# A NETWORK FEEDBACK ARCHITECTURE FOR SELF ADAPTIVE APPLICATIONS IN WIRELESS NETWORKS

By

Nagendra S. Singh

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science

2004

## ABSTRACT

By

Nagendra S. Singh

For the optimal utilization of resources in a wireless network, applications need to adapt their output rates to suite the current traffic and mobility conditions in the network. In a wired network, like the Internet, this can be achieved by using an end-to-end feedback protocol like TCP. In a wireless network, due to issues related to unfairness, a framework for network-based feedback generation becomes attractive.

This thesis presents a framework, Network Feedback for Application Self Adaptation (NEFASA), to generate network feedback for adaptive applications in wireless networks. NEFASA can be split into two major components – the component used to generate the adaptive feedback and the component used to propagate the feedback. NEFASA uses congestion detection to start the feedback generation process and uses the AODV protocol to actually propagate the feedback to adaptive applications. To implement the feedback propagation, significant additions are done to the AODV protocol in the form of additional packet types and functionality. NEFASA delivers only negative feedback to applications.

NEFASA is characterized and analyzed by simulations using the Network Simulator (ns). For simple scenarios involving one or two flows, the performance of NEFASA is compared against manually found optimal values. Fairness problems inherent in an end-to-end mechanism, like TCP, is demonstrated and compared with a network feedback mechanism like NEFASA.

*To My Parents*

*For Letting Me Pursue My Dreams*

*So Far Away From Home*

## ACKNOWLEDGMENTS

I would like to thank Dr. Subir Biswas, the advisor of this thesis, for giving me a lot of freedom and flexibility throughout this work. His guidance helped me come up with a lot of ideas and finish this thesis in a timely manner. I would also like to thank the other members of the committee, Dr. Matt Mutka, Dr. Hayder Radha, and Dr. Jonathan Shapiro for their time and advise. I would especially like to thank Dr. Shapiro for giving me an account on his multi-processor Linux machine. I would also like to thank the CSE department for supporting my graduate studies at the Michigan State University.

Many thanks to Rohit, Jayanthi, and Raymond for all the discussions and help during the thesis. I would also like to thank my friends, Kantha and Meena, for helping me manage the grueling final days of the thesis. Last but not the least, I would like to thank all the contributors of all the free tools – *ns*, Linux, LaTeX, Gnuplot and CVS – that were used in writing and researching this thesis. Needless to say, having these tools made it a lot easier to complete this work.

# Table of Contents

# List of Tables

# List of Algorithms

# List of Figures

# Chapter 1

# Introduction

With increasing deployment of 802.11 wireless networks, and the current proliferation of mobile devices, supporting multimedia applications could further fuel the demand and growth of such networks. Supporting such applications is challenging because of limited resources – bandwidth, power, computing resources etc. – available in such networks. With advances in multimedia coding techniques, such as scalable video coding ([1][2]), deployment of such applications is indeed possible. Adaptive multimedia coding techniques require feedback about the quality-of-service that the network can provide; such a feedback can be used to gracefully increase or decrease the quality of output produced by applications using these coding techniques. End-to-end feedback mechanisms working at the transport or application layer, have their limitations because of their inability to quickly detect changes due to mobility. They also suffer from issues related to fairness, wherein flows with smaller number of hops are favored and receive a higher throughput when compared to flows with bigger number of hops. It is in this context that a network based feedback for adaptive multimedia applications becomes a necessity.

## 1.1 Design Goals

The goal of this work is to design a state-less adaptive network feedback mechanism. State-based reservation schemes are attractive, if the session time and the route used by the session stays the same over a long period of time. But with mobility and fast changing channel conditions in wireless networks, route changes become frequent and the overhead related to resource reservation and maintenance becomes high. For such dynamic networks, a state-less feedback system is a natural option to explore.

To maintain the state-less nature of the system, no flow-specific information which has to be initialized when the flows are being set-up, or removed when the flow is being dismantled, should be maintained .

In the absence of multimedia adaptation, the degradation in the quality of output at the destination for even small variations in the quality-of-service provided by the network is severe. For such applications, a reservation based scheme which guarantees a certain performance can be necessary. On the other hand, an adaptive multimedia application can adapt its output as a response to varying quality-of-service from the network. It does this by changing the amount and type of data inserted, which gracefully degrades the quality of video received at the destination. For such an application, any form of guarantee is not necessary and it is sufficient if the network can provide the best-effort service. Hence no service guarantees are provided by the system presented in this work.

## 1.2 NEFASA

This work describes the Network Feedback for Application Self Adaptation (NEFASA) network feedback mechanism. It achieves the above goals by means of three important components, the component to produce the feedback, the component to propagate the feedback and the adaptive applications that receive the feedback.

As feedback generation and propagation mechanism is the primary focus of this work, the effect of mobility on feedback generation and propagation is not considered.

In NEFASA, the applications are provided only negative feedback, i.e., only when the events in the network cause disruptions in the quality of received output the network informs the applications. The applications in turn, respond by decreasing their output rates. The implicit assumption here is that the applications are well-behaved, and NEFASA does not enforce this behavior in any way. Applications use their discretion to increment their output rates, and this is generally achieved by increasing the output rate only when negative feedback has not been received for a predetermined duration of time.

NEFASA produces feedback by identifying the conditions and events in the network which can be of interest to an adaptive application. Mobility is not the only event that could be of interest to applications. Because of constraints in bandwidth, any new flow that starts in the network can cause significant changes in bandwidth availability. As such changes are not addressed by the routing protocol – for ex., by re-routing the flow – it becomes necessary to have system to provide such services to interested applications.

Once the feedback has been produced, a mechanism is required to propagate the feedback to the interested applications. The feedback propagation happens in a stateless manner. No flow-specific information is maintained by the system to transport the feedback to the applications. The feedback that is delivered by the network, does not contain any information which tells the application about the conditions in the network. Hence, the application cannot decide the amount by which it should reduce its rate by looking at the feedback message. If the decrease in output rate is not sufficient, NEFASA will inform the same application or different one, such that over a long run the quality at which the input is produced is the quality at which it is delivered.

### 1.2.1 Orthogonality

All the major entities that are involved in NEFASA – feedback generation, feedback propagation and the application – are functionally orthogonal. This is achieved by not carrying any information in the feedback message which relates to the method in which the feedback is generated and propagated. This is a significant feature in NEFASA, because, if the feedback message carried specific information which could be used by the application, the application design would have to depend on the network design, i.e., any changes in network design would necessitate changes in the applications. The same applies on the feedback generation mechanism too. As the feedback generation method need not place any information in the generated feedback, any changes in the feedback generation method will not effect the feedback propagation or application behavior. Hence developments and research in application behavior, feedback propagation and feedback generation can go on simultaneously and transparently.

### 1.2.2 Cross-Layer Interaction

The feedback generation in NEFASA happens in the link layer, whereas the feedback propagation happens in the network layer. When the generated feedback reaches the network layer of the final destination, the network layer interacts with the transport or application layer to provide the feedback. Hence in NEFASA, there is cross-layer interaction between various layers of the protocol stack. This form of interaction between layers, which can be beneficial for applications, is not found in end-to-end feedback mechanism like Transmission Control Protocol (TCP).

Cross-layer interaction, makes a lot of information, related local network conditions, available to a network feedback mechanism like NEFASA. Information like – mobility – cannot be inferred by an end-to-end mechanism like TCP. To make use of all the local information, is one of the important goals of the NEFASA feedback mechanism.

## 1.3 Fairness

Apart from adaptive applications, NEFASA can be a useful mechanism for non-multimedia applications, like File Transfer Protocol (FTP) or Hypertext Transfer Protocol (HTTP). Such applications predominantly use an end-to-end feedback mechanism, like TCP, which was researched and developed for wired networks. TCP in wireless networks, suffers from the hop-unfairness problem, i.e., flows with small hops tend to be favored over flows with large hops. Because of this unfairness problem, flows with small number hops tend to have better throughput than flows with larger hops. NEFASA, on the other hand, is more fair and flows have throughput performance which is independent of the number hops. Hence a transport protocol which provides TCP-like service in NEFASA system, will have better fairness characteristics.

## 1.4 Organization of the Thesis

Chapter 2 provides the background material related to Ad-hoc On Demand Distance Vector (AODV). It elaborates on several important aspects of AODV ,which is used by NEFASA for feedback propagation.

Chapter 3 does a literature review which explores existing research on congestion control and application adaptation in both wired and wireless networks.

Chapter 4 discusses and characterizes congestion in wireless network. It brings out the impact of congestion on route capacity and the methods which could be used to track congestion in wireless network.

Chapter 5 describes the NEFASA framework. All the three components of NEFASA – the component used to generate the feedback, the component used to propagate the feedback and the application behavior – are discussed in detail.

Chapter 6 compares the performance of NEFASA against optimal conditions of simple scenarios – with one or two flows – which could manually found. It also describes the fairness problem in end-to-end mechanism like TCP and compares the performance of NEFASA with TCP.

Chapter 7 describes the NEFASA feedback generation and propagation mechanism. It characterizes the system by discussing the impact of the various "knobs" which influence the behavior of the system.

Chapter 8 summarizes the main contributions of this work and discusses the future work.

# Chapter 2

# Background

In this chapter, distributed wireless routing protocols are briefly discussed. An overview of various components present in a wireless protocol stack is then followed by a discussion of the AODV protocol.

## 2.1 The Wireless Protocol Stack

Software and hardware in a wireless node can be split into six layers[3], viz.,application, transport, network, link, Medium Access Control (MAC) and physical layers. These protocol layers can be imagined to be stacked on top of each other, with the application layer at the top and the physical layer at the bottomFigure 2.1. The physical layer is responsible for the transmission and reception of packets on the wireless medium. The link layer is responsible for many services. It provides address mapping services (for example: mapping network addresses to MAC addresses and vice versa) and buffering services for packets received from higher protocol layers. The link layer buffer, which provides the buffering services, is interchangeably referred to as link layer queue or the Interface Queue (IFQ). In a wireless network like 802.11, the MAC layer is responsible for managing access to the shared medium. The network layer sets up and maintains routes between nodes in the network. The transport layer

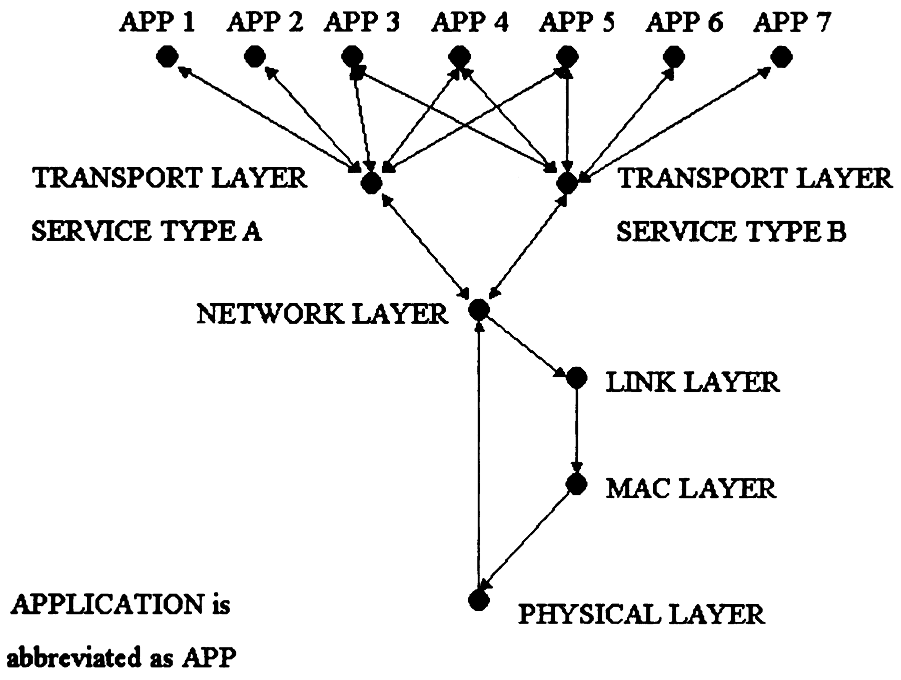

APP 1
APP 2
APP 3
APP 4
APP 5
APP 6
APP 7
TRANSPORT LAYER SERVICE TYPE A
TRANSPORT LAYER SERVICE TYPE B
NETWORK LAYER
LINK LAYER
MAC LAYER
PHYSICAL LAYER
APPLICATION is abbreviated as APP

Figure 2.1: Flow of Packets in a Protocol Stack

provides different kinds of services to the applications. It provide a highly reliable service by assuring in-order and loss-less delivery of packets (for example: TCP[4]). As a reliable service may not be required by all applications, a transport service may only provide very basic services like multiplexing and demultiplexing of packets at the source and destination (for example: User Datagram Protocol (UDP)[5]).

Figure 2.1 gives the flow of packets through the protocol layers. The application generated packets are processed by the transport layer. After the transport layer processes the packets, it passes them on to the network layer. The network layer figures out the next hop of the packet by looking into the routing table. The network layer receives packet from two sources. It can get packets from the transport layer or through the physical layer. It processes both the streams in the same way. If the packets belong to the applications, it passes those packets to the transport protocol. Otherwise it forwards the packets to the next hop towards its destination. Packets from the network layer pass through the link layer before getting transmitted on the physical layer. The link layer buffers the packet till the MAC layer is ready to receive it. If the buffer is full the packet is dropped. These drops are called buffer drops or IFQ[6] drops. The MAC layer tries to gain access to the shared physical medium. In the process it could either fail to get access to the channel or the packet could be lost during transmission due to errors. In either case the packet is dropped. If the MAC layer is successful in gaining access to the channel, it passes the packet to the physical layer for transmission.

## 2.2 Network Layer

The network layer is responsible for setting up and maintaining end-to-end routes. It maintains the information about the routes – in every node – in a database called the routing table. All the nodes in the network participate as defined by a protocol, called the routing protocol, to set up their routing tables.

In this section, the Ad-hoc On Demand Distance Vector (AODV) routing protocol is described. AODV is a popular routing protocol in distributed wireless networks. Routes in AODV are set up on-demand, i.e., they are setup only when needed by the applications. This is as opposed to a pro-active strategy of route set up, where routes may be set up before they are needed by the applications.

### 2.2.1 Route Set Up

Routes are set up by AODV when source nodes have data for a destination, which doesn't yet have a route. A route is set up by using a request-reply protocol; the request packets are called Route Request (RREQ) packets and the reply packets are called Route Reply (RREP) packets.

In AODV, RREQ propagation in broadcast based; every node that receives the

RREQ packet rebroadcasts it, if it cannot generate a reply. Because of this, once a RREQ is generated, all the nodes in the network will receive it at-least once. It is possible that a node receives the RREQ packet multiple times. To assist the intermediate nodes to identify multiple receptions of the same RREQ packet, the node originating the RREQ packet places a unique number, called the Route Request Identifier (RREQID), in the RREQ packet.

When an intermediate node receives a RREQ packet for the first time, it creates the reverse route in the routing table for the source of the RREQ packet. When the node receives the same RREQ packet again, it may be because of multiple routes from the source of the RREQ to itself. If the path taken by the current RREQ packet is shorter than the path taken by the previous RREQ packets, the node updates the reverse path from the current RREQ packet. In order to compare the distance traveled by RREQ packets, a hop count field is maintained in the RREQ packet. It is initialized to zero by the originating node, and every node which receives the RREQ packet increments it by one before rebroadcasting it again on the MAC layer.

Because of the broadcast nature of RREQ, all possible routes from the source to every node in the network – including the destination of RREQ packet – is found. This will lead to the generation of many RREP messages for one RREQ message. A hop count field, similar to the one present in the RREQ packets, is maintained in the RREP packets. The source will use the hop count field in the RREP packet to find and use the shortest route to the destination.

When the RREQ packet reaches its final destination, a reply is generated by the destination by sending a RREP message. The RREP packet travels back to the source of the RREQ packet by taking the reverse path.

| Field | Explanation |
|---|---|
| *rt* | routing table entry in the node |
| *dst(rt)* | The node for which *rt* gives the route |
| *nexthop(rt)* | The next hop that is used to reach *dst(rt)* |
| *dst_seqno(rt)* | The sequence number of *dst(rt)* as maintained in *rt* |
| *precur(rt)* | The list of precursors for the route *rt* |

Table 2.1: Fields in a Routing Table Entry

### 2.2.2 Route Maintenance

Mobility or contention for the medium, may break the wireless link between two nodes. When a link is broken, all routes which use that link should be dismantled. The dismantling process is started when a node tries to send a packet over the MAC and fails after many retries. The node then forms a Route Error (RERR) packet, which is then propagated upstream along all the routes which which have their next-hop to be the neighbor which failed to receive the packet. At the end of the RERR propagation, the RERR message is delivered to all the upstream sources which use the broken link.

The RERR propagation mechanism uses the precursor list, which is maintained for every route in the routing table. The precursor list of a route, is the subset of all neighboring nodes which use it to forward packets to the routes destination. The precursor list is built during the route request and route reply generation process. The algorithm used by a node to generate an RERR message is given in Algorithm 1. Table 2.1 explains various terms used in Algorithm 1.

When a node receives an RERR message, it follows Algorithm 2. Table 2.2 describes the various terms used in this algorithm.

### 2.2.3 Sequence Numbers

To ensure loop free routes, AODV employs the concept of sequence numbers. Every route in the network is associated with a number, called the sequence number.

```
forall routes, rt, that are present in the routing table and are usable do
    if nexthop(rt) is the unreachable neighbor then
        add dst(rt) to the RERR message
        increment dst_seqno(rt) by one
        add dst_seqno(rt) to the RERR message
        mark the route as unusable
    end
end
broadcast the RERR message in the MAC
```

**Algorithm 1**: Generating a RERR message

| Field | Explanation |
|---|---|
| *X* | RERR message |
| *src(X)* | source of the RERR message, *X* |
| *unreachabledsts(X)* | List of destinations that are reported to be unreachable by *X* |
| *seqnos(X)* | List of sequence numbers of the unreachable destinations present in *X* |
| *dst_count(X)* | Number of unreachable destinations reported in *X* |
| *rerrseqno(X, Y)* | Sequence Number of the unreachable destination, *Y*, which is listed in *X* |

Table 2.2: Terms used in Algorithm 2

```
for all usable routes,rt, and X is the RERR message that has been received do
    if dst(rt) is listed in unreachabledsts(X) and dst_seqno(rt) $\leq$
    rerrseqno(X,dst(RT)) and nexthop(rt) = src(rerr) and precur(rt) is not
    empty then
        dst_seqno(rt) $\leftarrow$ rerrseqno(X,dst(rt))
        add dst(rt) to the NEW_RERR message
        add dst_seqno(rt) to the NEW_RERR message
        mark the route as unusable
        clear the precursor list of the route
    end
    Broadcast the NEW_RERR if new destinations have been added
end
```

**Algorithm 2**: Forwarding an RERR message

The destination of a route, i.e., the node at which the route terminates assigns the route its sequence number. As the nodes assign sequence numbers independently – the nodes do not co-ordinate between themselves to assign sequence numbers that are unique throughout the network – many routes may have the same sequence number. To uniquely identify routes, the pair consisting of the destination, and the sequence number of the route has to be used. The primary motivation of using sequence numbers, is to compare the "freshness" of two routes; the route with higher sequence number is fresher than the route with a lower sequence number. Whenever a node discovers a "fresher" route, it discards the older route and starts using the fresh one. Even though the destination of the route assigns the sequence number, all the nodes in the network participate in maintaining them.

#### 2.2.3.1 Sequence Numbers and Route Set Up

When a node originates RREQ packets, it places its sequence number in the RREQ packet after incrementing it by one. This ensures that all the reverse routes – RREQ packets may set up reverse route to its originating node – have the "freshest" sequence number.

When a node originates RREQ packets, it also places the last known sequence number of the destination in the RREQ packet. This is used by an intermediate node – a node which is not the destination of the RREQ packet – to generate a reply. If the intermediate node has a route to the destination of the RREQ packet, and it is "fresher" than the route that is requested – as found out by comparing the sequence number of the route in the routing table and the sequence number of the destination carried by the RREQ packet – a reply is generated. If the route maintained by the intermediate route is not fresh, a reply is not generated.

When a RREQ packet reaches its final destination, the destination updates its own sequence number to be one more that the largest of the sequence number requested in the RREQ message or the sequence number maintained by itself. This updated sequence number is also used in the generated RREP message; This ensures that this route is the "freshest" in the network.

# Chapter 3

# Related Work

This chapter provides a literature review related to application self-adaptation in wireless networks. To provide adaptive feedback to applications, various kinds of design approaches could be taken. They can be broadly classified into two types, viz., state based schemes and stateless scheme. In a state based scheme every node in the network will keep information about every flow that passes through it. The information kept could be about the resources reserved for that flow in the node. Only information about locally reserved resources is kept in the node and no global information is maintained. A signaling system is then designed to set up and update the flow information in every node. Two such schemes from the literature are discussed in this chapter.

In a stateless scheme, no reservation information is maintained in any of the nodes. Nodes collect local information about network conditions, such as bandwidth available or congestion conditions. This information is then processed and transported to the sources of the affected flows. Different schemes could be designed which differ in the kind of information that is collected at the nodes and the way in which the information is disseminated and used. This chapter discusses two such systems for wireless networks and a system which is employed in the wired domain.

## 3.1 Stateless Schemes in Wired Networks

The Random Early Detection (RED) gateways [7] use queue length as a metric to detect and avoid congestion. A very high level algorithm for RED gateways is given in *Algorithm 3*. Whenever a packet enters the queue, a metric which depends on the queue length is first calculated. Two thresholds - $min_{th} and max_{th}$ - are enforced on the metric. If the metric is below the lower threshold, $min_{th}$, then no action is taken. If the metric is above the higher threshold, $max_{th}$, then the packet is dropped. If the metric is between the two thresholds, then the packet is dropped with a probability of $p_a$, which again depends on the *metric.* The probability of dropping packets increases as the occupancy of the buffer increases.

Applications can use dynamic rate control to improve the overall throughput, with the help of transport protocols like TCP. These applications use packet drops to get the required feedback about network conditions and regulate the application data rate. Packet drops in wired networks, is predominantly due to buffer overflows. By the time the applications take corrective action by decreasing the data rate, a lot of packets may be dropped which may affect a lot of applications. Hence, the basic thrust of the RED algorithm is to detect congestion early, so that applications which use TCP or TCP-like protocol benefit from it. To meet this goal, RED starts dropping packets randomly before the buffer becomes completely full. Dropping packets, in itself, may not prevent buffer overflows or decrease congestion; but applications get the implicit feedback early and hence many applications will not be penalized for buffer overflows.

Dropping packets need not be the only mechanism to provide feedback to applications. An explicit feedback can be generated from the Random Early Detection (RED) queue, which will expand the range of applications which can benefit from RED. To generate an explicit feedback, a RED queue may mark a bit field, called the Explicit Congestion Notification (ECN)[8], in the packet to be dropped. Instead of dropping the packet, it is forwarded to its destination. When the marked packet reaches the destination, it can invoke any application specific feedback mechanism to deliver information back to the source.

```
for each packet arrival do
    calculate the metric. The metric is a function of the queue length.
    if $min_{th} \leq metric < max_{th}$ then
        calculate the probability $p_a$. $p_a$ is a function of metric
        with probability $p_a$, drop the arriving packet
    else if $max_{th} \leq metric$ then
        drop the arriving packet
    end
end
```

**Algorithm 3**: High Level RED Algorithm

The RED scheme is similar to the method used by NEFASA to detect congestion. In NEFASA, unlike RED which has two thresholds, only one threshold is used. Once this threshold is crossed, NEFASA does not drop or mark packets, but it sends feedback messages to the applications.

## 3.2 Stateless Schemes in Wireless Networks

### 3.2.1 Stateless Wireless Ad hoc Networks (SWAN)

Stateless Wireless Ad hoc Networks (SWAN)[9] is a non-reservation based approach to provide service differentiation in wireless network. Two types of services are offered and differentiated by a SWAN network, real-time and best-effort. SWAN builds a system to provide soft-real time service to real-time applications.

To make real-time services in an ad hoc network feasible, admission control of real-time UDP traffic is performed. The end-to-end bandwidth is estimated – the source does this – by sending a probe request to the destination of the flow. If sufficient bandwidth is available, the real-time UDP traffic is admitted into the system.

Once a real-time traffic is admitted, SWAN uses local bandwidth measurement and an ECN based mechanism to provide feedback to real-time applications. This is done to regulate real-time traffic during excessive traffic conditions and mobility. SWAN does not guarantee that a fixed bandwidth will be provided throughout the session of a flow. SWAN only informs the applications during congestion and mobility. The application then tries to re-establish a session, as if it were starting all over again. If it successfully re-establishes a session, no permanent service disruption is experienced. If it cannot, the application either shuts down or manages with the available bandwidth. Hence the service provided by SWAN is termed soft real-time.

Real-time service is regulated, so that bandwidth consumed by real-time services does not cross a threshold. This ensures that the delays experienced by packets in a flow stays constant over a period of time. Best-effort traffic, on the other hand, is bandwidth regulated locally so that excessive best-effort traffic does not disrupt real-time services.

The SWAN has four (see Figure 3.1) major software components. An admission controller, as the name suggests, is responsible for admission control of real-time UDP applications. The shaper determines the rate at which best-effort traffic is pumped into the network by the node. Only best-effort traffic should pass through a shaper. This is ensured by a classifier which separates out real-time packets and sends them directly to the MAC. The rate controller is responsible for the calculation of the output rate of the shaper based on the delay measurements by the MAC. The MAC is responsible for measuring packet transmission delays and the bandwidth utilized for real-time traffic.

#### 3.2.1.1 Admission Control in SWAN

Every wireless network could have a threshold ($R_{thresh}$) rate [10] at which delays would be excessively high. SWAN tries to keep the bandwidth utilization of real-time

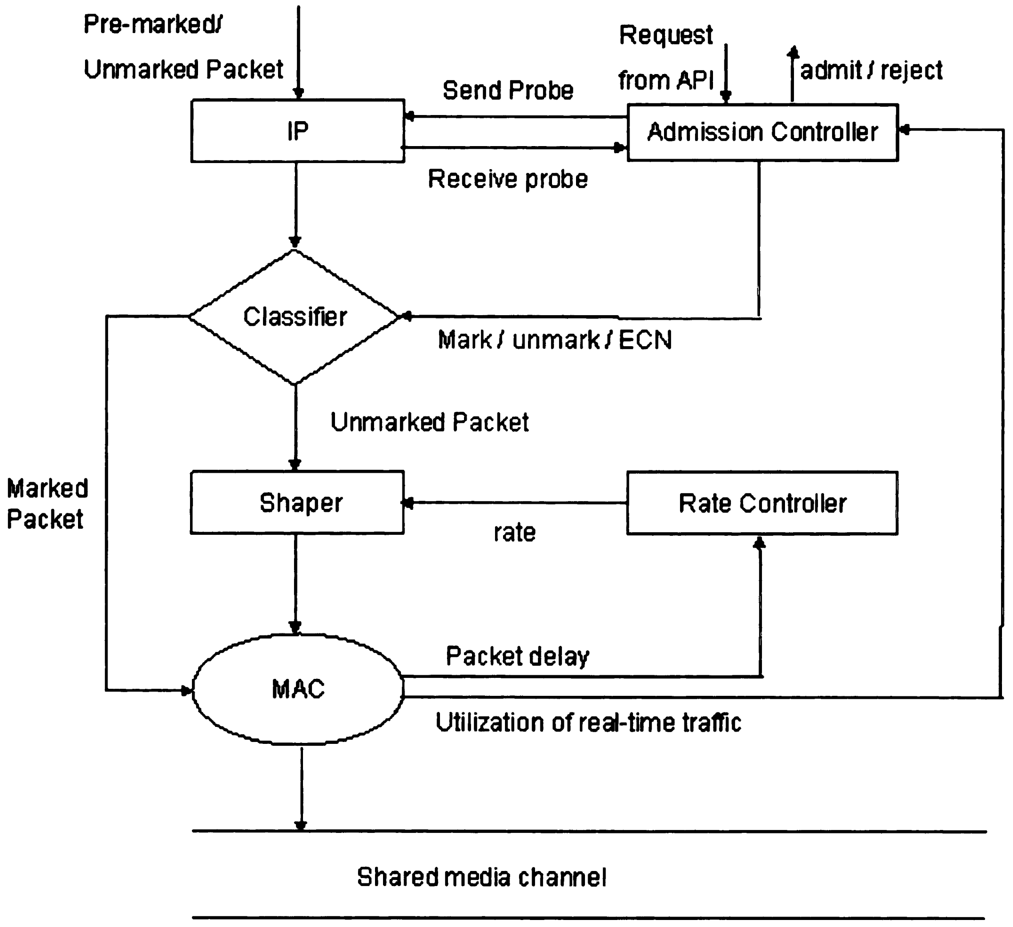


Figure 3.1: Swan Software Components

and best-effort traffic below this level. In-spite of admission control, there may be variations in the amount of local bandwidth consumed by a flow during its life time. These variations may cross the threshold ($R_{thresh}$). This may not be disruptive for best-effort traffic. But real-time applications will notice such variations. To minimize the effect of such variations, real-time traffic is admitted only if the current total utilization of bandwidth by all real-time traffic ($R_{rt}$) is below the admission threshold ($R_{admit}$). The admission threshold, $R_{admit}$, is below ($R_{thresh}$). The difference between $R_{admit}$ and $R_{thresh}$ is used for best-effort traffic.

```
receive probing request
extract the bottleneck bandwidth from the probing request ($R_{bn}$)
calculate the bandwidth utilized by existing real-time flows that pass through
the node ($R_{rt}$)
$R_{available} \leftarrow R_{admit} - R_{rt}$
**if** $R_{available} < R_{bn}$ **then**
    place $R_{available}$ in probing request
**else**
    do not change the bottleneck field in the probing request
**end**
```

**Algorithm 4**: Admission Control of Real-Time traffic in SWAN

To implement the above policy, every node in the network, before it starts a real-time application, sends out a probing request toward the destination. The probing request packet contains the bottleneck bandwidth field. Every node in the network, upon receiving a probing request executes the admission control algorithm (see Algorithm 4) and forwards the probing request. The destination upon receiving the probing request sends it back to the source as a response. When the source receives the request, it starts the application. So in effect, the bandwidth consumption at the bottleneck node is figured out and if its less than $R_{admit}$ the request is successful.

#### 3.2.1.2 Rate Control of Best Effort Traffic

SWAN uses a simple Additive Increase Multiplicative Decrease (AIMD) algorithm (see Algorithm 5) for the rate control of best-effort traffic. The algorithm is run every *T secs.* Depending on the MAC delay experienced by the packets transmitted after the previous execution of the algorithm[11], the shaping rate of the algorithm is either increased additively or decreased multiplicatively. There may be instances where the shaping rate is significantly greater than the actual rate of best-effort traffic. In this case, the shaping rate is decreased to match the actual rate.

```
n ← number of packets which have a MAC delay > the threshold delay
($d_{thresh} secs$) ;
if $n > 0$ then
    //Multiplicatively Decrease the Shaping Rate $s$
    $s \leftarrow s * (1 - r/100)$ ;
else
    //Additively Increase the Shaping Rate $s$
    $s \leftarrow s + c$ ;
end
//decrease $s$, if s is > actual rate ($a$) by g%
if $(s - a) > a * g/100$ then
    $s \leftarrow a * (1 + g/100)$ ;
end
```

**Algorithm 5**: Rate Control of Best Effort Traffic in SWAN

#### 3.2.1.3 Regulation of Real-Time Traffic

All nodes in the network monitor the utilization of bandwidth by real-time traffic. If the bandwidth utilization of real-time traffic crosses the threshold $-R_{thresh}-$ the nodes start marking real-time packet that they forward. The destination upon receiving the marked packets sends regulate messages to the source. The source then tries to re-establish the session by sending out probing request messages as previously discussed. If nodes mark all forwarded packets during the time of congestion, sessions may not be re-established correctly. To prevent this, two approaches can be taken.

One approach is for the source node to wait for a random amount of time before trying to re-establish a session. This will prevent simultaneous establishment of multiple sessions. The other approach is to mark packets from only a subset of real-time flows. Information about the subset of real-time flows being marked is maintained in the intermediate node. The subset may be recomputed periodically. Note that, maintaining information about a subset of real-time flows does not necessarily mean that per-flow information has to be maintained in the node.

SWAN does not provide adaptation services to the applications. It tries to provide services which enable an application to access the amount of bandwidth available in the network. If the applications find the bandwidth to be sufficiently available, they start the flow according to the available bandwidth. If more bandwidth becomes available to the network at a later time, or even if there is more bandwidth is available when the flow is started, the application is not informed and it continues to work at the data rates at which it started.

In NEFASA, all applications start at their native data rates, and then depending on the feedback from the network, they adapt their quality of output. Hence NEFASA provides adaptation services to the applications. NEFASA, on the other hand, does not provide any kind of service differentiation between real-time and best-effort traffic, which is provided by SWAN. Though such services can be built into NEFASA, the current implementation does not support such services.

### 3.2.2 Quality-Modified AODV (QMAODV)

QMAODV[12][13] relies on measuring the available bandwidth on a wireless link and using AODV to develop a distributed algorithm to provide network feedback to adaptive applications. For a multi-hop network, due to unique contention experienced by a flow existing between a (source,destination) pair, total link bandwidth available to an intermediate node may not be completely available for a flow that exists between the (source,destination) pair. Hence instead of making measurements for total bandwidth available locally, a node makes measurements for bandwidth available for each (source, destination) pair that it supports (see Equation 3.1).

For an 802.11 network, throughput for a packet is measured as Equation 3.2. Let us say that $u$ is the fraction of time for which the wireless link is utilized at a node. Let us say that this node is an intermediate node for a flow between nodes (A,B). If $Throughput_{packet}$ is the throughput of a packet belonging to the flow (A,B) as measured by Equation 3.2, then the bandwidth available for a flow between (A,B) is given by Equation 3.3.

$$\begin{aligned}\text{Throughput}_{(\text{source,destination})} &= \text{Throughput of a packet } (\text{Throughput}_{\text{packet}}) \in \\ &\quad \text{to a flow between (source,destination)} \end{aligned} \tag{3.1}$$

$$\text{Throughput}_{\text{packet}} = \frac{S}{t_q + (t_s + t_{ca} + t_{overhead}) \ * \ R + \sum_{r=1}^{R} B_r} \tag{3.2}$$

$$\begin{aligned} S &= \text{Number of bits transmitted} \\ t_q &= \text{MAC queuing time} \\ t_s &= \text{Transmission time for S bits} \\ t_{ca} &= \text{collision avoidance time} \\ t_{overhead} &= \text{control overhead Time.} \\ &\quad \text{e.g. RTS, CTS , propagation delay etc.} \\ R &= \text{Number of necessary transmissions} \\ B_r &= \text{Backoff time for } r_{th} \text{ retransmission} \end{aligned}$$

$$\text{Throughput}_{\text{Available}} = (1-u) * \text{Throughput}_{\text{packet}} \tag{3.3}$$

$$u = \text{Fraction of the time for which link is utilized}$$

$$c = t_{RTS} + t_{CTS} + t_{HDR} + \tilde{t}_{waitACK} + t_{transmissionoverhead} \tag{3.4}$$

$$t_{RTS} = \text{Transmission time for RTS}$$

$$t_{CTS} = \text{Transmission time for CTS}$$

$$t_{HDR} = \text{Transmission time for the MAC header}$$

$$\tilde{t}_{waitACK} = \text{Wait time for the arrival of the acknowledgment}$$

$$t_{transmissionoverhead} = \text{Time spent for SIFS, DIFS and extra propagation times}$$

In order to make the measurements got from Equation 3.2 more reliable, instead of calculating the throughput on a per-packet basis, it is calculated as an average of a packet window. The window size is typically kept at around 16 or 32. To make the measurements from Equation 3.2 independent of packet size, a constant value $c$ is subtracted from each throughput sample collected. The constant is given by Equation 3.4.

Once the available throughput is calculated, the AODV RREP based mechanism is used to to propagate the available bandwidth back to the source. The RREP mechanism is modified and is called "event triggered RREP". During the route setup phase, when the RREP message is begin sent back to the source, every intermediate node reports its available bandwidth. Using this information, bandwidth of the bottleneck link is figured out. In order to detect changes in the available bandwidth, the "event triggered RREP" is triggered by a node, automatically, when the available bandwidth changes by more than RELAY_FACTOR (which is expressed as a percentage) of the previously reported available bandwidth. The node sends this "event triggered" RREP upstream, just like the RREP messages of AODV. The only difference being, to propagate the "event triggered" RREP upstream, the permissible bandwidth in the RREP message should be more than RELAY_FACTOR of the previously reported available bandwidth by the forwarding node.

The QMAODVrelies on bandwidth measurement to provide feedback to applications. The bandwidth measurements done in the MAC layer, are not exact measurements but only indirect inferences based on observable quantities like collision avoidance time and back-off time. These measurements get approximated further, with the subtraction of the constant $c$ (Equation 3.4) from the throughput measurements. Bandwidth measurements can be accomplished in higher layers, but these are more approximate than the measurements in the MAC layer.

Furthermore, in routing protocols like AODV, bandwidth availability may vary significantly when routes are being set up. This variance in bandwidth availability is not taken into consideration in QMAODV.

## 3.3 State Based Schemes in Wireless Networks

To give feedback to adaptive applications, a flow and reservation based approach can be taken. Every node in the network will keep information about every flow that passes through it. The information kept could be about the resources reserved for the flow in the node. A signaling system could then be designed to set up and update the flow information in every node. Two such systems, Adaptive Reservation and Pre-allocation Protocol (ASAP)[14] and INSIGNIA[15], are discussed in this section.

In ASAP, flows are classified into two types, QoS flows and best-effort flows. Two types of reservations can be made by a node for a QoS flow, soft reservation and hard reservation. When bandwidth is soft reserved it can also be used for best-effort flows. But when bandwidth is hard reserved, it can only be used for QoS flows. Two types of messages, which flow in two phases and in opposing directions, are used to manage the two types of reservations. In the first phase, signaling messages flow downstream from the source to the destination and a soft reservation is made. These are the SR messages. The HR messages which flow in the second phase, travel in the opposite direction and convert the soft reservations to hard reservations. These messages have a field, *SetBW*, which indicate the amount of soft reservation to be converted to hard reservations. Reservations are setup by the first transaction (SR from source to destination and HR from destination to source) of these messages. Reservations are maintained by periodically inserting SR messages into the flow. If a HR message is received in response to a SR message, the source adapts its output according to the reservations made. If a HR message is not received, then it maintains its previous output quality. Reservations can be released by either sending a HR message with the *SetBW* field set to zero or allowing the reservations to time-out.

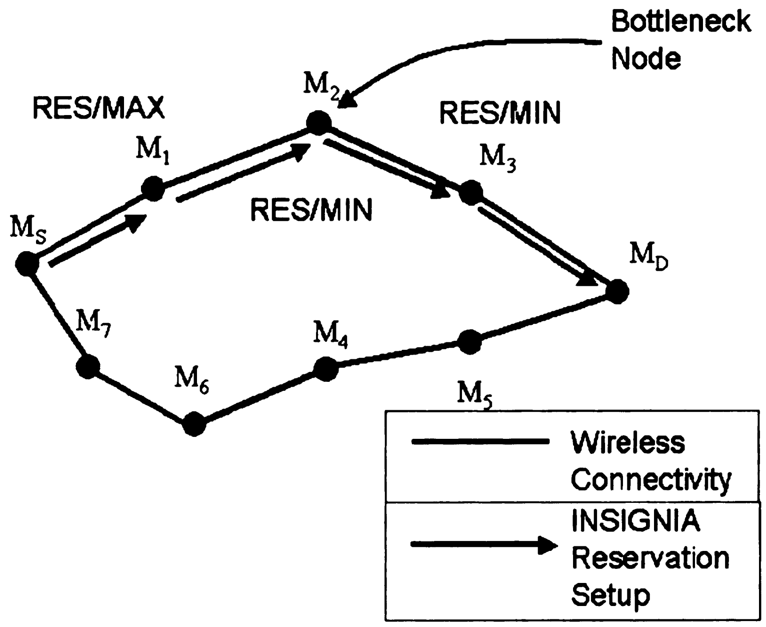
INSIGNIA[15] also supports two types of flows, best-effort and real-time flows. Real-time flows carry data from adaptive applications and can further be classified into two types, base-layer(BL) and enhancement-layer(EL). An adaptive application can pump both BL and EL data. Bandwidth can be requested in two levels, the minimum bandwidth level called MIN and maximum bandwidth level called MAX. If a MAX request is made and it is approved by the network, then both BL and EL traffic are given priority and travel as real-time traffic. If a MAX request is made, but only a MIN level is granted by the network, no EL traffic is given priority and is demoted to best-effort traffic by the network. If a MAX request is made and the network cannot even grant a MIN level, then both BL and EL traffic are demoted to best-effort. In this case it is up-to the application to give up transmission or continue with the level of service available. MIN level requests are treated similarly. As the request itself is at the MIN level, both BL and EL traffic receive best-effort service.

INSIGNIA uses an in-band signaling protocol to setup and maintain reservations.

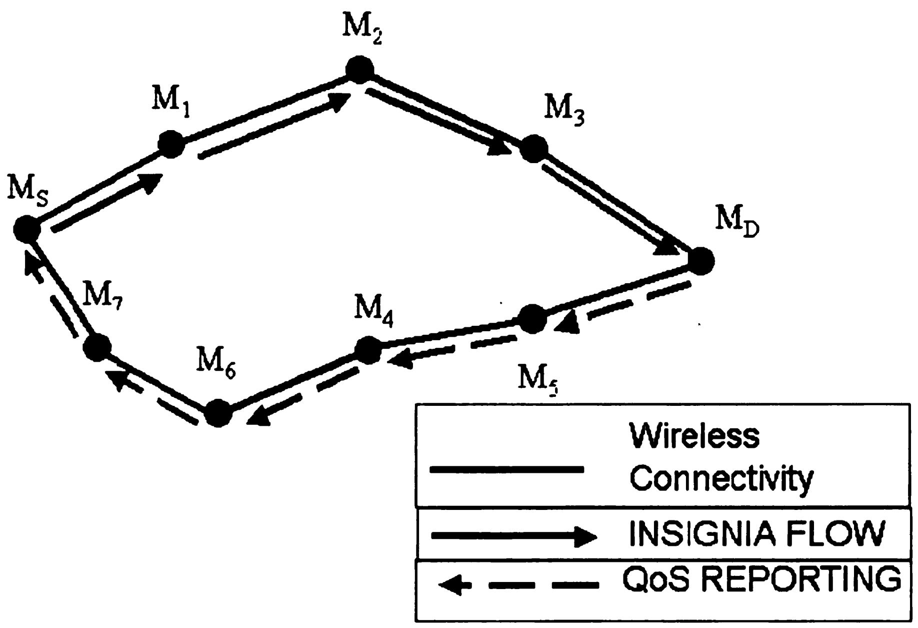
The INSIGNIA control information is piggybacked into data packets and separate INSIGNIA control packets are exchanged between the source and destination. Also, INSIGNIA plays no role in route setup phase and routes are setup independently by routing protocols.

The source, requests for a reservation by placing five pieces of information in the control portion (for ex: IP options in an IPv6 packet) of the data packet. They are Reservation mode, service type, payload indicator, bandwidth indicator and bandwidth request. The reservation mode says whether the packet is requesting for a reservation or using a previously made reservation. The service type specifies the flow type, viz. best-effort or real-time, for which the reservation is being made. If packet carries real-time data then the payload indicator specifies if its BL or EL traffic. If a request is being made, then the bandwidth indicator field specifies whether MIN level or MAX level bandwidth is being requested. The bandwidth field itself specifies the amount of bandwidth to be reserved.

When the first few data packets are exchanged between the source and destination, the reservations are setup as illustrated by figure 3.2(a). The source node $M_S$ requests for reservation by placing REQ in the reservation mode field. The values RT and MAX specify that a MAX level reservation is being requested for real-time traffic. First few nodes ($M_1$) are able to make the requested reservations. But when the packet arrives at the bottleneck node ($M_2$), MAX level bandwidth reservations are not made; but MIN level reservations are successful. The bottleneck node, now forwards the packet as REQ/RT/MIN to indicate to the downstream nodes that one or more upstream nodes can support only MIN level reservations. When the packet reaches the destination, it can deduce the type of reservations made at the intermediate nodes by looking at the packet. It then sends a QoS report to the source 3.2(b). The QoS report may take a route which is different than the one taken by the data packets as shown in 3.2(b).



(a) Reservation Setup



(b) QoS Reporting

Figure 3.2: INSIGNIA Reservation Setup and QoS Reporting

The reservations made at the intermediate nodes is maintained by periodically refreshing them. When packets are forwarded by a node, it automatically refreshes the reservation information for the flow to which the packet belonged. Hence a separate refresh related signaling need not be designed. This is called the soft-state approach to state management. As long as the information is used it is not discarded. Once state information is unused beyond a specific time interval, called the soft-state timer interval, it is discarded. In case of INSIGNIA, if a packet is not forwarded during a soft-state timer interval, the reservation information is discarded. This happens when a node moves away from a route and is not in radio contact with other nodes of the route.. The node that moves away discards its information after soft-state timer interval as it won't receive any packets for that flow. Soft-state timer interval may also be used to discard excess reservations made in the nodes that was made during the flow setup phase.

During the life-time of a flow, any of the nodes may scale up or scale down the current reservation. Let the current reservation be for MAX bandwidth level and let us say node $M_1$ degrades the flow to MIN level of bandwidth. The first packet that is forwarded using this level, will have its bandwidth indicator set to MIN. The bandwidth field will specify the bandwidth that was reserved at this level. When this is received by downstream nodes, they too degrade the reservation for the flow to MIN level. Finally when the destination received the first data packet at the MIN level, it informs the source in the next QoS report. The source, then may take different actions which include continuing with EL traffic as it may not mind the degradation of the flow, dropping all the EL traffic or stopping the flow itself. This depends on the instructions that the user has supplied to the application.

In wireless networks, apart from bandwidth constraints, energy and processing power is also an issue. To provide a reservation based scheme, finding a mapping algorithm which can set aside processing power and energy to meet specific bandwidth reservations may be difficult. Also, due to dynamic nature of ad hoc networks, a reservation based scheme may have significant overhead associated with maintaining reservation information in networks. Hence such a system does not scale well, when the number of nodes and mobility is high.

## 3.4 Summary

In this chapter, different schemes designed to provide adaptive feedback to applications in wireless and wired networks were considered. To improve the scalability of the system and to simplify the system design, a reservation based scheme for providing adaptive network feedback may not be suitable for wireless networks. In wireless routing protocols like AODV, due to the broadcast nature of route set up, bandwidth available at a node may show sudden, temporary dips. A bandwidth measurement scheme in the MAC or higher layer, can be inaccurate and it may not be able to detect such sudden dips. A more reliable method of detecting the actual bandwidth available to applications should be used is designing a network based feedback scheme.

# Chapter 4

# Congestion in Wireless Networks

The capacity of routes in a network is influenced by congestion conditions; as congestion develops in a region, it decreases the capacity of all routes that pass through that region. As congestion starts to develop in the network, corrective action should be taken by throttling all the adaptive applications. A lot of different metrics, like MAC delays, MAC drops, or route changes can be used to identify congestion. In this chapter, one such metric, occupancy levels of the buffers in the link layer, is used characterize and locate congestion in a wireless network. This is then used in NEFASA, to provide adaptive feedback to applications.

## 4.1 Contention and Route Capacity

A route can be made up of one or more intermediate nodes. The maximum effective capacity of a route, depends on the capacity of each intermediate node, which in turn is determined by various factors like topology, traffic patterns, transmission speeds and transmission technologies. As the capacity of every intermediate node varies, the maximum effective capacity of a route is equal to the capacity of the intermediate node with the smallest available capacity. This is analogous to a pipe which is not uniform (Figure 4.1). Some regions of the pipe have higher thickness

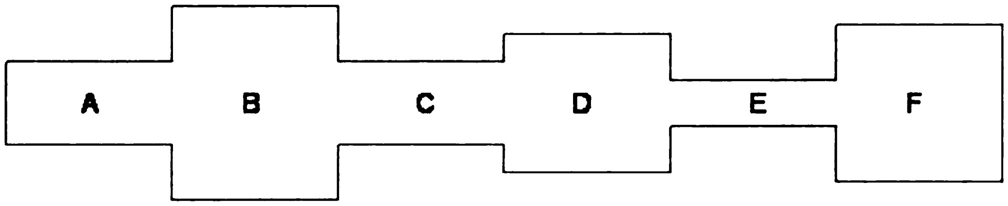


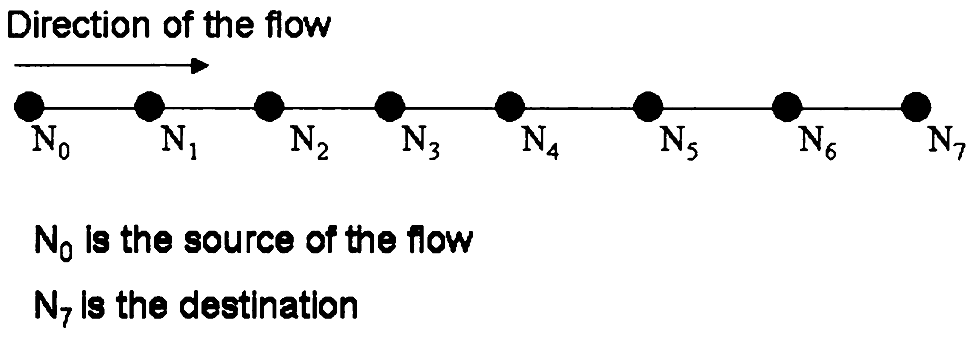
Figure 4.1: Pipe Analogy for Optimum Throughput

than others, but the maximum flow that the pipe can support is determined by the narrowest region of the pipe.

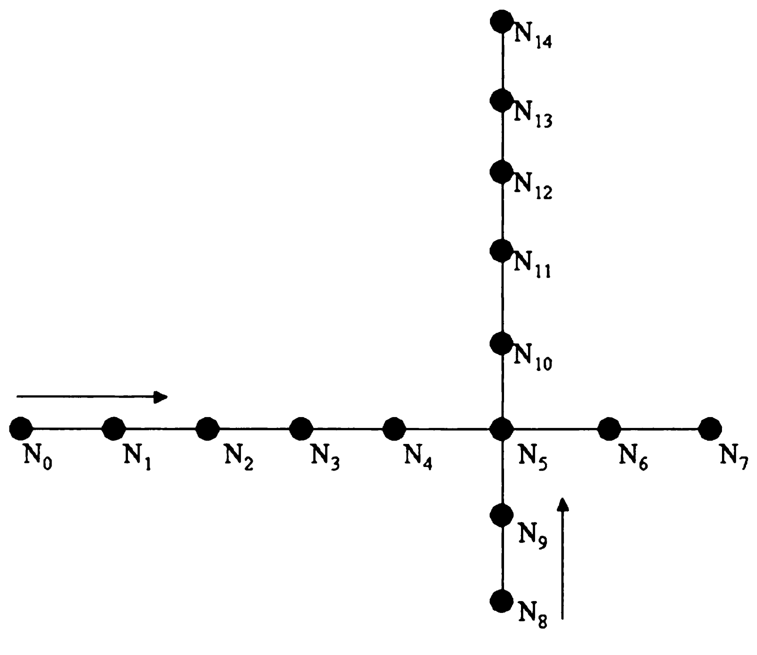
In a wireless network – where all the nodes share the same MAC and physical layer properties – the available capacity of a node is directly proportional to the amount of contention it experiences; higher the contention, lower the capacity. As the node with minimum capacity determines the maximum effective capacity of the route, contention determines the maximum effective capacity of the route.

To demonstrate the effects of contention on route capacity, consider a wireless network with a single flow (Figure 4.2(a)). It has eight equidistant (inter-node distance is 245MTS) nodes, which are arranged linearly. All the nodes in the network share the same MAC and physical layer properties. Only one flow – from source node $N_0$ to destination node $N_7$ – exists in the network. A *ns* simulation – with the application packet size of 512 bytes, carrier-sense range of 550MTS, and receive range of 250MTS – is conducted by increasing the load on the network and observing the PDP achieved by the network. If maximum effective capacity of a route is defined as, the maximum input load at which the PDP at the destination stays above 99%, 132Kbps (Figure 4.3(a)) is the maximum effective capacity for this route.

To see the effects of flow interaction in a linear, single flow network, let's add another flow on top of the existing flow (Figure 4.2(b)). The second flow starts at node $N_8$ and ends at node $N_{14}$. The two flows share the common node $N_5$. An experiment, similar to the one with the single flow, is performed by increasing the



(a) Single Flow Scenario



(b) Two Flows Scenario

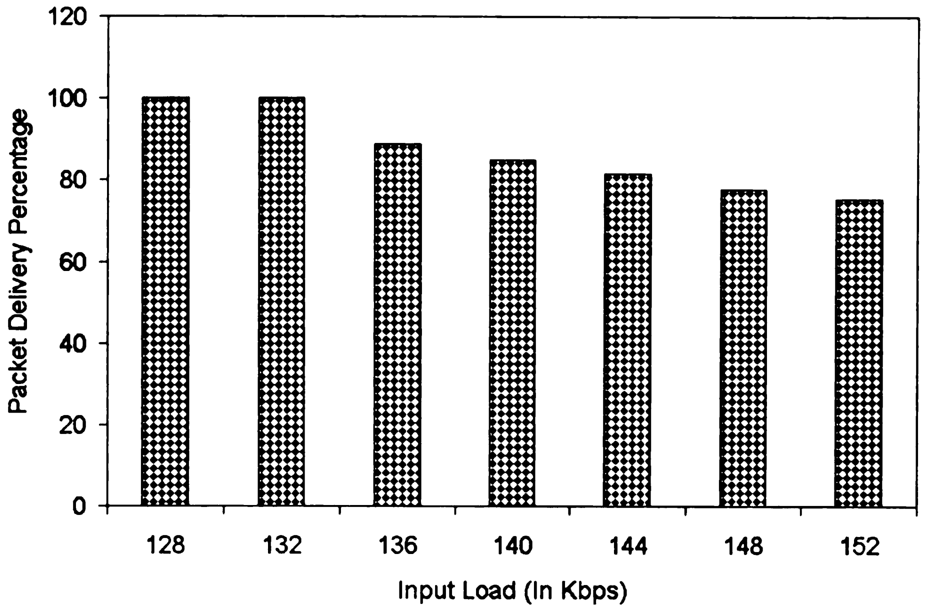
Figure 4.2: Influence of Contention on Route Capacity

input load on the network. The input load is increased by varying the rates of both the flows. If the maximum load (also optimum load) of a network is defined as, the load experienced by the network when all routes are operating at their maximum effective capacity, it can be seen that (Figure 4.3(b)) 136Kbps is the optimum load for this network. Of the 136Kbps, the flow between $N_0$ *and* $N_7$ contributes a load of 88Kbps. Because of the increased contention at $N_5$, the maximum effective capacity of this route is significantly less than than a similar flow in the single flow scenario. In the single flow scenario, the maximum effective capacity was 132Kbps. Hence it can be seen that with increasing contention the maximum effective capacity of a route decreases.

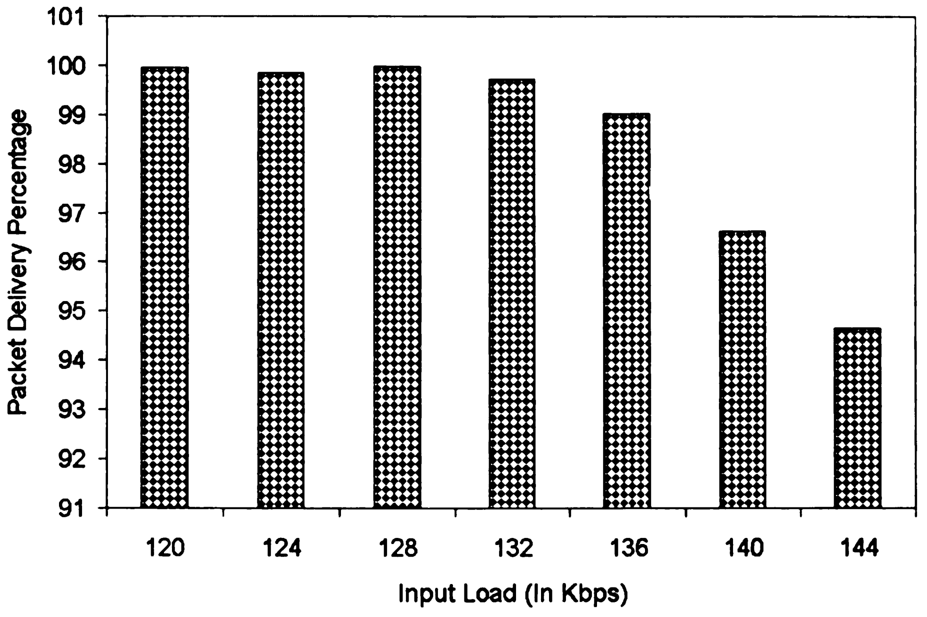
## 4.2 Congestion Identification

Once the input load exceeds the maximum effective capacity of a route, it manifests as a congestion in the network. The goal of NEFASA mechanism is to identify regions of congestion which develop in the network, and act upon them by providing adaptive feedback to applications. The challenge in developing in such an mechanism, is to reliably identify congestion by observing different parameters in the network. Care should be taken such that the mechanism work in networks with different topologies and traffic conditions and is not fine-tuned to specific types of networks.

One of the reliable measures of congestion is the occupancy levels of the buffers in the link layer. As the congestion in the network increases, the occupancy levels should increase. To observe this effect of congestion, in the single flow scenario, the effective load on the network is gradually increased from a level which is below the maximum loading conditions (found in Section 4.1) to loads above it. Once the input load crosses the optimum levels, $N_0, N_1, N_2$ *and* $N_3$ start to see a significant increase in the average queue length (Figure 4.4(a)). This indicates that average queue length



(a) Single Flow Scenario



(b) Two Flows Scenario

Figure 4.3: Max Capacity of Single and Two Flow Scenarios

can be used as an indicator of congestion. To check the validity of this, a similar experiment is performed in the two flow network. Nodes $N_2, N_3, N_4$ *and* $N_5$ see an increase in queue length once the loading conditions are above the maximum effective capacity of the route. This demonstrates that queue length can be a reliable measure of congestion in the network.

## 4.3 Region of Congestion

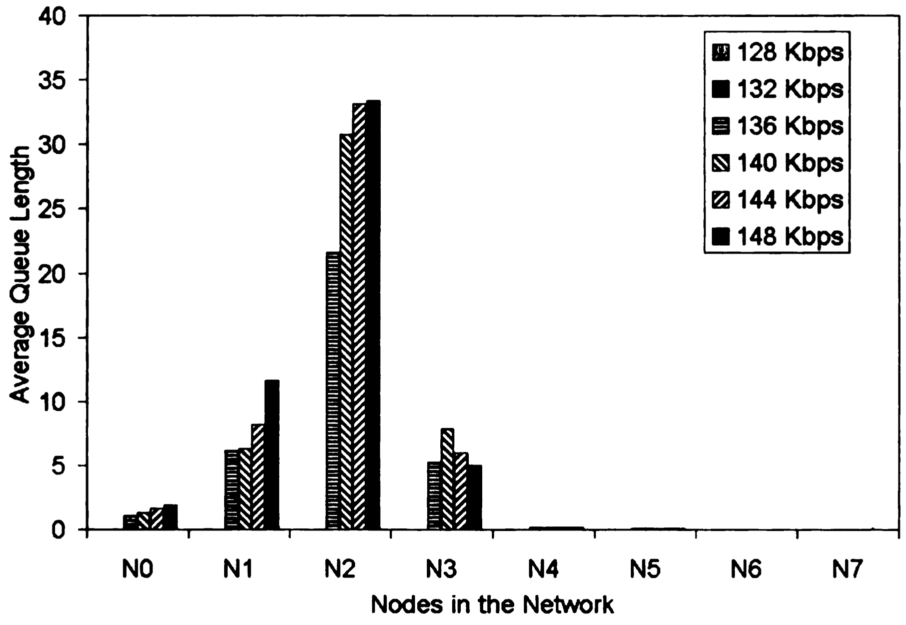
In the single and two flow scenarios, the region of congestion can be different. In the single flow scenario, as there is no cross-traffic and the topology of the network is uniform, the network cannot cause congestion. Hence the source can increase its rate till it crosses the limits imposed by the MAC and physical layer. Once these limits are crossed, the region in the vicinity of the source is expected to observe congestion. This is supported by the queue length data collected from the single flow scenario, as only nodes $N_0, N_1, N_2$ *and* $N_3$ see increases in queue length, and other downstream nodes hardly see any increase. Also $N_2$ sees the maximum amount of congestion, because it has to contend with four nodes. Two of these four nodes – $N_0$ and $N_1$ – are before $N_2$ and other two nodes – $N_3$ and $N_4$ – are after $N_2$ in the flow [1]. As $N_0$ and $N_1$ contend with only two and three nodes respectively, the amount of contention experienced by $N_2$ is higher than that of $N_0$ and $N_1$, and it sees more congestion. As $N_2$ drops packets when it is congested, nodes downstream to $N_2$ do not see any congestion.

In the two flows scenario, as the common node cannot treat both the flows as two independent flows, the common node is expected to be the bottleneck. But in wireless networks, because of the interference in the wireless channel, nodes which are a few hops before the common node can experience congestion. Hence, because of the

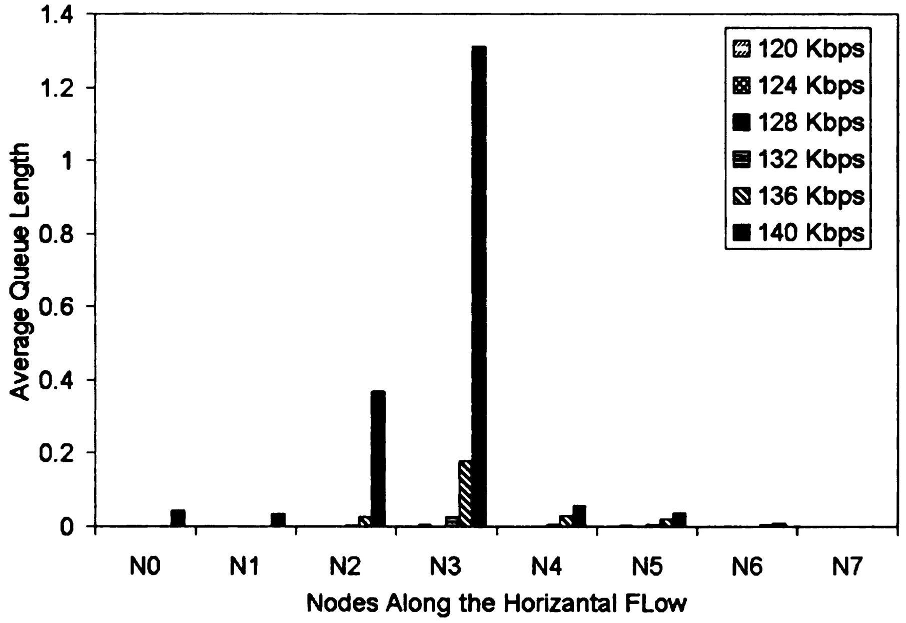
[1]This is for a linear network with inter-node distance of 245MTS, application packet size of 512 bytes, carrier-sense range of 550MTS and receive range of 250MTS

increased activity at the common node, $N_5$, nodes in its vicinity see an increase in queue length. The queue length data collected from the two flows scenario supports this, as nodes – $N_2, N_3, N_4$ – upstream to the common node see an increase in queue length with an increase in input loads. This also shows that the region of congestion has moved from the vicinity of the source to the vicinity of the common node in the two flows scenario.

The experiment with the two flows scenario shows that, congestion in the network can be near the source, or it can be away from it (due to cross-traffic). TCP and other end-to-end mechanisms, cannot detect such congestion because network end-to-end delays and packet drops may fail to detect their existence and if they do, they might fail to quantify the magnitude. Such information can be used in an network-based feedback architecture and queue length is not only used to detect congestion but also to accurately identify its region.



(a) Queue Lengths in Single Flow Scenario



(b) Queue Lengths in Two Flows Scenario

Figure 4.4: Identifying Congestion using Queue Lengths

# Chapter 5

# Network Feedback for Application Self Adaptation (NEFASA)

This chapter discusses Network Feedback for Application Self Adaptation (NEFASA) – the framework used to generate and deliver adaptive feedback to applications in a wireless network. NEFASA framework is comprised of three major components – the component used to generate the feedback, the component used to propagate the feedback and the adaptive applications. In NEFASA, congestion is used to generate the feedback for adaptive applications; queue length of the buffer queues are used for this purpose. To deliver the generated feedback, the feedback propagation is implemented in the network layer with the help of the routing protocol. For wireless ad-hoc networks, AODV is a popular and well-researched network protocol. NEFASA uses AODV to deliver the generated feedback to the adaptive applications. No major changes to AODV are done, and the existing mechanism used to generate and deliver RERR messages is re-used in NEFASA. As one of the important goals of NEFASA is to have a stateless feedback generation and propagation process, no state information is maintained within AODV during any part of the process. In Chapter 2, AODV was discussed in detail and this chapter discusses only the extensions related to the feedback propagation process. In addition, the default application behavior used in all future simulations is described at the end of this chapter.

## 5.1 Overview of the Three Components

Feedback protocols can be designed in various protocol layers. If applications have their own special needs, they can implement a specialized feedback protocol in the application layer. If many applications benefit from a specific type of protocol, the feedback mechanism can be implemented in the transport layer. An example of this is TCP, which is used in applications like FTP, HTTP and TELNET. Feedback protocols in the application and transport layers are quite common in wired networks, like the Internet. Network based feedback is not common and if used, it is designed to work with application and transport protocols[8].

One of the important issues in designing feedback protocols in the network layer is its scalability. A feedback protocol in the network layer, should scale well irrespective of the number of applications running on the network, and the amount of data that is exchanged between the applications. An important factor which determines scalability is the amount of feedback data produced and distributed in the network. As the feedback data is an overhead and decreases the amount of actual data that can be exchanged in the network, it is necessary that it is only a small fraction of the total amount of data that is exchanged in the network.

In order to achieve these goals, NEFASA is divided into three components (Figure 5.1). These are

1. Feedback Generation.

   This component is responsible for identifying congestion and starting the feedback propagation process when congestion is identified.

2. Feedback Propagation.

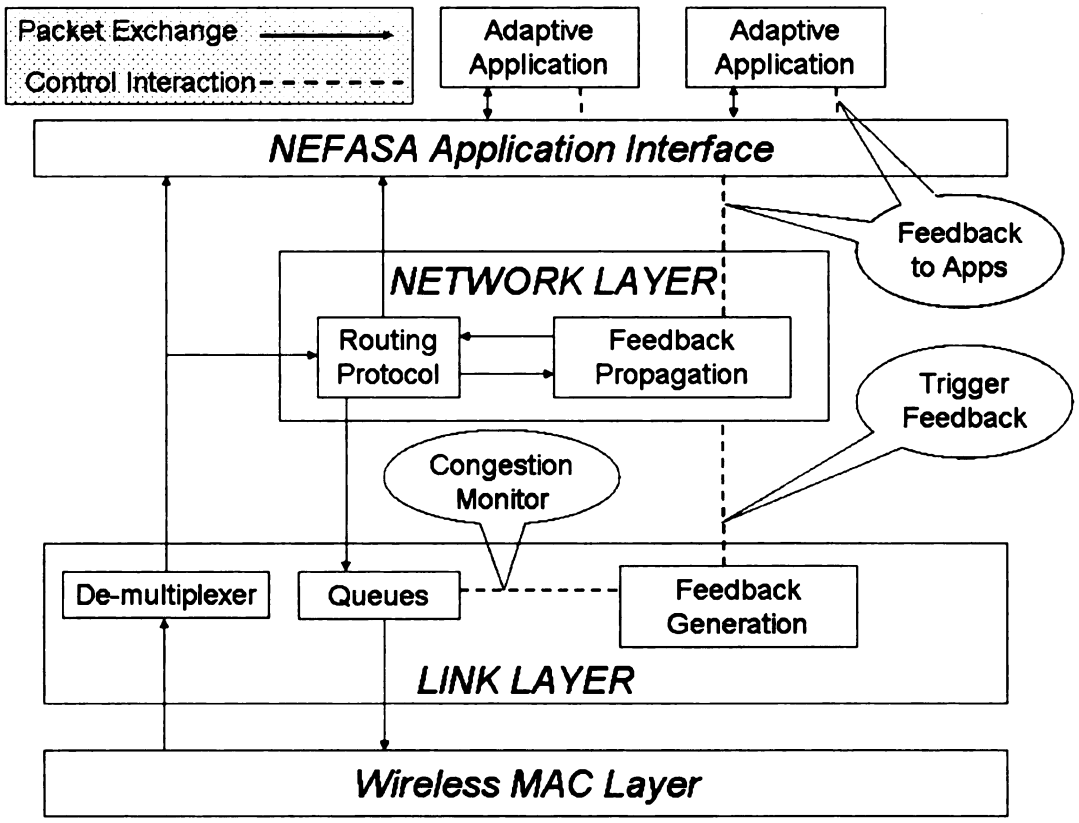


Figure 5.1: The Three Main Components of NEFASA

This component is responsible for propagating the feedback messages.

3. Adaptive Applications.

   The adaptive applications respond to the generated feedback by responding in ways defined by NEFASA.

### 5.1.1 Feedback Generation

In NEFASA, feedback generation is accomplished in the link layer (Figure 5.1), by identifying regions of congestion in the network. As described in Chapter 4, queue length of the link layer queues can be used to identify regions of congestion in a wireless network. To produce adaptive feedback, a threshold is placed on the queue length[1] and every time a packet enters the queue and the queue length is above the threshold, feedback is generated by the link layer queues.

Scalability is an important issue in the design of feedback protocols. The amount of feedback generated and its timing should be such that it controls congestion and does not aid congestion. The feedback generation mechanism plays an important role is deciding the scalability of a system. In Section 5.2 this is discussed in further detail, and it is shown that the queue length based feedback generation mechanism does control congestion and the system operates near optimal levels.

In order to aid the feedback propagation process, the single link layer queue is divided into many queues; a different queue is maintained for every neighbor of the node. A threshold is placed on every queue and if the threshold is crossed a feedback message is generated. Depending on the neighbor for which the feedback is generated, the propagation mechanism can use different methods to deliver the feedback to the applications.

One of the important design goals of NEFASA, is to produce a stateless mechanism of producing and distributing feedback messages. As the link layer queues do

[1]Queue length is measured in units of packets and not bytes

not maintain any state information related to the flows that pass through the node, the feedback generation method is stateless.

In NEFASA, the feedback generation method does not place any information in the feedback message. This increases the modularity of the whole system, as the feedback propagation method can be used with any method of generating feedback messages. This also de-couples applications from network design, as the applications now need not be modified every-time the network design changes.

### 5.1.2 Feedback Propagation

The component used to propagate feedback messages, is triggered by the component used to generate feedback messages (Figure 5.1). Once triggered, the feedback propagation component creates the feedback message and takes the responsibility of propagating it in the network. The feedback propagation happens in the network layer with the help of the network layer protocol. The basic idea is to identify all the sources that use the route for which the feedback is generated. This is done by using the precursors list maintained by AODV. Once the feedback propagation ends, all the sources responsible for the congestion along the route would have received the feedback. In AODV, NEFASA achieves this goal by using a mechanism similar to the one used to propagate RERR messages.

Many sources may be simultaneously using a route to a destination, and each of them may have different data rates. If the feedback is provided to all the applications, it may lead to under-utilization of network resources due to excessive throttling. To prevent this, a rate based mechanism is used to select a subset of high output rate applications. Once the feedback propagation ends, only the nodes which belong to this subset will receive the feedback.

### 5.1.3 Application Adaptation

Once the feedback propagation ends, the applications are informed by the feedback propagation component using the NEFASA application interface (Figure 5.1). In NEFASA, only negative feedback is generated by the network. If the applications do not receive any feedback from the network, they can assume that their current output rates are sustainable and do not lead to congestion in the network. Once the applications receive negative feedback, they take corrective action by throttling their output rates. If this does not remove the congestion, the network continues to provide feedback till the applications throttle their output rates to a level which does not cause congestion. The implicit assumption here is that applications are well behaved, i.e., they reduce their output rate when they receive feedback.

## 5.2 Feedback Generation

In order to generate feedback from the congestion information, a threshold is placed on the queue length. Every time a packet enters the queue and the queue length crosses the threshold, a feedback message is generated and propagated to the applications. The threshold used on queue length has an influence on application and network behavior. Also, generating feedback for every packet which crosses the threshold may lead to sub-optimal application behavior. These aspects of the feedback generation are characterized along with the feedback propagation method in Chapter 7 .

### 5.2.1 Optimality

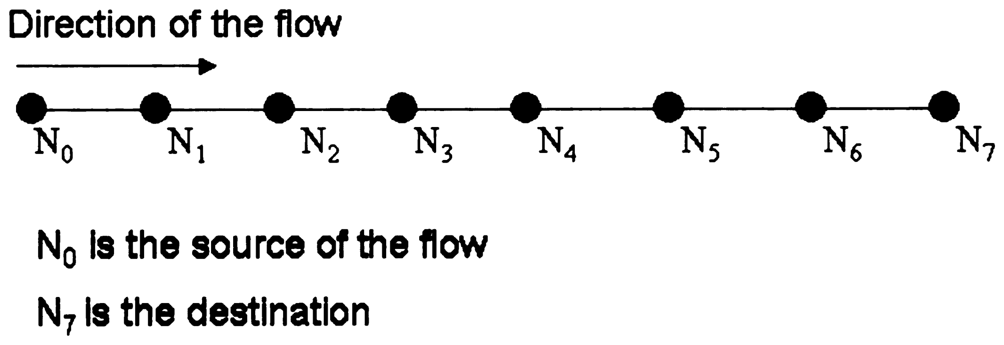
One of the important goals of any application feedback mechanism, is to ensure that the network operates in the region of optimal throughput. The mechanism used to trigger the feedback generation plays an important role in achieving this goal. If the triggering mechanism is too conservative, feedback may be generated long after congestion develops in the network. Because of less feedback from the network, and as a consequence, less throttling from the applications the load on the network may be above optimal levels. The PDP for such a network will be low as the network cannot deliver all the packets that it may receive.

If the triggering mechanism is too sensitive, more triggers than necessary may be generated and propagated. This will lead to excessive throttling of the applications and in some cases may aid congestion instead of alleviating it. The PDP for such a network will be high, but the network throughput will be below the optimal level. Hence it is necessary that the triggering mechanism is tuned well and is generic, so that it works in networks of different topologies and traffic patterns.

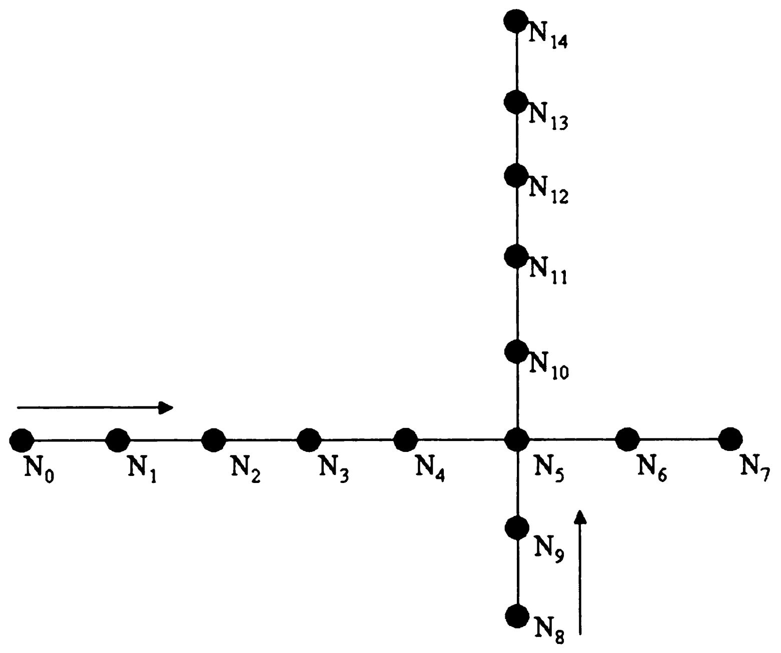
In simple scenarios, where the number of flows is less, the optimum loading conditions on the network can be manually found by trial and error. Once this optimum load is found, the effect of operating the network above and below optimum levels can be found out. Consider the single and two flows scenarios present in Figure 5.2. As there are at-most two flows, the optimal loading conditions on the network can be manually found out. A *ns* simulation is performed to find out the performance of NEFASA and the optimal loading conditions on these networks. The *ns* simulation uses application packet size of 512 bytes, carrier-sense range of 550 MTS and receive range of 250MTS. The inter-node distance is 245MTS.

Figure 5.3(a) compares the performance of NEFASA mechanism against the optimum throughput, for the single and two flow scenarios. NEFASA mechanism operates within 80% of the optimum levels. The PDP achieved by the NEFASA is comparable to the PDP achieved in the optimal case (Figure 5.3(b)). This shows that the feedback generation method is able to contain the congestion and does not aggravate the congestion by producing excessive feedback.

Figure 5.3 has two loads, *high load* and *low load*, which respectively are above



(a) Single Flow Scenario



(b) Two Flows Scenario

Figure 5.2: Optimality of Feedback Generation

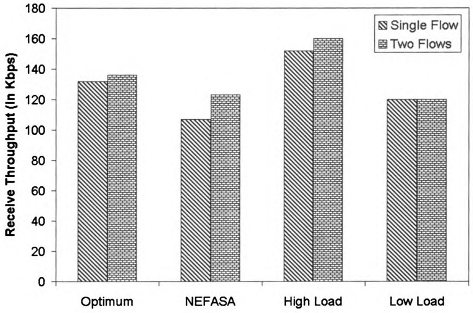
and below the optimum loading conditions. For *low load*, the PDP is high but the network is below the optimum load conditions. For *high load*, the network is above the optimum loading conditions, but the PDP is low. This demonstrates the effects of a conservative and very sensitive feedback mechanism.

### 5.2.2 Source of Feedback

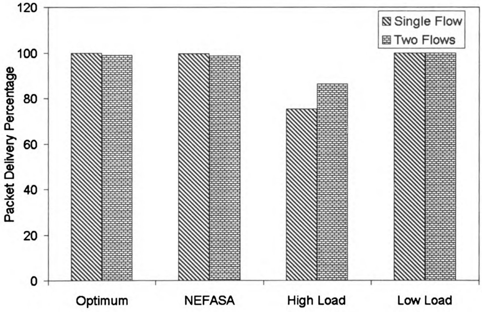
It is important that feedback be generated only from the congested nodes, and there are no false alarms. In order to verify this, the threshold based feedback generation, and the propagation mechanism which will be described in the later sections, are used in the single and two flow networks (Figure 5.2). Figure 5.4(a) shows the number of feedback messages generated from each node for the single flow network. Figure 5.4(b) has similar information, for the same flow, in the two flow network. For the single flow network, as the congestion is near the source, the feedback must be generated from nodes which are at the beginning of the flow. For the two flow network, the feedback must be generated from the vicinity of the common node, $N_5$, as this is the region of congestion. This is indeed the case as seen from Figure 5.4.

## 5.3 Feedback Propagation

The basic idea used in the propagation of feedback messages, is to deliver the feedback message to all the sources which use a congested route. A route gives the next-hop which is to be taken to reach a destination; this helps in forwarding packets from the source to the destination. But with additional help from the routing protocol, not only can packets move in the forward direction of a route, the packets can also flow in the reverse direction and trace their way back to all the sources which use the route. This is not used for data packets, but it may be of use for implementing some functions of the routing protocol.

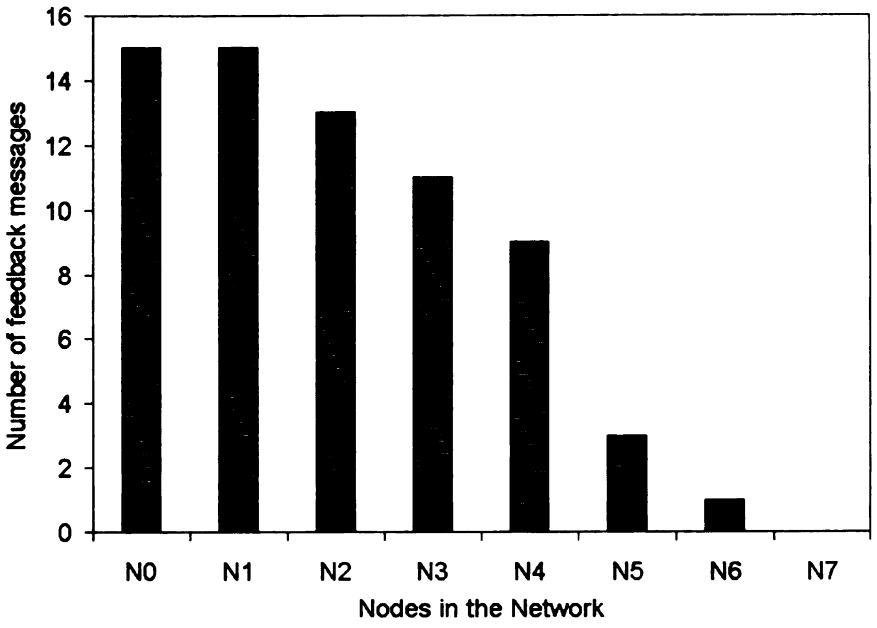


(a) Throughput Comparison

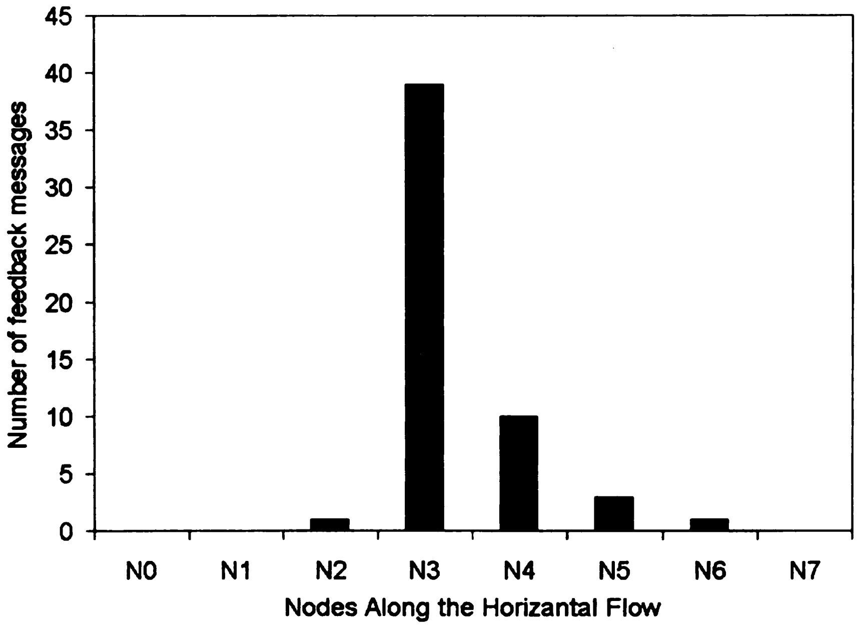


(b) Corresponding PDPs

Figure 5.3: Optimality of NEFASA Mechanism



(a) Feedback Generation in Single Flow Scenario



(b) Feedback Generation in Two Flows Scenario

Figure 5.4: Feedback Generated from Different Nodes

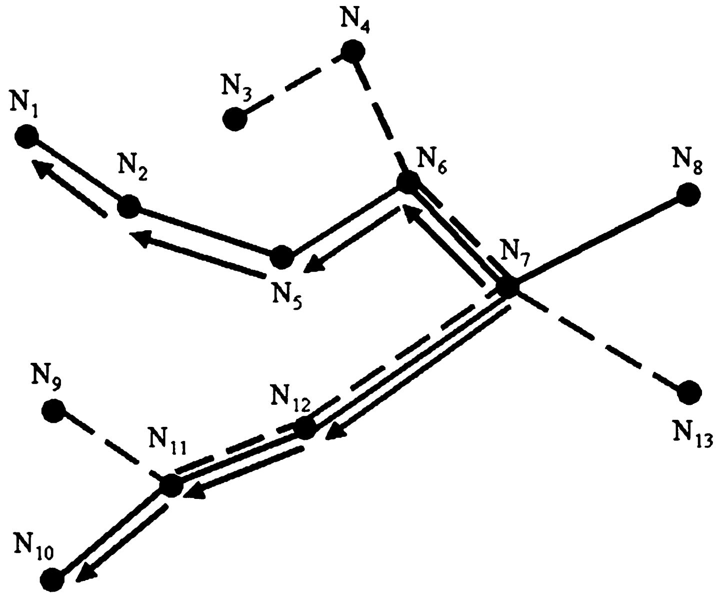


Figure 5.5: Tracing Back All the Sources of a Route

For example, consider the wireless network in Figure 5.5. It has two routes – one to destination $N_8$, which is in solid lines, and the other to destination $N_{13}$, which is in dotted lines. If $N_7$ wants to trace back the routes to reach all the sources which pump data to $N_8$, it can do so by sending packets along the direction which is marked in arrows in Figure 5.5. In a routing protocol like AODV, the mechanism to accomplish this (in the form of precursors list), is already in place and it used in NEFASA to propagate the feedback messages.

### 5.3.1 Precursors List

In AODV, every node in the network maintains a list of precursors for every route that passes through it. The precursors list is set-up during the route set-up phase, when the request and reply messages are exchanged. The routing table has an entry for every route that it maintains. The precursors list can be found in every entry of the routing table.

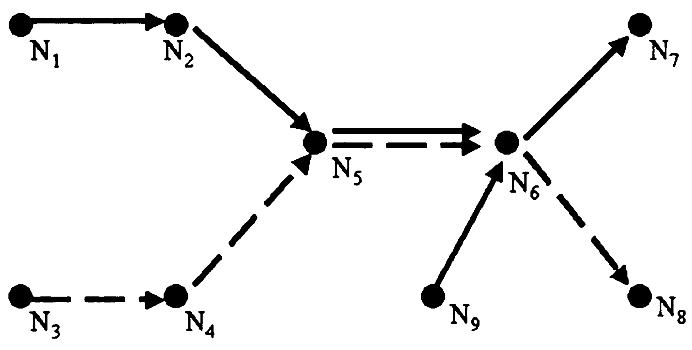


Figure 5.6: Wireless Network with Two Routes

The precursors list for a node maintaining a route, is the subset of its neighbors which use itself as the next-hop to forward packets to the destination of the route. Consider the network in 5.6. It has three flows in it; they are from $N_1$ to $N_7$, $N_2$ to $N_8$ and $N_9$ to $N_7$. As there are only two destinations, there are only two routes in the entire network – one to $N_7$ which is in solid lines and the second to $N_8$ which is in dotted lines. Table 5.1 gives the list of precursors maintained by all the nodes in the network.

1. As nodes $N_1$ and $N_9$ send data to destination $N_7$, they maintain a single route to the destination $N_7$. For the destination node $N_7$, they do not receive any packets from their neighbors; hence their precursors list for $N_7$ is empty. As nodes $N_1$ and $N_9$ do not generate or forward packets to node $N_8$, they do not have a routing table entry for $N_8$.

2. Node $N_3$ is similar to nodes $N_1$ and $N_9$, but it only maintains a route to destination $N_8$. The precursors list for this route is empty because, none of its

$$\begin{tikzcd}
 & N_6 \arrow[dl] \arrow[dr] & \\
N_5 \arrow[dr] & & N_9 \\
 & N_2 \arrow[dl] & \\
N_1 & &
\end{tikzcd}$$

(a) Precursor Tree for Destination $N_7$ from $N_6$

$$\begin{tikzcd}
 & & & N_6 \arrow[dl] \arrow[dr] & \\
 & & N_5 \arrow[dl] \arrow[dr] & & N_9 \\
 & N_2 \arrow[dl] & & N_4 \arrow[dr] & \\
N_1 & & & & N_3
\end{tikzcd}$$

(b) Precursor Tree for Destination $N_7$ and $N_8$ from $N_6$

Figure 5.7: Precursor Trees

| Node | Precursors for Destination $N_7$ | Precursors for Destination $N_8$ |
|---|---|---|
| $N_1$ | $\emptyset$ | N/A |
| $N_2$ | $\{N_1\}$ | N/A |
| $N_3$ | N/A | $\emptyset$ |
| $N_4$ | N/A | $\{N_3\}$ |
| $N_5$ | $\{N_2\}$ | $\{N_4\}$ |
| $N_6$ | $\{N_5, N_9\}$ | $\{N_5\}$ |
| $N_7$ | N/A | N/A |
| $N_8$ | N/A | N/A |
| $N_9$ | $\emptyset$ | N/A |

Table 5.1: Precursors List for the network in Figure 5.6

neighbors forward packets to $N_8$, through it.

3. Node $N_2$ maintains a route to the destination $N_7$, as it forwards packets received from node $N_1$. Hence the precursors list for destination node $N_7$, as maintained by $N_2$, has $N_1$ as its only element. As node $N_2$ does not maintain a route to $N_8$, there is no precursors list for node $N_8$.

4. Node $N_4$ is similar to node $N_2$, and it maintains $N_3$ as its precursor for destination $N_8$.

5. Node $N_5$ receives packets for both destinations $N_7$ and $N_8$; hence it maintains a route for both the destinations. As only $N_2$ forwards packets to destination $N_7$ through $N_5$, only $N_2$ is in the precursors list for destination $N_7$. Similarly, $N_5$ only maintains $N_4$ in its list of precursors for $N_8$.

6. Node $N_6$ maintains routes to both destinations $N_7$ and $N_8$. Its precursors list is similar to $N_5$, with $N_9$ added to the precursors list for destination $N_8$.

7. As $N_7$ and $N_8$ do not communicate with any other nodes in the network, they do not maintain any routes in their routing tables. Hence they do not maintain any precursors list.

### 5.3.2 Precursor Trees

The intention behind maintaining the precursors list is to trace back the route from an intermediate node, to all the upstream (w.r.t the direction of the route) sources which use the route. For example, in the network shown in Figure 5.6, if the route to destination node $N_7$ is traced back from the intermediate node $N_5$, the precursors list comes handy. In the routing table maintained by $N_5$, the only precursor for destination $N_7$ is $N_2$; and in the routing table of $N_2$, the only precursor for destination $N_7$ is $N_1$. Hence from $N_5$, the route can be traced back as, $N_5$-$N_2$-$N_1$.

All precursor lists need not have a single neighbor as their precursor. If there are multiple nodes in the precursor list, it leads to a tree-like structure when the route is traced back. In the routing table for $N_6$, for the destination $N_7$, there are two precursors, $N_9$ and $N_5$. When the route is traced back, a tree-like structure as shown in Figure 5.7(a) is formed.

A precursor tree only gives information about upstream (w.r.t the direction of the route) sources and intermediate nodes. There may be many more sources which can be reached from the precursors list of the downstream nodes. The following properties can be inferred about a precursor tree.

1. The precursor tree can be uniquely identified by the intermediate node from which the tracing is started, and the destination nodes for which the tracing is done.

2. The root of the tree, is the intermediate node from which the tracing is started.

3. All the leaves are the upstream source nodes. The source nodes have at-least one application which sends data to the destination node of the precursor tree.

4. All the leaves are upstream sources, but it is not necessary that they are the *only* upstream sources. The non-leaf nodes, including the root, can have applications which send data to the destination node of the precursor tree.

5. Precursor trees can be drawn for multiple destinations. As the number of destinations increase, the number of nodes in tree increases. Figure 5.7(b) shows the precursor tree for two destinations, $N_7$ and $N_8$.

In AODV, the precursor tree is used by the RERR propagation mechanism to dismantle an existing route. If $N_7$ becomes unreachable due to mobility, $N_6$ will discover this when it fails to deliver a packet. It will then start the RERR propagation mechanism to dismantle all the routes which use $N_7$ as the next-hop. As there are two routes which $N_6$ hosts, and only one of them – the route to destination $N_7$ – uses $N_7$ as the next hop, $N_6$ starts the dismantling process by sending the RERR message along the precursors of this route. As $N_5$ is the only precursor along this route, after it receives the RERR message from $N_6$, it takes over the responsibility of completing the RERR propagation. It sends the RERR message to $N_2$, which is in its precursors list for destination $N_7$. When the RERR propagation mechanism terminates, it would have reached all the nodes in the precursor tree for the intermediate node $N_6$ and destination $N_7$, and the route would have been dismantled.

### 5.3.3 RCHANGE Messages

In NEFASA, whenever a node identifies itself as a congested node – using the threshold based mechanism described in the previous chapter or some other mechanism – it triggers the generation and propagation of a feedback message called Route Change (RCHANGE). The propagation of RCHANGE is very similar to that of the RERR messages, which was described in detail in Section 2.2.2; however, the intention here is not to dismantle the route, but to inform all the sources about the congestion in an intermediate node. When the nodes get the RCHANGE message, they take corrective action by throttling their data rates.

Similar to the RERR propagation algorithm, RCHANGE is propagated with the help of two different algorithms; the generation (Algorithm 6) and the forwarding

| Field | Explanation |
|---|---|
| *rt* | routing table entry in the node |
| *dst(rt)* | The node for which *rt* gives the route |
| *nexthop(rt)* | The next hop that is used to reach *dst(rt)* |
| *dst_seqno(rt)* | The sequence number of *dst(rt)* as maintained in *rt* |
| *precur(rt)* | The list of precursors for the route *rt* |

Table 5.2: Fields in a Routing Table Entry

| Field | Explanation |
|---|---|
| *rch* | RCHANGE message |
| *src(rch)* | source of the RCHANGE message, *rch* |
| *rchange_dsts(rch)* | List of destinations for which the RCHANGE message, *rch*, is generated |
| *seqnos(rch)* | List of sequence numbers of the destinations present in *rch* |
| *dst_count(rch)* | Number of destinations reported in *rch* |
| *intended_precursor(rch)* | The neighbor which is supposed to process *rch* |
| *rchange_seqno(dst,rch)* | Sequence Number of the destination, *dst*, which is listed in *rch* |

Table 5.3: Terms used in Algorithm 7

algorithm (Algorithm 7). The generation algorithm is triggered when the congestion detection algorithm, as described in Chapter 4, detects a congestion. In the case of NEFASA, the congestion detection algorithm runs in the link layer. The information that is gathered in the link layer, triggers the network layer to start the feedback generation and propagation process. This is an example of cross-layer interaction that can be found in NEFASA.

#### 5.3.3.1 Generating RCHANGE Messages

The algorithm to generate the RCHANGE message is very similar to the RERR generation process; the only difference being, the source may run an algorithm to select a subset of precursors to which it sends the RCHANGE message. In contrast, the RERR generation protocol sends the RERR message to all the precursors. The

```
//The different terms used in this algorithm are explained in
//Table 5.2 and Table 5.3

//Collect all the precursors in set_of_precursors

forall routes, rt, that are present in the routing table and are usable do
    if precur(rt) is not empty then
        set_of_precursors ← set_of_precursors ∪ precur(rt)
    end
end

//Prune the set of precursors using the select_subset algorithm

subset_of_precursors ← select_subset(set_of_precursors)

if subset_of_precursors is not empty then
    //Form a new RCHANGE message for each selected precursor and
      broadcast it

    foreach precursor, i, in the set subset_of_precursors do
        write i to intended_precursor(new_rchange)

        write the destinations, which the precursor i tries to reach and are
        listed in recd_rchange, to the new_rchange message

        write the sequence numbers of the destinations found in the previous
        step the new_rchange

        set dst_count field in the new_rchange message

        broadcast the new_rchange message in the MAC
    end
else
    Stop further processing and return from the function
end
```
**Algorithm 6**: Generating a RCHANGE message

motivation for doing this, and the various algorithms that can be used to select the subset of precursors will be discussed in the later sections.

The RCHANGE message contains the following fields. The *intended_precursor*, contains an address which is used by the neighbors which receive the RCHANGE message; the neighbors process the RCHANGE only if their address is present in this field. The RCHANGE message, also has a list to identify the destinations for which the RCHANGE is triggered. This list is called *rchange_dsts* and the number of elements in this list is present in *dst_count(rch)*. The sequence numbers corresponding to the destinations listed in *rchange_dsts* is present in the field *dst_seqnos*.

The RCHANGE generation process consists of three steps. In the first step, the set of routes for which the RCHANGE has to be generated is identified. In NEFASA, it is generated along all the routes.

Once the routes are figured out, in the next step, the precursors to which the RCHANGE has to be sent is decided. The RCHANGE generation algorithm starts out by collecting the precursors of all the routes. Once all the precursors are known, an algorithm to prune the precursors is executed. Once the pruning of the precursors is completed, a subset of precursors to which RCHANGE has to be sent would have been selected.

In the third and final step, the RCHANGE message itself is formed by placing all the information discussed previously. One RCHANGE message is formed for each selected precursor. The precursor's address is placed in the field *intended_precursor*. All the destinations which the precursor (for which the RCHANGE is being generated) tries to reach and are listed in the received RCHANGE, are placed in the field *rchange_dsts*. Other fields in the RCHANGE message are filled out similarly. Once this is done, the RCHANGE message is broad-casted on the MAC.

```
//The different terms used in this algorithm are explained in
//Table 5.2 and Table 5.3
if this node is in the intended_precursors_list(recd_rchange) then
    for all routes, rt, that are present in the routing table and are usable do
        if dst(rt) is listed in recd_rchange and dst_seqno(rt) $\leq$
        rchange_seqno(dst,recd_rchange) and nexthop(rt) = src(rchange) and
        select(precur(rt)) is not empty then
            set_of_precursors $\leftarrow$ set_of_precursors $\cup$ precur(rt)
        end
    end
    //Prune the set of precursors using the select_subset algorithm
    subset_of_precursors $\leftarrow$ select_subset(set_of_precursors)
    if subset_of_precursors is not empty then
        //Form a new RCHANGE message for each selected precursor and
          broadcast it
        foreach precursor, i, in the set subset_of_precursors do
            write i to intended_precursor(new_rchange)
            write the destinations, which the precursor i tries to reach and are
            listed in recd_rchange, to the new_rchange message
            write the sequence numbers of the destinations found in the previous
            step the new_rchange
            set dst_count field in the new_rchange message
            broadcast the new_rchange message in the MAC
        end
    else
        Stop further processing and return from the function
    end
end
```

**Algorithm 7**: Forwarding an RCHANGE message

#### 5.3.3.2 Forwarding RCHANGE Messages

As the RCHANGE message is broadcasted in the MAC layer, neighbors which are not in the precursors list may get the RCHANGE message. When a neighbor receives the broadcasted RCHANGE message, it first looks in the *intended_ precursor* field before processing the message; only if its address is present there, the node processes the RCHANGE message.

The node receiving the RCHANGE message follows the same three steps – identify the routes, select the precursors, form the RCHANGE message – that was used in the generation process. The only difference is in the first step, when it looks for the routes along which the RCHANGE message has to be propagated. Instead of generating the RCHANGE message for all the routes, it looks at the list of all the destinations that are present in the received RCHANGE. If for any of these destinations it uses the source of the RCHANGE message as the next-hop, it uses the routes of only these destinations to propagate the RCHANGE message. It then follows the precursor selection step as described for the RCHANGE generation phase. It then completes the RCHANGE propagation by forming a new RCHANGE message and broadcasting it on the MAC.

The feedback propagation ends when the pruning of the precursors leads to an empty list. Before the RCHANGE propagation terminates, it would have visited a lot of nodes which could have sources causing the congestion. When the node receiving the RCHANGE message runs the algorithm to select the subset of precursors, and finds its own address in the subset that is selected, the RCHANGE message is delivered to all the applications which use the destinations present in the RCHANGE message. This is another example of cross-layer feedback which is generally not found in transport and application layer feedback protocols.

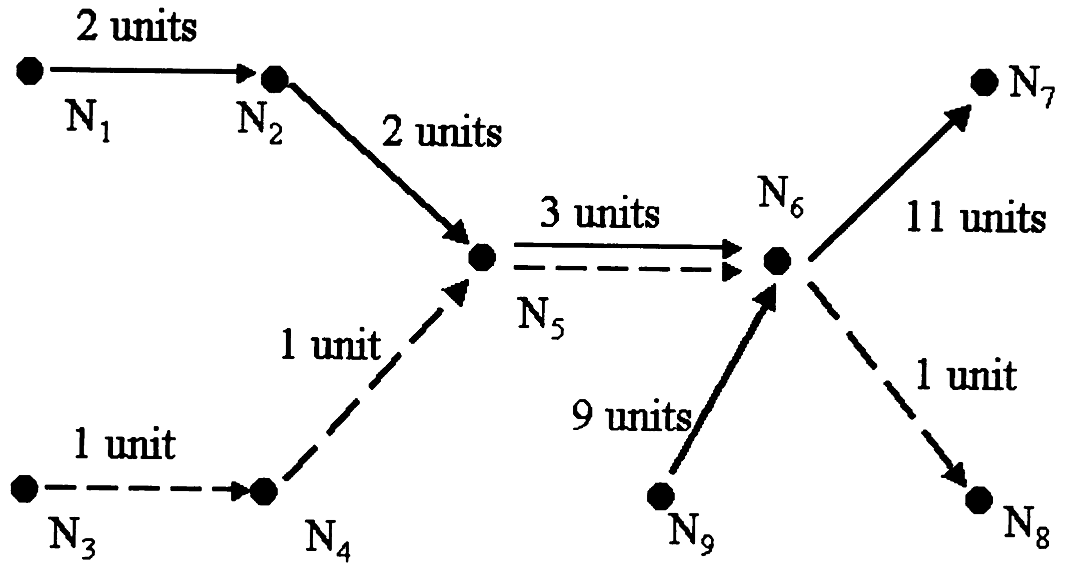
The RCHANGE message itself does not have any information which can be used by the application. The only intention of the RCHANGE message is to alert the applications about an impending congestion, and request them to take corrective action. Because the RCHANGE message does not carry any information that is specific to the mechanism which triggered the feedback generation process, in NEFASA the feedback generation and triggering are strongly de-coupled. The RCHANGE method of propagating feedback can be used equally well with any method of triggering the feedback generation process (for example, MAC delays, MAC drops etc.).

One of the important goals of the NEFASA mechanism, is to have a stateless mechanism of generating and propagating feedback messages. During the feedback generation and propagation phase, only information available from the AODV protocol is used. As AODV does not maintain any information specific to flows, the RCHANGE algorithm is completely stateless and distributed.

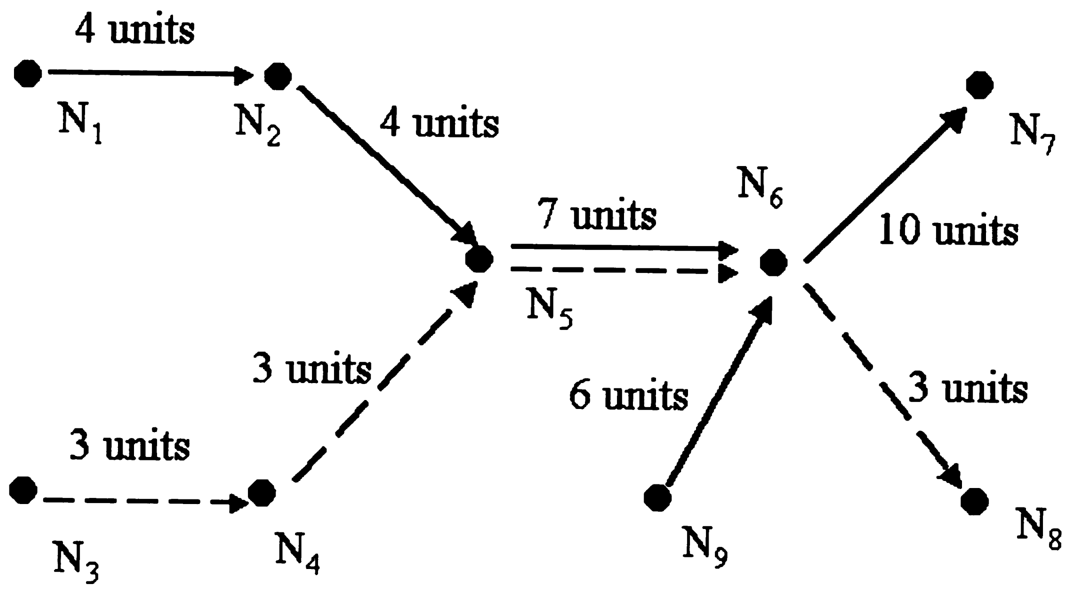
### 5.3.4 Rate Based Pruning

If the RCHANGE messages are propagated like RERR messages, all the upstream sources will receive the feedback from the network. This will lead to aggressive throttling of many applications, and in some cases it may be unfair to applications with small output rates. Consider the scenario presented in Figure 5.8(a), where source nodes $N_1$, $N_3$ and $N_9$, have rates of two, one and nine units respectively. Because of the high loads in the neighborhood of $N_6$, when $N_6$ sends a RCHANGE message, it will reach the sources $N_1$,$N_3$ and $N_9$ and throttle all of them. Throttling $N_1$ and $N_3$ may not be necessary as the congestion is being caused by $N_9$.

With this in mind, a rate based RCHANGE propagation method is used to prune the precursor tree and throttle only those applications with high output rates. To implement the rate based pruning policy, every node in the network estimates the rate at which each of its neighbor's sends data to itself. The rate estimation can be based on a simple scheme, which calculates the number of packets – if the size of all the packets in the network are the same – that are received from the neighbor within



(a) Case A



(b) Case B

Figure 5.8: Pruning of RCHANGE Messages Based on Rates

a specific interval. A history based rate estimator can be implemented, which takes into consideration the rates of the previous intervals when calculating the rate of the current interval. The rate estimator is best implemented in the network layer, as only AODV uses these estimates. Two different cases of rate calculations for the network in Figure 5.6 is shown in Figure 5.8.

Once every node in the network has an estimate for each of its neighbors, the rate data can be used to prune the precursor tree during the RCHANGE propagation. Pruning is done by every node when it receives or generates an RCHANGE message. Every node in the network will use the rate data to select only a few precursors with the highest rates. The RCHANGE message is then sent to only these precursors.

One of the policies that can be used to prune the RCHANGE messages is to select the precursor with the highest rate. Consider the scenario presented in Figure 5.8(a); if $N_6$ triggers the RCHANGE message, $N_9$ will receive the feedback immediately and no other node will receive the RCHANGE message. This will be more fair to small rate applications – like $N_1$ and $N_3$ – which are not the real cause of the congestion.

Selecting the precursor with the highest rate does not necessarily lead to selecting the node – among-st only the upstream sources – with the highest rate. Consider the scenario presented in Figure 5.8(b). If $N_6$ triggers the RCHANGE, $N_5$ will receive the RCHANGE message first, as it is the precursor with the highest rate. $N_5$ then forwards the RCHANGE message and this will eventually throttle $N_3$, when the propagation of the RCHANGE message ends. Hence $N_3$, which has a lower output rate than $N_9$ gets the RCHANGE message.

## 5.4 Application Behavior

In order to characterize and understand the RCHANGE propagation and threshold based feedback triggering mechanism, a simple, greedy application behavior is defined and used.

All applications start (Algorithm 8) with a very small output rate which would not cause congestion. When congestion conditions develop in the the network, only negative feedback is given to the applications. Hence it is left to the discretion of the applications to increase their data rate. The default application behavior defined for NEFASA, increments its rate every *N*secs, which is called the increment interval. The applications can do so with the help of a timer which is set to go off once every *N*secs. Applications can choose any appropriate value for the increment interval; the NEFASA framework does not impose any constraints on it and only characterizes the performance that could be expected with different values of increment interval.

As an RCHANGE message indicates congestion in the network, the applications should not wait before reducing their data rate. Hence when a RCHANGE message is received (Algorithm 9) , the applications immediately reduce their output rates. The application may receive multiple RCHANGE messages within a short interval. This happens when the reduction in the data rate is not sufficient to reduce the congestion in the network. Hence an application decreases its data rate every time it receives an RCHANGE, even if they arrive within a short interval of time. The applications also keep track of the number of RCHANGE messages that have been received in an increment interval.

At the end of an increment interval (Algorithm 10) , the applications increase their output rate, only if they have not received any RCHANGE messages during that interval. If an RCHANGE message has been received during the increment interval, the application does not increment its data rate and waits for the congestion in the network subside before incrementing its output rate. It does so only at the next increment interval during which it does not receive any RCHANGE messages.

```
Start a timer to go off every $N$ secs
Set the initial rate of the application
```

**Algorithm 8**: Initialization Performed by an Application in NEFASA

```
Decrease the data rate (The magnitude of decrease can be application specific)
RchangeCount $\leftarrow$ RchangeCount + 1
```

**Algorithm 9**: Negative Adaptations by an Application in NEFASA in Response to RCHANGE

```
if RCHANGECount is zero then
    Increase the data rate (The magnitude of increase can be application
    specific)
else
    Do not Increase the data rate
    RchangeCount $\leftarrow$ 0
end
```

**Algorithm 10**: Positive Adaptations by an Application in NEFASA

# Chapter 6

# Performance

In this chapter the performance of the NEFASA framework is studied. These comparisons are done in scenarios with varying complexities; starting from simple scenarios like a linear topology network with one or two flows, and in complex scenarios where multiple flows are present in an arbitrary mesh topology. For all the simulations in this chapter, optimum values of the various parameters of the NEFASA framework – disable period of two seconds, threshold of 2% and application increment interval of ten seconds, as discussed in Chapter 7 – are used.

## 6.1 Linear Networks

If the number of flows in a network is small, the optimal loading conditions on the network and the throughput of the individual flows can be manually found out. This is done by trial and error, by performing many simulations. The rate of all flows is held constant in each simulation. The load on the network is progressively increased with each experiment, till the maximum loading conditions on the network is reached. The throughput of the individual connections at this point gives the maximum throughput for these connections.

In a wireless network, even under low-load conditions there may be losses due to

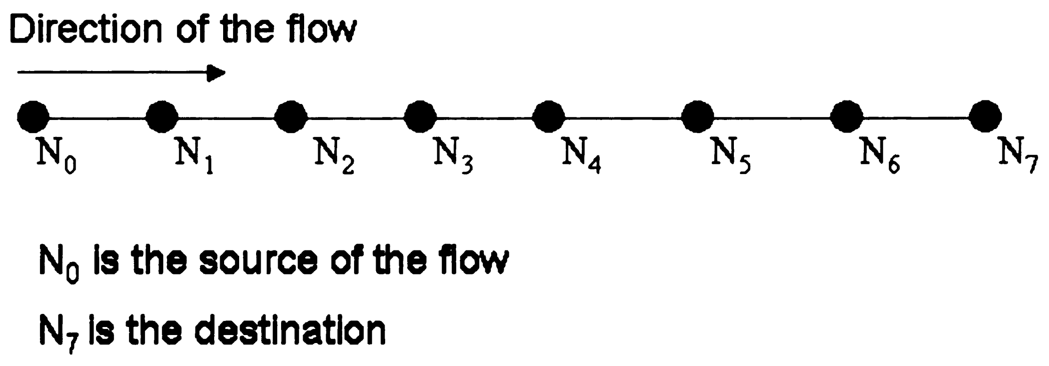


Figure 6.1: Linear Topology Network with a Single Flow

contention. Hence even under optimum loading conditions, losses in the network can be expected.

Once the optimum loading on a network is found out, the NEFASA feedback mechanism is used for the same flows and the throughput and PDP are found out.

In order to find out the optimum loading conditions on a network, two scenarios are considered; one with a single flow and the other with two flows. The single flow scenario uses the topology in Figure 6.1; $N_0$ acts as the source and $N_7$ acts as the destination. The two flow scenario is tried in four different topologies as shown in Figure 6.2 (in page 68) to Figure 6.5 (in page 69). In all the four topologies, the first flow is from $N_0$ to $N_7$ and the second flow is from $N_8$ to $N_{14}$. Hence we have five different configurations – linear, two flows-1, two flows-2, two flows-3 and two flows-4 – corresponding to the topologies in Figure 6.1 and Figure 6.2 (in page 68) to Figure 6.5 (in page 69).

Figure 6.6 shows the throughput and PDP achieved in the optimum loading conditions and the NEFASA mechanism. It can be seen that the performance of NEFASA is comparable to optimal loading conditions in the linear and two flows scenario. In the two flows scenario, the throughput (Figure 6.6(a)) levels are within

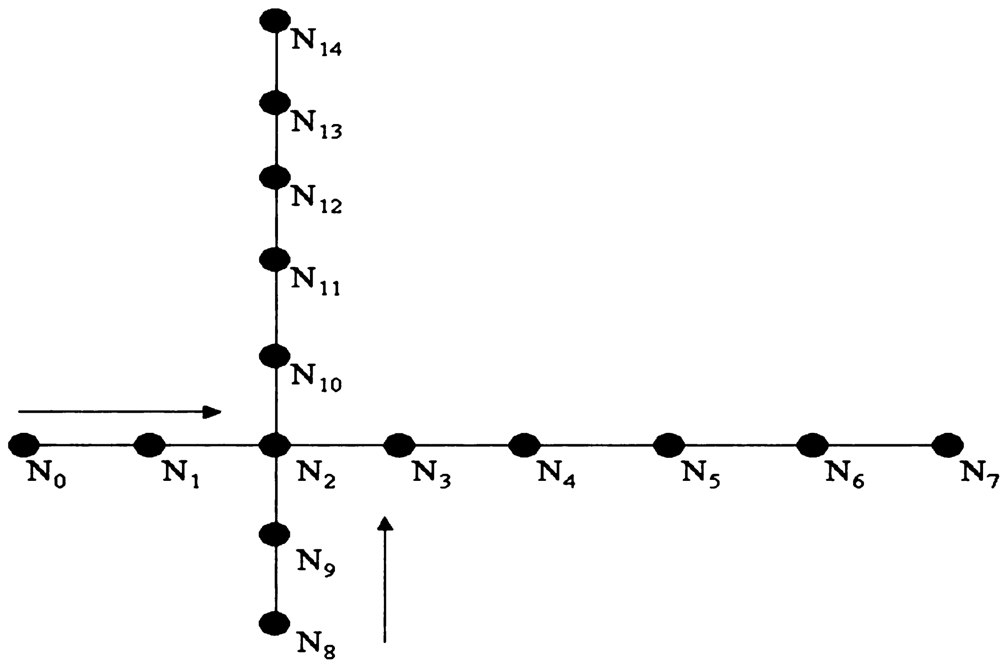


Figure 6.2: The First Configuration of the Two Flow Scenario

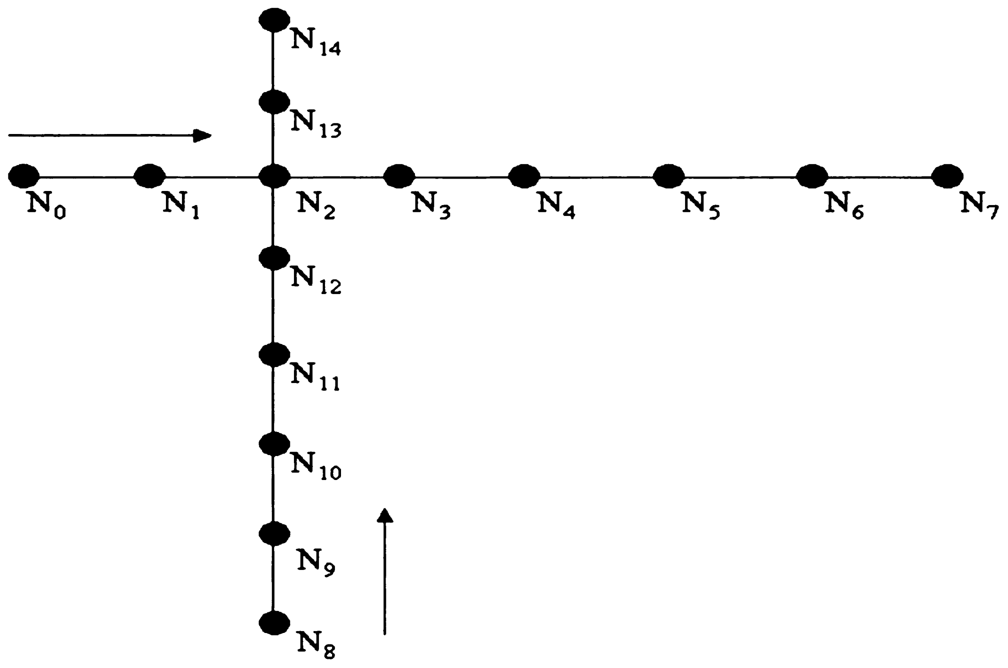


Figure 6.3: The Second Configuration of the Two Flow Scenario

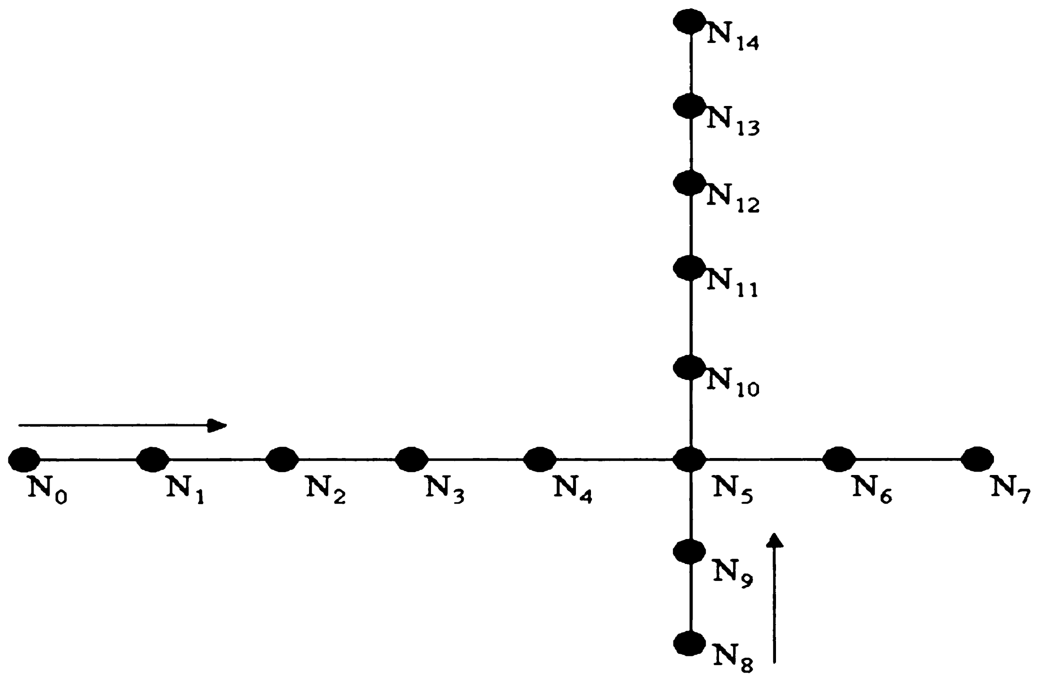


Figure 6.4: The Third Configuration of the Two Flow Scenario

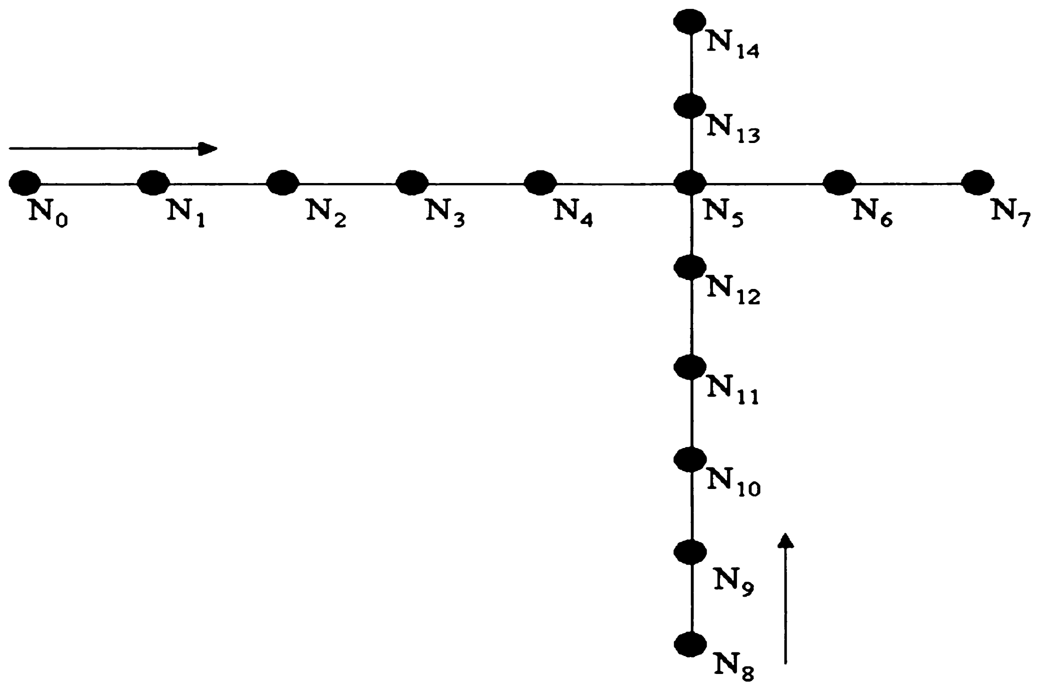
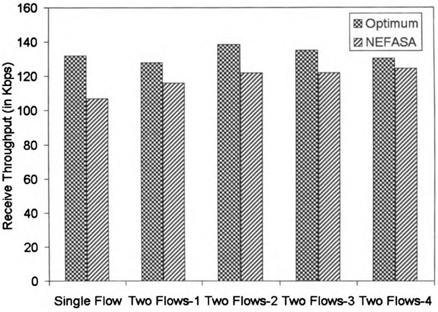
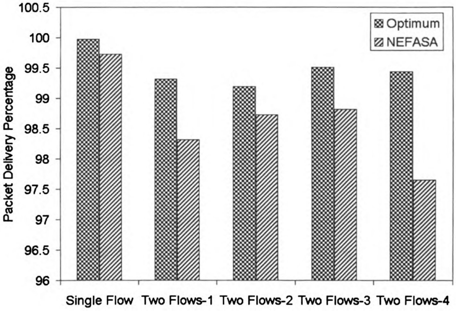


Figure 6.5: The Fourth Configuration of the Two Flow Scenario



(a) Throughput in the Single and Two Flow Scenarios



(b) PDP in the Single and Two Flow Scenarios

Figure 6.6: Comparison Against Optimum in Single and Two Flow Scenarios

90% of the optimum and in the single flow case the performance is within 80% of the optimum. The PDP (Figure 6.6(b)) achieved by NEFASA is also comparable to that of the optimum loading conditions and is above 97%.

In the single flow case, performance of the NEFASA is comparatively less because of the aggressive throttling done by the flow source. As the threshold is just 2% of the queue size (queue size is fifty packets), i.e., just one, the source is not able to increase its throughput to the optimum levels. At the same time, its interesting to see that the source can achieve such high throughput values for such a low threshold.

### 6.1.1 Multiple Flows in Arbitrary Mesh Networks

In complex scenarios which have multiple flows in an arbitrary mesh network, manually finding out the optimal loading conditions may not be possible. In order to check the performance of the NEFASA mechanism in complex scenarios, the effect of increasing the load on the network beyond the levels achieved by NEFASA is found out. So if NEFASA mechanism achieves a load of $X$ in a scenario, the effect of increasing the load on the system by a small percentage of $X$ is found out. When the load on the system is increased by a small percentage, the NEFASA mechanism is not used and the rates of all the flows is held constant throughout the simulation, such that the overall loading conditions on the network accurately reaches the desired percentage levels.

To perform the simulation on multiple flows, an arbitrary mesh topology having 36 nodes (shown in Figure 6.7) is used. Four configurations of traffic loads are used; they are 5 flows, 10 flows, 15 flows and 20 flows. The *ns* simulation for all the four configurations use application packet size of 512 bytes, feedback triggering threshold of 2%, disable period of two seconds and application increment interval of ten seconds.

Let the loading conditions achieved by NEFASA in any one of the four configurations – 5 flows, 10 flows, 15 flows and 20 flows – be $X$. When the load on the system

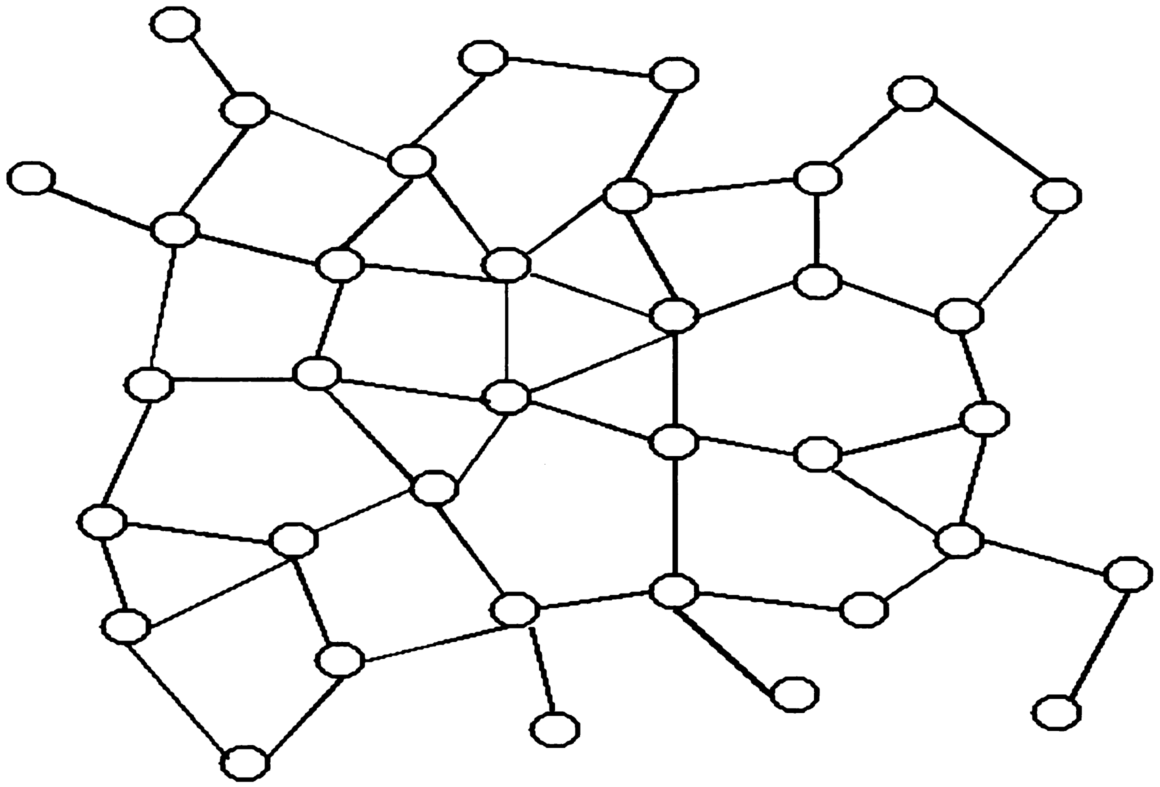


Figure 6.7: The Arbitrary Mesh Topology Used in Performance Comparison of NEFASA

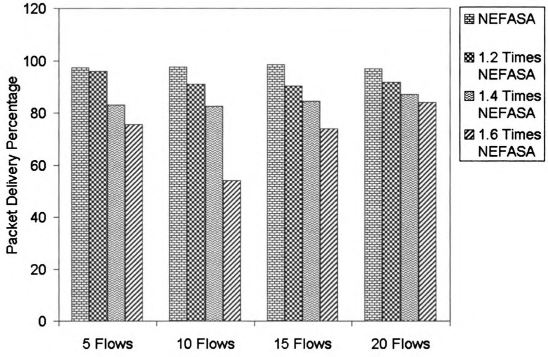


Figure 6.8: The Effect of Increasing the Load Levels Beyond NEFASA Levels

is increased by a small percentage of $X$, 20%, 40% and 60%, such that the load on the system becomes 1.2 times, 1.4 times and 1.6 times of $X$ respectively, it can be seen that (Figure 6.8) the PDP of the system falls; higher the increase greater is the fall in PDP. This shows that the loading conditions achieved by NEFASA is near the optimum.

## 6.2 Performance Comparison with TCP

TCP is an end-to-end system of feedback generation and propagation; the peer of the transport layer protocol at the destination takes the responsibility of providing the source with the required feedback. In the following sections, the performance of NEFASA is compared against TCP.

### 6.2.1 Hop-Unfairness in TCP

802.11 wireless ad-hoc networks have an inherent bias for flows with smaller number of hops[16]. Consider a single hop flow, i.e., the source and the destination are in radio contact over the wireless link. Let the capacity of the wireless network be $C$. Let $X$ be the throughput reached by this single hop flow, when it pumps data into the network such that it consumes all the available capacity ($C$). Now consider a two-hop flow. If it wants to pump data at the rate of $X$ units, the effective load on the network is $2X$ units because of the additional transmission by the single intermediate node. Because the capacity of the network remains the same at $C$, the two hop flow may not reach a throughput of $2X$ and it will be less than $X$. Generalizing this, a smaller hop flow can find it easier to achieve a higher throughput than a flow with bigger number of hops.

To verify this, consider a set of experiments which try to analyze the effect of hop count on the throughput achieved. A single TCP flow, but with different number of hops is used in each experiment. There are no other flows in the network other than the TCP flow, and the only nodes in the network are the ones that participate in the single TCP flow. The nodes are all equidistant and are 200MTS apart; this is similar to the single flow scenario present in Figure 6.1, but in each experiment a flow with different number of nodes is used. A similar set of experiments is perform with the NEFASA system, but the TCP flows are replaced with applications which conform to the application behavior described in Section 5.4.

As shown in Figure 6.9, a TCP flow with a single hop gets a very high throughput, and this drastically reduces as the number of hops increase. NEFASA on the other hand, achieves much better fairness when compared to TCP. This can be attributed to the aggressive throttling that is done by NEFASA at the source. Because of tight constraints on the threshold, the source is not as greedy as TCP when it tries to increase it output. This shows that TCP aggravates the hop-unfairness problem in

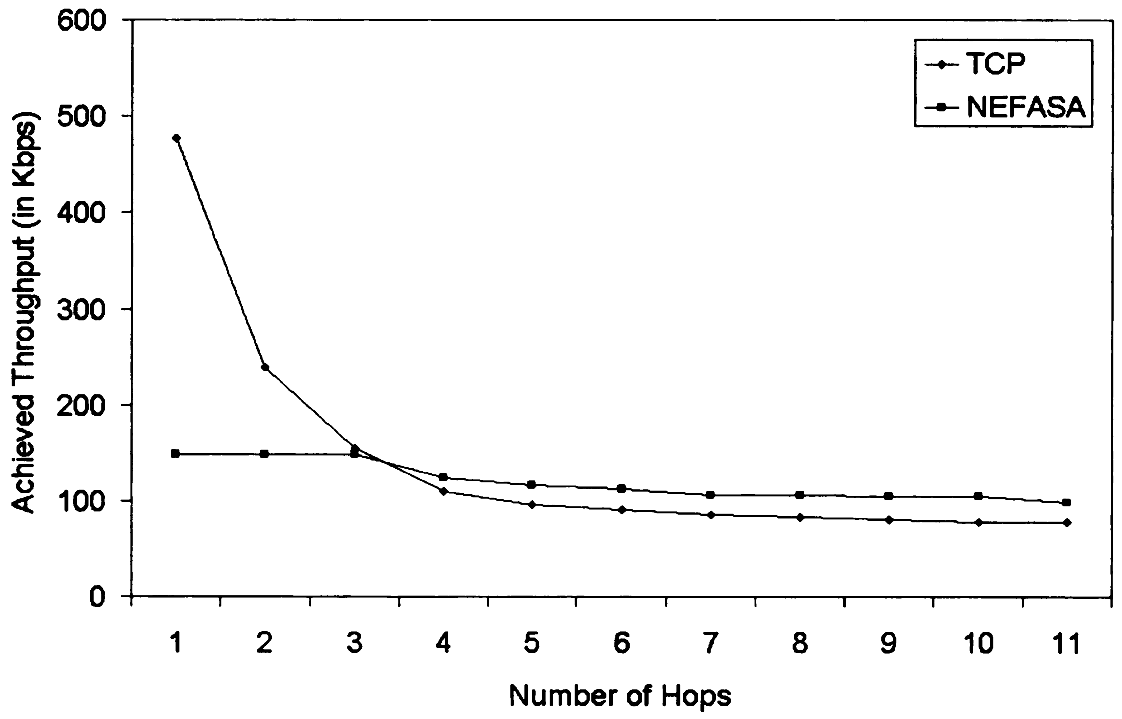


Figure 6.9: Hop-Unfairness in TCP

multi-hop networks.

### 6.2.2 Hop-unfairness in Arbitrary Mesh Networks

Unfairness in itself may not be an issue, if it does not effect other flows in the network. In a multi-flow scenario, if there are flows with varying number of hops, better performance of small-hop flows comes at the cost of other big-hop flows. Consider a single-hop flow in a multi-hop wireless network. Not only does a the single-hop flow get a relatively higher throughput, it also manages to disrupt all the flows within its carrier-sense range. A smaller hop flow (especially the ones which have less than six hops) can achieve significantly better throughput than bigger hop flows. At these very high throughput levels they can manage to suffocate all the big hop flows which pass through not only their receive range but also their carrier-sense range. This is because, the Request To Send (RTS) packets of bigger hop flows has to

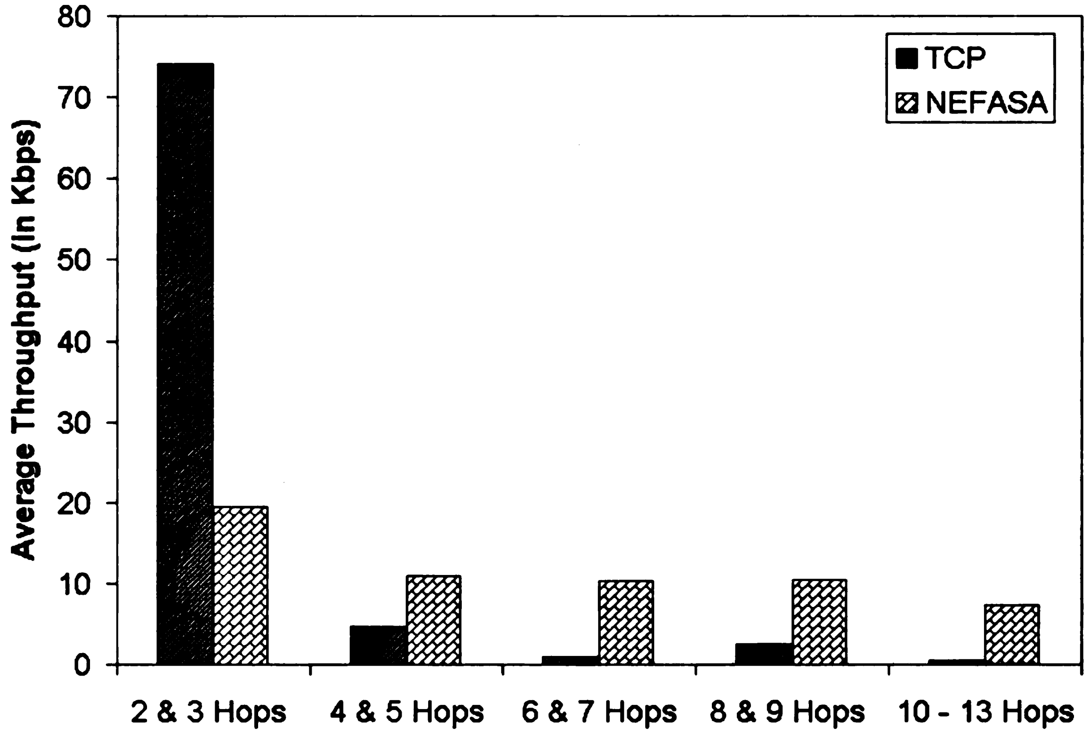


Figure 6.10: Hop-Unfairness in Arbitrary Mesh Topologies

be sent without errors, within a very small interval when the smaller hop flow is not trying to transmit the data [16]. This interval gets very small when there are flows with very small hops (less than 4) in the network. Hence the probability that a big hop flow succeeds in sending out an RTS packet becomes less and the big hop flows suffer.

Consider the arbitrary mesh wireless network, Figure 6.7 in page 72 with a dimension of 1300MTS X 1300MTS. The carrier-sense range of the nodes is 550MTS. The network has 15 TCP flows, and the performance of these flows for hop-unfairness is analyzed. A similar experiment in the NEFASA system is conducted, but with the TCP flows replaced with flows which confirm to the application behavior in Section 5.4.

As can be seen from Figure 6.10, TCP still exhibits hop-unfairness as flows with two or three hops perform better in terms of average throughput; NEFASA system on the other hand is more fair. Hence a network based feedback mechanism can be beneficial in a wireless ad-hoc network, as it can achieve better fairness among applications.

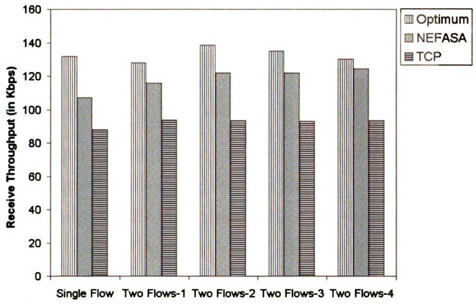
### 6.2.3 Throughput Comparison

In order to compare the throughput performance of NEFASA with TCP, consider the five different configurations – linear, two flows-1, two flows-2, two flows-3 and two flows-4 – described in Section 6.1. Instead of running flows which conform to the NEFASA feedback mechanism, TCP flows are used. In all the five configurations the hop count of all the flows is constant, i.e., six. As can be seen from Figure 6.11, when the hop counts do not matter, NEFASA performs better than TCP. In an arbitrary mesh topology, which has flows with very small hops, TCP has a better throughput performance when compared to NEFASA. But when the flows have bigger hops, NEFASA has a throughput performance comparable to TCP. This is can be seen in an arbitrary mesh topology (Figure 6.7 in page 72) where all the flows have at-least four hops. The performance of NEFASA is compared against TCP for four different – 5 flows, 10 flows, 15 flows, 20 flows – loading conditions. As shown in Figure 6.12, at comparable PDP, the throughput of NEFASA system is within 90% of TCP and in some cases better. In case of devices with energy constraints, the higher PDP levels achieved by NEFASA can be of a significant advantage.

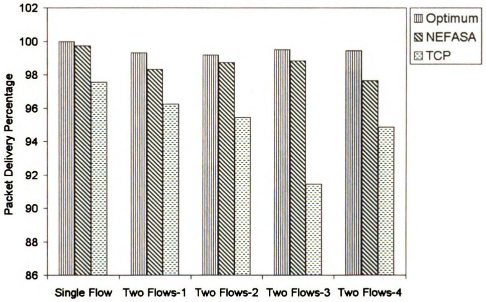
## 6.3 Conclusions

1. The performance of NEFASA, as measured by the overall throughput achieved by the network, is comparable to manually found optimum loading conditions.

2. TCP exhibits hop-unfairness where flows with smaller hops not only manage to get higher throughput, but also affect flows with bigger hops within their

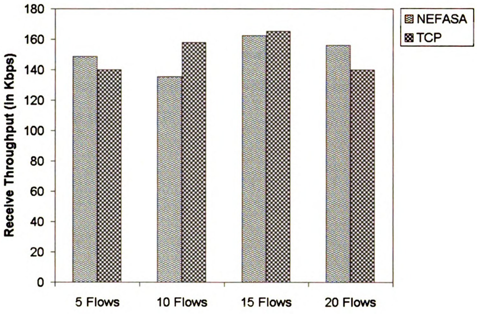


(a) Throughput

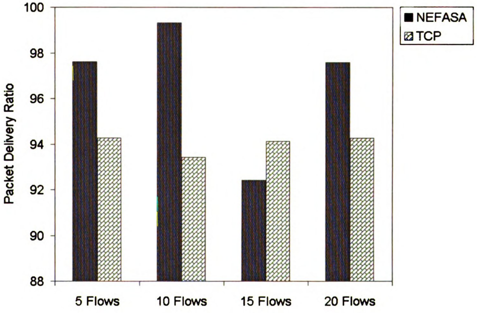


(b) PDP

Figure 6.11: Comparison of NEFASA Against TCP in Single and Two Flow Scenarios in Linear Networks



(a) Throughput



(b) PDP

Figure 6.12: Comparison of NEFASA Against TCP in Arbitrary Mesh Scenario

carrier-sense range.

3. Throughput performance of NEFASA is better to that of TCP when the hop count of all the flows is same. In networks which have flows with a hop count of at-least four, the performance of NEFASA, as measured by throughput is comparable to that of TCP

# Chapter 7

# Parameter Tuning

In the previous two chapters, the NEFASA framework and its performance were discussed. In this chapter, NEFASA system is studied in further detail and the various parameters which effect the performance of the NEFASA and their impact on the system is discussed.

## 7.1 Congestion Triggering Sensitivity

To understand the NEFASA mechanism, consider an arbitrary mesh topology with thirty-six nodes in Figure 6.7 (in page 72). There are five flows in it, and the default application behavior described in Section 5.4 is used for these flows. A *ns* simulation is performed with all the components of NEFASA incorporated into it. A threshold of 2% is used for generating the feedback messages and the application increment interval is set to 10secs. Figure 7.1 gives the adaptation curve for one of the five flows in the network

The adaptation curve in Figure 7.1 shows wild oscillations, and at different points of time in the simulation the application rate touches the minimum. These are not due to the network, as it is still lightly loaded. Upon further observing Figure 7.1, it can be seen that most of the crests in the adaptation curve are preceded by peaks,

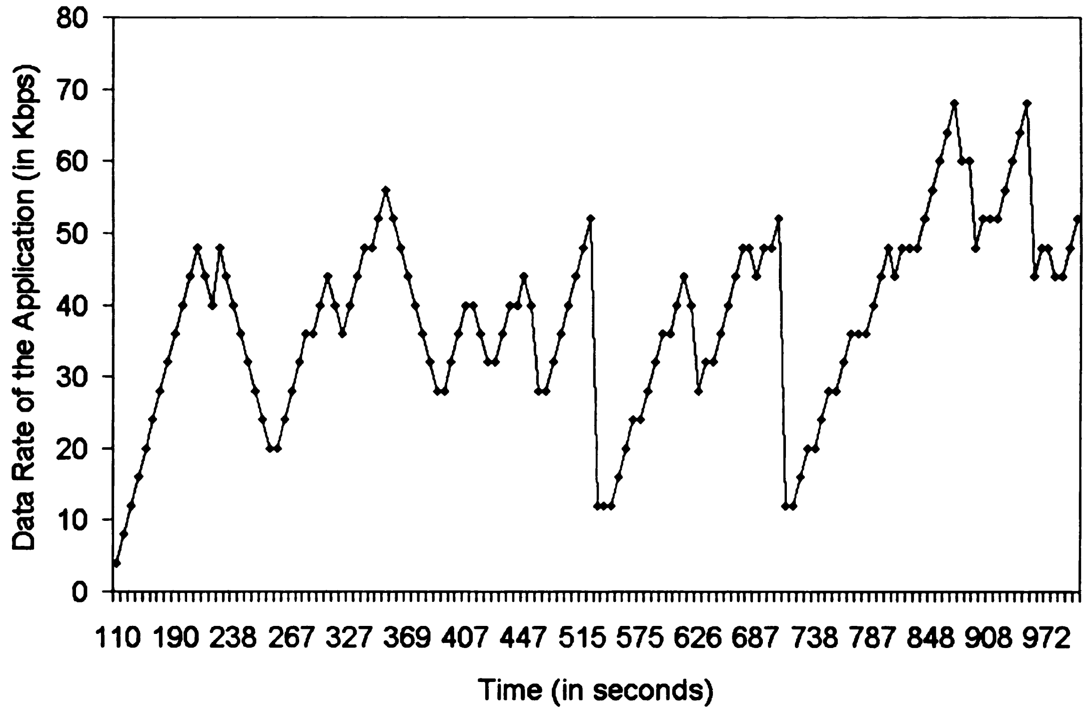


Figure 7.1: The Effect of Having No Disable Period

i.e., there are steep falls in the curve. As the application responds immediately to an RCHANGE message, the steep falls are due to a sequence of consecutive RCHANGE messages that are received by the application.

The root of this problem can be traced to the manner in which the RCHANGE messages are triggered. An RCHANGE message is triggered every time a packet enters the queue and the queue length is above the threshold. If the threshold was five, and the current queue length was five, every consecutive packet arrival (assuming there are no packets exiting the queue during this time) causes a RCHANGE message to be generated. This is highly likely, because, there is a time lag between the generation time of the packet, which caused the RCHANGE message to be triggered, and the the application response to the RCHANGE. All the packets that are generated by the application during this time lag have the potential to trigger RCHANGE messages at the point of congestion.

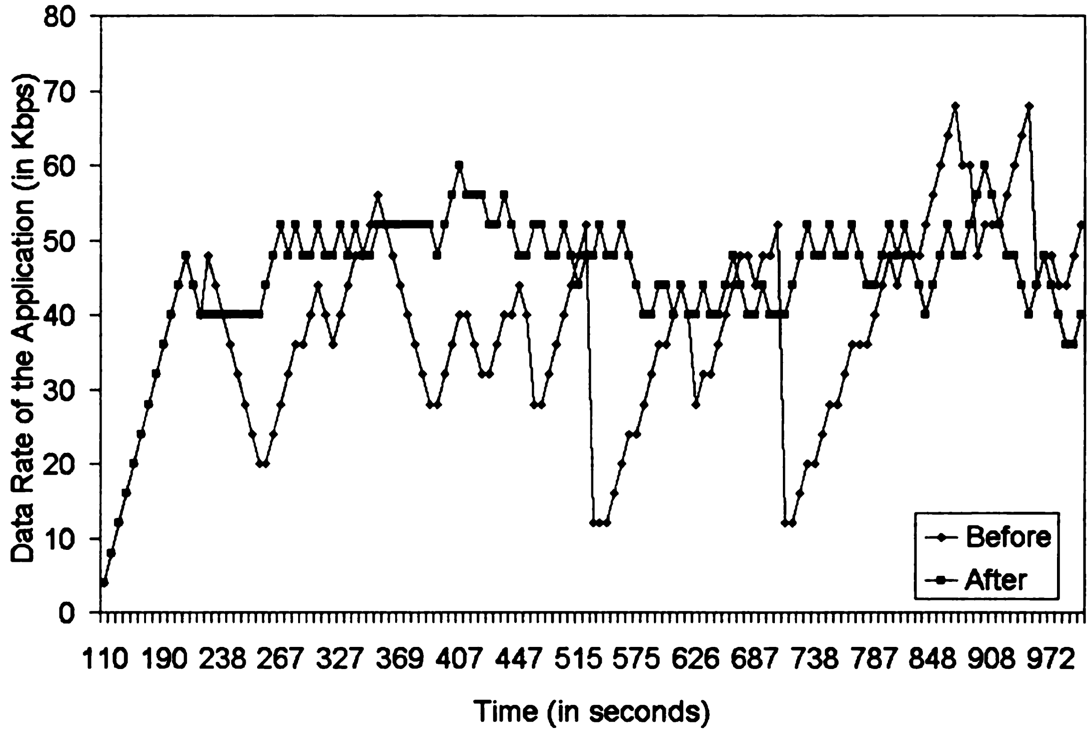


Figure 7.2: The Effect of Introducing Disable Period

After the first time an RCHANGE message is generated, if the node which generates the RCHANGE message disables further RCHANGE generation for a short duration of time, this problem can be alleviated. The adaptation curve of the flow after the disable period is introduced, is shown in Figure 7.2. As can be seen from Figure 7.2, the falls from the peaks are now much smaller and the application is stable and has less variance.

Introducing the disable period at the intermediate node has its disadvantages. If the disable period is too long, the node becomes insensitive to changes in the network. This is because, if the congestion for which RCHANGE message was generated gets cleared because of application throttling, any future congestion conditions in the network will go undetected during the disable period. Hence the disable period should be kept small to keep the network responsive to changes in the network.

In order to characterize the influence of disable period, the impact of using different disable periods on the application behavior is studied. As the main motive behind introducing the disable period, is to reduce the variance in the application adaptation curve, the influence of disable period on application variance is characterized. This is done by plotting the variance in the application adaptation curve, as measured by the co-efficient of variation, for different disable periods. The *ns* simulations are performed with a thirty-six node network – Figure 6.7 (in page 72) – with five flows in it. For this simulation, a threshold of 2% is used for generating the feedback messages and the application increment interval is set to 10secs.

As can be seen from Figure 7.3, all the five flows benefit from the introduction of the disable period, and show a significant decrease in variation when the disable period is increased from zero to two seconds. But after two seconds, there is no noticeable upward or downward trend in the variance. Hence it can be seen that a disable period of around two seconds is suitable for a network which uses an application increment interval of at-least 10secs.

## 7.2 Effects on Throughput and PDP

Disable Period has other effects on the application behavior. when the disable period is increased from zero seconds to two seconds, due to less throttling by the network, applications see an increase in average throughput (Figure 7.4(a)). After two seconds, there is a small upward trend in the throughput (as seen at the destination); but large values in disable period cannot be used in the network, because it would make the nodes insensitive for a very long time.

The PDP of the applications shows a decrease (Figure 7.4(b)) once the disable period is decreased. The applications are excessively throttled when the disable period is zero seconds; this decreases the load on the network below congestion levels very quickly. Hence the packet looses suffered in the network due to congestion is very low.

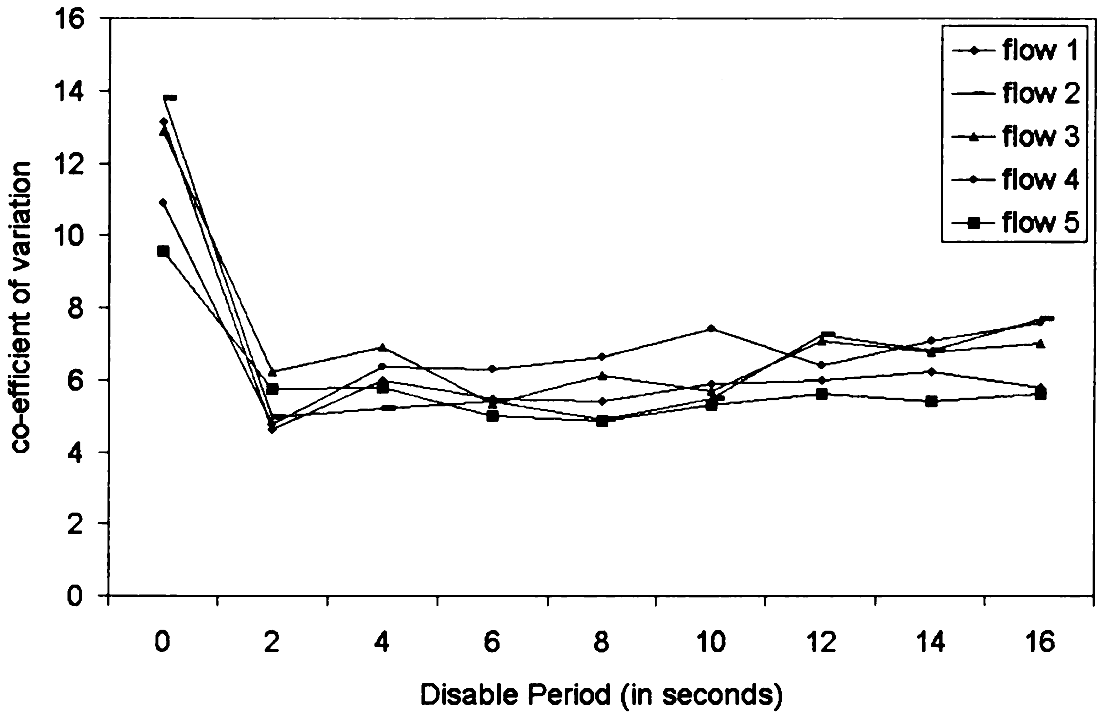
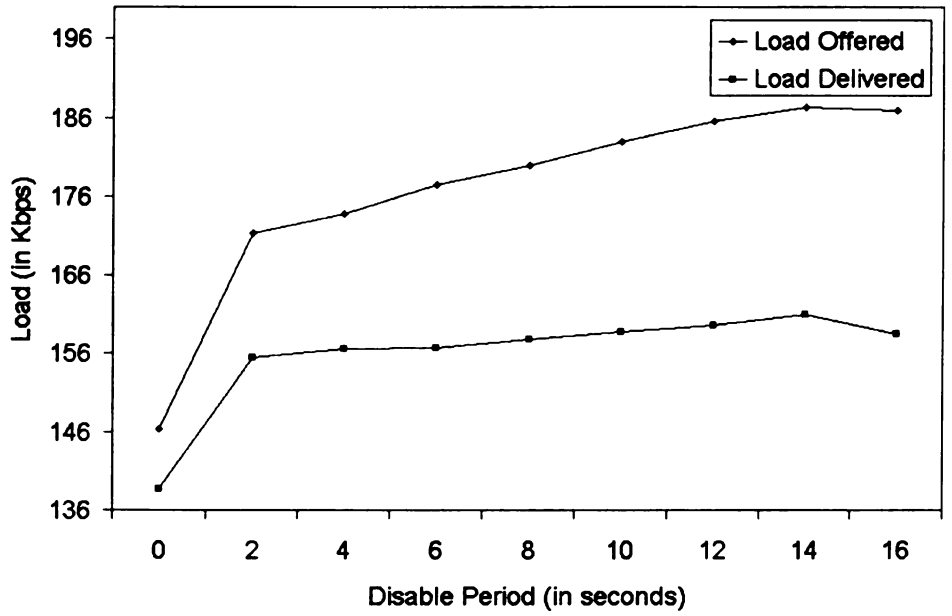


Figure 7.3: Decrease in Variation due to Disable Period

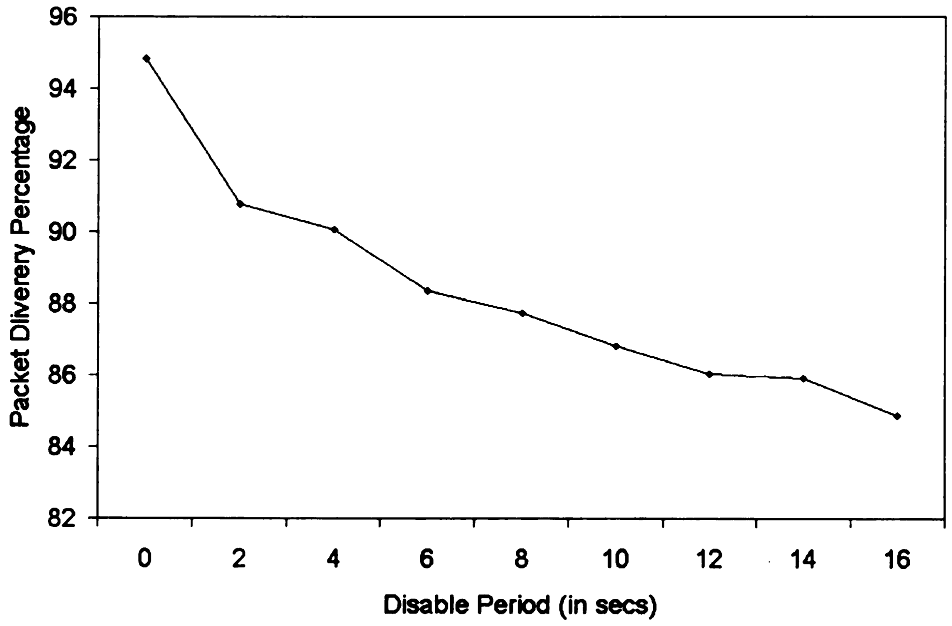
If the disable period is high, it takes more time for the applications to bring down their output rates to levels below congestion; this is because of less throttling by the network. Due to this, all the packets that are generated during the disable period in the intermediate node have a high probability of being dropped. Hence applications see low PDP levels.

## 7.3 Congestion Detection Sensitivity

Threshold on link layer queues, is the key mechanism that is used to trigger the RCHANGE propagation. In NEFASA, the link layer queues are modified such that every neighbor has its own queue, i.e., the queues are neighbor specific. Thresholds are uniformly placed on all the queues, and if the threshold is crossed in one of the queues the RCHANGE mechanism is triggered.

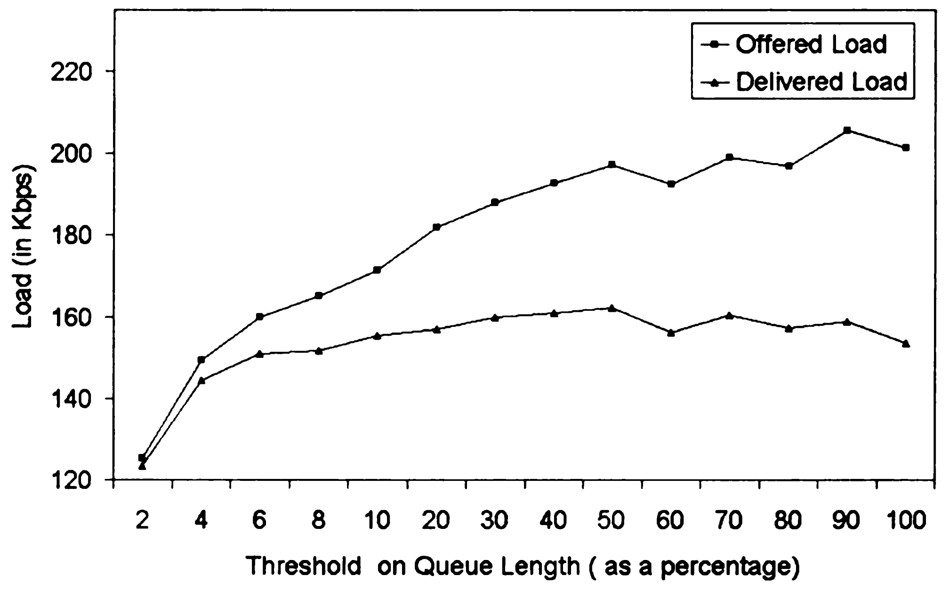


(a) Effect of Disable Period on Throughput

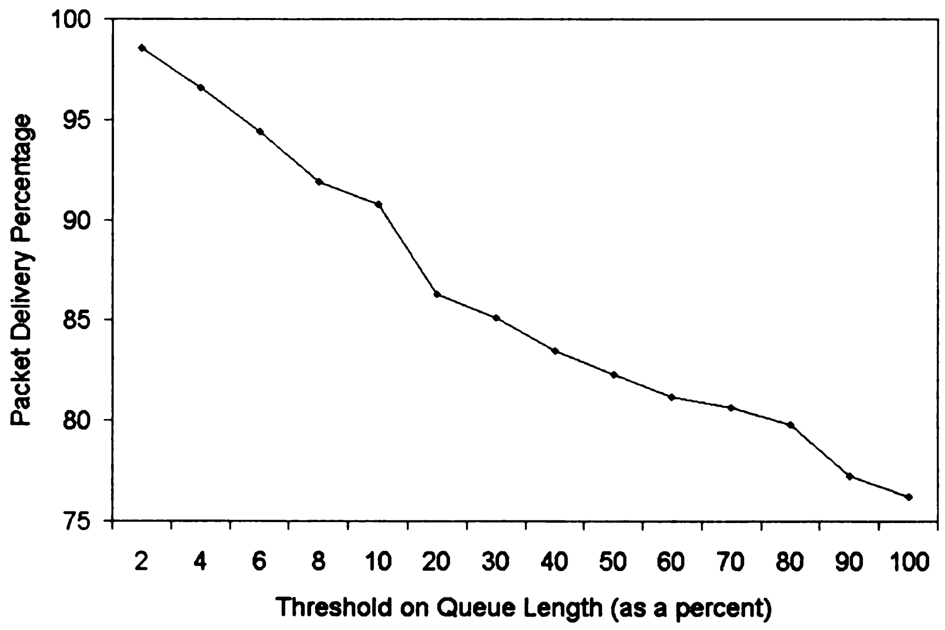


(b) Effect of Disable Period on PDP

Figure 7.4: Side Effects of Disable Period



(a) Effect of Threshold on Throughput



(b) Effect of Threshold on PDP

Figure 7.5: Effects of Threshold

Having lenient thresholds makes the network more tolerant to congestion conditions. Hence higher thresholds will trigger the RCHANGE propagation sometime after the congestion begins to build. This higher tolerance to congestion will allow the applications to increase their output rate during the congestion build-up phase. This has the effect of increasing the overall load (Figure 7.5(a)) on the network – and hence a higher throughput from individual applications – with increasing thresholds. Allowing applications to increment their rates when congestion is building up has the effect of decreasing the overall PDP (Figure 7.5(b)) of the entire network. This is because of the packets that are lost during the congestion build up phase.

## 7.4 Application Behavior

An application in NEFASA (application behavior is described in Section 5.4), expects only negative feedback from the network, and in the absence of negative feedback it is free to increment its output rate. Because of this policy, the disable period – the interval during which no bad news is generated – has an influence on the increment interval of the application (see Section 5.4 for a description of increment interval).

Lower values of increment interval means that the application is very aggressive in pumping data into the network. This should generate a similar response from the network, which on its part tries to aggressively throttle the applications by generating more RCHANGE messages. For the network in Figure 6.7 (in page 72), a threshold of 2% for generating the feedback messages, and a disable period of two seconds, the network sees a decrease in number of RCHANGE messages with an increase in the increment interval. This increase is more pronounced when the interval is decreased from two to zero seconds, during which it almost sees an 100% increase in the number of RCHANGE messages.

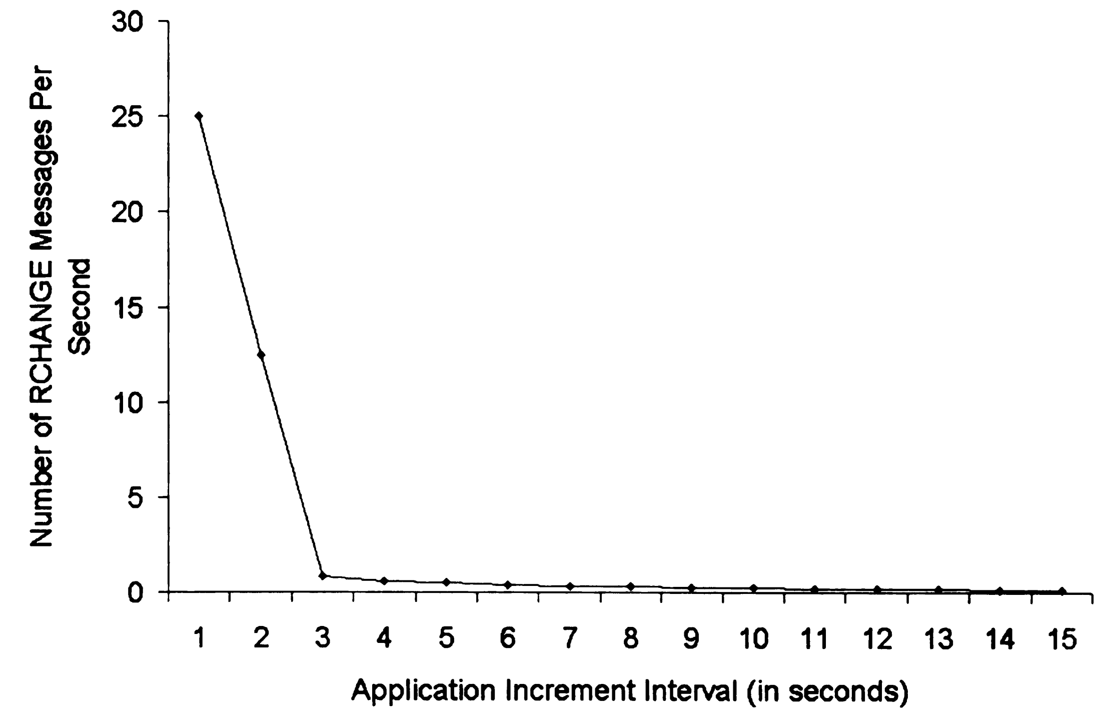
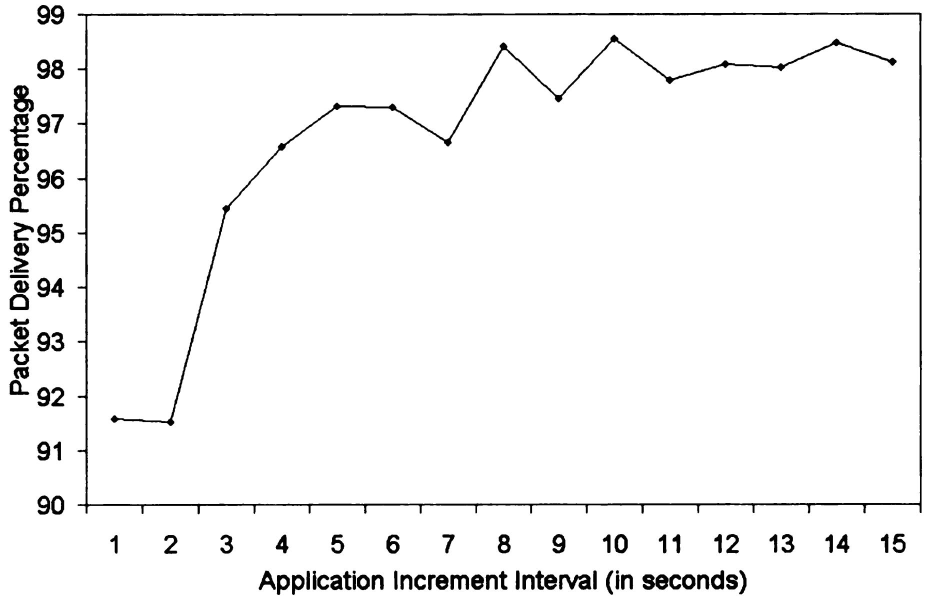


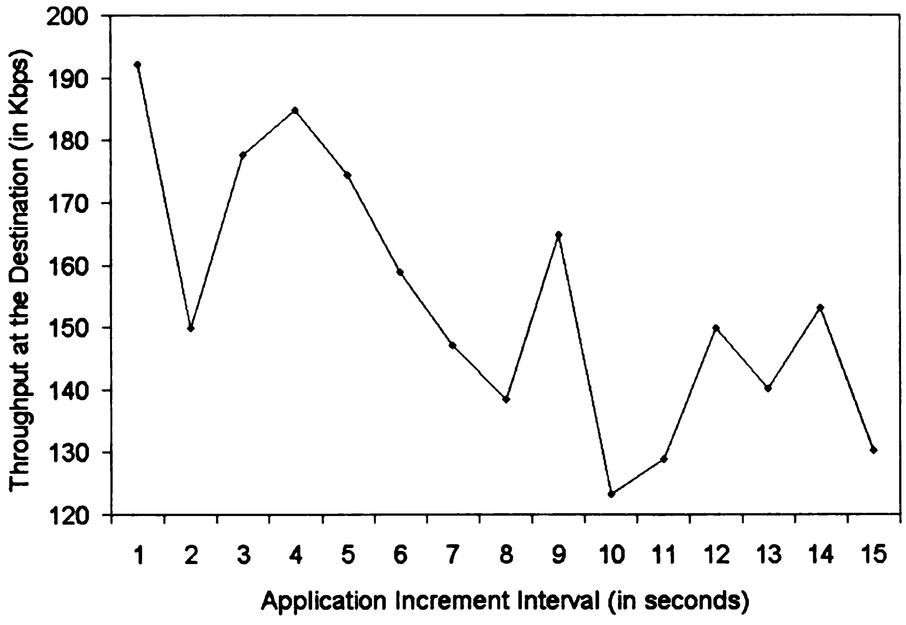
Figure 7.6: Number of RCHANGE Messages and Increment Interval

An aggressive increment in data rates by the application, when it has a increment interval below the disable period, leads to severe packet drops (Figure 7.7(a)) and the application could see a PDP of only around 90%. When the increment interval is above the disable period, the PDP stays healthy and is above 95%. As the application tries to increase its rate when some of the nodes ,which previously had reported congestion in the network, cannot generate the negative feedback the application suffers packet losses due to the slow response from the network.

With decreasing increment period, the overall load on the network decreases (Figure 7.7(b)) as the applications become less aggressive in pumping data into the network.



(a) Effect of Increment Interval on PDP



(b) Effect of Increment Interval on Throughput

Figure 7.7: Effects of Increment Interval

## 7.5 Stability

As the application behavior defined in NEFASA (see Section 5.4) is greedy, the applications experience oscillations in steady state. Consider the network configuration in Figure 6.7 (in page 72)), a threshold of 2% for generating the feedback messages, a disable period of two seconds and an application increment interval of 10 secs. Because of the greedy behavior of the applications, all the five flows experience steady state oscillations as shown in Figure 7.8.

In order to check if greedy behavior is responsible for the oscillations, the average throughput of each of the five flows when all the five flows are greedy (Figure 7.8), is calculated. With these average values, another simulation is conducted where the maximum data rates of all the five flows is set to their respective calculated average throughput values. With these settings, all the applications exhibit stability with very little oscillations (Figure 7.9(a)).

In order to further verify this, a similar simulation is conducted by setting the maximum output rates of only four applications; hence only one application – flow 1 in Figure 7.9(a) – is allowed to be greedy. When the adaptation curve of the applications is plotted (Figure 7.9(b)), it can be seen that only the application whose maximum rate was not set shows steady state oscillations. Also, due to less contention from other non-greedy applications, the average throughput achieved by this application is higher when compared to its average throughput when all applications are greedy.

Hence we can conclude that greedy application behavior is responsible for the steady-state oscillations. As the oscillations are bounded, and they are about the average throughput of the flows we can conclude that the NEFASA feedback propagation mechanism promotes network stability.

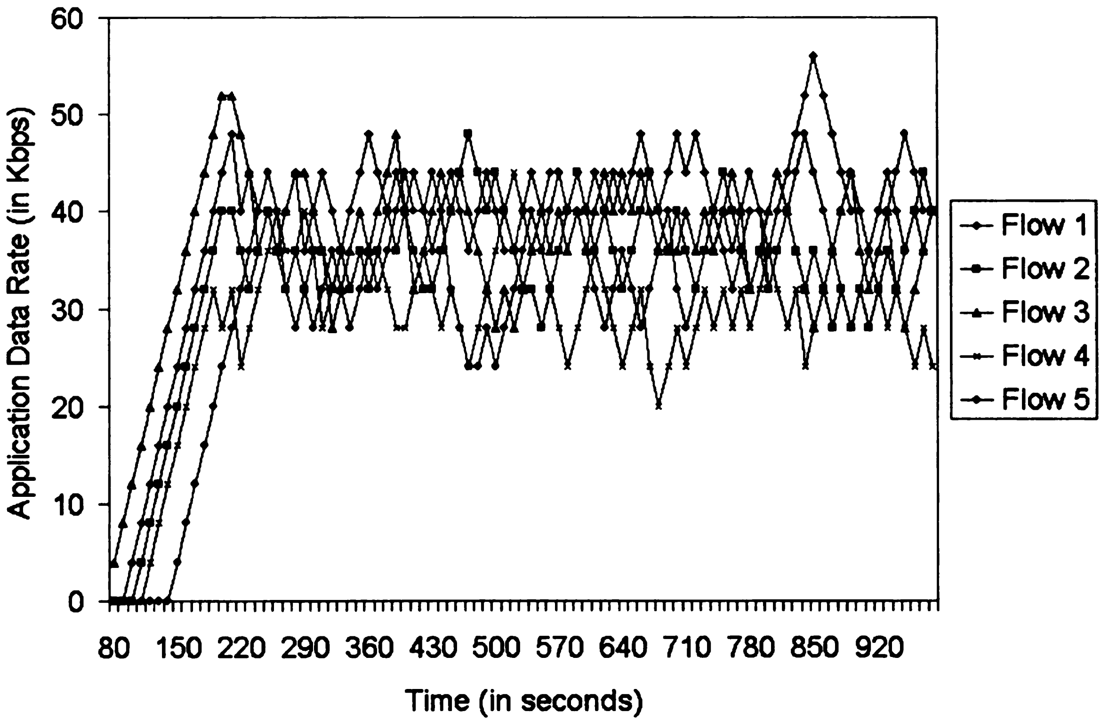
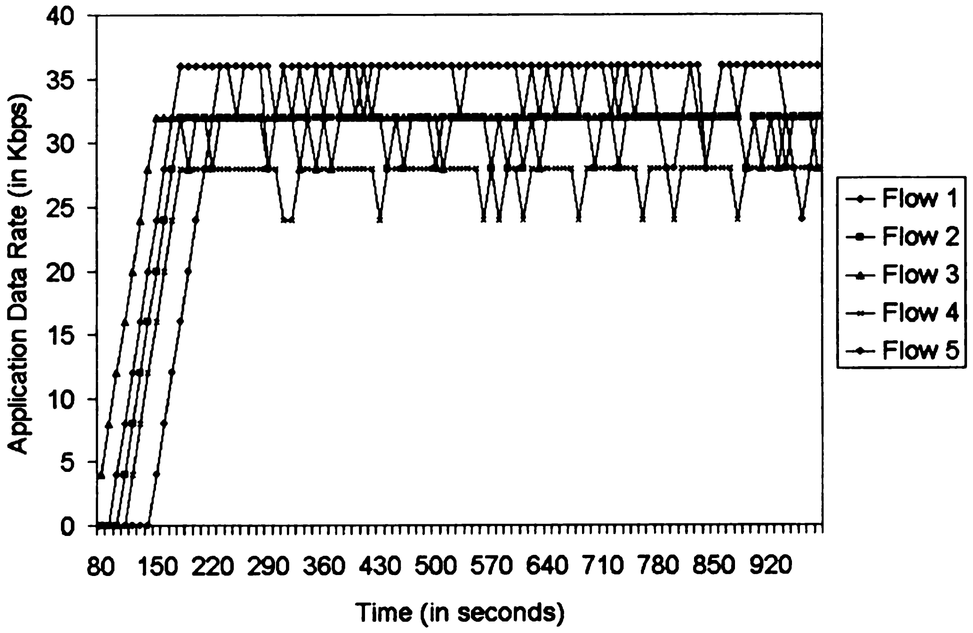
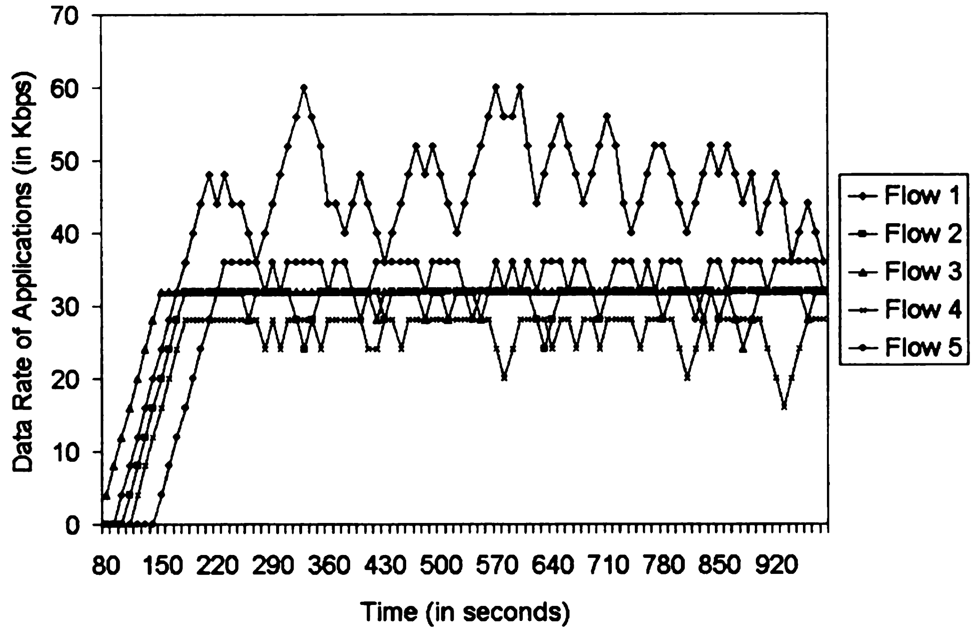


Figure 7.8: All Applications are Greedy



(a) No Greedy Applications



(b) One Greedy Application

Figure 7.9: Stability Analysis in NEFASA Network

## 7.6 Conclusions

In this chapter the various parameters which influence the performance of the NEFASA system were studied. The following conclusions can be drawn from this study.

1. Introducing disable period into the NEFASA system leads to a decrease in variance, and hence instability, experienced by the applications. A disable period of around two seconds is sufficient to stabilize the applications, and higher disable periods will not increase stability of the applications and only make the network insensitive to changes.

2. Increasing disable period will have the effect of increasing the throughput of the applications but at the cost of decreased PDP.

3. Increasing thresholds leads to increasing the throughput of the applications but at the cost of reduced PDP.

4. Having very short application increment intervals leads to significant increases in throughput, but the PDP achieved at short application increment intervals can be as small as 60%. The application increment intervals should be at-least as big as the disable periods.

5. Because of greedy application behavior, the applications exhibit oscillations around their average throughput. If the applications are not greedy, the system reaches stability.

# Chapter 8

# Conclusion and Future Work

## 8.1 Conclusion

It is feasible to develop a network-triggered feedback mechanism for application adaptation in wireless networks. Such a mechanism provides a fair bandwidth allocation and does not depend on specific transport layer mechanisms.

To develop such a framework, AODV can be used for feedback propagation with only very minor modifications to its mechanism of propagating RERR messages.

The main contributions of this work are

1. AODV-like precursor lists, can be very helpful to provide state-less feedback to applications

2. Buffer occupancy levels in the link layer can not only indicate congestion, they can also used to generate feedback to adaptive applications. This method of generating feedback can drive stability in the network in the absence of greedy application behavior.

3. TCP is bias toward flows with smaller number of hops, hence these flows have better throughput performance when compared to flows with larger hops.

4. Network based feedback can be used to achieve better fairness among flows with different hop lengths.

## 8.2 Future Work

The following are the areas where further work could lead to interesting results and improvements in the NEFASA system :

1. To extend the NEFASA system to take into consideration mobility in the nodes.
2. To have a TCP-like transport protocol for NEFASA and compare the fairness characteristics of the resulting protocol against TCP.
3. To modify TCP to work with NEFASA, so that it benefits from its fairness characteristics.
4. The filtering methods used to reduce the number of feedback messages in the link and network layer, takes away control from the applications. This may not suite specific types of applications which would want to do a more sophisticated congestion and rate control. This should be explored further and the interaction between feedback propagation and different application models should be studied.
5. To further explore and characterize other – MAC delays and MAC drops – feedback generation mechanisms to work with NEFASA
6. To improve NEFASA's throughput performance in an arbitrary mesh topology with multiple flows.

# APPENDICES

# Appendix A

# Modifications to *ns*

Network Simulator (ns) version 2[6] was used for all the simulations in this work. A lot of modifications were done to incorporate the NEFASA system into *ns*. Some of them are listed here.

1. The Constant Bit Rate (CBR) application was modified to conform to the default application behavior as described in Section 5.4.

2. RERR generation and propagation was stopped, as mobility was not considered in the work.

3. RCHANGE generation and propagation was implemented as described in Algorithm 6 and Algorithm 7.

4. Packet types for RCHANGE were created.

5. The simple priority queues used in *ns* was modified such that every neighbor has its own queue.

6. The threshold based RCHANGE trigger was implemented on the neighbor-specific queue.

7. Queue-monitors were implemented for the per-neighbor queues.

# Appendix B

# Abbreviations and Acronyms

| | |
|---|---|
| **ACK** | Acknowledgment |
| **AIMD** | Additive Increase Multiplicative Decrease |
| **AODV** | Ad-hoc On Demand Distance Vector |
| **CBR** | Constant Bit Rate |
| **CTS** | Clear To Send |
| **ECN** | Explicit Congestion Notification |
| **FTP** | File Transfer Protocol |
| **HTTP** | Hypertext Transfer Protocol |
| **IFQ** | Interface Queue |
| **IP** | Internet Protocol |
| **MAC** | Medium Access Control |
| **MPDU** | MAC protocol Data Unit |
| **MSDU** | MAC Service Data Unit |
| **NEFASA** | Network Feedback for Application Self Adaptation |
| **NS** | Network Simulator |
| **OSI** | Open Standards Interconnect |
| **PDP** | Packet Delivery Percentage |

|  |  |
|---|---|
| **PDR** | Packet Delivery Ratio |
| **PPS** | Packets Per Second |
| **RCHANGE** | Route Change |
| **RED** | Random Early Detection |
| **RERR** | Route Error |
| **RREP** | Route Reply |
| **RREQ** | Route Request |
| **RREQID** | Route Request Identifier |
| **RTS** | Request To Send |
| **DIFS** | Distributed (Coordination Function) Interface Space |
| **SIFS** | Short Inter-frame Space |
| **SWAN** | Stateless Wireless Ad hoc Networks |
| **TCP** | Transmission Control Protocol |
| **TTL** | Time To Live |
| **UDP** | User Datagram Protocol |
| **ns** | Network Simulator |

# BIBLIOGRAPHY

[1] Hayder Radha; Mihaela van der Schaar and Yingwei Chen. The MPEG-4 Fine-Grained Scalable Video Coding Method for Multimedia Streaming Over IP. *IEEE Transactions on Multimedia*, 3(1), March 2001.

[2] Weiping Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Trans. on Circuits and Systems for Video Technology*, 11(3):301–317, March 2001.

[3] A. Tanenbaum. *Computer Networks.* Prentice Hall, thrid edition, 1996.

[4] Transmission Control Protocol. RFC 793, September 1981. URL: http://www.ietf.org/rfc/rfc0793.txt.

[5] J. Postel. User Datagram Protocol. RFC 768, August 1980. URL: http://www.ietf.org/rfc/rfc0768.txt.

[6] Kevin Fall and Kannan Varadhan. *The* ns *Manual.* The VINT Project, December 2003. URL:http://www.isi.edu/nsnam/ns/ns-documentation.html.

[7] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. In *IEEE/ACM Transactions on Networking*, August 1993.

[8] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):10–23, October 1994.

[9] G-S Ahn; A.T Campbell; A. Veres; and L-H Sun. SWAN: Service Differentiation in Stateless Wireless Ad Hoc Networks. In *Proc. IEEE INFOCOMM 2002*, New York,New York, June 2002.

[10] A. Veres; A.T Campbell; M Barry and L. Sun. Supporting Service Differentiation in Wireless Packet Networks Using Distributed Control. *IEEE Journal of Selected Areas in Communcations(JSAC)*, 19(10):2094–2104, October 2001. Special Issue on Mobility and Resource Management in Next-Generation Wireless Systems.

[11] Seoung-Bum Lee; Jiyoung Cho and Andrew T. Campbell. HMP: Hotspot Mitigation Protocol for Mobile Ad hoc Networks. In *11th IEEE/IFIP International Workshop on Quality of Serivce*, Monterey, CA, June 2003.

[12] Manthos Kazantzidis; M. Gerla and S-J. Lee. Permissible Throughput Network Feedback for Adaptive Multimedia in AODV MANETs. In *Proc. of IEEE ICC 2001*, Helsinki, Finland, June 2001.

[13] Manthos Kazantzidis. *Adaptive Wireless Multimedia.* PhD thesis, University of California, Los Angeles, Department of Computer Science, 2002.

[14] Jianbo Xue; Patrick Stuedi and Gustavo Alonso. ASAP: An Adaptive QoS Protocol for Mobile Ad Hoc Networks. In *Proc. of IEEE PIMRC*, Beijing, China, September 2003.

[15] Seoung-Bum Lee and Andrew T. Campbell. INSIGNIA: In-band Signaling Support for QOS in Mobile Ad Hoc Networks. In *Proc. of* 5th *International Workshop on Mobile Multimedia Communications (MoMuC,98)*, Berlin, Germany, October 1998.

[16] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad Hoc Networks? *IEEE Communications Magazine*, June 2001.