ \alpha\beta N_T}{N_A \ + \ \beta N_T} \tag{3.1}$$

This approach differs from discrete protocols such as AODV which considers a link as either on or off depending on the most recent communication. SAMPLE is using recent historical data in an attempt to distinguish a broken link from temporary congestion.

[3]SAMPLE only considers events which have occurred in a certain time interval τ. The interval τ is discretised into a number of increments m ,such that every $\frac{\tau}{m}$ seconds the oldest increment is discarded

A routing end point is defined as either a source of or destination for a packet. The current estimate (V value) for the cost of a routing end point is calculated by calculating the optimal value of a set of Bellmann equations. The result is presented in Eq. 3.3, further details can be found in references [7] [21] [22]

C_L : *Last advertised cost of a route end point by neighbour*
T_E : *Time elapsed, difference between current time and last time route end point was advertised*
D : *Decay per second,1.01*

$$V(Neighbour) \; = \; C_L D^{T_E} \tag{3.2}$$

Advertised costs in SAMPLE are considered to decay, therefore stale routes can be removed. If a route has not been advertised within a defined period ROUTE_TIMEOUT, (10secs) the route is considered stale and the last advertised value set to COST_MAX, (-1000). This is a similar concept to "pheromone evaporation" used in Ant Colony Optimisation (ACO) protocols.

R_F : *Reinforcement received for a failed packet transmission, -7*
R_S : *Reinforcement received for a successful unicast transmission, -1*
P_S,P_F : *Probability of success and failure respectively as described by Eq. 3.1*
V(Neighbour) : *see Eq. 3.2*

$$max \left[V(Neighbour) + R_F + \frac{P_F}{P_S} R_F \right] \tag{3.3}$$

3.1.2 Forwarding Packets

The SAMPLE routing agent on each node maintains a routing table with the current estimated cost to a route end point as described in Eq. 3.3 above. This current estimated cost is updated every time the SAMPLE daemon receives a network event, a packet. Every neighbouring node who has advertised a cost for this routing end point

is kept in a next hop table. When a routing agent wishes to forward a packet to a destination it will calculate the cost value (Q values) of each neighbouring node for this destination. SAMPLE will not blindly considering every neighbouring node as a potential next hop for the packet. It employs a greedy heuristic to minimise the length of time it takes to deliver a packet and restrict exportation to the most profitable parts of the network. This greedy heuristic prevents a routing agent from forwarding a packet to a neighbouring node whose advertised cost for the destination is greater than the current estimate plus MINIMUM_REWARD for the destination maintained by the routing agent.

As well as considering the cost advertised for all neighbouring nodes SAMPLE also gives the possibility of Broadcasting the packet a cost. Broadcasting a packet is considered for two reasons:

1. In the case of bootstrapping or when existing routing information has gone stale, a routing agent has no next hop to unicast the packet too. In such a scenario SAMPLE must behave correctly, therefore broadcasts are used when no next hop exists.

2. In order to discover new routes SAMPLE allows suboptimal decisions to be made. A suboptimal decision may be either a unicast to a node whose total route cost is not the lowest or a broadcast when unicast options are available. A comprise has to be achieved between *exploration* and *exploitation* . If optimal decisions were taken whenever possible the quality of SAMPLES routing information would be severely effected. On the other hand if some limitation is not placed on when suboptimal decisions can be taken the cost of routing packets is greatly increased. SAMPLE controls the frequency of suboptimal decision occurring by placing a virtual cost for a broad cast.In addition the temperature value for the Bolzmann discussed can be changed to reduce or increase the likelyhood of suboptimal decisions being taken.

Once the total route costs have been calculated the next hop is picked probabilistically using the Boltzmann distributed-exploration. The probability of picking **N** as the next hop is:

M : *Set of all eligible*[4] *next hops*
T : *TEMPERATURE, 1*

$$P(N) = \frac{e^{\frac{Costofrouteva N}{T}}}{\sum_M e^{\frac{CostofroutevaM}{T}}} \tag{3.4}$$

As **T** increases it is more likely suboptimal decisions will be taken. The IP destination address is set to the value of the next hop and source and destination costs added to the SAMPLE header. Once the destination receives the packet it uses the source and destination costs included in the SAMPLE header to improve its view of the network. In addition the destination node has the option of sending a proactive response to help disperse routing information through out the network. The MAXIMUM_RECEIVES_WITHOUT_SEND is set to 2.

3.1.3 SAMPLE Routing Example

In Fig. 4.2 (adapted from [7]) node **A** joins a MANET using SAMPLE as their routing protocol. **A** wishes to open an ssh session to **F** but as it has just joined the network and has not overheard an other traffic it has to broadcast its first packet. When the packet arrives at **B** it has to decide on the which next hop will **A**'s packet be sent on. **G** and **H** are ruled out because of the greedy heuristic due to the fact their advertised costs for **F** are greater than **B**'s current estimate for **F**. **B** then calculates the total cost of routing through **C** as 10 (9+1), this includes the probability of successfully transmitting the packet to **C**. Like wise the total cost through **E** is calculated as 13. Due to the fact that SAMPLE picks its next hop probabilistic **B** has now possible next hops:

- The optimal unicast via node **C**
- A sub optimal unicast via node **E**
- A suboptimal Broadcast

Using the Boltzmann distributed-exploration the routing agent at **B** selects **E** as the next hop, opting for exploration over exploitation. **G** overhears this unicast by promis-

[4]All neighbours whose current estimate for the destination is greater then the routing agents estimation plus MINMUM_REWARD, 0.5

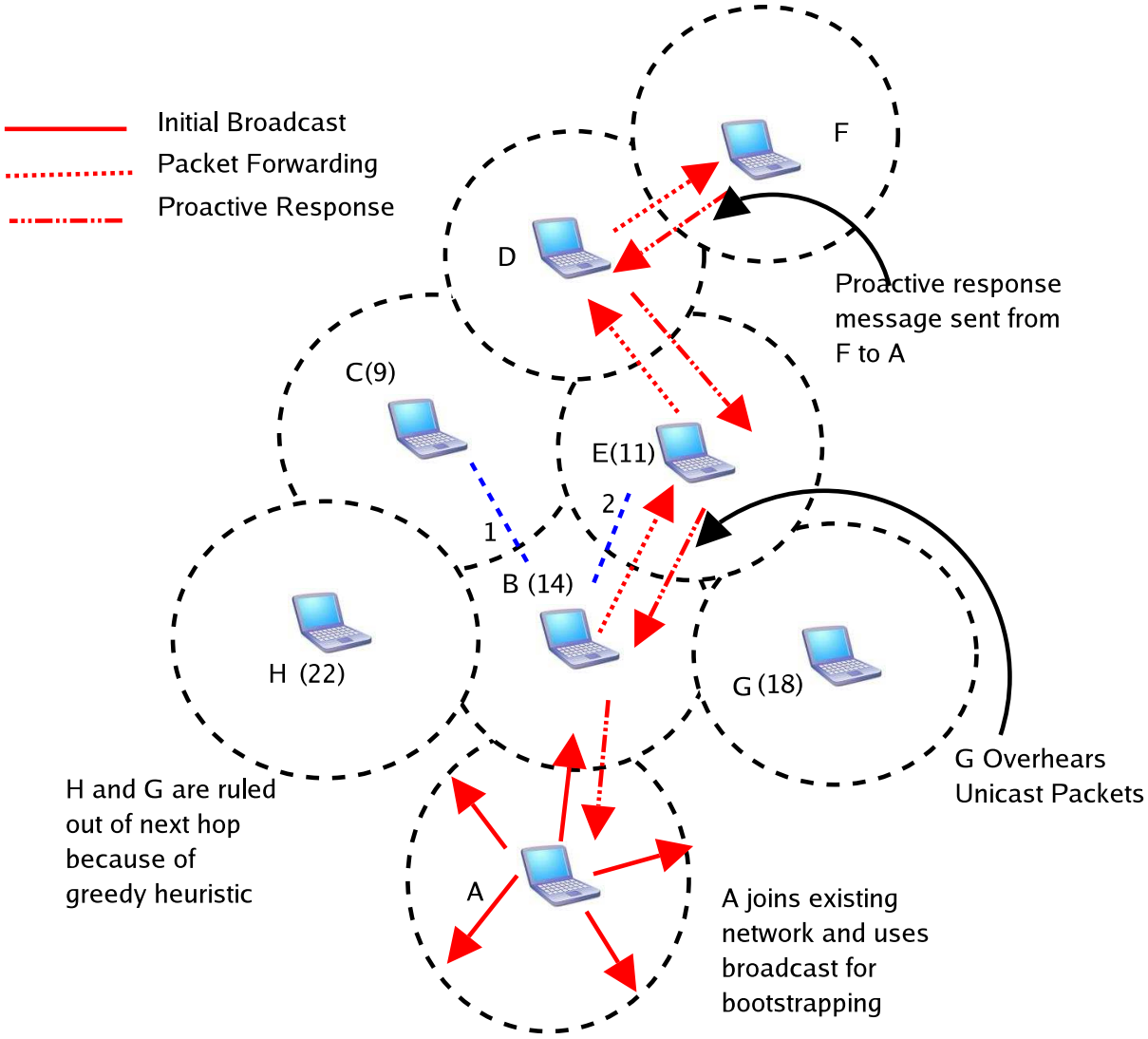

Figure 3.1: *Routing in SAMPLE*

cuously listen for all unicasts in range. **E** then forwards the packet to **F** via **D**. **F** sends a proactive response back to the source **A**. For the rest of the ssh session **A** can exploit the stable routing **A-G-C-D-F** while carrying out exploratory options by choosing suboptimal next hops.

Chapter 4

Implementation

This chapter describes the implementation approach taken and presents some of the important components of the SAMPLE design. The chapter is concluded by looking at how packets are handled in the SAMPLE protocol.

4.1 Overview

Creating a working implementation of MANET protocol is a necessary step in the validation of a new protocol. An implementation will test the protocol under real conditions and verify whether any assumptions made were valid. This prepares the way for field tests to be carried out and eventually full deployment of the protocol. The first step in this process is to find a design which will allow the protocol to be ported from its simulation environment, in SAMPLEs case NS2, and integrated it into a full operating system. This is not a trivial task.

The most fundamental decision is where the protocol should reside: in userspace, kernelspace or both? It is possible to implement a protocol in userspace by using events external to the protocol to trigger actions. For example an Address Resolution Protocol (ARP) message is generated when the host does not know the MAC address of the next hop. This message can be used to indicate that no path to the destination is know and therefore a route discovery process needs to be initiated. The advantage of this approach is that no kernel space code is required, reducing the complexity of the development and installation process. Disadvantages of this process include the

dependency on ARP for initiating protocol events. This can cause problems if the ARP cache and routing tables are out of sync. For example the ARP cache might contain an entry for a destination not in the protocols routing tables. Also since route discovery is initiated by outgoing ARP packets these are an unnecessary overhead and waste of bandwidth. It is possible to avoid these problems by modifying ARP to suit the needs of the protocol.

Another possibility is to support the protocol events directly with the kernel. While this increases efficiency it comes at cost of a longer and more tortuous development cycle. In addition installation requires a kernel rebuild plus the protocol is now directly dependent on the operating system, creating potential porting issues for the future.

The third option is to implement a kernel module which sits on top of the netfilter architecture. This approach has the advantage of intercepting incoming and outgoing packets directly in kernel space with out having to modify kernel source code. Installation is greatly simplified, loadable kerenel modules are installed and uninstalled by using a bash command. Portability and maintainability are greatly increased by replacing a dependency on the the kernel core which is constantly in flux with the more stable netfilter architecture. This was the approach taken for the implementation of the SAMPLE protocol.

The Linux operating system was not built with MANET protocols in mind. Implementation problems surface when required features of the protocol are not directly supported by the operating system. The following sections introduces some of the implementation issues which arose.

4.2 IOCTL- Input/Output Control

Promiscuous listening allows the local node to gather information shared between other nodes but not directly destined to the local node. This provides an alternate mechanism for propagating information through out the network at no extra cost in terms of number of transmissions. Network interfaces do not normally listen for frames other than those addressed to its 6 byte mac address or broadcast address. The *ioctl* function call can be used to manipulate the underlying device parameters for special files, in this case the network interface. See Fig. 4.1 for an overview of promiscuous listening

in SAMPLE.

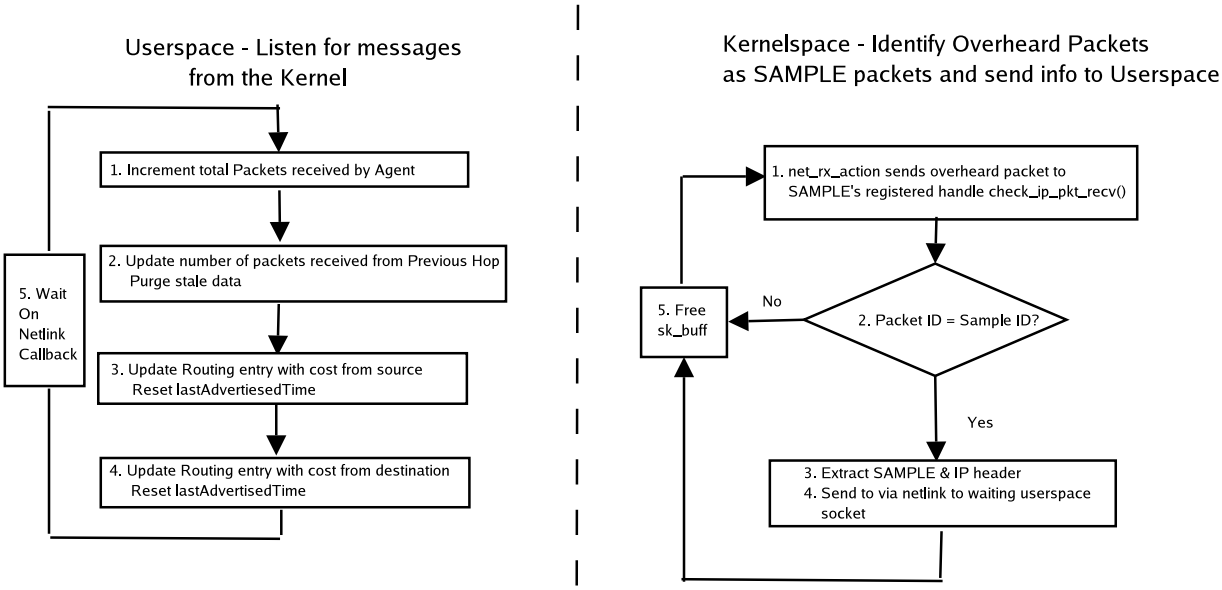

Figure 4.1: *Promiscuous Listening*

The call which allows the interface PROMISC flag to is be set is as follows:

```
#include <sys/ioctl.h>

ifr.ifr_flags |= IFF_PROMISC
ioctl (sock, SIOCSIFFLAGS, &ifr)
```

The paramters are

- 1. An open socket descriptor "sock"
- 2. requests the flags of the interface
- 3. is an address of an interface request structure ifr which holds the name of the interface that will be manipulated

4.3 Headers

The sample header Fig. 4.2, is included in every out going IP packet sitting between the IP and transport layer headers. Each packet buffer is expanded to include the SAMPLE header at the local outgoing hook, NF_IP_LOCAL_OUT. The SAMPLE header is stripped on arriving at its destination.

Figure 4.2: *SAMPLE header is placed between the IP and Transport Layers*

By registering a hook in the kernel module, where pf in the hook structure below can be set to NF_IP_LOCAL_OUT for example, all packets which traverse this hook can be dropped,stolen,accepted or in SAMPLEs case queued to userspace using NF_QUEUE and the IPQ library.

```
struct nf_hook_ops
{
        struct list_head list;
        nf_hookfn *hook;
        int pf;
        int hooknum;
        int priority;
};
```

In order to read packets sent using LIBIPQ in userspace an IPQ handle needs to be registered, after which packets can be read using the ipq_read function and returned to kernel space using ipq_set_verdict.

```
h = ipq_create_handle(0, PF_INET);
ipq_read(h, buf, sizeof(buf), 0);
ipq_set_verdict(h, pkt->packet_id,  NF_ACCEPT, 0, NULL);
```

Manipulation of the sample header is carried out by the *memmove* function. For example the code below moves the ip payload, including transport headers, by an amount equal to the size of the SAMPLE header structure.

```
memmove ( (char *) sh + sizeof (struct sample_header),
        (char *) sh, ntohs (ip->tot_len) - (ip->ihl << 2) );
```

The SAMPLE header is composed of

- Source Address, The IP address from where the packet originated
- Destination Address, The final destination of the packet
- PrevHop Address, Previous Hop of the packet. This is necessary because the IP source address can't be equal to the source address of the local host
- SAMPLE Type, Defined in the implementation of NS-2
- Protocol, Original IP protocol id
- Sequence Number, Checks the freshness of routing information
- Source Value, Cost from the source
- Destination Value, Cost to the destination
- Checksum , Verify data integrity
- TTL, Time to Live, This prevents packets from circulating in the network continuously
- Broadcast, Identify broadcast packets

The packet header is of fixed size, 31 bytes, and contains the information necessary to route itself through through the MANET and convert back to its original IP format.

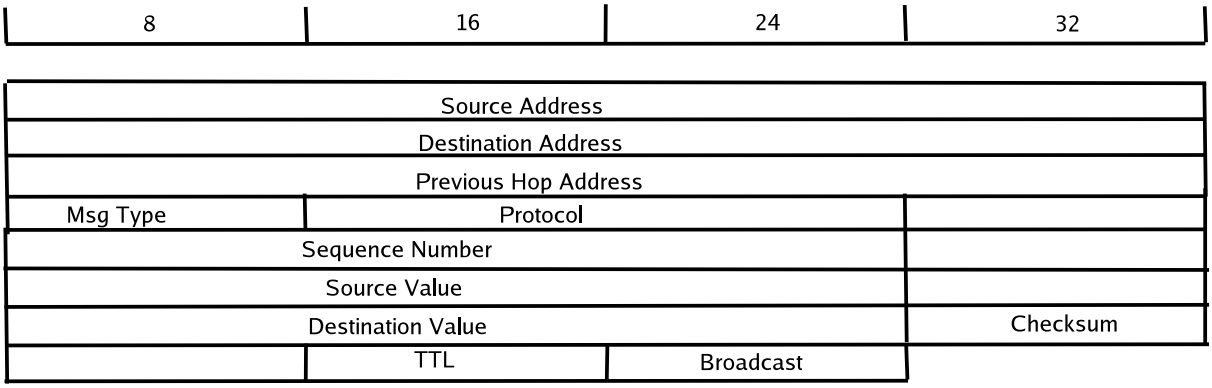

Figure 4.3: *Sample header*

4.4 NETLINK

Once an overheard packet has been successfully retrieved and the SAMPLE and IP headers extracted they must be passed to the SAMPLE daemon in userspace. At the IP layer, in the kernel core the IPQ library can be used to communicate between kernel and userspace. No such support is provide for the link layer. An alternative option is to used to use Netlink. As described in[23], Netlink socket is a special IPC used for transferring information between kernel and user-space processes. It provides a full-duplex communication link between the two by way of standard socket APIs for user-space processes and a special kernel API for kernel modules. Netlink socket uses the address family AF_NETLINK, as compared to AF_INET used by TCP/IP socket. Each netlink socket feature defines its own protocol type in the kernel header file include/linux/netlink.h. SAMPLE takes the used protocol type of 15 and so avoids having to modify netlink.h. The kernel-space netlink API is supported by the netlink core in the kernel, net/core/af_netlink.c. In the code below a netlink socket is created with a protocol type of NETLINK_SAMPLE.

```
struct sock *
ksamplenl = netlink_kernel_create(NETLINK_SAMPLE,
           ksample_netlink_rcv_sk);
```

Once a netlink socket has been created, messages can be sent using netlink_broadcast with the struct sk_buff holding the SAMPLE and IP headers. Below SAMPLEGRP_NOTIFY is the ORed bitmasks of all the receiving groups and GFP_USER is the allocation is the kernel memory allocation type.

```
netlink_broadcast(ksamplenl, skb, 0, SAMPLEGRP_NOTIFY, GFP_USER);
```

4.5 Forwarding

Once inter process communication between kernel and userspace has been established at the IP and link layer, and the SAMPLE daemon has the ability to manipulate and redirect packets the final step is to allow the forwarding of packets through kernel space. Packet forwarding refers to the process of taking a packet, consulting the SAMPLE daemon, and sending the packet towards its destination as determined by next hop.

For unicast packets this is achieved by manipulating the appropriate proc file. The /proc directory is a virtual hierarchy of special files which represent the current state of the kernel, allowing applications and users to observe the kernel's view of the system. As a general rule, most virtual files within the /proc directory are read only. However some proc files particularly in the sys directory can be used to adjust settings in the kernel. The following proc file allows unicast packets to be forwarded.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

It should be noted that the kernel will drop all forwarding packets whose source address is equal to that of the local host. This in part helps to prevent packet spoofing.

While enabling the forwarding of unicast packets is only a matter of manipulating the appropriate proc files the forwarding of broadcasts is not so trivial.Fig. 4.4 presents an overview of packet forwarding in SAMPLE.

The Linux operating system does not feel the need to support the forwarding of broadcasts messages to the same sub-net. The reasoning behind this decision lies in conserving network bandwidth. In a manner similar to some Denial Of Service [24] (DOS) attacks if a node were to ping [1] every machine on a subnet and each node were to

[1]In a DOS attack the original ping request has the spoofed source address of the victim and is replicated a large number of times.

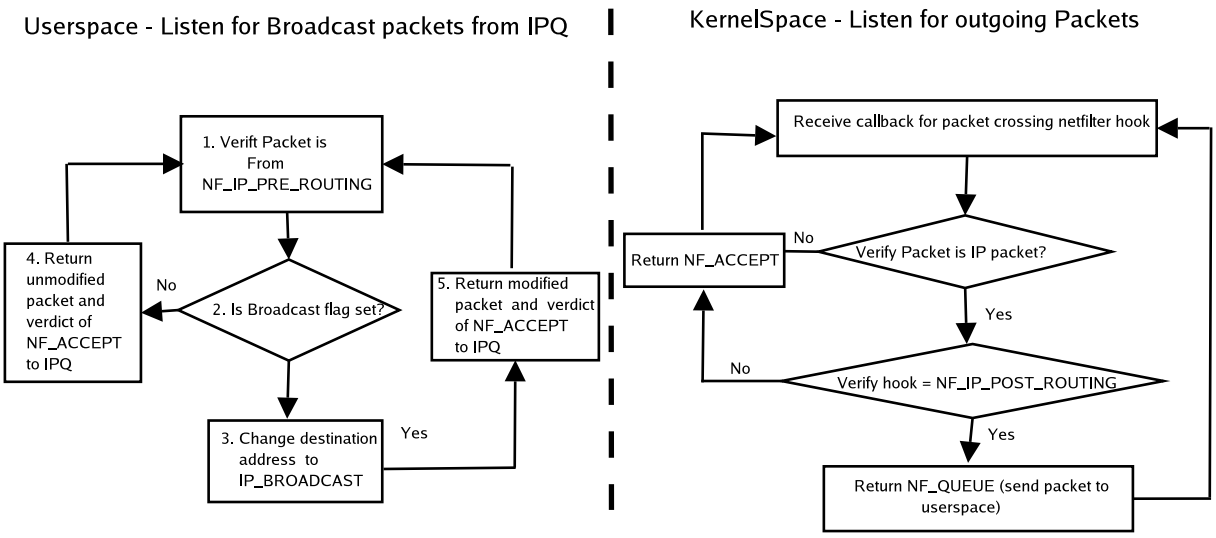

Figure 4.4: *Broadcast forwarding in SAMPLE*

simultaneously respond to the original request and forward the packet this would cause a huge increase in network traffic. SAMPLE prevents such as scenario by not indiscriminately forwarding broadcast packets.This is achieved by inserting a Time to Live (TTL) field and using the sequence number to drop stale packets. SAMPLE enables broadcast forwarding by setting a broadcast flag at the NF_IP_PRE_ROUTING hook and leaving the IP destination hook to be updated by the NF_IP_POST_ROUTING hook.

4.6 Packet flow in SAMPLE

Now that the main technical difficulties impending the porting of SAMPLE from NS-2 have been resolved, the rest of the SAMPLE code can be ported and placed in a userspace daemon. Apart from promiscuous packets which have been looked at in section 4.2 packets arrive in SAMPLE daemon in three other ways

- Packets generated by applications running on the local host
- Unicast packets addressed to the local host
- Broadcast packets destined for all nodes on the subnet

4.6.1 Packets generated by the local host

Packets generated by the local host are intercepted at the NF_IP_LOCAL_OUT hook, see Fig 4.5.

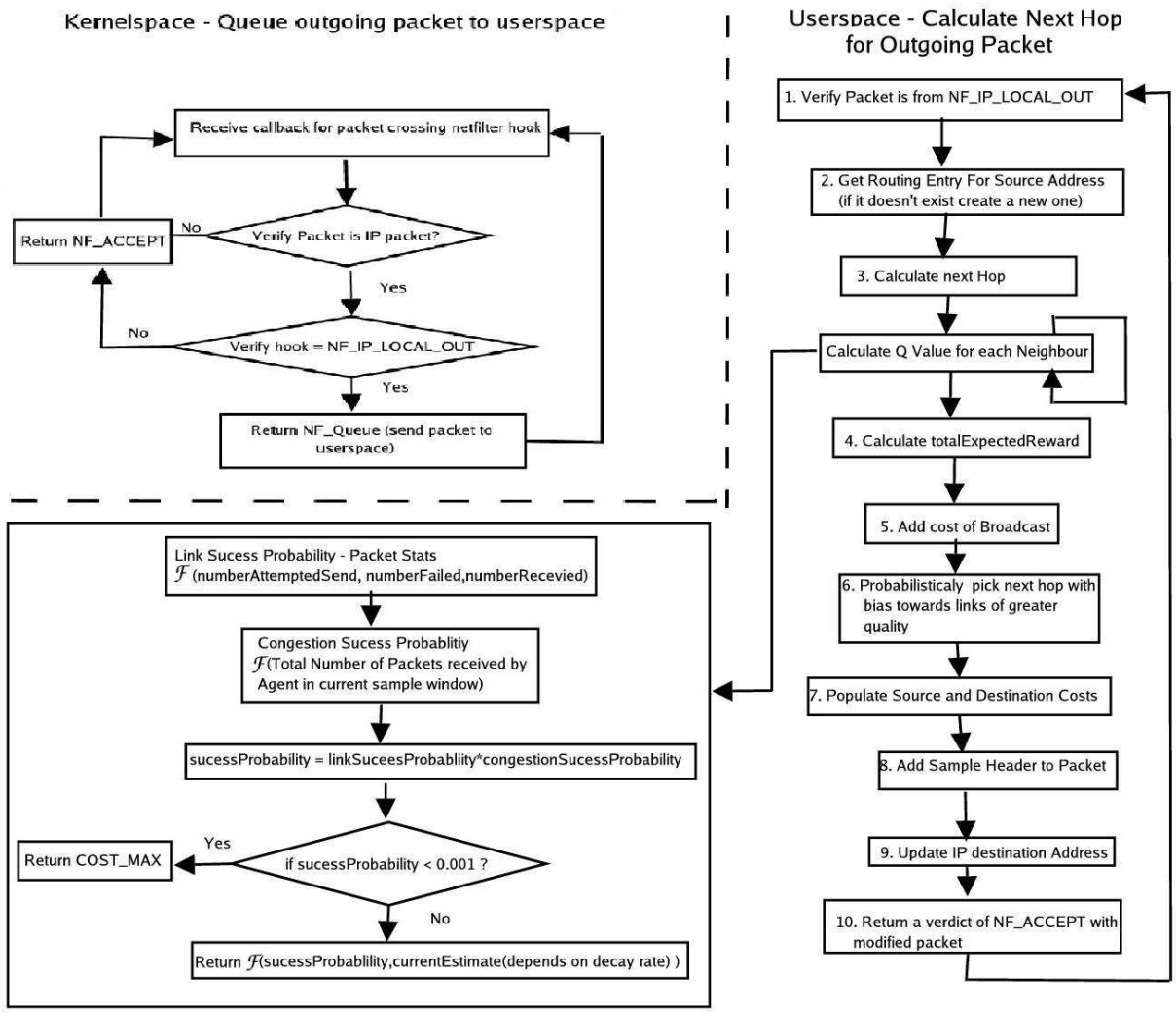

Figure 4.5: *Packets arriving from the application layer*

If no route exists then the broadcast address is chosen as the next hop. Otherwise as described in chapter 3 the cost of all the neighbouring nodes with the associated probability of successfully sending a packet to them is calculated. The probability of successfully sending a packet to a neighbouring node is calculated from the number of times the local node has attempted to send and failed to send packets to the neighbour plus the number of packets successfully received. The next hop is chosen probabilistically with links of higher quality having the a greater chance of been chosen. The SAMPLE header is added, with the IP protocol field set to the SAMPLE protocol ID and the destination address set to the next hop. Following this the packet is returned to kernelspace where it will eventually be queued for transmission by the network card.

4.6.2 Unicast packet reception

Upon reception of a unicast packet, see Fig. 4.6 the packet is queued to userspace at the NF_IP_POST_ROUTING hook. Upon reception the packet counters associated with the neighbour node and local routing agent are incremented. If the packet is stale, that is its sequence number is less than the minimum sequence number a verdict of NF_DROP is returned to the kernel. At this point all packets addressed to the local are stripped of their SAMPLE header. The SAMPLE protocol id used to identity SAMPLE packets on the network is replaced with the original IP protocol id. A verdict of NF_ACCEPT is returned to the kernel and the packet will continue on its way up the protocol stack to the waiting userspace application. However if the packet is not addressed to the local host the next hop will be chosen in the manner described in section 4.6.1

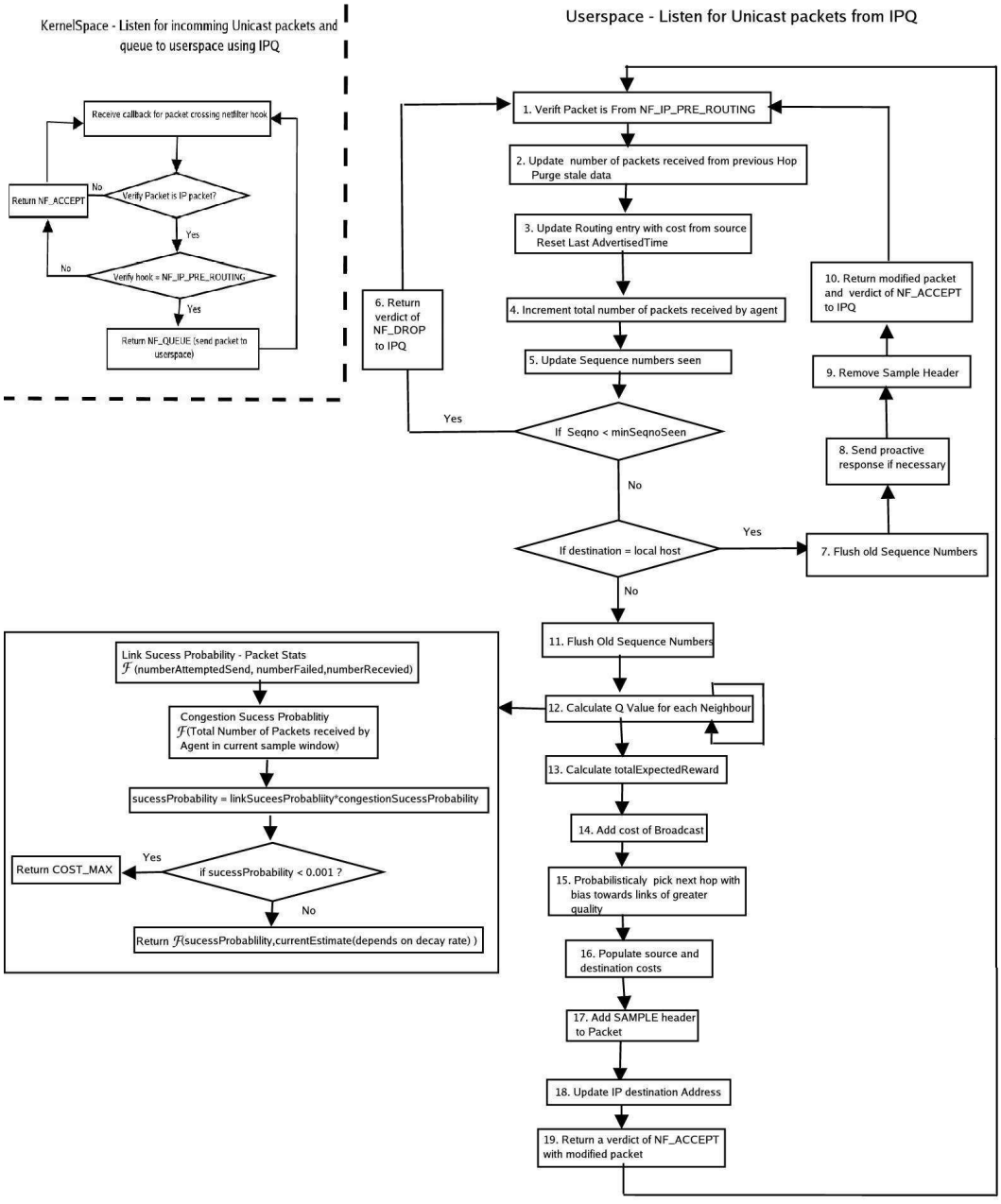

Figure 4.6: *Unicast packet reception in SAMPLE*

4.6.3 Broadcast packet reception

Reception of Broadcast packets is very similar with two exceptions see Fig 4.7.

- Broadcast packets whose source address is equal to that of the local host will be dropped to prevent circular forwarding of broadcast packets.
- If the calculated rewarded for forwarding the broadcast packet to a destination is less than the minimum reward, the packet will be dropped.

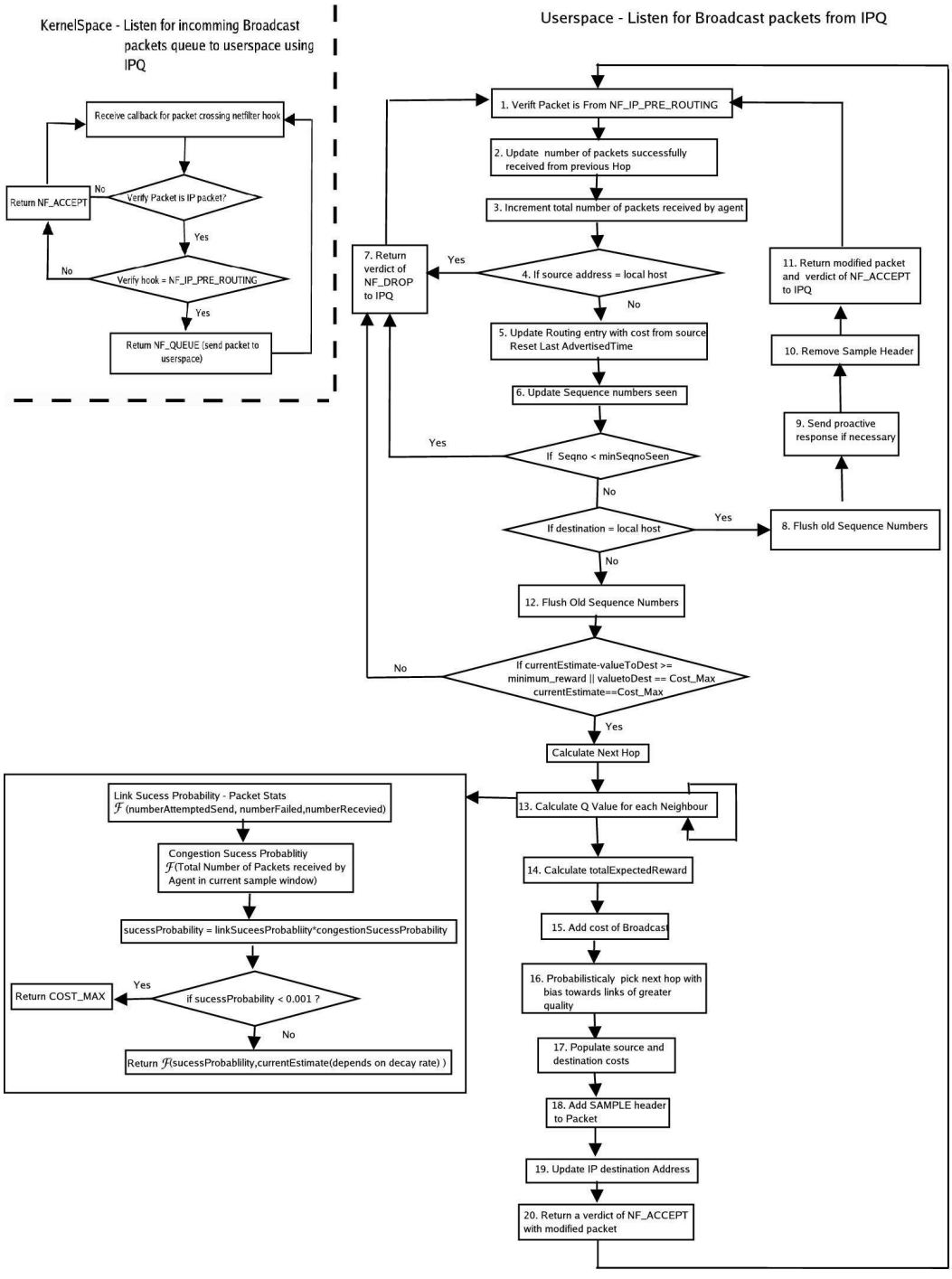

Figure 4.7: *Broadcast reception in SAMPLE*

Chapter 5

Results

5.1 Overview

This chapter reviews the implantation of the protocol. Evaluation was carried out by creating a virtual network using VMware [25]. Vmware provides an abstraction of x86 PC hardware so that multiple operating systems can run unmodified and at the same time on a standard PC. Using Vmware a SAMPLE node can be cloned extremely quickly and virtual networks created with full operating systems within a half an hour. This offers huge advantage over having to build a physical network from scratch. Simulating wireless nodes out of range is relatively simple but recreating contention and signal degradation would be quiet difficult. While Vmware does provide a test environment at relatively little expense in terms of time and hardware this step is seen only as the first step in a a full evaluation of the protocol.

5.2 Single Hop

The experiment test is to evaluate packet flow over a single hop. Two VMware nodes, **A** and **B**, are set up within wireless[1] communication range of each other, see Fig. 5.2.

[1]Within wireless range is defined as whether or not a node can process packets from another node. In these experiments nodes which are set up to be out of range artificially drop packet after the *netif_rx()* function, see section 2.6, so neither SAMPLE nor the upper protocol layers will see this packet.

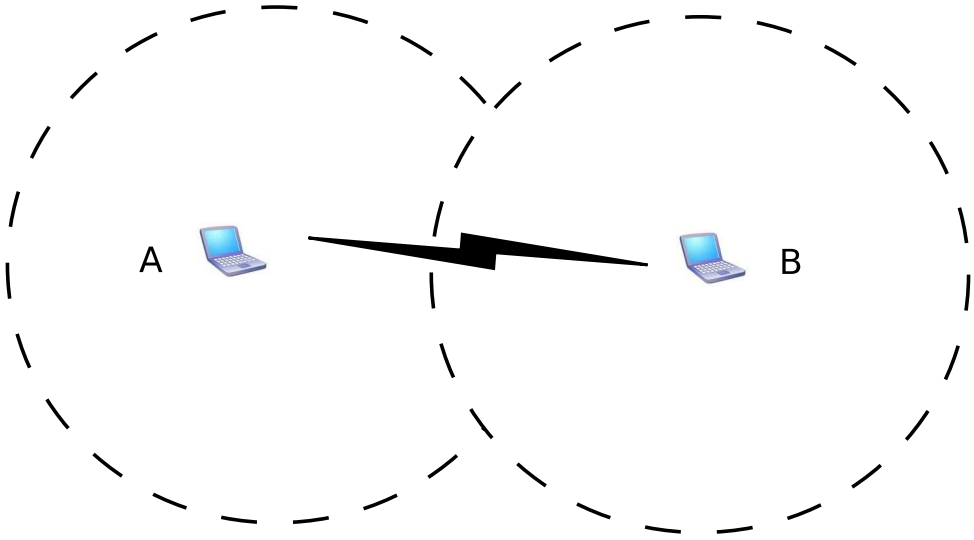

Figure 5.1: *Single Hop set up*

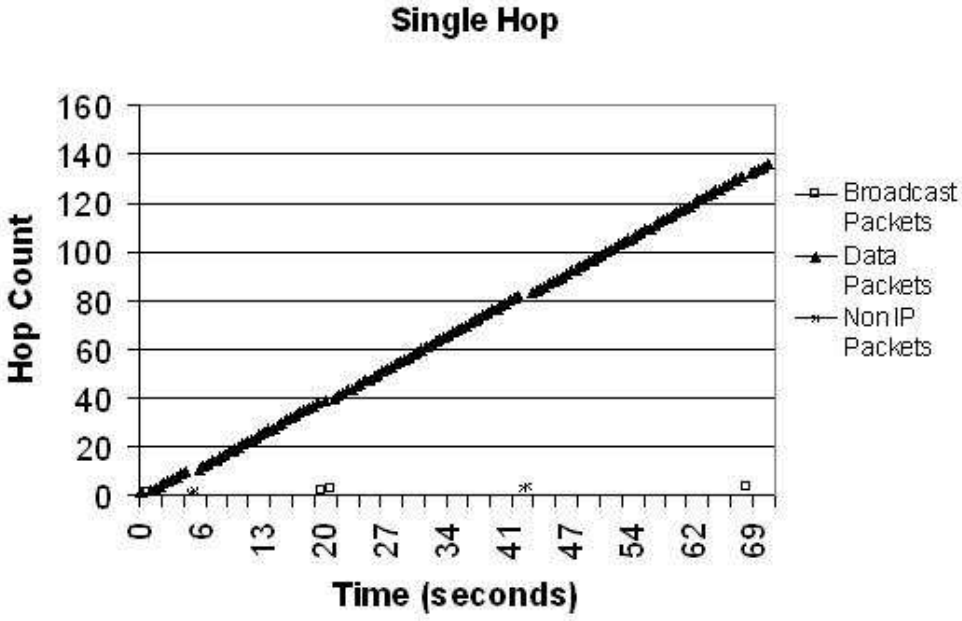

Figure 5.2: *Sub optimal decisions in SAMPLE between two nodes*

When two nodes have not communicated for a period of such that existing routing information is stale[2] or on bootstap the node which initiates communication will broadcast the initial packet. This can be see in Fig. 5.2. The initial packet is broadcast. Once routing information has been gathered by both nodes, packets are unicast. At times 20.1 seconds, 21.2 seconds and 67.6 seconds the packets are broadcast. These are suboptimal decisions in an attempt to discover alternative lower cost routes. The non IP traffic in this experiment are ARP requests and responses.

5.3 Multihop Hop

The second experiment is to evaluate packet flow over multiple nodes. For this experiment nodes **A** and **C** are out of wireless range and are therefore forced to communicate via **B**, see 5.3

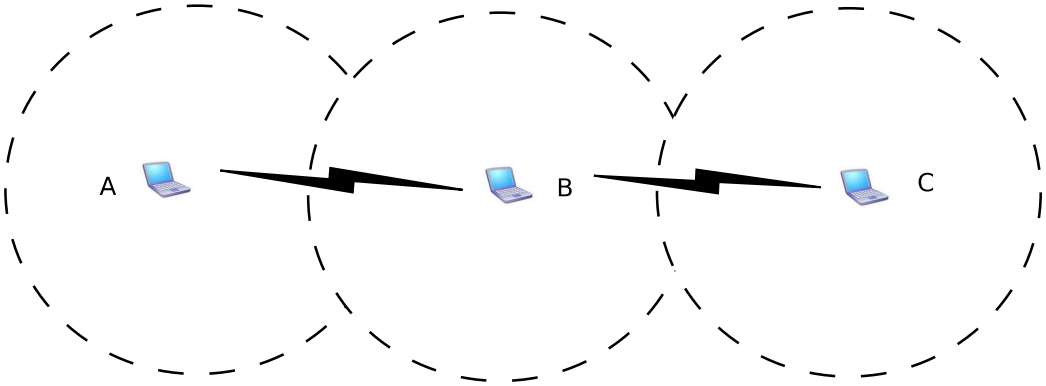

Figure 5.3: *Multi hop set up*

As in section 5.2 nodes begin communication by broadcasting the first packet. The number of data packets per second seen across the network has increased, see Fig. 5.3. While **A** is sending packets to **C** at the same frequency as in the single hop experiment above each packet has to be forwarded via **B**. Therefore it takes at two transmissions to send a packet from **A** to **C**. At 11 seconds **A** sends a broadcast packet in an attempt

[2]This is defined by the ROUTE_TIMEOUT configurable parameter in SAMPLE which for this experiment is set to 10 seconds.

at exploration. As in single hop case non IP packets are ARP packets.

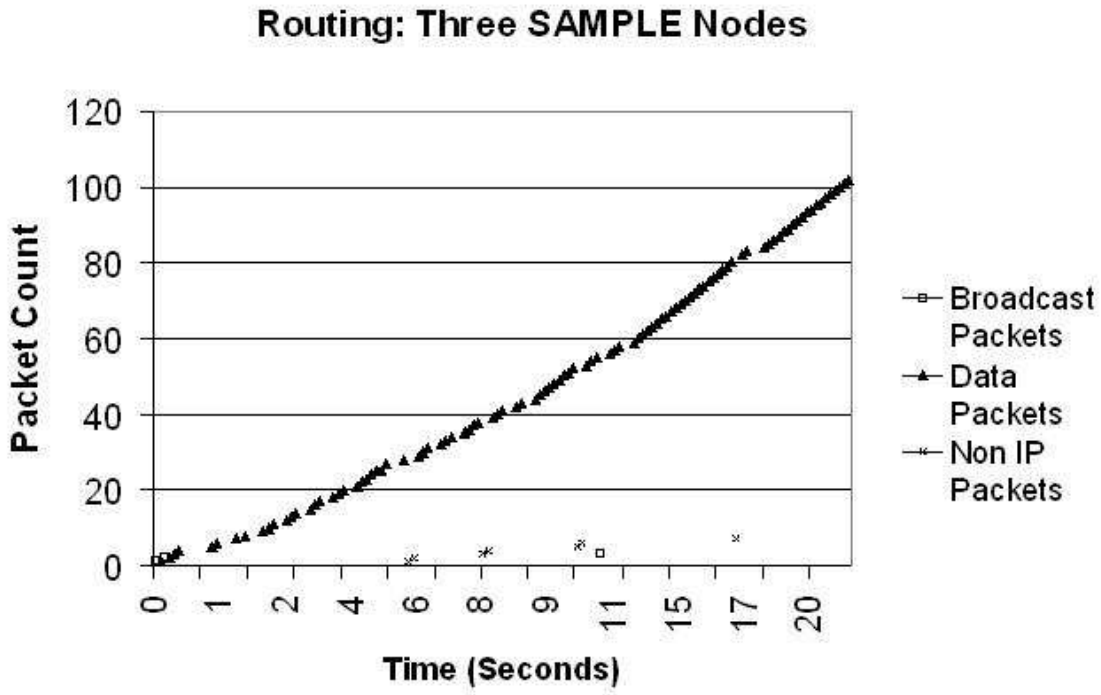

Figure 5.4: *Sub optimal decisions in SAMPLE routing*

5.4 Proactive Response

In order for routing information to be propagated effectively throughout the MANET a certain amount of communication from destination to source is desirable. This is controlled by the MAXIMUM_RECEIVES_WITHOUT_SEND configurable parameter and is reset every time a packet is sent to a destination. Using the same setup as in Fig. 5.3, Fig. 5.5 shows that on bootstrap the initial number of proactive replies is quiet high but reduces quiet significantly once nodes **A**, **B** and **C** have send a certain level of traffic.

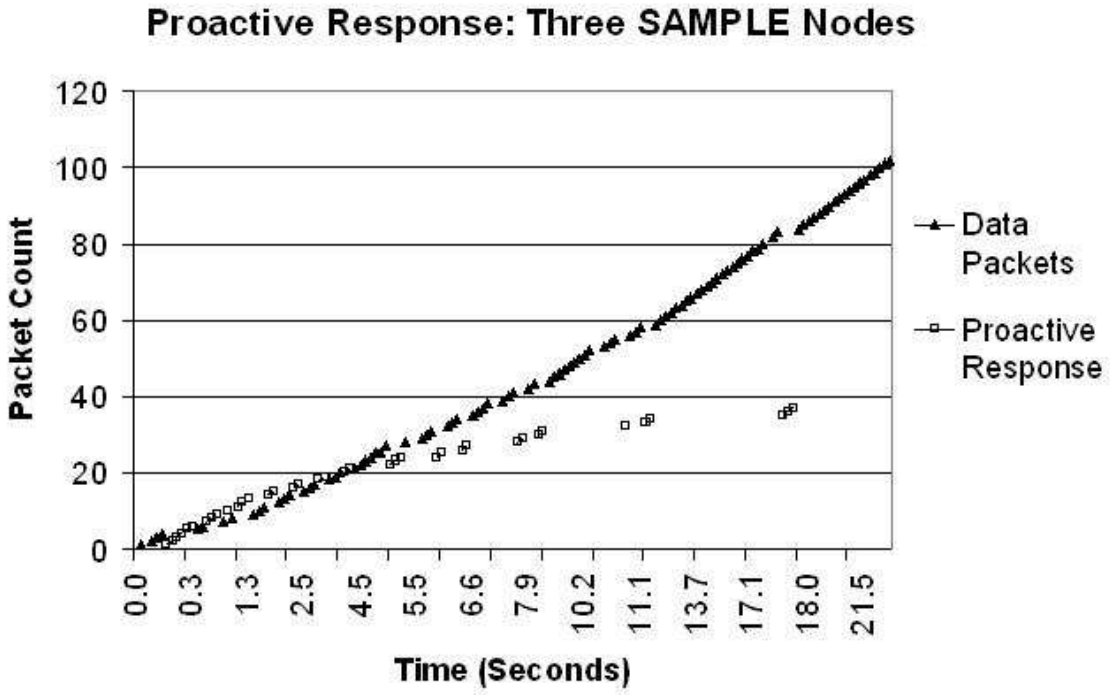

Figure 5.5: *Proactive response in SAMPLE routing*

5.5 Design Analysis

The SAMPLE protocol was implemented using a kernel module, communicating with the kernel via the netfilter [8] architecture and userspace via IPQ. Originally it was planned to place more of the SAMPLE code directly into the kernel module but as well as increasing the programming difficulty communication between kernel and userspace was becoming overly complicated. Therefore code such as adding and removing SAMPLE headers was moved to userspace along with the rest of the SAMPLE code. This had the advantage of reducing dependencies on the kernel and increasing easy of programability. As the number of lines of code in the kernel module was significantly reduced stability was dramatically increased reducing the number of kernel crashes to zero since the change in design. But does increased stability come at the expense of consuming local host resources to the point where context switching is a major overhead? At the moment there is no official benching marking[26] data on context switching for the netfilter architecture. However, netfilter is being widely used for network administration purposes and in other MANET routing protocols. For example Guido Albertengo et al [27] have used the netfilter architecture in the emulation of satellite links which had to handle 180Mb/s of traffic. As the theoretical maximum data rate for 802.11 networks is 54Mb/s this would indicate that IPQ/netfilter should be suitable for the implementation of MANET protocols.

Chapter 6

Conclusions

6.1 Achievements

The goal of this project was to provide a real world implementation of the SAMPLE protocol on Linux 2.4. This involved porting SAMPLE from the NS2 simulator and providing solutions for SAMPLE features not directly supported by the Linux operating system. The main achievement of this project was to present an architecture which would allow SAMPLE to integrate successfully with the Linux Kernel. The decision to use loadable kernel modules, which interface with the stable netfilter architecture was taken with a view to:

- Increasing portability by replacing dependencies on the kernel with the more stable netfilter architecture.
- Ensuring the SAMPLE implementation was stable by avoiding an over complex kernel solution.
- Increasing the ease of supporting and developing the project without compromising performance.

In addition a preliminary evaluation of SAMPLE was carried out using VMware verifying that the SAMPLE protocol behaves as expected in the following areas:

- bootstrap,

- forwarding of unicast and broadcast packets,
- correctly balancing exploration and exploitation
- advertising of routing information proactively
- listening promiscuously to overheard traffic

6.2 Future Work

Since the WAND operating system is Linux 2.4, this provides an excellent testing environment for the next step in the evaluation of the protocol. Detection of failed unicast packets has not been integrated into the current implementation. This can be achieved in a number of ways. Passive acknowledgement can be achieved by listening to neighbouring nodes forwarding packets. The 802.11 layer provides acknowledgement of unicasts packets over a certain size or a specific SAMPLE acknowledgement message could be used similar to DSR. In addition the feasibility and value of neighbour node and unidirectional link detection could be investigated. Finally as promiscuous listening places a significant strain on battery resources increasing the energy efficiency of the code could therefore be worth investigating.

Appendix A

Configurable Parameters

Parameter	*Significance*	*Value*
COSTMAX	Maximum cost of a route	-1000
DECAYPERSECOND	Routing information will decay over a period of time	1.01
PROBABILITYHISTORY	Width in seconds of sliding sample window	20
PROBABILITYSAMPLES	Number of samples to keep	50
RECEIVESIGNIFICANCE	The importance of receiving packets compared to sending them	0.2
UNICASTSUCCESSREWARD	Reward for successful Unicast	-1
UNICASTFAILREWARD	Reward for failed Unicast	-3
BROADCASTREWARD	Reward for Broadcasts	-4
MINIMUMREWARD	Reward necessary to consider a node as a potential next hop	0.5
TEMPERATURE	Influences probability of picking sub optimal paths	1
NUMBEROFSEQNOSTOSTORE	Number of sequence numbers stored	20
MAXIMUMRECEIVESWITHOUTSEND	Number of packets received before sending a proactive response	2

Table A.1: SAMPLE Configurable Parameters

Bibliography

[1] N Abramson. The aloha system - another alternative for computer communications. volume 37, pages 281–285, 1970.

[2] J. Zheng and MJ Lee. Will ieee 802.15.4 make ubiquitous. networking a reality? *IEEE Communication Mag.*, Jun 2004.

[3] Tomasz Imieli and Julio C. Navas. Gps-based geographic addressing, routing, and resource discovery. *Commun. ACM*, 42(4):86–92, 1999. http://doi.acm.org/10.1145/299157.299176.

[4] Ajay Chandra V. Gummalla and John O. Limb. Wireless medium access control protocols. *IEEE Communications Surveys and Tutorials*, 3(2), 2000. http://www.comsoc.org/livepubs/surveys/public/2q00issue/gummalla.html.

[5] S. Corson and J. Macker. Rfc 2501: Mobile ad hoc networking (manet):routing protocol performance issues and evaluation considerations, jan 1999. http://www.ietf.org/rfc/rfc2501.txt.

[6] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, and David Culler andJohn Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications(WSNA'02)*, Atlanta, GA, September 2002. citeseer.nj.nec.com/mainwaring02wireless.html.

[7] Eoin Curran and Jim Dowling. Sample: Statistical network link modelling in an on-demand probabilistic routing protocol for ad hoc networks. In *WONS*, pages 200–205, 2005. http://doi.ieeecomputersociety.org/10.1109/WONS.2005.30.

[8] M. Josefsson and et al. The netfilter/iptables project, 2003. http://www.netfilter.org.

[9] Raymond Cunnigham Jim Dowling, Eoin Curran and Vinny Cahill. Using feedback in collaborative reinforcement learning to adaptively optimize manet routing. *IEEE Transactions on systems, man, and cybernetics-parta:systems and humans*, May 2005.

[10] Josh Broch, David B. Johnsona, and David A. Maltz. The dynamic source routing protocol for mobile ad hoc networks (dsr). Internet-draft, IETF MANET Working Group, March 1998. Expired.

[11] Zygmunt J. Haas and Marc R. Pearlman. The zone routing protocol (zrp) for ad hoc networks. Internet-draft, IETF MANET Working Group, November 1997. http://www.ics.uci.edu/ atm/adhoc/paper-collection/haas-draft-ietf-manet-zone-zrp-00.txt.

[12] Charles E. Perkins. Ad hoc on-demand distance vector routing protocol. Internet-draft, IETF MANET Working Group, November 1997. Expiration: Mai 20, 1998.

[13] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *ACM Conference on Communications Architectures, Protocols and Applications, SIGCOMM '94, London, UK*, pages 234–244. ACM, ACM, August 1994. http://people.nokia.net/charliep/txt/sigcomm94/paper.ps.

[14] Valeri Naoumov and Thomas Gross. Simulation of large ad hoc networks. In *MSWIM '03: Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 50–57, New York, NY, USA, 2003. ACM Press. http://doi.acm.org/10.1145/940991.941001.

[15] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. The intrazone routing protocol (iarp) for ad hoc networks. Internet-draft, IETF MANET Working Group, January 2001. Expiration: July ,2001.

[16] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. The interzone routing protocol (ierp) for ad hoc networks. Internet-draft, IETF MANET Working Group, January 2001. Expiration: July ,2001.

[17] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. The bordercast routing protocol (brp) for ad hoc networks. Internet-draft, IETF MANET Working Group, January 2001. Expiration: July, 2001.

[18] Zygmunt J. Haas and Marc R. Pearlman. The performance of query control schemes for the zone routing protocol. In *SIGCOMM '98: Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 167–177, New York, NY, USA, 1998. ACM Press. http://doi.acm.org/10.1145/285237.285279.

[19] Ian D. Chakeres and Elizabeth M. Belding-Royer. Aodv routing protocol implementation design. In *ICDCS Workshops*, pages 698–703, 2004. `http://csdl.computer.org/comp/proceedings/icdcsw/2004/2087/06/208760698abs.htm)`, crossref = DBLP:conf/icdcsw/2004, bibsource = DBLP, http://dblp.uni-trier.de.

[20] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998. citeseer.ist.psu.edu/dicaro98antnet.html.

[21] E. Curran. Swarm: Cooperative reinforcement learning for routing ad-hoc networks, sept 2003.

[22] R. Bellman. Dynamic programming, 1957.

[23] Kevin He. Why and how to use netlink socket, Jan 2005. http://www.linuxjournal.com/article/7356.

[24] J. Mirkovic, J. Martin, and P. Reiher. A taxonomy of ddos attacks and ddos defense mechanisms, 2001. citeseer.ist.psu.edu/article/mirkovic02taxonomy.html.

[25] Jason Nieh and Ozgur Can Leonard. Examining VMware. 25(8):70, 72–74, 76, August 2000.

[26] Randy Appleton. Understanding a context switching benchmark, Jan 2003. http://www.linuxjournal.com/article/2941.

[27] Guido Albertengo and Stefano Petroianni. Ace/2: A scalable modular satcom system emulator. In *DS-RT*, pages 119–126, 2002. http://doi.ieeecomputersociety.org/10.1109/DISRTA.2002.1166897.