# Quality of Service in Ad Hoc Networks

*A study of the Stateles Wireless Ad Hoc Network mechanism*

**Fredrik Andreassen**


Department of Informatics

Faculty of Mathematics and Natural Science

UNIVERSITY OF OSLO

# Abstract

*Mobile Ad Hoc Networks (MANET) is an autonomous collection of mobile nodes with wireless connection. The nodes are free to move and there exist no infrastructure for these wireless connected nodes. The Stateless Wireless Ad hoc Network model (SWAN) is a proposal trying to solve the Quality of Service concerns in MANET's. This study presents an introduction to the wireless world, the SWAN network model, a theoretical analysis of the SWAN model, and finally my work and analysis of the result gained with the simulation of the SWAN model in Network Simulator 2 (ns-2).*

# Preface

This master thesis is written as part of my Civil Engineering degree in Computer Science at the University of Oslo, department of Informatics. The topic of my thesis was introduced to me by Andreas Hafslund at the Unik University Graduation Center. Hafslund have been my supervisor throughout this project.

In my early days of studying computer science I had most interest of computer networks. Thus, studying and doing simulations for wireless Ad Hoc networks was an exciting task for me. Most of the areas where new to me, and there have been many challenges and a lot of hard work, but at the same time I have learned a lot about Quality of Service, wireless Ad Hoc networks and the many sides that appear in large projects like a master thesis.

I would like thank my two supervisors PhD. Andreas Hafslund and Prof. Knut Øvsthus for guidance and helping me accomplish this master thesis project.

Fredrik Andreassen
Oslo, May 1st, 2005

# Table of Contents

# List of Figures

# Table of Tables

# 1 Introduction

## *1.1 About my work with this thesis*

The general idea behind my work on this master thesis is to study wireless Ad Hoc networks, Quality of Service in MANET, and particularly take a deeper look at the SWAN network model and the implementation of that model made for ns-2. The people behind SWAN have done several tests to show how the model performs in the ns-2 network simulator. The two major tasks in this thesis are; first, to make a theoretical analysis of the SWAN model, and second, making analysis of the simulations of the SWAN network model in ns-2.

My effort is to install the ns-2 simulator for Linux and then to set up the same scenario which they ran, and then verify those tests. I will also expand my simulations with scenarios, which seems realistic for Real-Time applications in an Ad-Hoc network environment. After accomplishing my simulations I will analyse the output of these simulations and try to find out the status of the SWAN network model. That means how well is the SWAN network model improving the Quality of Service in an Ad Hoc network, what do works, what does not works and what improvement can be done in both the model and the implementation. New suggestions will be enclosed in suggestions for future work.

## *1.2 Chapter overview*

Chapter 1 is an introduction to this thesis and also to the different network communities. Chapter 2 introduces the reader to world of wireless networks and particularly Mobile Ad Hoc networks. Explaining the term “Quality of Service” and its conceptions is carried out in chapter 3. The Stateless Wireless Ad Hoc Networks (SWAN) model is presented in chapter 4. The problems and troubles I came across during my work with SWAN are presented in chapter 5. Chapter 6 is dedicated to a short description of two related QoS network models. Chapter 7 includes a presentation of the ns-2 networks simulator, my work with this simulator and my preliminary simulations schemes carried out in this simulator. In chapter 8, a description of the different simulations scenario intended for testing the performance of the SWAN implementation. Chapter 9 is dedicated to the presentations of the result of the simulation scenarios described in chapter 8. Chapter 10 summarizes my work, my result and concludes the status of the current SWAN model. This chapter also contains suggestions for future work and research.

## *1.3 Who is who in the network community*

In the internet research network community a lot of vendors, suppliers, organizations, universities etc. exist. In this chapter I will give a short overview of who is who, of the most important organizations in the network community, their tasks and major areas of work. To ensure that computers are able to communicate despite inequalities in both hardware and software from all the

vendors that exist, network standards are needed to ensure word wide compatibility.

One of the earliest founded standardization organization is the International Telecommunication Union (ITU). The task of the ITU's standardization unit ITU-T, is to make technical recommendations about data communication, telegraphy, and telephony. Another major contributor, which is the largest professional organization in the world, is the Institute of Electrical and Electronics Engineers (IEEE). Every year IEEE host hundreds of conferences, publish journals and articles in addition to develop standards in their electrical engineering and computing standardization group. One of their most successful groups is the 802 committee.

Yet another important organization is the part of the Internet Architecture Board (IAB) which is called the Internet Engineering Task Force (IETF). IETF concentrates its work on short-term engineering issues. The IETF is divided into specific working groups with individual problem to solve. For a proposal of a technical contribution to the Internet to become a Proposed Standard, in addition to have sufficient interest, it must first be described completely in an Internet draft. After the publishing, the draft is made available for informal reviews and comments for a six months period. A working implementation must have been thoroughly tested upon advancing to a Request For Comments (RFC). The archival series of RFC document is the official publication channel for Internet standards documents and other publications. After this, the proposal is ready to be declared as an Internet Standard if the software works and the IAB is convinced [46] [51].

# 2 Wireless Networks and Mobile Ad Hoc Networking

This chapter will present background material of wireless computer networks, its operations, architecture and its protocols.

## *2.1 Introduction to Wireless Networks*

In recent years, wireless radio networks have become increasingly popular. Among the most popular devices are mobile phones, PDAs and computer laptops. The nomadic lifestyle of people in modern society has to be supported by future wireless system. Wireless computer networks can be divided into three broad categories; System interconnection (e.g., Bluetooth), Wireless LAN, and Wireless WAN (e.g., IEEE 802.16). For the further purpose of this paper, Wireless LAN (WLAN) is the one of interest. Several standards for WLAN have been developed, such as IEEE 802.11 standard and the European developed standard; HIPERLAN. So far the IEEE 802.11 protocol is the one that is implemented in most products for WLAN and the one that is most widely used. Throughout this paper the IEEE 802.11 is the basis for further investigation of Mobile Ad Hoc Networking. [6] [51]

## *2.2 The IEEE 802.11 Wireless LAN*

Since its initiation in 1990, and the completion of the international standard in 1997, the IEEE 802.11 Wireless Network (WLAN) has become incredibly popular. As an alternative to the high installation and maintenance costs experienced in the traditional LAN, WLAN was developed to equip the users in limited geographical areas with high bandwidth. Typical communication range of WLAN lies between 100-500 meters. The IEEE 802.11 is operating in the Industrial, Scientific, and Medical (ISM) band (See Figure 2.1).

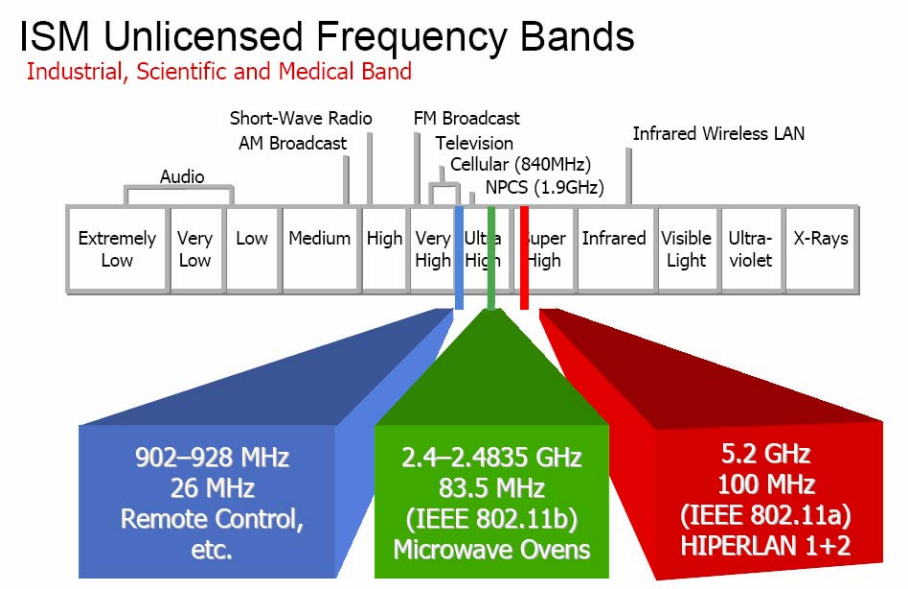



**Figure 2.1: The ISM Unlicensed Frequency bands. Figure from [23]**

The original 802.11 standard operates in the 2.4 GHz band with data rates up to 2 Mbps. Examples of places where WLAN is used are conferences, classrooms, office buildings, and military environments. When the first standard was released, the IEEE initiated two new task groups, to work on two new initiatives. The first resulted in the 802.11a, which was introduced as a standard in 1999, and which operates in the 5 GHz band with data rates up to 54 Mbps. The 802.11b standard was also released in 1999, operating in the 2.4 GHz band with data rates up to 11 Mbps. It has so far been the most successful standard of the IEEE, and it has millions of users every day. Other WLAN standards such as the European HIPERLAN2, is also present today, in addition to the many 802.11 standards.

A substantial part and a building block of the IEEE 802.11 architecture is the *basic service set* (BSS). The definition of a BSS is simply a group of stations which are under control of a single coordination function, and therefore is able to communicate with each other. The *basic service area* (BSA) is the diffuse geographical area covered by the BSS. When a station finds oneself in a BSA it can communicate with the other member of the BSS. The BSS is divided in two configuration modes:

- **The *Independent* BSS mode (IBSS)** is the formal name of this mode, more commonly known as Mobile Ad Hoc Network (MANET). In an Ad Hoc network the nodes (stations) are operating on a peer-to-peer basis. A group of nodes in a single BSS can communicate directly with each other. There is no need of any centralized access point (AP) or any wired network connections, (see Figure 2.2).

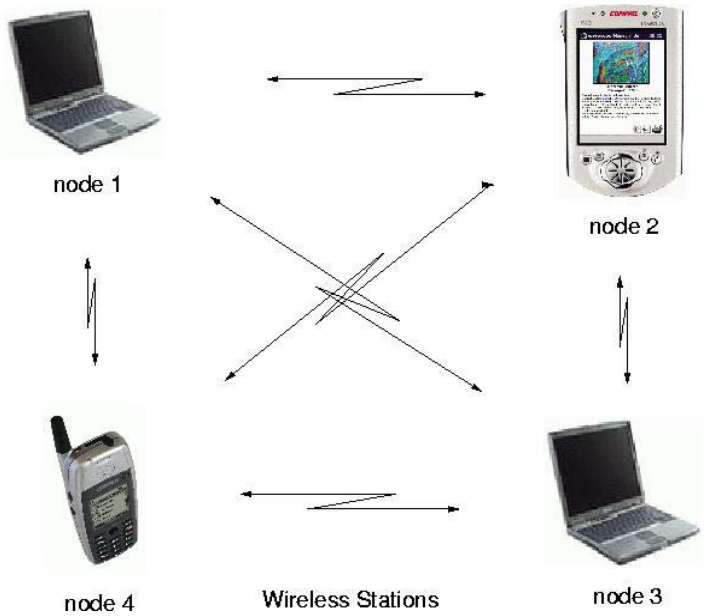



**Figure 2.2: Independent mode BSS (IBSS)**

- **The *infrastructure* mode (BSS)**, where at least one central access point (AP) is present. In infrastructure networks all communication is flowing through the AP, including communication between the mobile nodes in the same BSA. The AP is normally connected to the Internet, (see Figure 2.3).

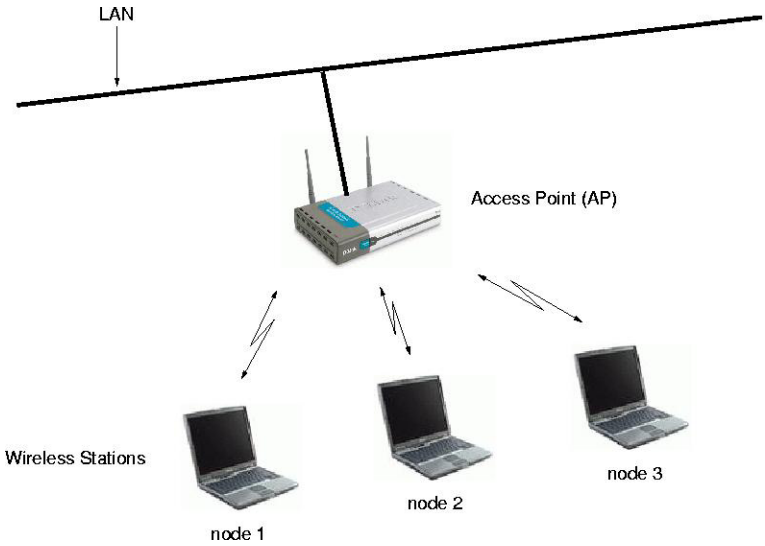



**Figure 2.3: The Infrastructure mode (BSS)**

In addition, the IEEE 802.11 standard allows connectivity between multiple BSS's, linked together in somewhat called *extended service set* (ESS), (see Figure 2.4). The ESS appears as one large BSS to the mobile node [5] [6] [7] [8].

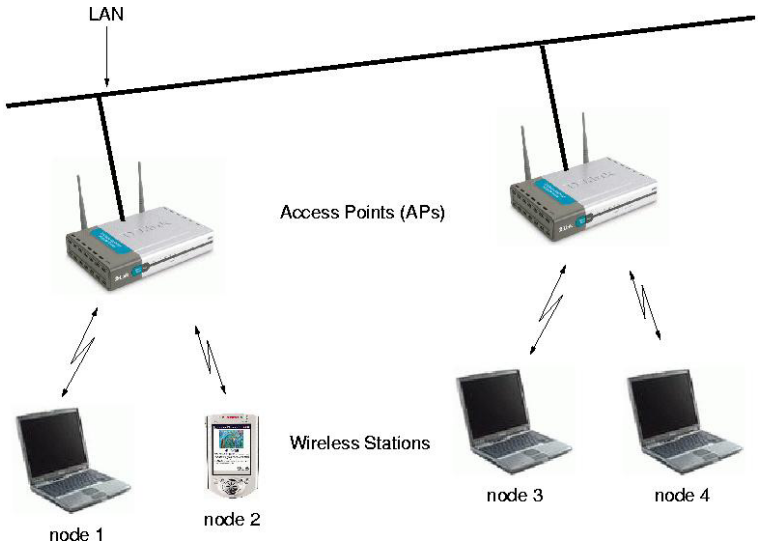



**Figure 2.4: The Extended Service Set (ESS)**

### 2.2.1 The Protocol Stack

Compared to the OSI model, the physical layers looks pretty much the same, but in 802.11 the layer that corresponds to the data link layer in the OSI model is divided into two sub layers. In the 802.11 standard the MAC (Medium Access Control) sub layer is responsible for the protocol data unit (PDU) addressing, frame formatting, fragmentation, and how the channel is allocated. The Logical Link Control (LLC) sub layers of the Data Link layer establishes and maintains the connections between network devices and hides the differences between the different 802 variants and make them indistinguishable to the network layer. The LLC is also responsible for the error correction and flow control at the Data Link layer. The SWAN module spans from the MAC layer up to the IP layer (see figure 2.5).

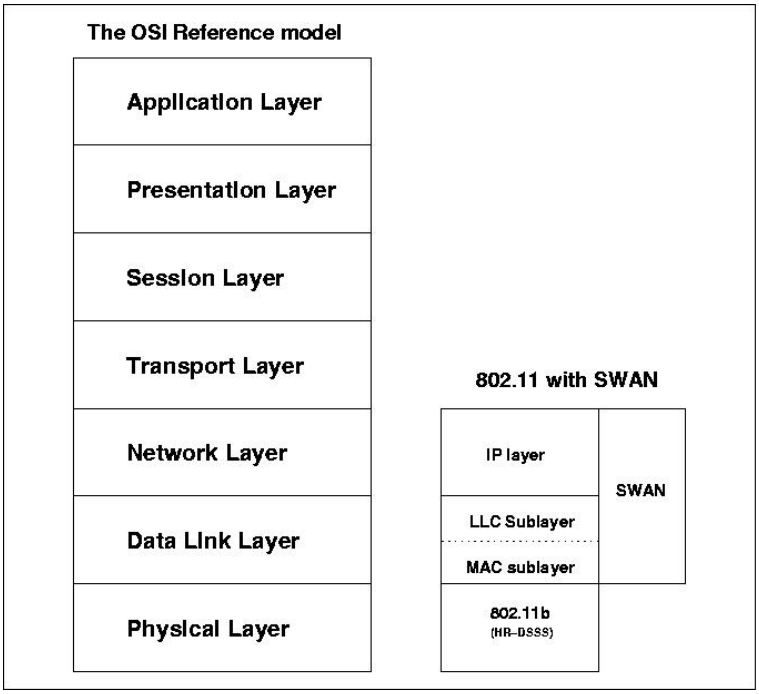



**Figure 2.5: The 802.11 protocol stack with SWAN and the OSI model**

One of the most pervasive channel access mechanisms at the 802.11 MAC layer is the *Distributed Coordination Function* (DCF), which is a *Carrier Sense Multiple Access/Collision Avoidance* (CSMA/CA) mechanism, with exponential back off. The DCF senses the channel before transmitting to determine if the channel is not in use by another station. The IEEE 802.11 also includes an alternative access method known as the *Point Coordination Function* (PCF), but only the DCF is suitable for the Ad Hoc mode. The main disadvantage of using the DCF is that it is only capable of delivering Best-Effort service to the network, and it is primary designed for asynchronous data transport. Activity has been established by the 802.11e Working Group to enhance the current 802.11 MAC protocol to support the QoS requirements for applications [5], [7], [21], [22] and [51].

## *2.3 Mobile Ad Hoc Networks*

In Latin, Ad Hoc literally means "for this", “towards this”, or "for this purpose only," and implies a spontaneous and temporary setting [32] [33].

### 2.3.1 Introduction to Ad Hoc Networks

Mobile Ad Hoc Networks (MANET) is usually simply referred to as Ad Hoc networks. In Ad Hoc networks the hosts are free to move, and no central agency is involved. A MANET is an autonomous system of mobile nodes; it can be created and used anywhere, anytime, and without any preexisting infrastructure. Ad Hoc networks also benefits from its simplicity, scalability and robustness due to its distributed nature. It may operate as a standalone network, or it could be connected to a larger network through a gateway, such as the Internet, for instance through a satellite connection. The hosts or mobile devices are usually called nodes.

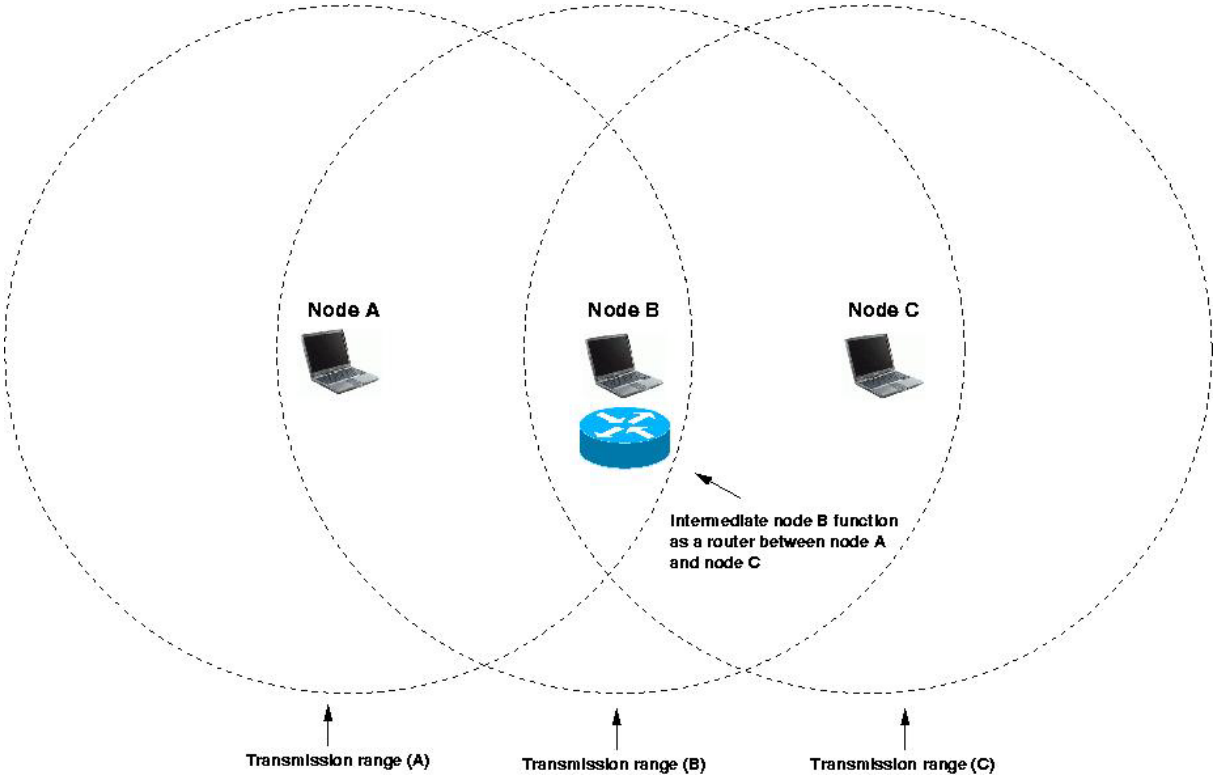



**Figure 2.6: A simple Multi Hop Ad hoc Network**

Wireless multi-hop ad hoc networks consist of several peer-to-peer connected nodes. A pair of nodes uses intermediate nodes to communicate when they do not have direct radio contact with each other. In a multi-hop ad hoc network the mobile nodes are not only edge nodes and therefore each mobile node must also be able to serve as a router as well as being able to forward packets generated by other nodes. Multi-hop communication is not supported by the 802.11 protocol, thus a special routing protocol suited for the dynamic multi-hop nature of ad hoc network must be used for this purpose. A loop free path from the source to the destination must be found so that packets do not

traverse the network infinitely. Such loop free path is called a *route*. Route updates may be required when the dynamic of the network changes the reach-ability relations among the nodes.

For instance, in figure 2.6, if node A wants to communicate with node C, it does not have direct radio contact with node C, because node A and C is out of each others radio transmission ranges. Instead, node A can reach node C through node B, because both node A and C are within B's transmission range. In order for node A to reach node C through node B, node B has to function as a router and forward packets it receive on its wireless interface from node A, which are destined for node C, out on the wireless interface again so they finally will reach the destination node, which here is node C. This is the simplest form of a multi-hop Ad Hoc network.

More nodes can easily join, but then we probably get more routes, and more nodes have to function as intermediate nodes in order for a source and destination pair to reach each other without direct radio contact.

#### 2.3.1.1 Usages of Ad Hoc Networks

The advantages of such networks immediately elicited interest from the military, rescue agencies and the police in the use of armed conflicts and disaster areas, where the conditions can be hostile and highly disorganized. Since the 1990s, Ad Hoc network have become increasingly important to commercial participants as well. Home or small offices networking with laptops and PDAs, such as in a classroom or conference situation, have emerged as other major areas of application. With the commercial participation, customers are demanding more and more features and functionality. Therefore, multimedia applications such as streaming of video and voice (VideoIP and VoIP) are important areas where special attentions have to be taken, to support the Quality of Service (QoS) requirement.

In near future, such networks are probably going to cooperate and coexist with other mobile systems such as the UMTS. It is also probably that a development of cooperative mobile robotic systems where the human presence is dangerous will use such technologies found in mobile Ad Hoc networks. [6] [13] [14]

### 2.3.2 Dynamics and Mobility

Mobility is one of the primary motivations and a very important feature for wireless networks. The nodes can move arbitrarily, thus the topology of the network may change randomly and rapidly. However, the mobility model may be very different in each Ad Hoc network. Mobility can occur within a constellation of mobile nodes, and the constellation can move as group in the same direction. Military networks may be highly mobile in a battlefield situation. On the other hand, the mobility patterns for commercial usage may be fundamentally different. Commercial users typically stand still or walk slowly using their PDAs and user tend to sit down when using their laptop computers. Typical examples of commercial Ad Hoc networks are a conference or a classroom situation. Thus the mobility in commercial networks tends to be more rarely and more slowly than in military

environment, unless the users stay inside a train or a car. The speed of the moving nodes can change the capacity of the network and hence the utilization of its resources.

Mobility is not the only contribution to the dynamics of wireless networks. Nodes can join and leave the network at any time, and new links and routes can be created, or old ones destroyed. Another contribution to the dynamic nature is the mobility of obstacles and interfering objects moving around in the network environment. Thus a wireless network in a static approach may face many of the same challenges as a network in a dynamic approach [28] [29] [30].

### 2.3.3 Ad Hoc Routing Protocols

Intermediate nodes are used in transmission when the source and destination node not are within radio range of each other. Each node on the intermediate path, process packets intended for the receiving node, and forwards the other packets to one of its neighbors. In that way all wireless nodes serve as routers. Route identification and reconfiguration is to be performed by the routing protocols, and the traditional TCP/IP *routing tables* are used to keep track of the routes to the hosts.

To ensure effective routing the first task is to find a suitable path between the source and destination. To prevent packets from traversing infinitely long paths, an obvious essential requirement for choosing a path is that the path must be loop-free. A loop free path between a pair of nodes is called a *route*. All the routes are kept in the routers local routing tables. A suitable QoS route means a path that supports the necessary resources to be able to meet the Quality of Service constraints desired. The QoS routing is often a NP-complete problem when more than one constraint is present. QoS routing includes request, identification and reservation of the available resources. Minimizing the number of intermediate hops is one of the principal ways of determining a suitable route to minimize the end-to-end delay and utilizing the network resources in a better way. The time needed to calculate routes certainly need to be low.

The dynamic nature of Ad Hoc networks may cause the reach ability relations among the nodes to change, and a route update may be required. In order to keep the network connected despite changes in the topology, the network performance is highly dependent upon the performance of the routing protocols. To be able to give any guarantee at all, it is important that the rate at which the topology changes is not “too” fast. In other words the topology updates of the routing tables must be faster than the topology changes. An Ad Hoc network is said to be *combinatorial stable*, if and only if the topology changes occur sufficiently slowly to allow successful propagation of all topology updates which is necessary.

Routing protocols in Ad Hoc networks have traditionally been *topology-based*; it uses the knowledge of the instantaneous connectivity of the network based on the state of the links. Topology-based routing protocols can be divided in three main categories, first the *proactive* (periodic or table-driven), which

attempts to maintain the knowledge of every current route to every other node in a dynamic way, regardless of packets flowing in the network based on a periodic exchange of control messages (e.g., OLSR), the *reactive* (on-demand) which only creates a route on-demand when it is necessary for carrying data packets (e.g., AODV). The third category is *hybrid* routing protocols. In addition, another class of routing protocols is the so called location-based routing protocols. In addition to *topology*-based information, this protocol uses information about the actual position to determine the routes [6] [30] [31] [50].

#### 2.3.3.1 Ad hoc On Demand distance Vector (AODV)

The AODV routing protocol is a routing protocol for ad hoc mobile networks, which is capable of both unicast and multicast routing. The algorithm is *on demand*, in the way that it only initiates new routes on request from a source node. These routes are maintained as long as they are needed by any of the source nodes. The freshness of a route is taken care of with sequence numbers. Furthermore, the AODV is loop-free, self-starting, and scales well on large number of mobile nodes. The AODV is the routing protocol used to support the current SWAN implementation.

Routes are built using *route request (RREQ)* / *route reply (RREP)* queries. A route request (RREQ) packet is broadcasted across the network when a source wants to reach a destination which it does not already have a route. Upon receiving the RREQ packet, intermediate nodes sets a backward pointer to the source in their routing tables. The intermediate nodes discard the RREQ if they receive one they already have processed, otherwise they rebroadcast the RREQ. A node unicast a RREP packet back to the source, if it is the final destination, or if it has a route to the destination, with a corresponding sequence number greater than, or equal to that contained in the RREQ. A forward pointer is set up to the destination as the RREP propagates back to the destination. The source node can start transmitting as soon as it receives the RREP. If a better route to the destination is found, the source updates its routing tables and starts using this route instead. The route is maintained as long as the route remains active. A timeout will occur once the source stops the transferring of data packets and the link will then be deleted from the intermediate nodes routing tables. A route error (RERR) message is sent to the source node if a link break occurs on an active route [27].

#### 2.3.3.2 Optimized Link State Routing (OLSR)

As mention above, the OLSR routing protocol have the routes immediately available when needed, due to its proactive nature. OLSR is an optimization of a pure link state protocol where all the links are flooded in the entire network. The multipoint relays (MPR) is one of the key concepts in the OLSR protocol. The set of MPR is selected such that it covers all the nodes that are two hops away. The routes to all known destinations are calculated through the nodes selected as MPRs, and these nodes are selected as intermediate nodes in the communication path. This significantly reduces the number of retransmissions in a flooded or broadcasted network. The OLSR protocol is also well suited for large and dense networks due to the optimization scheme

of the MPRs. It is also well suited for WLAN where radio transmission problems are frequent, because it does not relay on reliable transmission for its control messages [11] [12].

#### 2.3.3.3 Hybrid routing protocols

Hybrid routing protocols combines techniques from both the proactive and the reactive routing protocols. Zone Routing Protocol (ZRP) is such routing protocol. The idea behind ZRP is to divide routing into local and global zones [6].

### 2.3.4 Other considerations for Ad Hoc networking

When working with Ad Hoc network development, mobility management, power management and security are among the topics that might need to be taken into consideration. **Mobility management** can be divided in two broad categories: *Location management*, which keep track of the position of every unit (e.g., with a GPS), and *handoff*, which mainly concern about the continuity of a Real-Time stream. In order to have a seamless communication, a QoS capable Ad Hoc network has to support reestablishment and rerouting when handoffs occur. It is important with a fast reestablishment in the case of a handoff event [24]. An important part of **power management** is power control, where one is dealing with increasing and decreasing the level of transmitting power, in order to reduce the interference and increase the overall system capacity. It is also urgent for researchers to develop efficient and low power consumption algorithm and protocols [24]. The main concern in wireless **security** is the physical access to data transmission over radio waves. This makes 802.11 networks vulnerable to eavesdropping. In addition to the existing security concerns which are common on any networks, such as authentication and integrity, wireless networks are also vulnerable to possible *Denial of Service* (DOS) attacks by a malicious station [25] [26].

# 3 Quality of Service

In this chapter I will discuss Quality of Service both in general and in the special case of wireless ad hoc networks.

## *3.1 Beyond Best-Effort*

Since the ISM band is unlicensed and free for all commercial use, users will commonly experience significant interference in some locations. The interference can interact with the wireless signal and disturb or block it completely. Interference can be caused by simultaneous transmissions, when two or more 802.11 stations are trying to send at the same time. The interference can also be caused by other devices operating in the same geographical area using the ISM band such as a police car radio, or even a microwave oven. This type of interference is called noise. An 802.11 station can also experience echo from its own signal which again induces interference. Multi-path fading can be an additional source of interference to the 802.11. Multi-path fading arises when radio signals following different paths arrive out of phase at the receiver. When interference is experience, the bandwidth, which the 802.11 operate on is highly variable and far from the theoretical throughput limits. Yet another impact of interference is higher bit error and more frequent disconnection. In addition to the variable radio link quality, the new dynamics introduced by the WLAN gives us more stringent challenges.

Because of the interference problem, mobility, and because the 802.11 MAC sub layer protocol (DCF) only can deliver Best- Effort, the need for at least one additional mechanism, which can handle this problem and assure a certain level on the quality on the service delivered by the network.

Another problem arises when the network traffic gets excessive, then *congestion* occurs. In a congested network more data packets are entering a node than the packets leaving the node. A congested node is said to be a bottleneck. Then the nodes buffers get overloaded and the node have to drop packets. The delay for data packets entering a congested network is usually higher than for data packets entering an idle network [5] [6] [9] [17].

## *3.2 Quality of Service in General*

Quality of Service (QoS) is a widely used term, but it has different meanings to different people, and the issues associated with QoS are not very well understood. In RFC 2386, QoS is characterized as a set of service requirements that the network has to meet, during a flow from the source to the destination [50]. And The United Nations Consultative Committee for International Telephony and Telegraphy (CCITT) Recommendation E.800 has defined QoS as: "The collective effect of service performance which determines the degree of satisfaction of a user of the service" [49]. The previous definition is also used as basic definition of QoS by ITU and ETSI. At he IP layer, the QoS guarantees that can be offered are a product of the cooperation of operation of several mechanisms, such as policy management, traffic classification, bandwidth reservation and admission control, traffic conditioning, queuing, scheduling and discarding [44]. When considering the

Quality of Service, one can investigate and ensure the quality at all the layers in the TCP/IP protocol stack, but the overall Quality of Service is finally that experienced by the user at the application layer.

### 3.2.1 QoS Parameters

The IETF defines a flow as: "*set of packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties derived from the data contained in the packet and from the packet treatment at the observation point*" [34]. The four primary parameters that determine the QoS for each flow are; reliability, latency, jitter and bandwidth [51].

- **Reliability** concerns the need to manage *packet loss*. Packet loss is usually caused by network congestion and is defined as the ratio of the undelivered packets to the one that are sent. Packet loss serves to reduce the number of packets competing for the output link. When the sources discover that packets are lost it usually reduce the transfer rate. If the source does not receive an acknowledgement (ACK) for a packet it can be re-transmitted, depending on the type of application. The packet can also be delivered with bit errors. This is usually taken care of by verifying the checksum at the destination.
- **Latency** is the one-way end-to-end delay for a flow, experienced while passing through the network. The end-to-end delay may be introduced by intermediate routers and switches and due to the latency of the physical transmission media. Significant delays can appear when packets are queued for long periods of time.
- **Jitter** is the variations in latency between packets, mainly due to the variations in the volume of other traffic streams competing for the output link. Then packets may take different routes through the network, or they may encounter varying queue length. In a stream, delay variations must be removed on the receiver side in order to replay the stream. This can be achieved with a de-jitter buffer. Then packets are rearranged in a timely order. The delivery time of a packet must not exceeds the length of the receive buffer, because this packet then arrives too late, with respect to the replaying time, and the packet will be discarded from the buffer, which again have a degrading effect on the replay quality.
- **Bandwidth** concerns itself with how the network manages the entire stream of data packets flowing through it, particularly in times of network congestion.

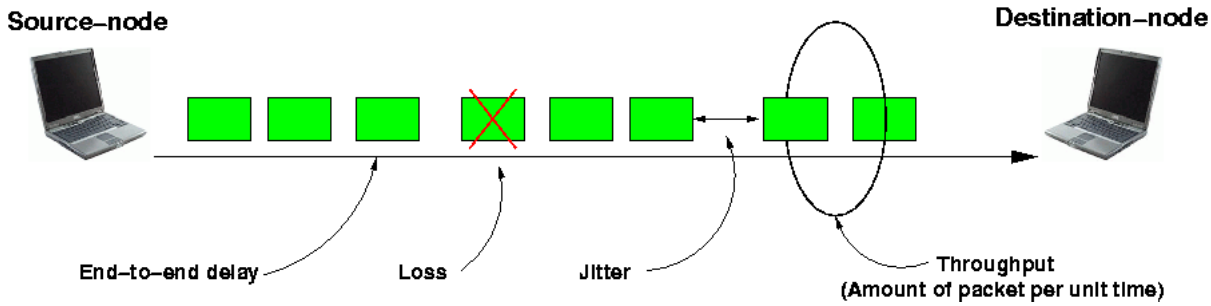



**Figure 3.1: Quality of Service requirements**

The treatment experienced by packets while traversing the network can be described by these parameters, which again can be translated into particular parameters of the network architecture components to ensure the QoS. The stringent of the QoS requirements for some selected applications are visualized in Table 3.1 [16] [35] [52].

| Application | Reliability | Delay | Jitter | Bandwidth |
|---|---|---|---|---|
| E–mail | High | Low | Low | Low |
| File transfer | High | Low | Low | Medium |
| Web access | High | Medium | Low | Medium |
| Remote login | High | Medium | Medium | Low |
| Audio on demand | Low | Low | High | Medium |
| Video on demand | Low | Low | High | High |
| VoIP | Low | High | High | Low |
| Videoconferencing | Low | High | High | High |

**Table 3.1: QoS requirements. Figure taken from [51]**

The **reliability** requirement is most important to the first four applications. This is usually taken care of by using a checksum to verify the content of each packet. The last four applications can handle bit errors, without using any control routine as the checksum correction. The likelihood of loosing a packet usually grows as the packet size increases. In an unreliable channel, as often is the case in a MANET, smaller packets are preferred. A drawback of using small packets payload compare to the fixed headers is waste of useful bandwidth.

When it comes to **delay**, file transfer application including video and e-mail, are not sensitive to delay. Web surfing and remote login have moderate sensitivity on delay, and audio and video on demand does not require low delay. On the other hand, IP telephony and videoconferencing have very strictly requirements on the delay [51]. The International Telecommunication Union (ITU), recommend that the one-way delay for general network planning, should not exceed 400ms. However, in highly interactive tasks, such as IP telephony, videoconferencing and interactive data application, these applications may be affected of delays below 150ms. Most applications would not be significantly affected, if the delays are kept below 150ms [16].

E-mail, file transfer, and web access is not very sensitive on packets arriving with irregular intervals. Remote login is slightly more sensitive to **jitter**, but applications involving audio or video are extremely sensitive to jitter. A typical audio- or video streaming application cannot handle more than a few milliseconds of variation before it is audible or the effect is visual.

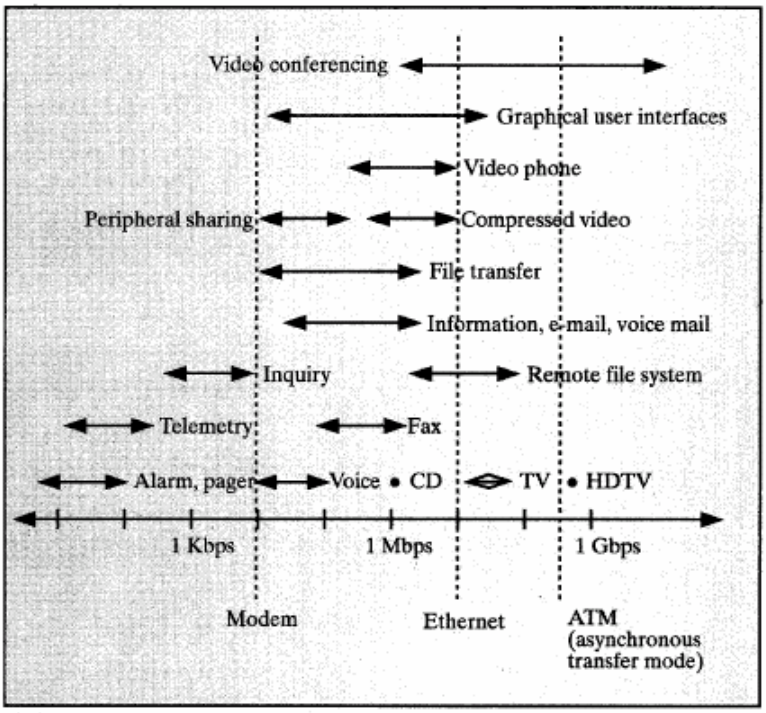



**Figure 3.2: Application bandwidth requirements in bits per second. Figure from [17]**

The applications demand for **bandwidth** can be categorized in sustained, bursty and interactive. For voice, 20 ms of speech is contained in speech packet from 20 to 160 bytes at 8 and 64 Kbps respectively, depending on the compression, while a high quality MPEG-2 video may require up to 10 Mbps. It depends on the quality, but typical rate for MPEG is about 1.2 Mbps. File transfers are bursty and tends to grab as much bandwidth as they can to make a fast data delivery. The bursty traffic is usually a common reason of congestion and must be controlled. Different applications bandwidth requirement can be viewed in Figure 3.2 [10] [35] [43].

As we can se from the table 4.1 both IP telephony and videoconferencing is sensitive on both the delay and jitter requirements. To satisfy those applications requirements on delay and jitter, the network has to offer special treatment so that the tight guarantees can be met. In order to enforce such network policies, classifying in traffic classes, marking of packets and flows, and scheduling using a queuing mechanism in the network is most likely necessary. The ability to varying the data rate upon feedback of the available bandwidth in the network, using compression and reducing the play back quality, may also be a likely solution to meet the QoS demands.

## *3.3 QoS Mechanisms*

Among the many service models and mechanisms proposed by the Internet Engineering Task Force (IETF) are the Integrated Service model (IntServ) and the Integrated Service model (DiffServ).

### 3.3.1 Integrated Services (IntServ)

The philosophy of the IntServ model is to provide special QoS for specific user packet streams, or "flows" through resource reservation in the routers, which in turn represents a fundamental change to the Internet model. Two service classes in addition to best-effort service (which is characterized by the absence of any QoS specification at all) are proposed:

- *Guaranteed service*s provide an assured amount of bandwidth for applications with tight bounds on both the delay and the jitter requirements.
- *Predictive services* also called controlled load services is intended for more tolerant and adaptive applications, which do not require a perfect reliable bond on the delay requirement. The provided service will be as close as possible to the service received by a best effort flow in a lightly loaded network, even though the network as a whole may be heavily loaded.

The implementation framework in [36] includes four traffic control components: admission control, the classifier, a packet scheduler and a resource reservation protocol. The *admission controller* is responsible for the reservation policies and determines whether the QoS request for a new flow can be granted without any devastating effect to earlier guarantees. The purpose of the *classifier* is to map each incoming packet into some class. Forwarding of the different packet streams is managed by the *packet scheduler* who uses queues and perhaps other mechanisms like timers. Packets' belonging to the same class receives the same treatment from the packet scheduler. To create and maintain flow-specific state in the endpoint hosts and in routers along the path of a flow a *resource reservation protocol* is needed. The protocol should be able to find a route that supports resource reservation and has sufficient unreserved resources for a new flow. It should also adapt to route failure and route change without failure. The protocol recommended by the IntServ group is The Resource ReSerVation Protocol (RSVP), which reserves a portion of the output link in each router along the path of a flow [36] [37] [38].

### 3.3.2 Differentiated Services (DiffServ)

In the differentiated services model, traffic entering a network is classified and possibly conditioned at the boundaries of the network, and assigned to different behaviour aggregates. Unlike IntServ, no signalling are exchanged between the source and the destination. It does not attempt to guarantee a level of service. It rather tries to make some relative ordering of aggregations. This causes one traffic aggregation to receive better or worse treatment relative to other aggregations. Within the core of the network, packets are forwarded according to the *Per-Hop Behaviour* (PHB). A DiffServ domain is a contiguous set of nodes which operate with a common service policy and set of PHB groups.

Two primary PHB forwarding classes in addition to the Best-Effort service are proposed by the IETF:

- *Expedited Forwarding* (EF) – The EF class also described as Premium service is intended for applications requiring low latency, low jitter, low packet loss, assured bandwidth and end-to-end service.
- *Assured Forwarding* (AF) – The AF class is divided in four separate classes intended for applications requiring better reliability than best-effort service.

The boundary of a DiffServ domain is well-defined and consists of boundary nodes that act both as ingress and egress nodes for traffic in different directions. A traffic flow enters a DiffServ domain at an ingress node and leaves at an egress node. The boundary nodes classify and possibly stipulating ingress traffic to ensure that packets passing by the domain are appropriately marked to select a PHB from one of the PHB groups. Based on the content of some portion of the packet header, the packet classifiers select packets in a traffic stream and lead the packets to a logical instance of a traffic conditioner. Traffic conditioning ensures that the traffic entering the DiffServ domain conforms to the rules specified in the given policy. Elements included in the traffic conditioner may be: meter, marker, shaper, and dropper.

The traffic *meters* measure a stream of packets temporal properties and passes state information to other conditioning functions to trigger a particular action for each packet. To add a packet to a particular DiffServ behaviour aggregate, the packet *marker* set the Type of Service (TOS) byte (DS field) of a packet to a particular codepoint. Several differentiated service classes can be created by marking the DS fields of the packets differently. In order to bring a stream into accordance with a traffic profile, the *shaper* delays some or all of the packets in a traffic stream based on their DS fields. Packets are discarded if there is not sufficient buffer space to hold the delayed packets. The process known as *policing the stream* is executed by the *dropper* who discards some or all of the packets in a stream, to bring a stream into compliance with a traffic profile.

For a customer to be able to receive differentiated services, a Service Level Agreement (SLA) must be arranged with its Internet Service Provider (ISP). In an SLA, which can be either static or dynamic, the basically service classes

supported and the amount of traffic allowed in each class is specified [38] [39] [40] [41] [42].

### 3.3.3 Adapting Service models to MANET

A lot of work has been done to achieve QoS in wired networks, but unfortunately this cannot be directly applied in Ad Hoc networks. In the IntServ/RSVP proposal, there are several factors which make the model not suitable for MANET's. First, huge storage and processing overhead due to the state information that has to be maintained is a drawback. Second, in the case of a topology update, all the reservations would have to be renegotiated simultaneously. This may lead to congestion of several routers. Third, in a MANET, every node has to function as a router, thus each node also has to perform the admission control, classification and scheduling. These tasks will require resources, which usually are not available in an Ad Hoc network. In addition, the RSVP signalling packets will compete with other data packets for the usually scare resource of bandwidth.

On the other hand, DiffServ is a more proper model for MANET's, even though the model cannot be applied directly in such a network. A clearly drawback on using the DiffServ model is that the dynamic nature of Ad Hoc network cause an unclear definition of what is the core network, ingress and egress routers. For the interior routers, DiffServ is a more lightly weighted model than IntServ.

For the future, in order to make suitable QoS model for MANET's one approach could be to make proper changes to the DiffServ model to make it fit the dynamic nature of Ad Hoc networks. Another approach could be to take advantage of the benefits from both the IntServ and the DiffServ model. Several models have been proposed, among them are the SWAN model [3] [49].

## *3.4 Quality of Service in Ad Hoc Networks*

Due to the dynamic nature of Ad Hoc networks, the challenges in utilizing the resources in such networks are more numerous and more difficult than in a traditional wired IP network such as the Internet. These challenges include subjects as effective routing, channel access, mobility management (hand-off e.g.,), security issues (e.g., Denial of Service attacks), power management and Quality of Service, mainly pertained to bandwidth management, and to the delay and jitter requirements. All these challenges are potential sources to degraded Quality of Service experienced by the users.

So far, Ad Hoc networks have only been able to support best-effort service, without any guarantees. To satisfy the demand of using the same applications (web browsing, interactive multimedia e.g.,) which are common in the traditional wired networks and to carry out other complex operations, various Quality of Service attributes for these applications must be satisfied. Quality of Service solutions for wired networks cannot be directly applied to wireless Ad Hoc networks.

**Bandwidth management** means administrating the scarce resource of the wireless channel among the users. Wireless networks have until recently delivered lower bandwidth than their wired counterparts. With an optimizing of the coding of bit in the transmission, it is expected that the IEEE 802.11n will be able to deliver a theoretical data rate of 320 Mbps by 2005 [9]. In a BSA, the network bandwidth is divided among the users. In addition to the raw transmission bandwidth, an important measurement of the network capacity is the deliverable bandwidth per user, which is dependent of the density and distribution of nodes. There are two ways to increase this capacity. One approach is to divide a BSS into two or more new BSS's operating on different wavelength. The other approach is to decrease the geographical area of the BSA, which means decreasing the radio range of the antennas. Due to interference and the dynamics, the available bandwidth of Ad-Hoc networks is highly time varying, and need to bee managed efficiently and in a fair way among the users to achieve a guaranteed Quality of Service. Figure 3.3 shows the application Network Stumbler measuring the Signal/Noise ratio.

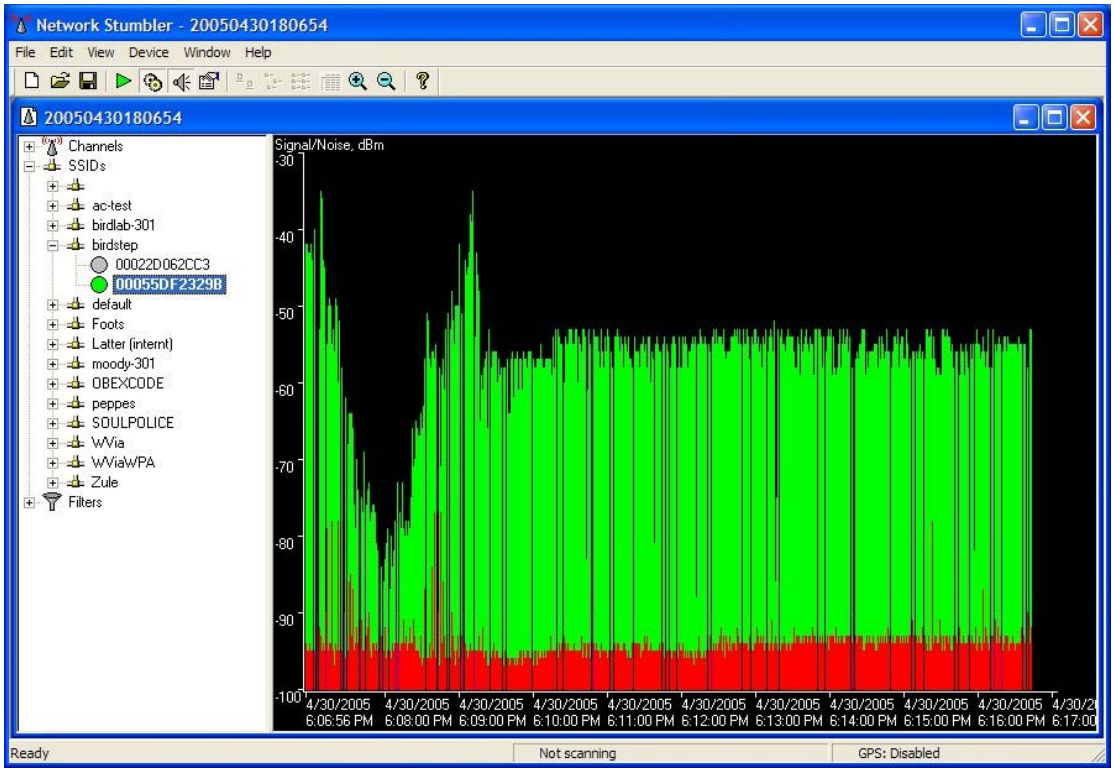



**Figure 3.3: Wireless Signal/Noise ratio measured in Network Stumbler**

Another concern, is how often and how much administrative and control information that may be exchanged due to the limited bandwidth available and due to the generally hostile transmission characteristics of wireless Ad Hoc networks. A good implemented solution is needed to satisfy the existing requirement to achieve the desired level of the Quality of Service. Together with performance analysis studies of Ad Hoc networks with QoS constraints this is still an open area for more research [6], [15], [17], [50].

# 4 Stateless Wireless Ad hoc Networks (SWAN)

In this chapter I will introduce the reader to SWAN, a network model proposed by the COMET Group at Columbia University in New York.

## *4.1 Introduction*

SWAN is a stateless network model proposal, which is supposed to ensure a certain level of QoS in wireless Ad Hoc networks. The SWAN mechanism, which is based on a best-effort MAC protocol, is placed between the IP and the MAC layer. SWAN uses *rate control* for Best-Effort traffic, and sender-based *admission control* for UDP Real-Time traffic. A *classifier* and a *shaper* is also part of the SWAN mechanism. The admission controller located at the source nodes, uses a bandwidth probe message to find the instantaneous bottleneck capacity path in the network. There is none state information about the ongoing flows kept in the network. Instead SWAN uses feedback information from the network.

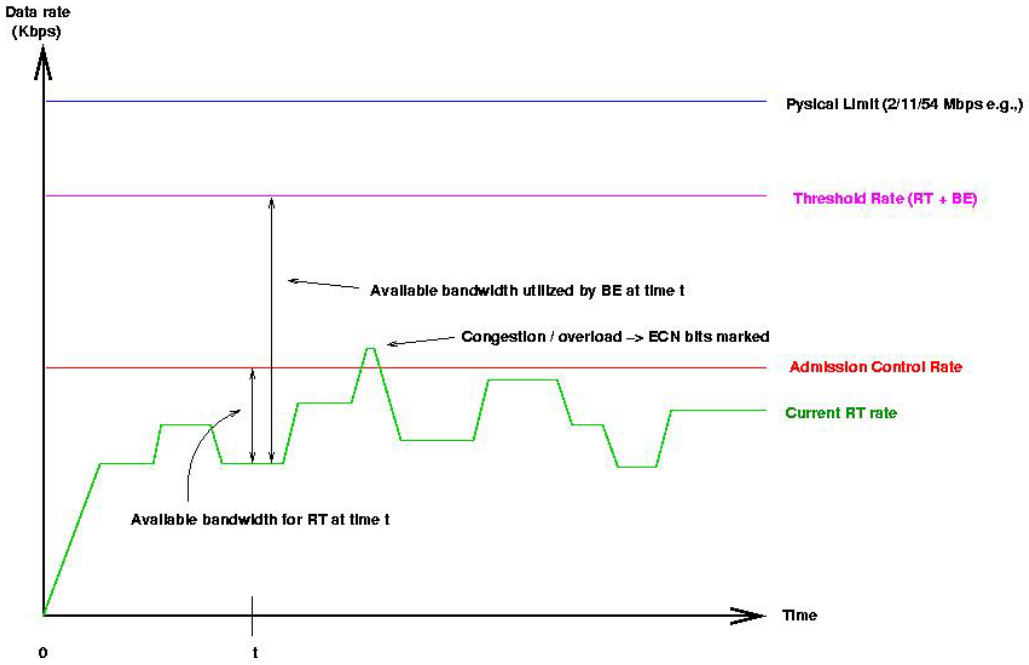



**Figure 4.1: Graphical presentation of data-rates and -limits**

For regulation of admitted Real-Time traffic, SWAN uses Explicit Congestion Notification (ECN) as feedback, when the network experience overload. MAC delay measurement from packet transmission is used as feedback for rate control of both TCP and UDP Best-Effort traffic. The admission controller gets feedback from measurement of the aggregated real-time traffic. To limiting the experienced delay, the total rate of both real-time and best-effort in a local shared media channel should be maintained below a *threshold rate*, see Figure 4.1. For this purpose SWAN adopts the Adaptive Increase Multiplicative Decrease (AIMD) rate control mechanism in order to improve the performance of Real-Time UDP traffic and at the same time let Best-Effort traffic efficiently utilize any remaining bandwidth. Unfortunately, for a number

of reasons it is not desirable to admit the Real-Time traffic up to the threshold rate. First, the flexibility upon channel dynamics will be intolerant. Second, if the Real-Time traffic is allowed up to the threshold rate, Best-Effort traffic would be starved. Therefore, SWAN uses a simpler approach and only admits Real-Time traffic up to a more conservative rate than the threshold rate. The local available bandwidth, of a shared media channel, estimated by each node, is then the difference between the rate of the Real-Time traffic and this new conservative "admission control rate" [3] [4].

## *4.2 The architecture*

In chapter 2.2.1 I presented how the SWAN network model fits in to the OSI stack. A deeper picture of the SWAN architecture can be viewed in Figure 4.2. Here we can see how each data packet traverse the SWAN network model. First, upon a request from higher layers, the admission controller probe to see if enough resources are available. If a data packet is admitted, it is marked by the admission controller, otherwise it is left unmarked. The classifier then inspects the IP header to see if the packet is marked or not. An unmarked packet is passed down to the shaper, while an unmarked packet bypasses the shaper and is passed straight down to the MAC layer. The shaping rate at which the shaper forward the data packet to the MAC layer is adjusted by input from the rate controller, which again takes its decisions upon feedback from the MAC layer delay. From the MAC layer the packet is passed to the Physical Layer and converted to radio signals.

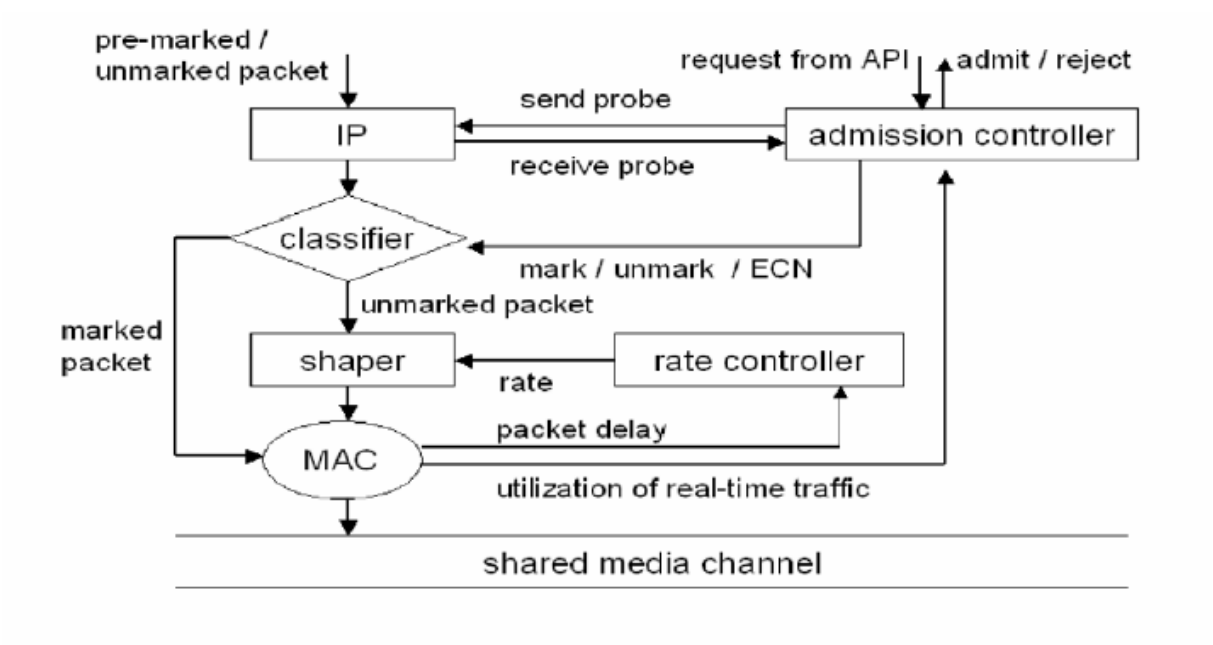



**Figure 4.2: The architecture of the SWAN network model. Figure from [1]**

### 4.2.1 Rate Control

Best-effort traffic is independently regulated in every mobile node. The transmitting rate used by the shaper, is determined by the rate controller using

a modified AIMD rate control algorithm. In the AIMD algorithm, the TCP sender increases its congestion window by one packet each roundtrip time (RTT), in times of no congestion. When congestion is indicated, the AIMD mechanism cut the window size aggressively. Then the TCP sender decreases its congestion window, so that the new congestion window is half of the minimum of the congestion window and the receiver's advertised window.

In SWAN the rate is based on the packet delay measured by the MAC layer. The packet delay is simply the time it took to send the packet from the source to the destination, including the acknowledgement from the receiver. At the source node, the delay ($\mathbf{Delay}_{MAC}$) is measured by subtracting the time the MAC layer receives a packet ($\mathbf{T}_{send}$) (from the upper layer), from the time an ACK is received from the destination $\mathbf{T}_{ACK/recv}$.

$$\mathbf{Delay}_{MAC} = \mathbf{T}_{ACK/recv} - \mathbf{T}_{send}$$

Every mobile node increases its transmission rate every **T** seconds until the packet delays become noticeable, (*Additive Increase* with the increment rate of **c** Kbps). This delay is detected by the rate controller when one or more packets have greater delay than the threshold delay **d** seconds. When the rate controller discover excessive delays, it backs of the rate (*Multiplicative Decrease* by **r %**). Every **T** seconds the shaping rate is adjusted. To respond to the dynamics of Ad Hoc networks the time **T** should be kept below a certain limit.

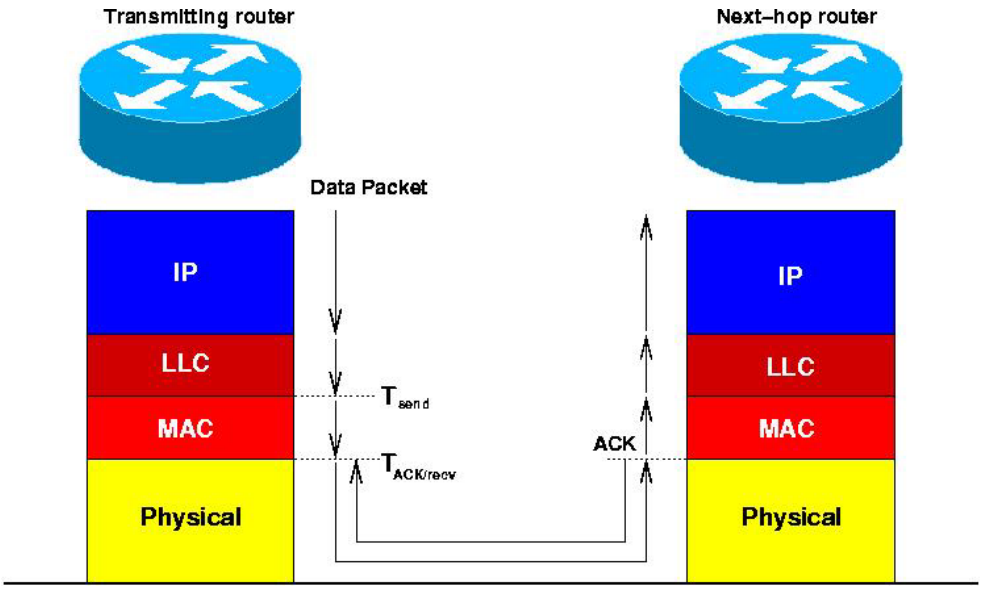



**Figure 4.3: MAC delay**

The shaping rate is the maximum rate determined by the Rate Controller in which the shaper is allowed to transmit. The actual transmission rate is the current rate at which the shaper actually is transmitting the Best-Effort

packets. If the difference between the actual transmission rate and the shaping rate is too large, the mobile node is enabled to send a burst, which might raise the delay for the real-time traffic. The rate controller monitors the actual transmission rate and adjusts the shaping rate to be **g%** above the actual rate if the difference between the actual rate and the shaping rate is greater than **g%** of the actual rate [3] [4].

### 4.2.2 Admission Control

The real-time traffic is measured by the admission controller in every node to find the local resource availability. In the shared wireless channel, nodes listen to packets sent within their transmission range in order to calculate the rate of Real-Time traffic. An average of these measures is used to smooth out small-scale traffic variations. Nodes actual available bandwidth is the difference between the current physical bandwidth and the total Real-Time packets sent within the transmission range. Since the SWAN mechanism is trying to improve the performance of Real-Time traffic, SWAN adopts the threshold rate, at a level below the physical limit, as a total maximum data rate of both Best-Effort and Real-Time traffic. Unfortunately, if the Real-Time traffic is allowed to consume the entire data rate up to the threshold rate, there would be no flexibility upon handling additional Real-Time traffic due to network dynamics and the Best-Effort traffic could be completely starved. Therefore, a more conservative rate than the threshold rate is used by SWAN. The difference between this conservative *admission control rate* and the current rate is then the available bandwidth for Real-Time traffic that a node can “see”. The Real-Time traffic is allowed up to this admission control rate. The static value of the admission control rate is coarse and approximated. All remaining unutilized bandwidth from the current rate of Real-Time traffic and up to the threshold rate will be absorbed by potentially best-effort traffic.

When a node wants to start a real-time session, the admission controller in this source node sends a bandwidth probing request towards the destination node, (see Figure 4.4). The task of this probing packet is to visit every intermediate node and then find the end-to-end bandwidth capability along the route from source to destination. If an on-demand routing protocol is used by the network and an intermediate node has no known route to that destination a route discovery process must be initiated before the probe can proceed to the next hop node. The nodes between the source and destination, updates the bottleneck field in the probing request packet, if the available bandwidth in that node is less than the current value in the packet. When the destination receives the bandwidth probe request packet it copies the bottleneck field into a bandwidth probe reply packet. The reply packet is sent directly to the source node and is not restricted to follow the request packets reverse path back to the source node. A simple admission control is done by the source node when it receives the bandwidth probe reply packet. The admission controller simply compares the new Real-Time session required bandwidth with the bottleneck field in the reply packet. If the bottleneck field in the reply packet is the same size or greater than the required bandwidth, then the session is admitted. Otherwise packets belonging to this session are refused and considered Best-Effort until a new probe request might find enough available resources in the network. In this model the intermediate nodes does not need to do any

reservations on behalf of a session, neither are any admission control executed in other nodes than the source node.

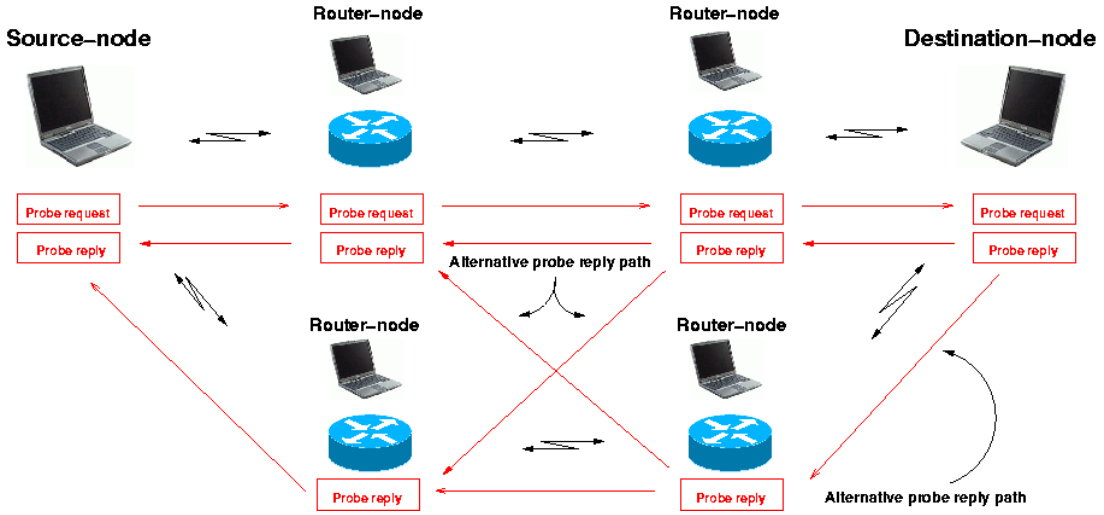



**Figure 4.4: Probe message**

The packet belonging to an admitted real-time session are marked RT (Real-Time) and will then bypass the shaper mechanism. An important assumption here is that each source nodes regulates its real-time sessions based on the decision of its admission controller [3] [4].

### 4.2.3 The Classifier

Best-effort traffic is separated from the real-time traffic by the classifier in order to let the shaper process only best-effort packets. When a packet leaves the IP layer the classifier examines the DS field in the IP header. If the packet is marked RT it will bypass the shaper, otherwise the packets are forwarded to the shaping mechanism [3]

### 4.2.4 The Shaper

The purpose of the shaper is to delay BE packets. This delay is adjusted by the rate calculated by the Rate Controller. The shaper is implemented as a simple leaky bucket algorithm where the input rate could be bursty and where the output rate is controlled.

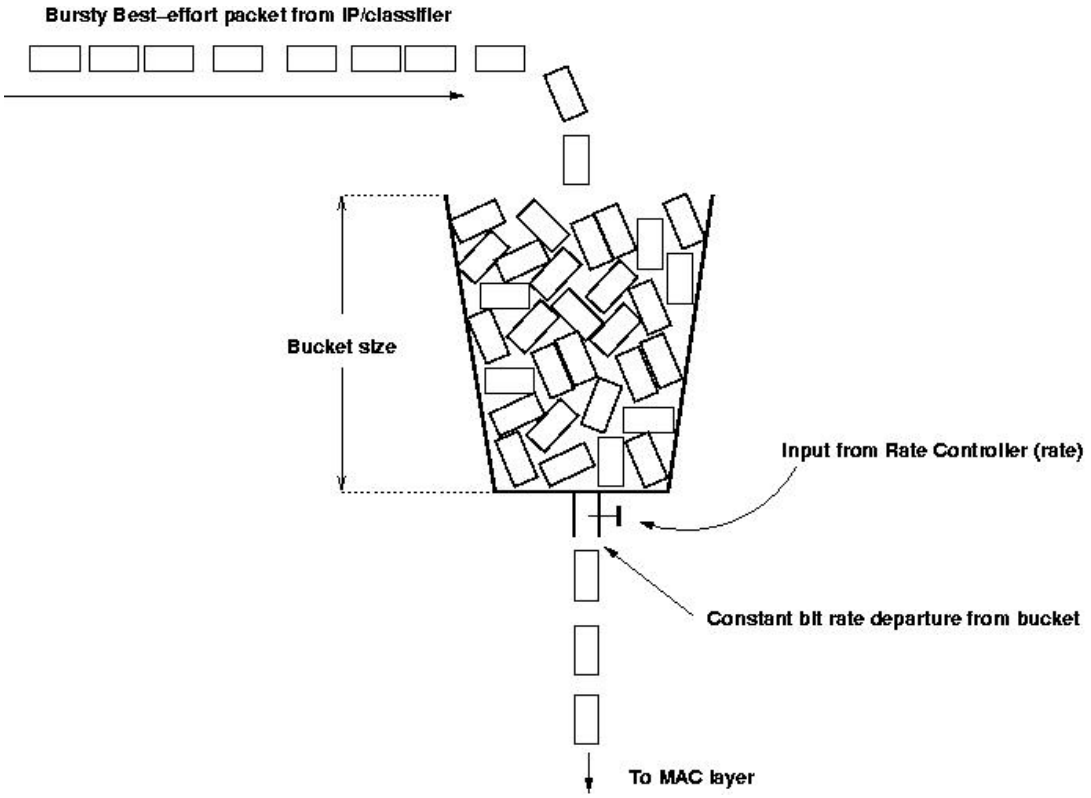



**Figure 4.5: The shaper; leaky bucket**

In Figure 4.5 the bucket represents an abstraction of a set of actions the network takes to monitor and control traffic. The bucket depth represents the tolerance of packet burst over a period of time. When the bucket is full, the incoming packets are dropped. The hole in the bucket represents the instantaneous rate at which the packets are allowed to transmit.

The departure rate of the shaper (bucket) is determined by the rate controller which is using the AIMD rate control algorithm based on feedback from the MAC layer [1] [3].

### 4.2.5 Probe Message Format

Two control messages are defined by SWAN's admission controller; "bandwidth probe request" and "bandwidth probe reply". Both are using UDP with a port reserved for the SWAN module. The probe request packet contains a "bottleneck bandwidth" field which contains the bottleneck of the path from the source to the receiver when it reaches the receiver. The total layout of the probe message format is illustrated in Figure 4.6 [4]. (In ns-2 implementation of SWAN port 252 are used).

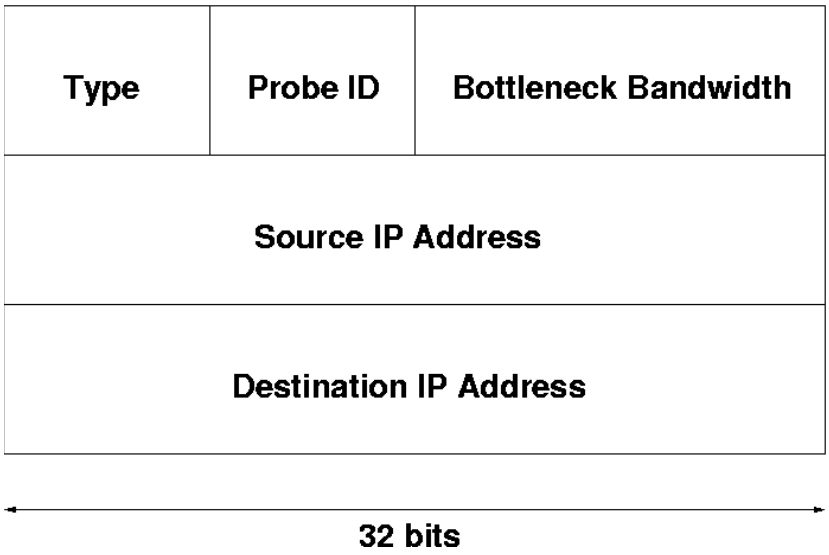



**Figure 4.6: The Probe message format**

The SWAN network model is using AODV as its corporate routing protocol, and the probe packet is placed as a piggyback message on a Route Request message in AODV.

### 4.2.6 Mobility and False Admission

In an Ad Hoc network, the nodes are not aware of how and when the *mobility* takes place. Harmful resource conflicts will most likely be the result, if re-routing due to mobility is not taken into consideration.

*False admission* arises when several nodes simultaneously probe the network for bandwidth and their paths from the source to the destinations are traversing one or more identical intermediate nodes. Both source nodes may then receive enough bandwidth, in the probe request, when the available bandwidth strictly is just enough for only one of the sources. This false admission is nothing the source nodes is aware of, and therefore admits new sessions of real-time packets, when the session actually should be denied.

The re-routing due to mobility and false admission can cause excessive delay and jitter for the real-time traffic if it is left unresolved.

### 4.2.7 Explicit Congestion Notification (ECN)

When running SWAN in an Ad Hoc network, every node periodically measures real-time traffic to get the exploitation ratio of local link such that the available bandwidth can be calculated. A node starts marking the ECN bits in the IP header of the real-time packets with Congestion Experienced (CE), when it detects violations such as congestion and overload conditions, due to re-routing and “false admission”. When the receiver discover the CE mark in the IP header it notifies the source node through a regulate message when it discover the ECN bits in the IP header. The source node try to re-establish its real-time session when it receives a regulate message. With the same bandwidth needs as the old session, the source node sends a probe request

toward the destination, in the same manner as when setting up a new session. If the bandwidth field in the new probe response packet is less than what the existing session needs, the session is terminated by the source node [3] [4].

### 4.2.8 Regulation Algorithm

If all real-time packets are to be marked CE, when the node detects overloaded condition or congestion, then every session that has this node in the path from source to destination has to re-establish at the same time. This is a behavior that is not wanted and is indeed vastly inefficient. Two and more systematic approaches, which penalize only a small number of sources and sessions, are considered in [4].

#### 4.2.8.1 Source-Based Regulation Algorithm

When overload or congestion is experienced by a node it marks all real-time flows with CE. Instead of re-establish the session when receiving a regulate message, the source node wait a random amount of time before it sends a new probe message with the intention to make a re-establishment of the session. With source-based regulation, situations where a number of source nodes simultaneously initiate re-establishment of their real-time session, and at the same time find the network overbooked, and in which they all drop their sessions will be avoided. Gradually the rate of real-time traffic will decrease until it gets below the admission control rate. Then the intermediate nodes, which experience congestion and overload will stop marking the packets. The source nodes need to be able to differentiate between regulation messages due to re-routing/mobility and false admissions. This can be taken care of by keeping some state information in the nodes about newly admitted flows and existing flows. With this approach, the source nodes can take immediate action, when a session is admitted by mistake, due to false admission [4].

#### 4.2.8.2 Network-Based Regulation Algorithm

In Network-Based regulation a randomly selected “congestion set” is picked out by the nodes that experience congestion/overload condition. Only real-time sessions associated with this congestion set is marked with CE. The congested set is marked for a period of time T, and then the congested nodes calculate a new congested set for a new period. When the measured real-time traffic fall below the admission control rate, the nodes stop marking the packets. Under congestion and overload, the intermediate nodes need to distinguish between flows which is admitted due to false admission, or flows which is rerouted due to mobility. In the Network-Based approach, the source nodes could mark packets in a real-time session to inform the intermediate nodes, whether the session is new or old. However, this case requires some calculations by the intermediate nodes in order to distinguish the new flows from the old so that they can take correct action upon false admission [4].

# 5 Problems and trouble with SWAN

In this chapter I will discuss the problems and troubles I came across, during my work with this master thesis and the SWAN network model.

## *5.1 The SWAN articles*

At first glance, the SWAN mechanism looks like a promising protocol in order to deliver a satisfying level of QoS for wireless Ad-Hoc networks. In [1] and [3], the SWAN mechanism claim to experience low and stable delays in an experimental wireless test bed, during various mobility, traffic and multi-hop situations. However, the delay measured in the wireless test bed in [1] and [3] is the average delay at the MAC layer, while the most critical issue is whether the end-to-end delay from application layer to application layer is low enough to support Real-Time traffic. In other words, this end-to-end delay should be kept below 150 milliseconds (see section 4.2). In [3] they even conclude that the result presented implies that the proposed SWAN mechanism is able to support Real-Time traffic within a single shared media channel. When it comes to the multi hop scenarios, this statement is found in [3]:

"*The average end-to-end delay of the real-time traffic in the original system grows linearly from 8 to 39 msec, as the number of TCP flows increase from 2 to 12 flows, respectively. In contrast, the average delays of real-time traffic in the proposed system remains around 5- to 7 msec*"

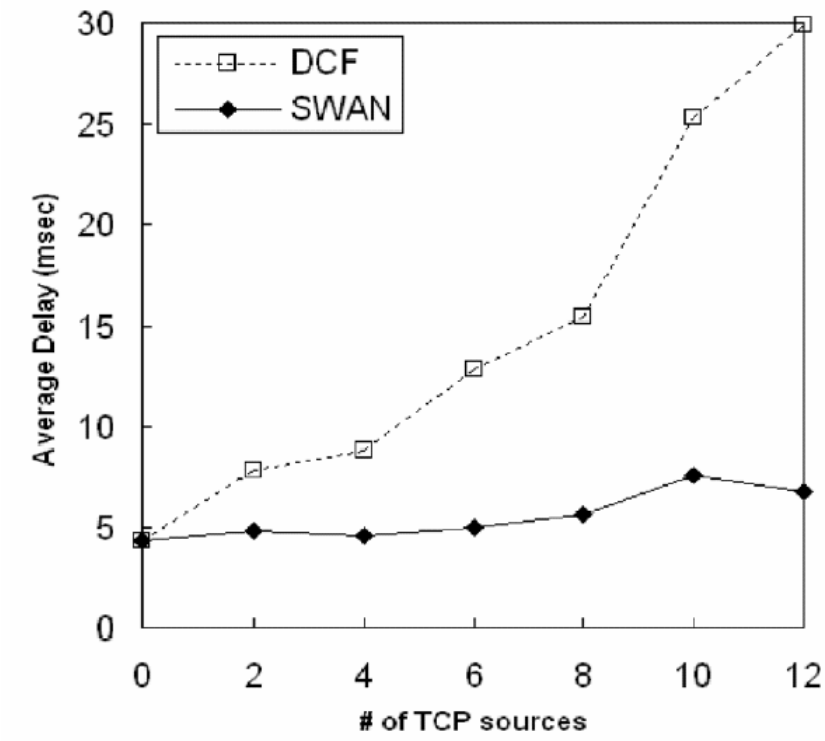



**Figure 5.1: Average delay of Real-Time traffic vs. Number of TCP flows [1].**

See Figure 5.1 for a graphical presentation of this. These results, where far from the experience during the early setup of my own simulation scenarios of SWAN in ns-2. I suspected this statement concerning the delay to be false, or

at least not telling us all that we need to know, in order to figure out if streaming of Real-Time applications could be supported. Therefore, it was important to recreate these simulations scenarios. Then measurement of the end-to-end delay experienced at the applications layer for both a single shared channel and a multi hop situation has to be made, so it is possible to figure out at what level the SWAN mechanism is able to support Real-Time traffic.

In [1] and [3] the term “goodput” is also widely used. I have done some research on this topic, and found a great variety in the definition of “goodput”. After e-mailing the people behind SWAN, it turns out that their definition of “goodput” is similar to throughput but only includes the packets that are useful to the applications. Throughout this paper, the term “throughput” which is the packets received at the application layer is used.

In [3] it is claimed that the action taken to improve the delay comes at a cost of only 15-20 percent reduction of TCP throughput compared to the original system. In addition to the end-to-end delay, it is important that measurements of the throughput also have to be taken into consideration when looking at the performance of the SWAN mechanism.

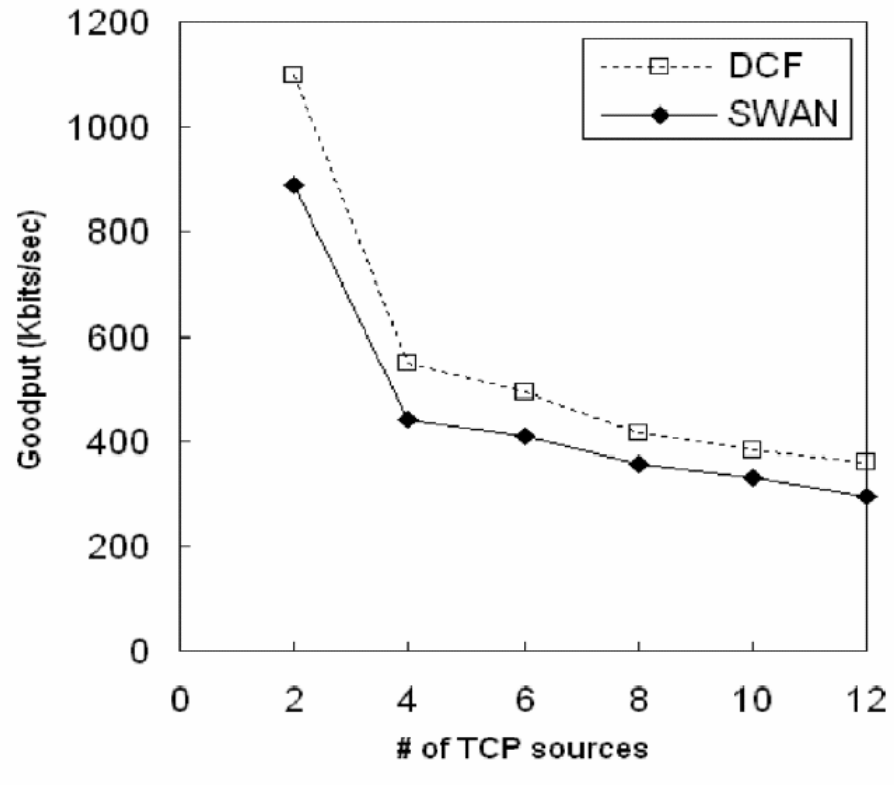



**Figure 5.2: Avg. “goodput” of TCP best-effort traffic vs. No of TCP flows [1].**

Due to the nature of wireless Ad-Hoc networks, mobility has an impact of the performance of such a network. Measurement of delay and throughput versus mobility is also carried out in [1] and [3]. It is clear that special treatment, to oppose the reduced performance due to this mobility have to be made.

However, it is fundamental that a network is able to satisfy the QoS requirements in a static approach, before mobility is taken into consideration. Therefore my simulations are concentrated on scenarios where no mobility occurs.

## *5.2 Classification and scheduling*

In SWAN only two packet classification schemes are proposed; Real-Time and Best-Effort. The classifier differentiate the packets and the BE packets go through a shaping mechanism, while RT packets bypass the shaper. Then the RT packets and the shaped BE packets are joined in a FIFO queue towards the MAC layer. SWAN treats all the UDP traffic equally. This may not be sufficient when trying to satisfy the many requirements and different behaviors of all the applications that exist (see chapter 4). First, the bandwidth and the delay requirements may be different for traffic belonging to different classes. Second, some flows may be more important than others. A more advanced classification scheme may be required. I chapter 4, the behavior for the most common applications was described. Based on the many applications different requirements, more classes should be added in order to differentiate data traffic to satisfy these applications service requirements. This scheme will give us a more complete QoS solution for a wireless Ad Hoc network. The classification scheme used in DiffServ (described in chapter 3.3.2) could be used as model for this improvement.

## *5.3 Compression and rate regulation*

When a network using SWAN gets congested, the ECN mechanism in the congested nodes starts marking the packets with Congestion Experienced. Then all the affected sessions has to be re-probed. The re-probing is most likely to cause unacceptable delay to a Real-Time session, and the playback buffer at the receiver will run out of packets. Then the sessions are interrupted and possibly teardown. This may have devastating effects on a Real-Time session, and could make the communication completely useless. Another drawback of re-probing is that the probe packets are competing with ordinary data packets for useful bandwidth. The probe packets are likely to passing through the congested routers and possibly making the situation even worse.

Here is a proposal of different scheme. Instead of just marking the packets of some flows with CE when congestion is experienced, packets belonging to all the flows passing through a router which experience *Heavily Load*, but not as much as experienced during excessive congestion, could be marked HL. Upon receiving the regulate messages, all or the least important of the transmitting nodes could either reduce the transmitting rate using compression, or in some way lowering the quality of the session. In this approach, all the affected transmitting nodes contribute to congestion avoidance, and any sessions may not be terminated at all. When the data traffic gets excessive, and some routers really experience congestion, the packets are marked CE as before. The HL marking serves for congestion avoidance, while CE marking is for congestion recovery. Due to the fact that all nodes is notified when the wireless channel is about to be exhausted, the false admission problem could easily be solved. All nodes know when their RT session was started, and if the time of their session is shorter than a fixed

threshold time, their session is falsely admitted, and must be terminated immediately.

When the receiving nodes detects HL or CE in the incoming packets, they send a regulate message to the transmitter. These regulate messages reverse paths will most likely follow the same path as through the nodes which experience overload and/or congestion. For that reason it is urgent that these regulate messages are given a high priority so that proper action can be taken as fast as possible in order to decrease the affected session's end-to end delay.

## *5.4 Admission Controller*

Another topic is the necessity of an admission controller. A well known solution to keep the QoS at a satisfying level in the Internet is over-provisioning the capacity of the routers and the links. It is believed that within 2005, the IEEE 802.11n standard will be able to deliver a theoretical throughput of 308 Mbps [9]. This will be a major improvement to the wireless world. If this bandwidth enhancement almost always will give us enough resources to initiate a new Real-Time session in typical Ad Hoc network, admission control might be superfluous upon starting a new session. The source node does not need to send a probing request towards the destination node to start a new session. This will again eliminate most of the start up delay, which is one of the drawbacks in the recent SWAN implementation. This is experienced in the simulation presented in chapter 7.2.2.2. Even though congestion occurs less frequently in an over-provisioned network, congestion may still occur. Therefore, the admission controller is still needed when sessions have to re-probe in times of congestion. If a partly removal of the admission controller in a QoS model for MANET is feasible, and the scheduling mechanism and congestion management is improved, the model would also be simpler and more suitable for a wireless environment.

## *5.5 Regulation algorithm*

Neither the Source-Based nor the Network-Based regulation algorithm proposed in [4], are optimal solutions to regulate network traffic due to congestion. Both proposals have certain disadvantages, and there are several trade-offs in both of them.

In the Source-Based proposal, upon receiving a regulate message, the source node does not immediately initiate reestablishment. Instead, it waits a random time before it initiates the reestablishment procedure to avoid simultaneously re-probing from all the affected sources. This may not be fair for the sources that wait the shortest period of time before re-probing, because these nodes are more likely to still find the path to heavily loaded, and therefore has to drop their sessions. Also, because the source node waits a random time before initiates their reestablishments, immediate countermoves against the congestion are not necessarily taken and the situation could be even worse. Another disadvantage is that this approach does not differentiate the different RT flows, and it may also force too many flows to be regulated.

In the Network-Based proposal, there is no differentiating of flows based on their importance, rather it randomly select a set of Real-Time sessions that has to reestablish their session. In this approach, important flows may have to drop their sessions in advantage of less important flows. Another disadvantage of this scheme is to determine if a flow is new or old in order to take correct action upon false admission, some intelligence at the intermediate nodes are required.

It is clear that the regulation of network traffic to dissolve congestion has to be quick. Another important fact is that the workload on the intermediate nodes should be kept low. In simulations presented in [swan] for the Network-Based and the Source-Based approach respectively, it took approximately 4 and 6 seconds for the Real-Time rate to reach below the Admission control rate. The Network-Based approach performed better than the Source-Based approach, at the cost of more workload and complexity in the intermediate nodes. When we know that some applications are delay sensitive in order of magnitude on hundreds of milliseconds, it may be to slow to dissolve the congestion in seconds. Before the regulate message have reached the source node, and proper action have been taken to regulate the transmission rate, the congestion may already have caused disruption to many of the ongoing RT flows.

Among my proposals, one immediate action in order to dissolve the congestion faster, could be for the congested node to broadcast one single CE/HL message to all it adjacent nodes. Then all the nodes, either directly disturbing or feeding the congested nodes with packets, could for a short period of time immediately back off all their transmitting traffic, or lowering the data rate for low priority traffic, such as BE and other non-delay-sensitive packets by either dropping those packets, or delay them in a buffer. This period of time should be long enough for the RT sources to adjust their data rate so the total RT rate again is kept below the Admission Control rate, without the necessity of terminating any RT sessions.

Also, the regulation of RT traffic could have two separate regulate messages. One regulate message for the lowest priority of RT traffic, and another regulate message to inform all the sources that the network is Heavily Loaded or Congestion Experienced. The destination nodes could select their set of “victim” flows among the least important flows, to reestablish their session based on their priority marks which could be located in the TOS field in the IP header.

Finally, since SWAN already trusts the source nodes, they could keep some state information about their newly admitted flows versus on-going flows in order to distinguish the flows that have been falsely admitted. Hence the complexity seen in the Network-Based regulation algorithm is moved from the network interior to the transmitting “edge” nodes.

In this proposal, faster and more thorough actions are taken to counter the upcoming or already on-going congestion, and the interior of the network is

spared from complex work, because most of the intelligence is kept in the end nodes.

## 5.6 Repeatability of the SWAN simulations

In order to test the performance of the SWAN network model, and figure out how the result claimed in [3] was received, a recreation of some of their simulations in ns-2 had to be made. First of all, TCL scripts of some simple simulations are enclosed in the SWAN source code. These scripts are just some simple scripts to describe how to set up the SWAN protocol in ns-2, and not the TCL scripts used in the simulations described in [3]. Also, the documentation of the simulations including all is parameters are only described by its major details in the SWAN articles, and a lot of important details are missing. Therefore, a lot of time was spent on figuring out how to put up the TCL scripts to recreate similar simulations as those described in [3].

## 5.7 Bugs in SWAN

In the implementation of the SWAN network model for ns-2, several bugs have been found and reported. The following bugs have been found by people working at Thales Communication U.K. Wrong variable have been set for the shaper queue size, so the default value is used instead. Second, SWAN uses the `data_packet` function to use if each packet is a valid `DATA_PACKET`. The variable `PT_UDP` is not defined for this function, so if you are going to use non CBR traffic over UDP it would not work unless you define it for this function. Third, the local channel usage, computed by the SWAN implementation, is not computed correctly. Code has to be added in the 802.11 MAC files of the ns-2 implementation. In addition to these errors, the modified `packet.h` file following the SWAN implementation code for ns-2 contained a lot of bugs that had to be fixed in order to re-compile the ns-2 source with the SWAN extension.

## 5.8 Summary

In chapter 5, problems with the SWAN network model proposal was discussed. In section 5.1 Questions whether the delay experienced by the Real-Time applications is low enough to properly support this type of traffic. Also, a question was raised whether the statements about the delay in the SWAN articles really are what they claim to be. This was suggested to be investigated in deeper details through further simulations. Section 5.2 claimed that one of the major drawbacks of the recent SWAN model have a poorly equipped classification scheme. More classes could, and must be added, in order to fully support a more complete QoS solution for Ad Hoc networks. Section 5.3 demands a more advanced notifications scheme than the existent one. In order to support both congestion avoidance and congestion recovery, it could be necessary with more than one notification message, without a heavy increase of the complexity of the network interior. Section 5.4 discussed the need for an admission controller, and that it could be superfluous with the increased capacity of the wireless bandwidth. Section 5.5 highlighted the drawback of the proposed regulation algorithms, and

suggested a regulation algorithm with more immediate action upon congestion, and a better solution concerning the differentiation of sessions to be re-established. Also, this section included a proposal to move the complexity of deciding whether a session is falsely admitted or not out to the “edge” routers. Sections 5.6 described the difficulties on repeating the simulations presented in the SWAN articles. In the final section of chapter 5, some bugs discovered during the work with the SWAN implementation was presented.

# 6 Related work

This chapter presents two additional and relevant protocols which also focus on the QoS in mobile ad hoc networks.

## *6.1 INSIGNIA*

The QoS protocol INSIGNIA is a proposal for an in-band signaling system that supports restoration and adaptive reservation service, in the continuously changing conditions of mobile ad hoc networks. The protocol supports "operational transparency" between several mobile ad hoc networking protocols, such as AODV, DSR and TORA.

INSIGNIA encapsulates control signals in the IP option of every IP data packet (INSIGNIA option), and can be characterized as an in-band RSVP signaling protocol. The flow state information, informing the source nodes for the status of their flows is maintained on an end-to-end basis. Whenever the INSIGNIA option is used in an IP packet, the INSIGNIA module is involved. If the resource requirement can be satisfied, it allocates bandwidth to the flow in coordination with the Admission Control module. If the requested resource can't be satisfied, the packet is left with only best effort service.

The two major drawback of INSIGNIA is that it, as the SWAN module, only enables the existence of two service classes (RT and BE), and that the flow state information is kept in the mobile host, which could lead to a scalability problem as the number of flow increases [47][49].

## *6.2 DS-SWAN*

Another protocol proposed to ensure QoS in wireless Ad Hoc network is the DS-SWAN protocol. The DS-SWAN protocol is based on the co-operation between the SWAN network model within the ad hoc network, and the Differentiated Services (DiffServ) in the fixed infrastructure network. The DS-SWAN protocol supports end-to-end QoS in ad hoc network connected to fixed DiffServ domains.

In the DiffServ domain the EF service class is used for Real-Time traffic. At the ingress edge router, the traffic are policed by a token bucket and traffic that exceeds the profile is dropped.

In DS-SWAN the nodes in the ad hoc networks are notified when the network congestion are about to be too excessive to support properly functioning of Real-Time applications. The affected nodes then react by adjusting the transmission rate for their Best-Effort traffic. If the end-to-end delay for a Real-Time session becomes greater than 140 milliseconds, the destination node sends a `QoS-LOST` warning message to the ingress edge router. The parameter values in the AIMD rate control algorithm is then modified by the nodes in the ad hoc networks receiving this `QoS-LOST` message [48].

# 7 Simulations

The general idea is to recreate some of the scenarios in the SWAN articles, and the results stated in those articles, and then investigate that those test and the implementation of SWAN, seems to be what it really claims to be. Additional performance tests will also be carried out in order to make a thoroughly investigation of the SWAN mechanism. Simulations will be executed in a single-shared-cannel medium and in a multi-hop scenario. An ideal approach would have been to run the test on real computers. When considering the amount of resources that will require, a network simulator is more suitable for this purpose. Therefore, it is important that the model on which the simulator is based matches as closely as possible to the reality, in order for the result of the simulations to be meaningful [18].

An implementation of SWAN for ns-2 has been made by the originators of SWAN, so ns-2 is the network simulator chosen throughout this work.

## *7.1 Ns-2 overview*

### 7.1.1 Introduction to ns-2

The network simulator (ns-2) is a discrete network simulator targeted at network researching. Ns-2 originated in 1989 as a variant of the REAL network simulator. As a part of the Virtual InterNetwork Testbed (VINT) project at the University of California in Berkley. The project was supported by DARPA in 1995.

Ns-2 is an object-oriented simulator with substantial support available for simulation of TCP, routing, and multicast protocols, initially intended for wired networks, but the Monarch Group at CMU have extended ns-2 to support wireless networks.

The core of the simulator, including the network protocols is implemented in C++, while object oriented TCL is used as an interface to describe, and set up the simulations. The implementation of ns-2 closely follows the OSI model. The essential of the wireless model consist the *MobileNode* at the core, with additional support for simulations of multi-hop ad-hoc network etc. A mobile node is derived from the basic Node *object*, with additional functionalities of a mobile wireless node, like the ability to receive and transmit signals to and from a wireless channel, and the ability to move within a given topology. In ns-2, an *agent* is used as a representation of an endpoint where network traffic are constructed, processed and terminated.

Ns-2 is able to trace the network traffic through the different protocols and produce output trace files, which can be used to calculation of the QoS parameters. Cmu-trace objects are used to support trace in wireless simulations. In the Network Animator (NAM) output from the trace files can be used to view the simulation, see Figure 7.1. Output from the trace files can also be plotted as graphs in graphical applications such as Xgraph or GnuPlot.

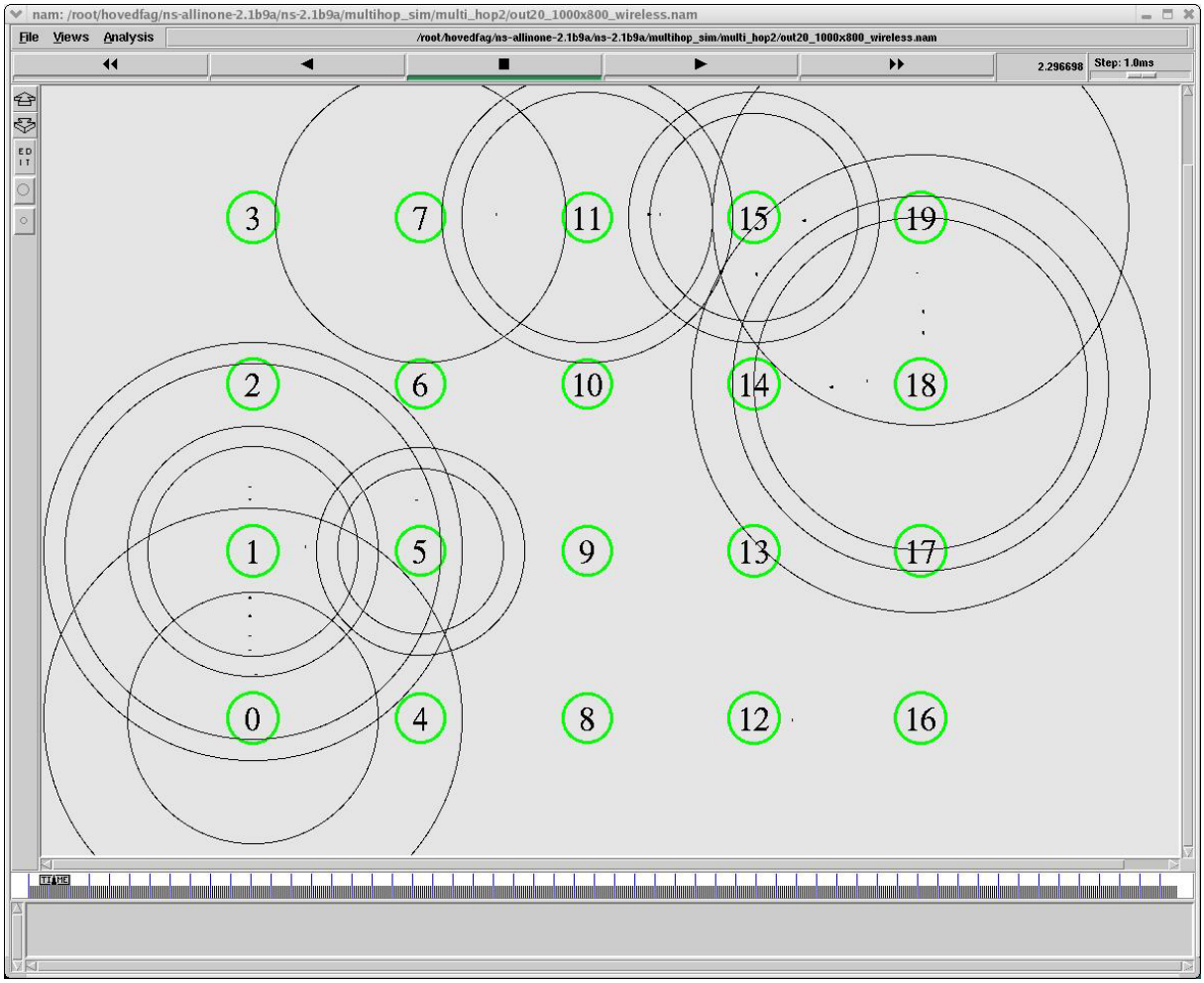



**Figure 7.1: Screenshot from NAM**

Due to its open source nature, ns-2 has certain limitations and disadvantages. Information and documentation are often limited and not *state of the art* in proportion to the current release of the simulator. Compatibility and consistency of code across releases may also cause problems. The last drawback of ns-2 is that it is poorly equipped with tools to prepare the simulation scenarios and to analyze the results [18] [19].

### 7.1.2 Credibility on simulations in ns-2

*"An opinion is spreading that one cannot rely on the majority of the published results on performance evaluation studies of telecommunication networks based on stochastic simulation, since they lack credibility. Indeed, the spread of this phenomenon is so wide that one can speak about a deep crisis of credibility."* [20].

When simulating network performance in a network simulator, it is important to consider the credibility of the simulator chosen for the modelling, and also in credibility of the results achieved in this simulator. In order to have a valid simulator model, it is of great importance to have appropriate assumptions about the network mechanisms and its limitations. The next step is to ensure that the simulation and analysis of the experiment is valid.

As mention previous in this paper, the simulator chosen for modelling this experiment is ns-2. When considering the size and complexity of ns-2, a complete validation of ns-2 is out of the scope of this thesis. Anyway, it is important to be aware of the weaknesses and limitations of the tools used in any experiment, and this experiment is no exception. In [18] several network simulators suited for MANET is tested. In [18] ns-2 is among the simulators in which the results show that there exists significant divergence between the simulators when equal models are executed on the simulators. Naturally, I have not found any complete validation on ns-2 ability to match the reality. Even though the match to the reality in fact could be close, a certain range of error in the simulation results is highly likely to occur. Anyway, this thesis has to rely on the correctness in the implementation of the ns-2, and then carefully considerations of the abbreviations that may exist upon making conclusions of the simulation results must be done.

### 7.1.3 The implementation of SWAN in ns-2

The SWAN module for ns-2 requires the ns-2 simulator version 2.1b9a. To install the SWAN module for ns-2 you need to download the SWAN module from http://www.comet.columbia.edu/swan/simulations.html, modify the makefile and compile the simulator with a C++ compiler. The additional SWAN files includes: *run*, *swan.tcl*, *mobility.tcl*, *traffic.tcl*, *swan_ac.cc*, *swan_ac.h*, *swan_rc.cc* and *swan_rc.h*. Modifications to some of the original ns-2 source code have also been made by SWAN. Those files are originally included in the downloaded version of ns-2 and have to be replaced by the modified one from SWAN. Those files are: *ip.h*, *packet.h*, *ll.cc*, *ll.h*, *ns-default.tcl*, *ns-mobilenode.tcl*, *ns-packet.tcl* and *cmu-trace.cc*. The SWAN extension for ns-2 includes an admission controller, a rate controller (AIMD), a mechanism for packet delay measurement, local utilization monitoring, a probe protocol and the ECN mechanism.

### 7.1.4 My work with ns-2

When I started my work on this paper I was completely new to ns-2, C++ and TCL. I started with the SWAN and ns-2 webpage, the SWAN articles and Marc Greis' Tutorial. I downloaded the 2.1b9a all-in-one package and installed it on my laptop computer running RedHat Linux. The installation was not without problems, and I had to fix a lot of bugs before I had a valid compilation of the simulator. When I finally had ns-2 up and running I had to learn how to write and run simulation scripts. I started with the small and simple scripts in Marc Greis' Tutorial. From there I went to more advanced simulation scripts and networks. Then I went into exploring the wireless part of ns-2. I looked at the descriptions of the simulation carried out in [3] and I had a hard time figuring out how their simulations were set up. Gradually, I approached similar simulations as the simulations in [3]. In order to have a reasonable output of the simulations, Agent/LossMonitor where used as traffic sinks since ns-2 use them to store the amount of bytes received. I wanted to use this information to calculate the bandwidth and delay of the simulations, but I could not get any output from TCP traffic with the LossMonitor. I later found out from the ns-2

mailing list that tracing of TCP traffic with the LossMonitor in ns-2 was not supported for a wireless simulation, so then I was stuck for a little while. Fortunately, I found out that I did have all the information I really needed in the trace files. Then I started making small scripts in Python with the trace files as input. Then I used the scripts to calculate the throughput and delay from my simulations. Finally, I had enough knowledge of ns-2 to add the SWAN module (as described in the previous chapter), and set up the scenarios I want to run, with and without the SWAN module. To fully understand the implementation of SWAN for ns-2, I also had to take a deeper look into the C++ files of the SWAN module and ns-2. The complete implementation of ns-2 including all the network modules is pretty large and complex, so I only had a chance to get a glance at how ns-2 is built up. The all-in-one package actually requires about 250 Mb of disk space. Since this installation and learning period, I have spent my time on setting up and running the different scenarios I want to simulate. During my simulations and when analyzing some of the results, I have also found many new scenarios I wanted to run and find out more about.

## *7.2 Preliminary simulations*

### 7.2.1 Maximum throughput tests

Due to the generally hostile transmission channel in wireless networks, the achieved throughput is usually far from the theoretical throughput dedicated to the wireless interfaces. The introduction of an additional protocol such as the SWAN module in an Ad Hoc network will also ad an extra burden to the network in terms of extra administrative and control information exchange. Before starting on more advanced simulations like the one in [1] and [3], it was desirable to see how SWAN performed in simple scenarios. Therefore I found it useful to first figure out what is the maximum throughput of TCP traffic in a wireless 802.11b simulation using ns-2 (version 2.1b9a).

#### 7.2.1.1 Testing maximum throughput with TCP traffic

These simulations were run both with and without the SWAN mechanism, in a single shared channel, under ideal and undisturbed conditions. No other nodes or traffic were present in the simulation area. To simulate this rather simple test, the simulation area included only 2 nodes and one single greedy FTP connection between two nodes. The bandwidth of the wireless media was set to 11 Mbps and SWAN's Threshold rate was set to 4000 Kbps when the SWAN modules where turned on. The packet size was 512 bytes and the RTS/CTS mechanism where turned on. The simulations lasted for a total period of 100 seconds. As we can see from the graph Figure 7.2, the simulation with the SWAN rate/admission controller turned off, the achieved throughput lay almost constantly around 3 Mbps from the beginning to the end of the simulation period.

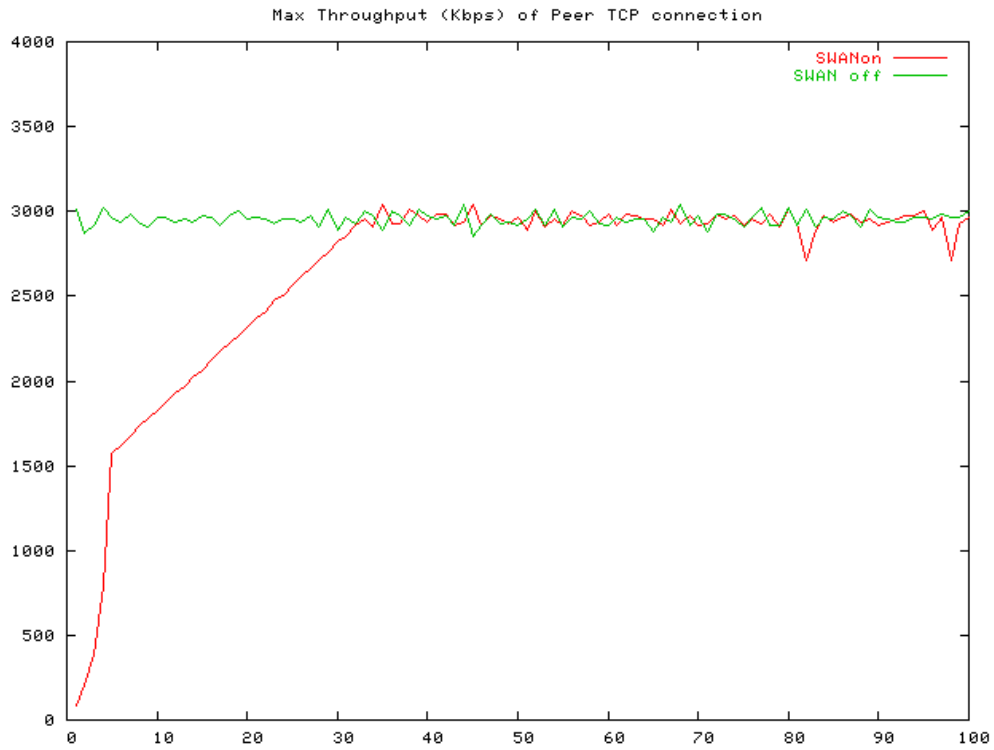



**Figure 7.2: Maximum throughput of TCP traffic with SWAN on/off vs. time**

On the other hand, a corresponding simulation with the SWAN mechanism turned on, the result shows that the achieved TCP throughput started at 0Mbps and then increased exponentially up to 1.5 Mbps for about 5 seconds. Then the AIMD protocol in SWAN started to use the Slow Start algorithm to increase the congestion window from a “cold start”. Then AIMD’s Slow Start Threshold (ssthresh) was reached, and AIMD used the “old” additive increase algorithm. For the next 25 seconds the TCP throughput grew linearly up 3 Mbps. The AIMD rate controller in the SWAN module uses more than 30 seconds to reach a stable throughput around 3 Mbps. After the first 30 seconds, the rate controller reached its “maximum” throughput. From now on, the extra administrative burden of SWAN control traffic was barely noticeable, and the achieved throughput stabilized approximately at the same throughput as achieved without the SWAN mechanism. For a single application running over TCP, it is not of great importance if a link at anytime is fully utilized. On the other side, there is no need to waste valuable bandwidth which indeed is a scarce resource in a wireless medium. For better utilizing of the link, the AIMD protocol in the SWAN module should be implemented and or adjusted so it faster adapts to the actually available bandwidth. Apparently it looks like a good idea to increase the ssthresh. Then the Slow Start algorithm will be used for a longer period of time, and the throughput will grow exponentially up to this new ssthresh. Thus, the utilization of the link will be more efficiently.

#### 7.2.1.2 Testing maximum throughput with UDP traffic

UDP traffic has different network characteristics than TCP traffic; therefore I had an analogous simulation with Real-Time CBR traffic where also accomplished. This simulation used 2 nodes and 10 CBR flows (5 in each direction); because some trouble with fast generation of packet occurred, and because the packet size over UDP is usually limited to 1500 bytes [45]. Instead of having only one flow and packet size of 10000 bytes, each flow had a packet size on 1000 byte and a sending interval on 0.01 seconds thereby making the simulation more realistic. The total CBR traffic generated in this simulation was then 8 Mbps.

The total simulation time of this simulation was the same as with TCP traffic; 100 seconds, and the RTS/CTS mechanism was turned on. Initially, the Threshold rate was set to 4000 Kbps and the Admission Control rate to 2000 Kbps in this simulation. The consequences of keeping this settings, was that the RT traffic was marked as BE traffic when it exceeded the Admission Control Rate, and then obviously treated as BE traffic by the Classifier and the Shaper. The outcome of this looked pretty much like the graph in the previous chapter, where the throughput first grows exponentially and then grows linearly up to a more stable throughput. In order to be capable of finding out how much RT traffic that could be achieved in this simulation, without having RT traffic converted to BE traffic, I had to put both the Threshold rate and the Admission Control rate high above the generated traffic of 8 Mbps. The bandwidth of the wireless media was set to 11 Mbps.

With the raised Threshold rate and Admission Control rate, the total throughput of this simulation was slightly above the previous simulation with TCP traffic. This could be due to the nature of TCP which introduces more overhead, and then produces a less effective throughput. Another reason why the achieved throughput in these simulations was higher than the previous with TCP traffic could be the packet size, which was nearly doubled. Increasing the packet size usually gives a higher throughput. The reason for this is that for each packet sent, the ratio of payload/header is higher with higher packet sizes.

Without the SWAN mechanism, an average UDP throughput about 3.6 Mbps was received in this simulation. Indeed, very small variations in the throughput were experienced, except for the very first seconds of the simulation. With the SWAN mechanism turned on, the throughput was slightly lower and the received average UDP throughput was about 3.4 Mbps (see Figure 7.3). Upon initiating a RT session, the probe packet has to be sent from the source to the destination. The session is not admitted to start before the source receives the probe-reply from the destination, thus introducing a start up delay, and naturally the network will not be fully utilized before all flows are admitted. It took almost 10 seconds before the simulation with the SWAN mechanism turned on, reached a maximum throughput at about 3.6 Mbps.

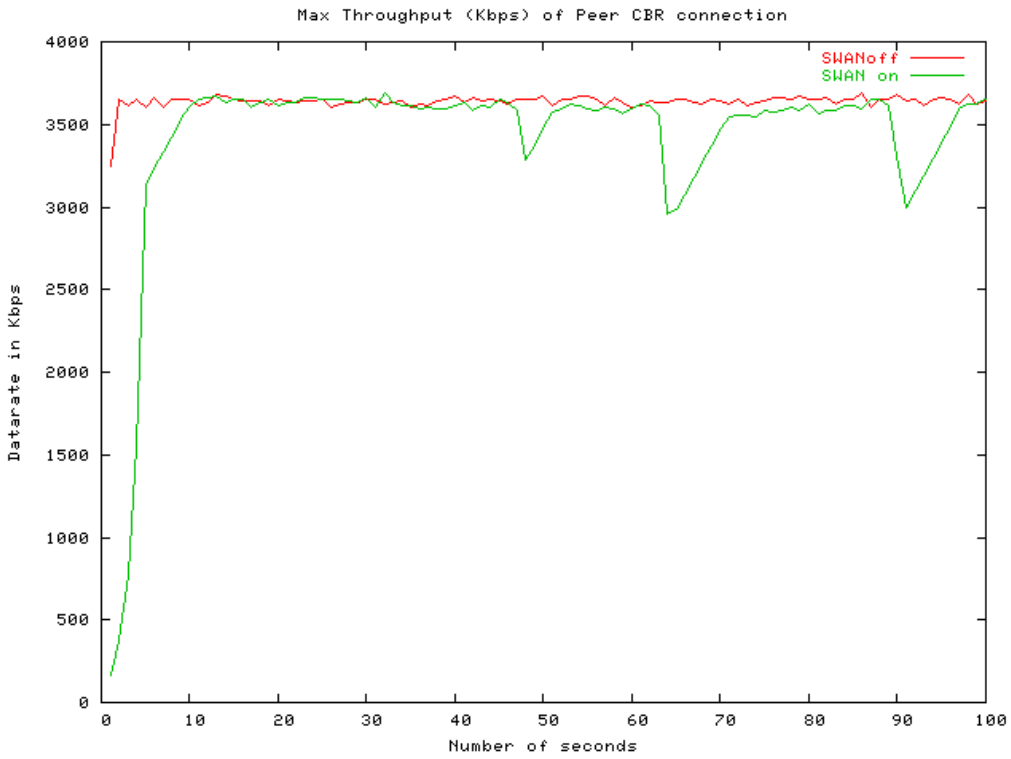



**Figure 7.3: Maximum throughput of CBR traffic with SWAN on/off vs. time**

### 7.2.2 Performance test of additional traffic

After running several simulations, it was discovered that the delay in some of the situations was extraordinary high. Due to this strange behavior, and after several trials it was reason to believe that the delay in some situations was dependent of whether the source or destination nodes where receiving or sending additional traffic. Therefore, a new simulation similar to the simulation above, were set up. This simulation contained two nodes, one source and one receiver and one single RT flow, once with small packet (voice), and once with larger packet (video). Additionally, to stress either the source or the destination node, a third node (node 2) which both aggregate and receive BE traffic from one of the two other nodes, was introduced. Respectively, in the first simulation sending additional FTP traffic to the source node of the RT traffic (node 0) and in the second simulation to the destination node of the RT traffic (node 1).

Also, it was desirable to see how the SWAN mechanism reacted in these situations. So, the simulation was executed both with and without the present of the SWAN module. As in the previous simulation, the bandwidth of the wireless media was set to 11 Mbps, the Threshold rate was set to 4000 Kbps, and the Admission Control rate was set to 2000 Kbps, when the SWAN modules where turned on. The packet size was 512 bytes. The simulations lasted for a total period of 100 seconds. Every second, the average delay was measured for the packets of the RT session, and the throughput was measured for the FTP connections.

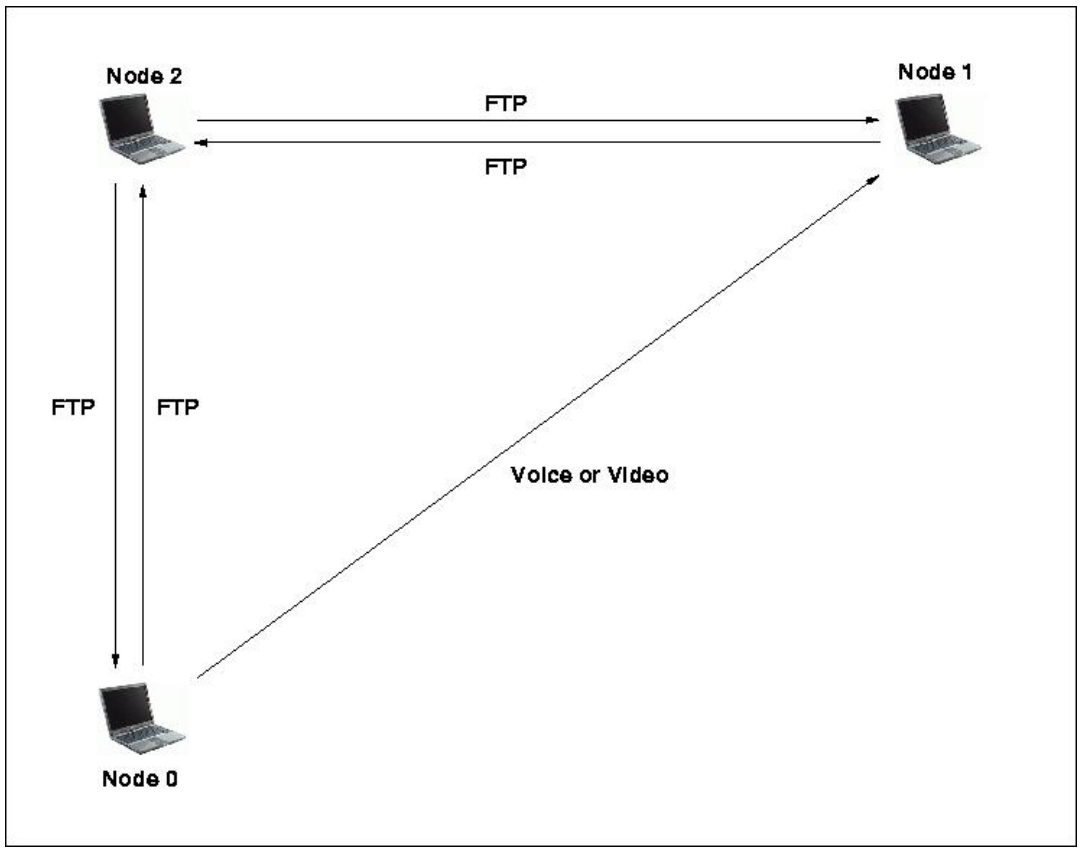



**Figure 7.4: Simulation scenario with additional traffic to node 0 or node 1**

### 7.2.2.1 Additional Traffic Exchange with RT Source

This simulation deployed voice traffic with a packet size of 80 byte, from node 0 to node 1, throughout the complete simulation period of 100 seconds. After the first 20 seconds of the simulation, node 2 starts aggregating FTP traffic to node 0. After 40 seconds node 0 starts to send FTP traffic back to node 2. From 40 seconds of the simulation we have an overlap of 20 seconds where FTP traffic is going in both directions between node 0 and node 2. After 60 seconds node 2 stops its aggregation of FTP traffic to node 0. Finally, after 80 seconds node 0 also terminate its FTP session to node 2. For the last 20 second of the simulation period, the only traffic present is the RT flow from node 0 to node 1.

From the graph in Figure 7.4, showing the delay of the voice flow from node 0 to node 1, scarcely no delay was imposed in the first 20 seconds of the simulation, where the only traffic present was the voice flow. After 20 seconds, where node 0 receives FTP traffic from node 2, the delay when using SWAN was still scarcely anything. For the same period the simulation without the SWAN mechanism showed a significant increase in the delay imposed on the RT flow. The delay rises from barely 0 up to about 80

milliseconds where the delay stabilized for the next 20 seconds until the next event occurd.

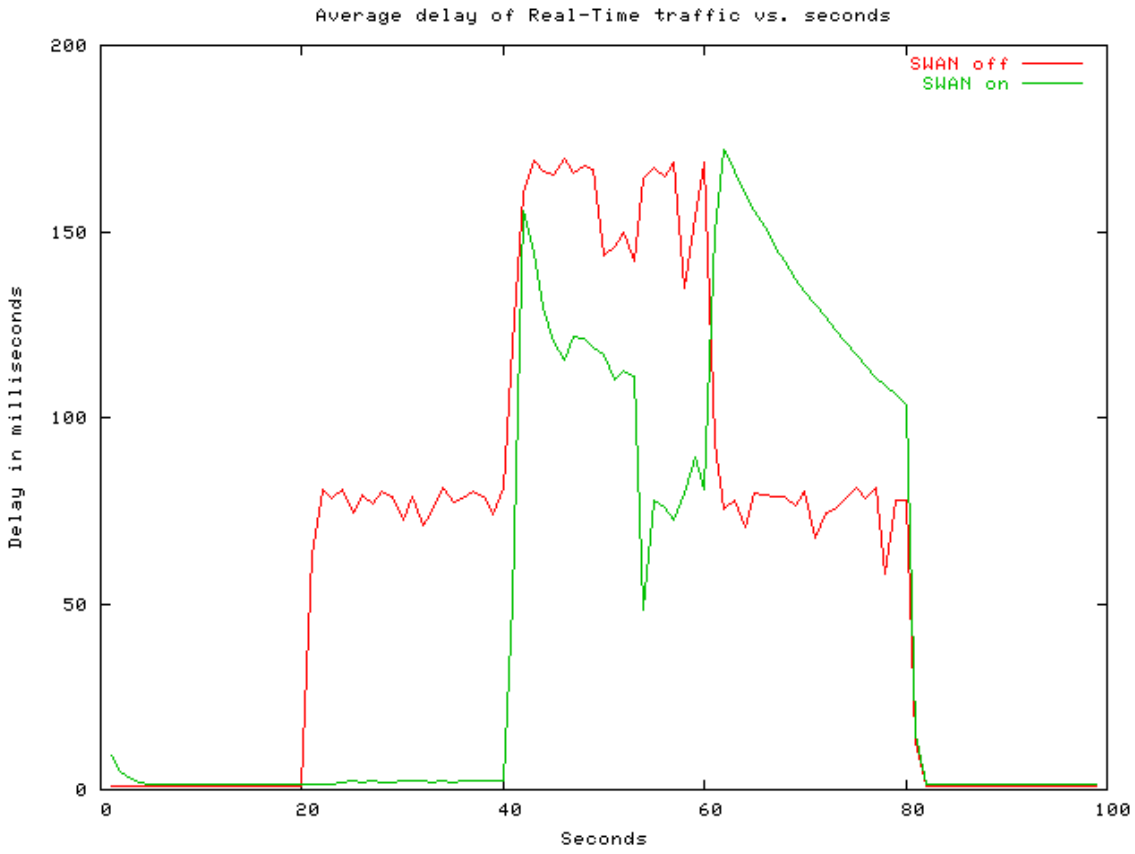



**Figure 7.5: Delay of 32 Kbps Audio stream**

At time 40, where node 0 started to send FTP traffic to node 2 in addition to the traffic it received from that same node, there was a considerable increase in the delay, both with and without the SWAN module. With the SWAN mechanism turned on, the delay raise almost instantly from nearly 0 to about 160 milliseconds, and then sharply declined for the next five seconds down to 120-130 milliseconds where it stayed for about 10 seconds. Then the delay suddenly dropped again, and until the next event occurred, the delay stayed at about 70-90 milliseconds. In the same interval (40-60 seconds) the simulations without the SWAN module showed a delay which was more than doubled compared to the last interval where only node 0 received FTP traffic. Indeed, the delay rises suddenly from about 80 milliseconds to between 160 and 170 milliseconds. Apart from some drops the delay stayed mostly around this level until the next event occurred.

After 60 seconds node 2 stopped generating FTP traffic to node 0. At that point, the graph in Figure 7.5 show us significantly different behavior of the delay experienced with and without the SWAN mechanism. The plot from the simulation with SWAN shows us a very strange behavior. The delay suddenly jumped from about 90 milliseconds, up to about 170 milliseconds, and then

the delay immediately decreased, approximately linearly down to 110 milliseconds, where all FTP traffic is stopped. In this same interval (60-80 seconds), the delay for the simulation without the SWAN mechanism was immediately reduced to less than half its value, when node 2 stopped sending FTP traffic to node 0. Indeed, for this interval, where the only FTP flow present, was the FTP flow from node 0 to node 2, the delay stayed at the same level as when the only FTP traffic present was the one in the opposite direction from node 2 to node 0 (interval 20 – 40 seconds).

All FTP traffic terminated after 80 seconds, and for the last 20 seconds where the RT flow from node 0 to node 1 was the only traffic present, the delay went back to almost 0 milliseconds throughout the simulation.

In Figure 7.6, the throughput of the FTP traffic from the same simulation can be viewed. The low delay experienced with SWAN in the second interval (20-40 seconds), could be explained by the behavior of the FTP throughput. As mention previous in this paper, SWAN is using the AIMD rate controller to regulate Best-Effort traffic. When node 2 starts to send FTP traffic after 20 seconds, the rate controller will only allow this flow to utilize the remaining bandwidth not used by the RT flow. Thus it was not making any disturbance on the RT session.

The reason why the delay was raised so heavily in the next interval (40-60 seconds), is probably because the node 0 was starting to send FTP traffic to node 2, and node 0 was not aware of the fact that node 2 is utilizing all the “free” bandwidth for best-effort traffic. All nodes get its measure from the MAC delay, and at the moment of initiating this FTP session, node 0 has no measure of the MAC delay. Therefore it probably misconceives the situation and thinks it can utilize the “remaining” bandwidth. Node 0 starts to send a burst of FTP packets, before it gets measurement of the excessive delay. When the delay gets excessive, node 0 adjust its transmission rate quickly. This was probably the reason why the delay jumps so high and then immediately backed off to a much lower level. The reason why the delay again reach a high peak in the forth interval (60-80 seconds) I have no good explanation for.

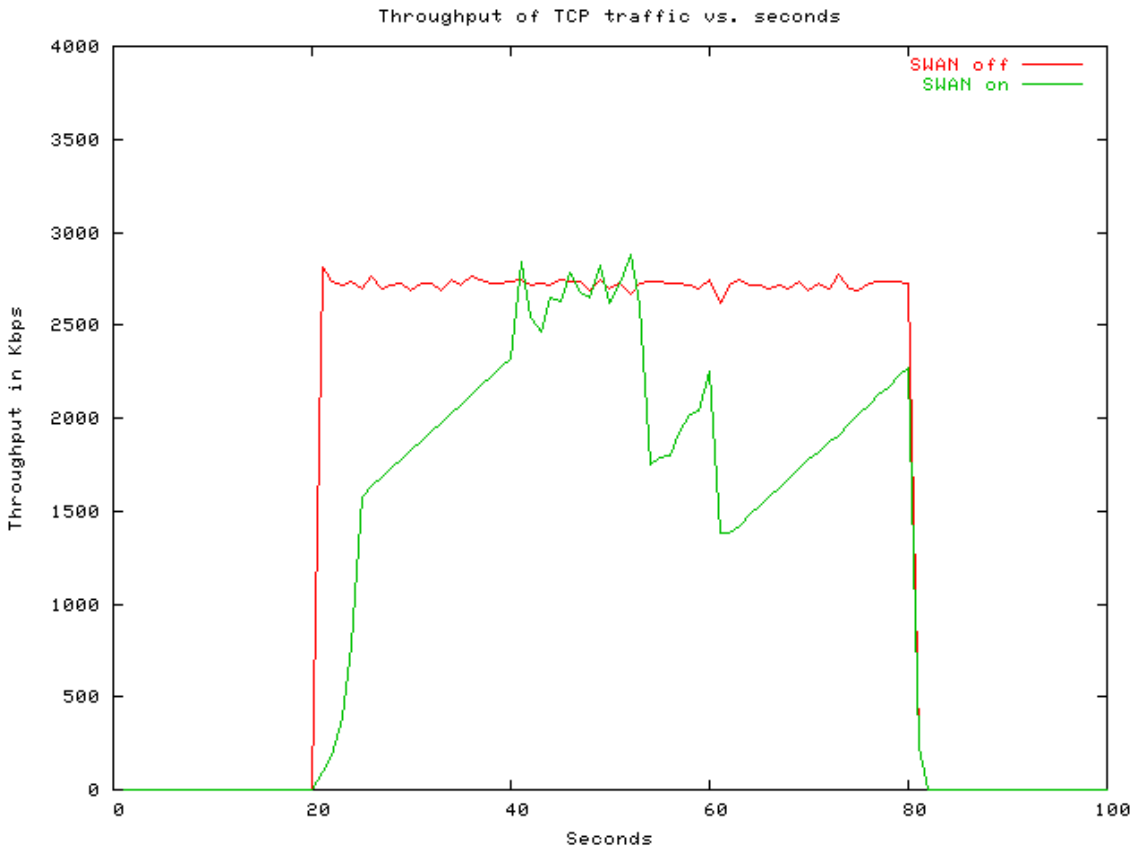



**Figure 7.6: Throughput of the greedy FTP traffic**

A solution to solve the problem of a node initiates a session with best-effort traffic without a MAC delay measurement, could be to in some way to retrieve this information from its most adjacent neighbor nodes. How this could be implemented I leave for future development of the SWAN mechanism.

Without the SWAN mechanism, there is no organized mechanism trying to prior, or in any way put any limitation on the traffic, except from the upper physical bandwidth limitation. As we can see from Figure 7.5, as long as there is at least one greedy FTP flow present, the TCP throughput was close to the physical limit of 3 Mbps measured in previous simulations. Here all the packets are equally competing for the bandwidth and as we can se from Figure 7.4 and 7.5, the delay was approximately doubled for the RT session. As reasonable to expect, this happens when the number of FTP flows was increased from 1 to 2 sessions, and at the same time no TCP throughput was increased.

Finally, some statistics in numbers: As expected the delay upon initiating a Real-Time session using SWAN was higher than without SWAN; the first RT packet was delayed 24.47 milliseconds, compared to 5.61 milliseconds without SWAN. The max delay of one single RT packet was 193.73 milliseconds with SWAN, and 181.55 milliseconds without SWAN. Also as expected, the average delay using SWAN; 48.92 milliseconds, was lower than

without SWAN; 62.46 milliseconds. Also, the total TCP throughput was lower with SWAN; 11 725 Kb, than it was without SWAN; 16 351 Kb.

#### 7.2.2.2 Additional Traffic Exchange with RT destination

Similar to the simulation above, another simulation was accomplished; instead of sending FTP traffic between the RT source (node 0), and the external one (node 2), a “2-way” connection between the external node, and the RT destination node (node 3) was set up. All network parameters where as in the previous simulations; the only difference was the swapping of the source and destinations nodes of the FTP traffic. So, here the RT connection from node 0 to node 1 also went throughout the simulation period. After 20 seconds, node 2 started to aggregate FTP traffic towards the RT destination node (node 1) and after another 20 seconds node 1 started aggregating FTP traffic back to node 2. For the next 20 seconds there was a “2-way” FTP connection between node 1 and node 2, until the FTP traffic from node 2 cease, after 60 seconds of the simulation time. After 80 seconds, the other FTP connection also terminates, and finally the simulation ends after 100 seconds. This simulation was also run separate for both voice and video.

Even though it still exist a start-up delay using the SWAN mechanism, in this simulation the average delay started at less than 10 milliseconds, with a fast decrease to only a few milliseconds. After 20 seconds, when node 2 started its FTP session to node 1, the average RT delay was increased by only a small fraction, and the delay was still down to only a few milliseconds. When the second FTP session was initiated by node 1, the delay seems to be doubled, but still the delay was very low. In this interval, it oscillated around 5 milliseconds (40-60 seconds). When node 2 terminated its FTP session after 60 seconds, the delay suddenly made a jump to about 13 milliseconds, and then a quick decrease down to only 2-3 milliseconds, where it stayed until node 1 also terminated its FTP session, and the delay decreased further to only a couple of milliseconds for the rest of the simulation period.

Without the SWAN mechanism, the simulation results shows a flat line, at only 1-2 milliseconds for the RT delay in the first 40 seconds. After 40 seconds, the second FTP session was initiated. When the first FTP session was terminated after 60 seconds it does not seems to affect the delay at all. In fact, the delay oscillated around 6 milliseconds, with some high peaks until the second FTP session also terminated after 80 seconds. For the rest of the simulation period, the only traffic present is the RT flow, and then the RT delay went back to only 1-2 milliseconds.

In Figure 7.7, the FTP throughput for this simulation can be viewed. Without the SWAN mechanism, the FTP session from node 2 to node 1 was not admitted to transmit at all. This seems very strange, and at first glance this could be due to a one-time error in this simulation, but the same thing also occurred during simulations of video. So, this clearly explains why the delay at the same period stayed at that low level. In the same period of the simulation with the SWAN mechanism, everything seems to work fine, and the FTP throughput was increased using the AIMD mechanism. After 40 seconds, the second FTP session started to generate traffic from node 1 to node 2. Then

the FTP throughput was fully utilized, and stayed just below 3Mbps, until this session ceased after 80 seconds. The behavior of the FTP throughput in the same period in the simulation using SWAN shows that the FTP traffic is controlled by the AIMD controller. The throughput drops certain times when the traffic gets excessive, and then it was raised linearly, without much disturbance to the RT flow.

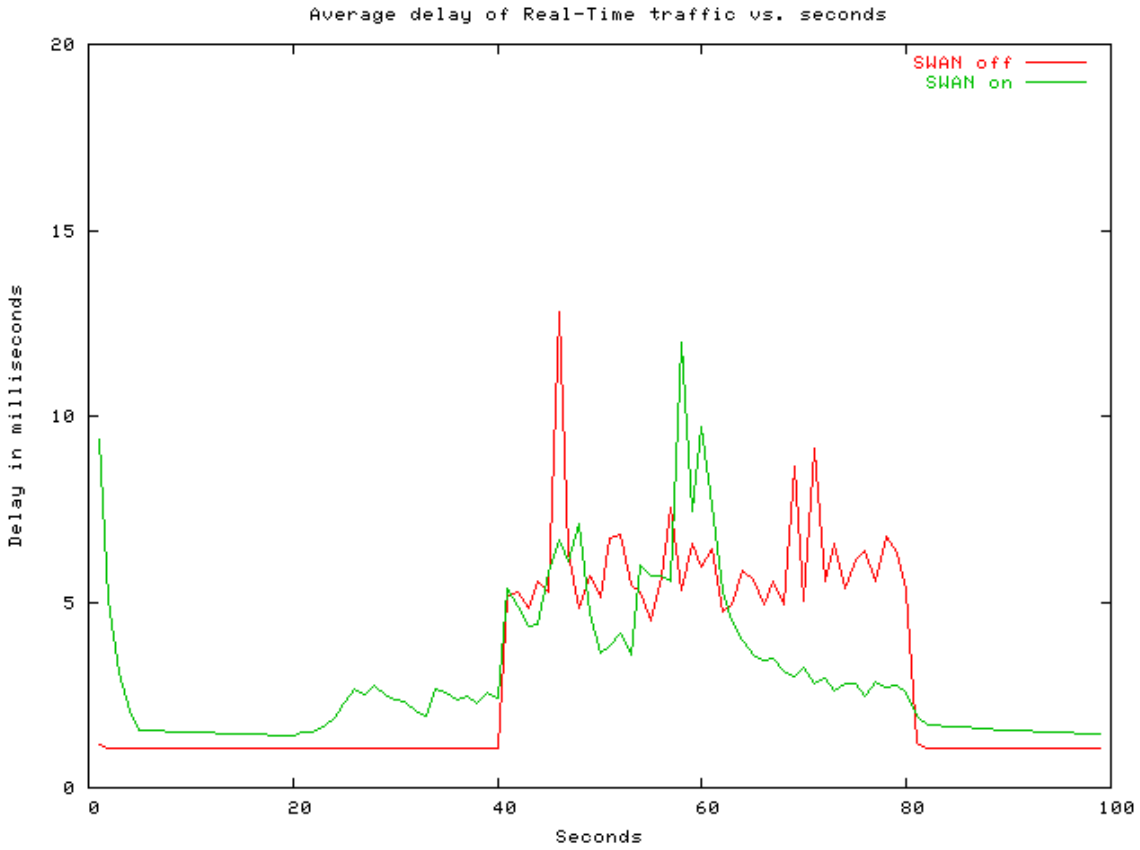



**Figure 7.7: Delay of 32 Kbps Audio stream with packet size of 80 byte**

To enumerate the statistics of this simulation, the delay for the first packet was the same as in the previous simulation, both with and without SWAN. This is exactly as anticipated, because in both simulations, only RT traffic was present in the first 20 seconds interval. Surprisingly, the average delay was down to only 3.03 milliseconds, which is very satisfying for the demands of RT applications. This was achieved both with and without the use of the SWAN mechanism. Also, the maximum delay of one single packet was much lower in these simulations. With SWAN, the maximum delay was 77.62 milliseconds and without SWAN the maximum delay was slightly lower; 62.58 milliseconds. On the other hand, the achieved FTP throughput was decreased; 11 006 Kb without SWAN, and as expected, slightly less with the usage of the SWAN mechanism; 10 374 Kb.

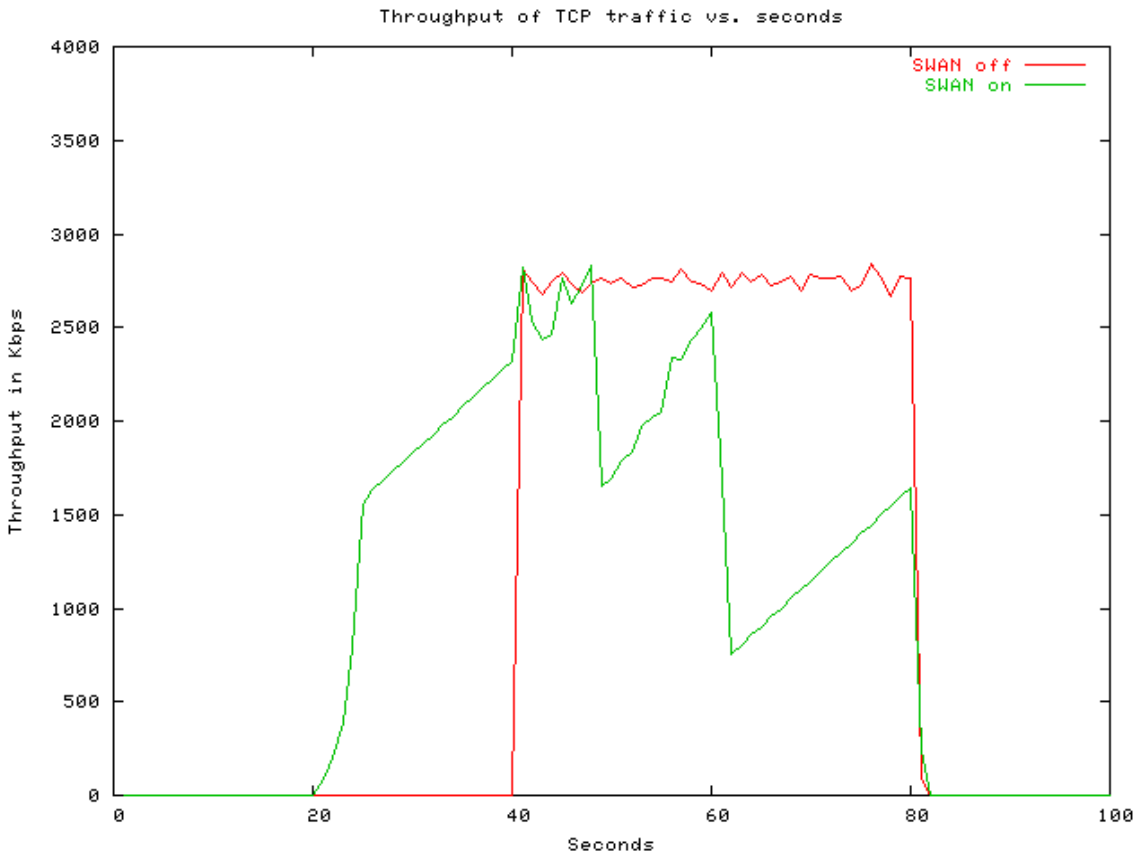



**Figure 7.8: Throughput of the FTP traffic**

### 7.2.3 Variable packet size test

As part of the preliminary tests, before running larger simulations scenarios like those posted in [3], it was desirable to find out how the performance varied with the size of the data packet. Applications like voice and video usually have different packet size. Also, in a Real-Time session the packet size can vary with different compressions, hence it is interesting to find out how the different packet size influence the performance, and in which situations different packet size triggers excessive delay. In an environment with high density of nodes, and an increasing amount of competing data flows, I wanted to show that this different type of data traffic have different behaviour. It was also desirable to show that different packet also needs to be treated separately in a QoS mechanism.

In this simulation, 40 nodes were present in the same BSA, so all the nodes shared the same 11 Mbps radio channel. Also, the Admission Control rate was set to 2000 Kbps and the Threshold rate was set to 4000 Kbps when the SWAN modules where turned on. The simulations lasted for a period of 100 seconds.

In 6 separate simulations, each with different packet size, the tests was executed in the same simulation environment. Results from three of the test are shown and discussed below. At time 0, 5, 10 and 15 a pair CBR sessions

was initiated. These CBR flows, one in each direction between a pair of nodes, was sent at a rate of 50 packets every second. In addition to the CBR traffic, one single greedy FTP session was also initiated every 10 seconds, from the 10th to 90th second of the simulation period, see table 7.1 for more details. Each FTP session has its own transmitter/receiver pair.

| Time in seconds | 5 | 10 | 15 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CBR | 2 | 2 | 2 | | | | | | | | |
| FTP | | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table 7.1: Initiation time of CBR/FTP flows**

The first simulation (see Figure 7.9) started with CBR traffic on 80 bytes. For the first 20 seconds of the simulation period, the average RT delay was kept very low both with and without the SWAN mechanism. Then if we look at the simulation with the SWAN mechanism turned off; after 20 seconds, when all the 6 CBR flows was transmitting, and the second FTP session was initiated, the delay started to grow up to such a level where Real-Time traffic no longer could be supported properly. Throughout this simulation, the delay continued to grow and when the simulation ends it had reached almost 3000 milliseconds. With the SWAN mechanism turned on the delay was kept very low throughout the simulation only with some exceptions ranging up to 100-200 milliseconds. Promisingly, the delay was kept below the upper limit of acceptable delay stated by the ITU in [16]. The SWAN mechanism was performing very well in this simulation environment, with a packet size on 80 bytes.

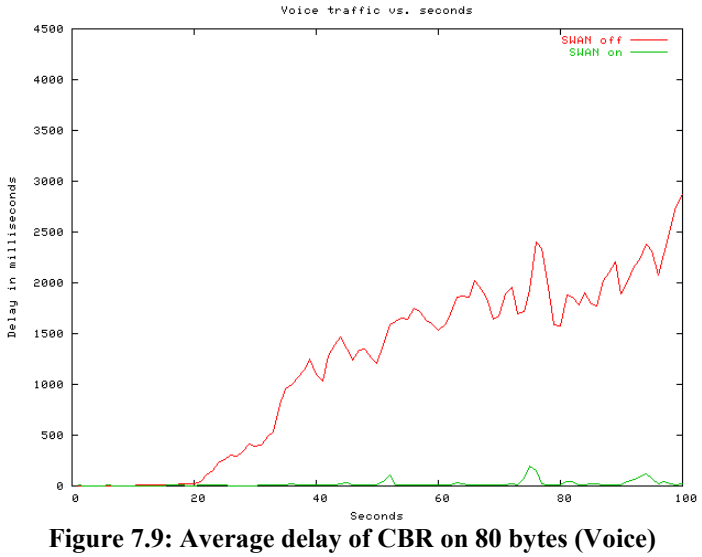



**Figure 7.9: Average delay of CBR on 80 bytes (Voice)**

In the next two simulations, with the packet size of hence 150 and 200 bytes, the simulation where carried out in the same simulation environment. In those simulations, the results were not much different from the one with a packet

size of 80 bytes, except for the last 10 seconds of those simulations. For this last period, the delay starts to increase fast using the SWAN mechanism.

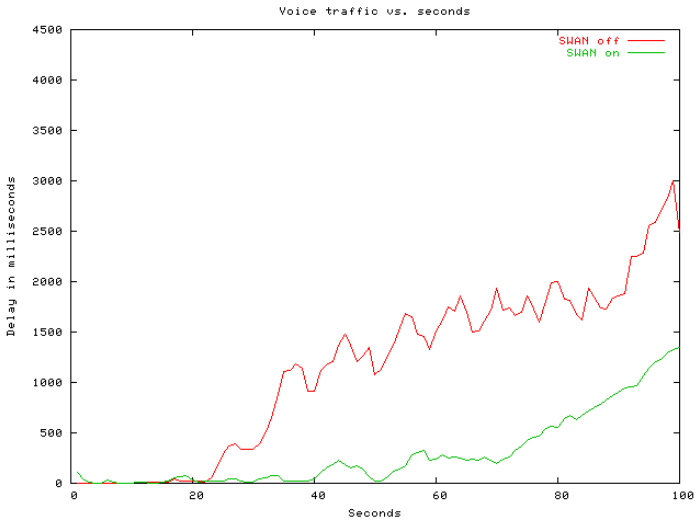



**Figure 7.10: Average delay of CBR on 250 bytes**

When the packet size was increased to 250 bytes the performance start to change radically. Without SWAN, the result was almost the same as in all the previous simulations. When running simulations with the SWAN mechanism, the delay started to grow after 40 seconds when all the 6 CBR sessions were running and the second FTP session was initiated. For the next 30 seconds the delay lied about 300 milliseconds with some minor variations, but the delay was below the limit of supporting RT traffic in this 30 seconds interval. After 70 seconds of this simulation the delay again started to increase. Actually it increases linearly from about 300 milliseconds to about 1400 milliseconds where the simulation ends at 100 seconds. For this last 30 seconds the network is poorly able to support any RT traffic at all. Another simulation where the packet size was set to 300 bytes where also accomplished. In that simulation, the delay when running with the SWAN mechanism was very excessive and closing in on the high delay explored without the SWAN mechanism.

The largest packet size used in these simulations was 512 bytes, which was the same packet size used for simulating video traffic. In the simulation without the SWAN mechanism, the measured delay was much like the delay experienced during the previous simulations; the delay was low until it starts to rise after about 20-30 seconds. Then the delay became so excessive, that hardly any properly functioning of RT traffic could be supported.

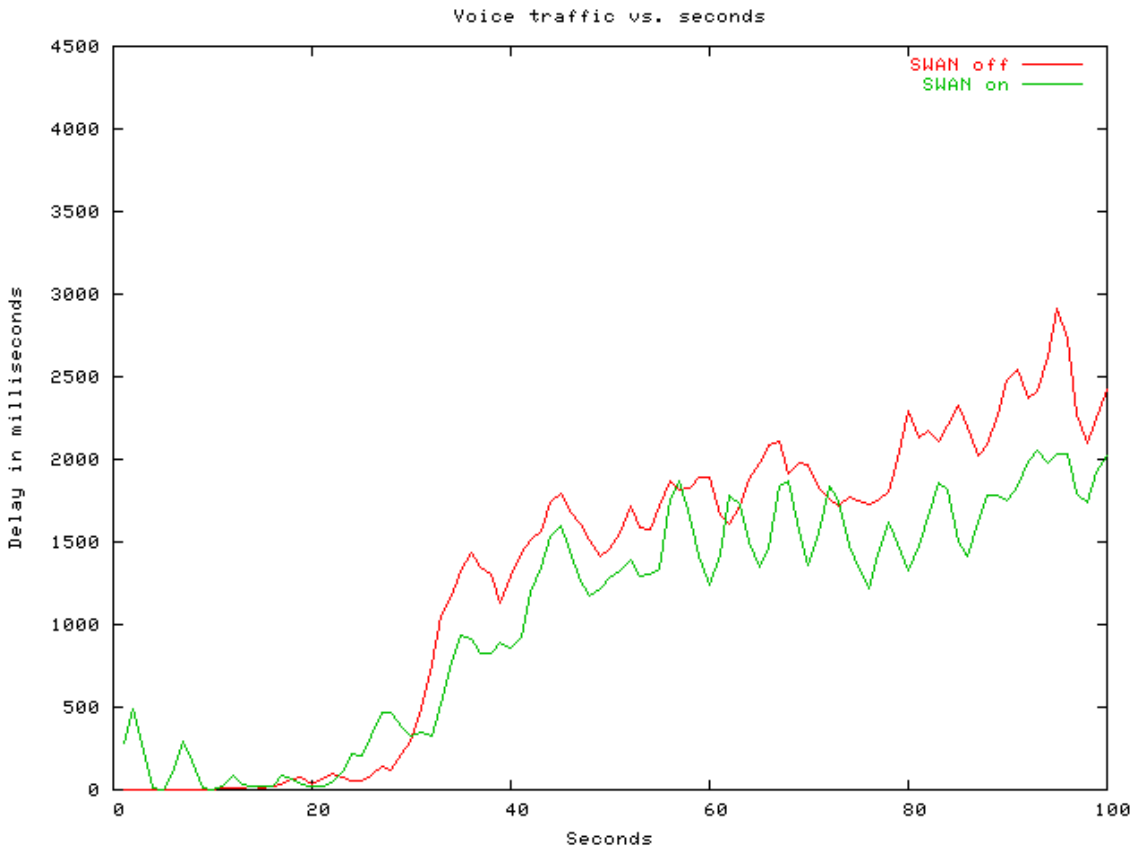



**Figure 7.11: Average delay of CBR traffic on 512 bytes (video)**

If we look at the graph in Figure 7.11, we can see that the experienced delay in the simulations with the SWAN mechanism has changed radically. In this simulation there are certain differences which separate the experienced results from all the analogous and previous simulations. When running simulations with the SWAN mechanism, the average RT delay was getting excessive within the first 5 seconds. Then the average RT delay smoothed out, and stabilized at an acceptable low level. After 20 seconds, the average RT delay started growing at almost the same rate as it grew without the SWAN mechanism. Even though the average RT delay lays 100-200 milliseconds lower with the SWAN mechanism, it gets too excessive after about 25 seconds. From that time and throughout the rest of the simulation, the RT delay is so high that functioning of any RT application could not be properly supported.

## *7.3 Summary*

In any simulations it is important to be aware of the weaknesses and limitations of the experiment tools and the aberrations this may lead to in the results. Before the simulations in [3] were recreated, some preliminary simulations was accomplished. First, the throughput of the wireless link using different type of traffic was tested. As expected, the throughput was far from the theoretical capacity of 11 Mbps. In these simulations, the SWAN mechanism was had slightly less throughput than without any QoS mechanism.

Section 7.2.2, introduced additional FTP traffic from an external node, when to nodes where having a Real-Time session. In these simulation, devastating impact on the experienced performance where shown when an external node started to exchange BE traffic with the source node of the RT session. However, this effect was not seen when the external node exchanged BE traffic with the destination node of the RT session. In these simulations, there were some differences in the behavior with or without the SWAN mechanism, but none could be said to perform better than another. The simulations from this section also showed us that there could be some problems when a best-effort session is initiated without any measurement of the MAC delay. The transmitting node misconceives the situation, but if the MAC delay could be retrieved from the most adjacent neighbors, this situation could probably be avoided.

Section 7.2.3, contained simulations with variations of the packet size of the CBR traffic. From these simulations, one can clearly see that for an Ad Hoc network, with the SWAN turned on, the performance is dependent on the packet size, when different nodes are present and transmitting in the same BSA. Even when treated equal by the SWAN mechanism, it is clear that applications like Real-Time video or voice, have different needs and different behavior preferences. Therefore, the SWAN mechanism should be able to differentiate Real-Time in different service classes (like the DiffServ mechanism) to better satisfying this needs, so it could deliver an overall improved performance to all applications. Also, from this simulations, where different packet size gives different performance, one can conclude that SWAN also should be able to let applications on the edge nodes, adjust their transmitting rate using compression, when congestion is experienced and the delay get too excessive. Then, a heavily loaded network can be released from its strain, and the delay might reach a level where the Quality of Service could be satisfied for most or all of the applications. The results from these simulations also show us that SWAN is performing very well in this scenario where the CBR packet size was kept low. When the packet size was increased, SWAN was not performing any better, than without any QoS mechanism.

# 8 Simulation scenarios for SWAN

This chapter presents the different simulations scenarios used both to recreate the simulations scenarios from the SWAN articles and in further testing of the SWAN mechanism. The simulations can be divided in two broad categories; the simulations in a Single Shared Channel and the simulations in a Multi Hop situation.

## *8.1 Single Shared Channel Simulation (Description)*

In this simulation scenario all nodes were within transmission range of every other node, thus al nodes shared the same 11 Mbps wireless transmission channel. In this scenario no intermediate routing was performed in order to reach other nodes, but the AODV routing protocol was still up and running. Also, the RTS/CTS mechanism was switched on during these simulations. The simulation area was a 150 m * 150 m flat square, and the transmission range of each node was 250m. The Admission Control rate was set to 2000 Kbps and the Threshold rate was set to 4000 Kbps. Every simulations consisted of UDP real-time traffic; 4 voice flows, 4 video flows versus a different number of TCP background traffic. The background traffic was modeled as a pair of 2, 4, 6, 8, 10 or 12 FTP flows. The simulation time was set to 100 seconds.

This test was carried out with two different scenario setups. One simulation setup where all the flows were evenly distributed, and where each flow had its own receiver/transmitter pair. This simulations setup is later referred to as Single Shared Channel 1. The other simulation setup, includes the same amount of flows, but with fewer nodes so that more flows share the same transmitter/receiver pair. This is referred to as Single Shared Channel 2.

These simulation setups closely match the setups of the simulations carried out in the SWAN articles. The purpose of this simulations test was mostly to recreate the simulations, and verify the validness of the results published in these articles. The purpose of having two different simulations setups, was to se how SWAN behave in situations where the ratio of flows/node where moderate or high.

### 8.1.1 Single Shared Channel scenario 1 (SSC1)

In this simulation scenario every flow had its own transmitter/receiver pair of nodes, which made up the total number of 40 nodes, as illustrated in Figure 8.1. Even though the simulations posted in the SWAN articles have been modeled with 50 nodes, there is no need for any “silent” nodes. That means, only nodes either transmitting or receiving data traffic are present in the simulation scenario.

Nodes with even number are transmitting and nodes on unequal numbers are receiving data traffic. Nodes 0 to node 7 are transmitter/receiver pairs of video traffic. Nodes 8 to node 15 are transmitter/receiver pair of voice. The rest of the nodes are transmitting/receiver pair of greedy FTP traffic. This is the nodes from 16 to node 39. In SSC1 all data traffic was simultaneously initiated at the start of the simulation and stopped when the simulations ended.

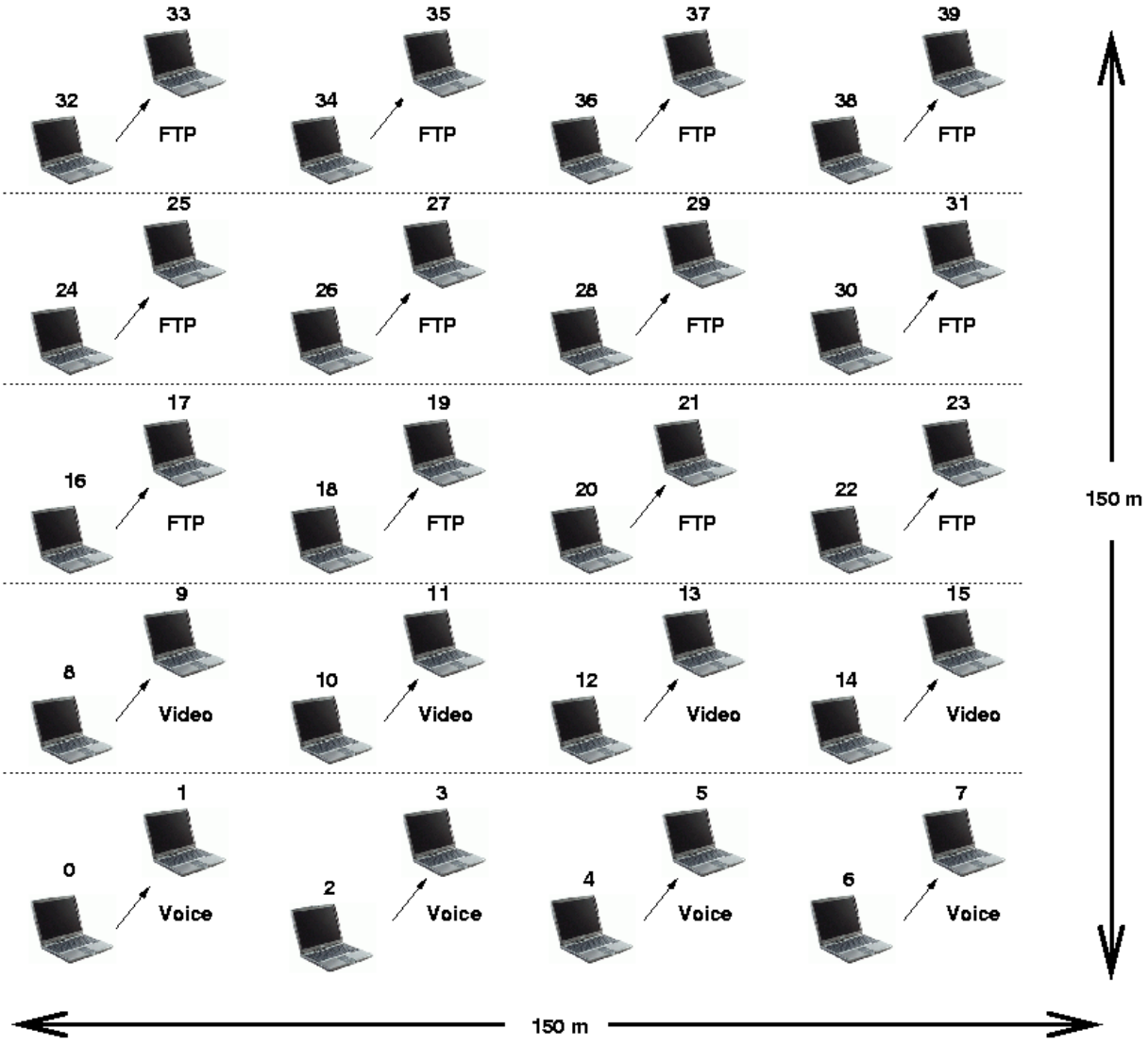



**Figure 8.1: Single Shared Simulation Scenario 1**

### 8.1.2 Single Shared Channel scenario 2 (SSC2)

In this simulation scenario the total number of nodes is reduced to only 8 nodes, but more data traffic are distributed on each node. The first four pair of nodes are both transmitting and receiving video and voice. This is a likely situation in a video conference situation, likely to occur in a rescue area. Here people using such applications on an end node are able to see and speak to the other opponent, at the same time.

Background traffic like FTP is also likely to be present in such situation. Therefore nodes 4 to nodes 7, was heavily stressed with the task of both transmitting and receiving FTP traffic. Nodes 4 to nodes 7 where transmitting or receiving from 0-3 different nodes at the same time. All the different traffic flows where in the same BSA.

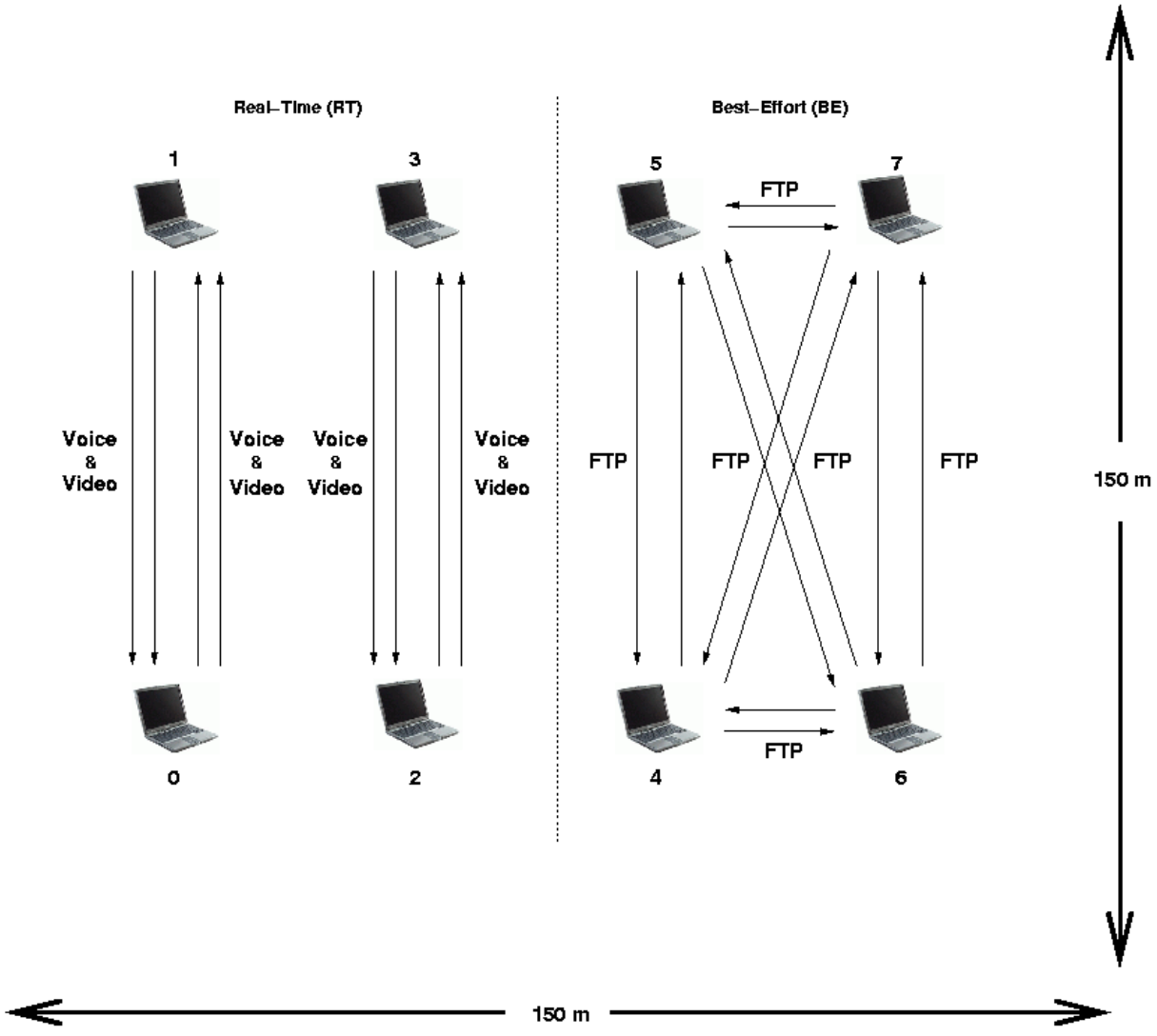



**Figure 8.2: Single Shared Simulation Scenario 2**

## *8.2 Multi Hop Simulation*

In a multi hop scenario the receiving node is not within the radio transmission range of the transmitting node. In order to reach the receiver, the transmitting node has to pass its data packet to the next intermediate node lying on the path between the transmitter and receiver.

The simulations of multi hop scenarios have been divided into two main simulation scenario categories. The first scenario is very similar to the scenario simulated in SSC1, except that all data traffic has to go through one intermediate node to reach its final destination. The second simulation scenario of multi hop contained less data traffic, but the transmitting and receiving nodes is moved further apart from each other and the data traffic has to go through a longer path of intermediate nodes to reach its destination.

### 8.2.1 Multi Hop Scenario 1 (MH1)

In this simulation scenario, the transmitting node is not able to reach the receiving node directly. Therefore the AODV routing protocol is used to keep track of the next hop information. The transmission range is still 250 meter and each transmitter/receiver pair is spaced by 260 meter. The capacity of the wireless medium is 11 Mbps and the RTS/CTS mechanism is turned on during the simulations. The simulation area is a 1500 m x 300 m flat square, and the CBR (video and voice) and FTP sessions are evenly distributed in the simulation area. Also, SWAN's Admission Control rate is set to 2000 Kbps and the Threshold rate is set to 4000 Kbps in the SWAN mechanism. Every session has its own transmitter/receiver pair. Every simulations consisted of UDP real-time traffic; 4 voice flows, 4 video flows versus a different number of TCP background traffic. The background traffic is modeled as a pair of 2, 4, 6, 8, 10 or 12 FTP flows. The simulation time is set to 100 seconds. All data traffic is simultaneously initiated at the start of the simulation and stopped when the simulations ends.

Nodes with even numbers are transmitting, and nodes with unequal numbers are receivers of data traffic. Nodes 0 to node 23 are handling FTP traffic, nodes 24 to node 31 are handling video traffic, and nodes 32 to node 39 are handling voice traffic.

The simulation of MH1 is very similar to the one of SSC1. Both the number of nodes, and the number of traffic flows are the same in these two simulation scenarios. The difference between MH1 and SSC1 is that the nodes are spread over a wider area in MH1 and then is dependent of intermediate routing in order to send data traffic. A consequence of the wide distribution of nodes is that the interference between the nodes radio signal probably will be reduced, and hence improve the performance of both Real-Time and best effort traffic. On the other hand, the fact that the traffic has to go over a longer path will probably decrease the performance and lower the Quality of Service. The purpose of these simulations is to se how a wireless network performing in a multi hop situation, both in only "Best-Effort service" and in the presence of the SWAN mechanism.


**Figure 8.3: Multi Hop Scenario 1**

### 8.2.2 Multi Hop Scenario 2 (MH2)

In this simulation scenario the mobile nodes are organized as a 5 x 4 grid, with the total number of 20 nodes. Both vertically and horizontally, the nodes are spaced by 200 meters. The simulation area is a 1000 m * 800 m flat square. The routing protocol used is the AODV routing protocol. The bandwidth capacity is 11 Mbps, and the RTS/CTS mechanism is turned on during the simulations. The radio transmission range is 250 meter, the Admission Control rate is set to 2000 Kbps, and the Threshold rate is set to 4000 Kbps. The total simulation time is 100 seconds.

Compare to the previous simulation (MH1) this simulation contains fewer flows, but more hops to reach its final destination. The purpose of this simulation is to se how a greater number of intermediate hops in a traffic path impact the performance in a wireless network running the SWAN mechanism.

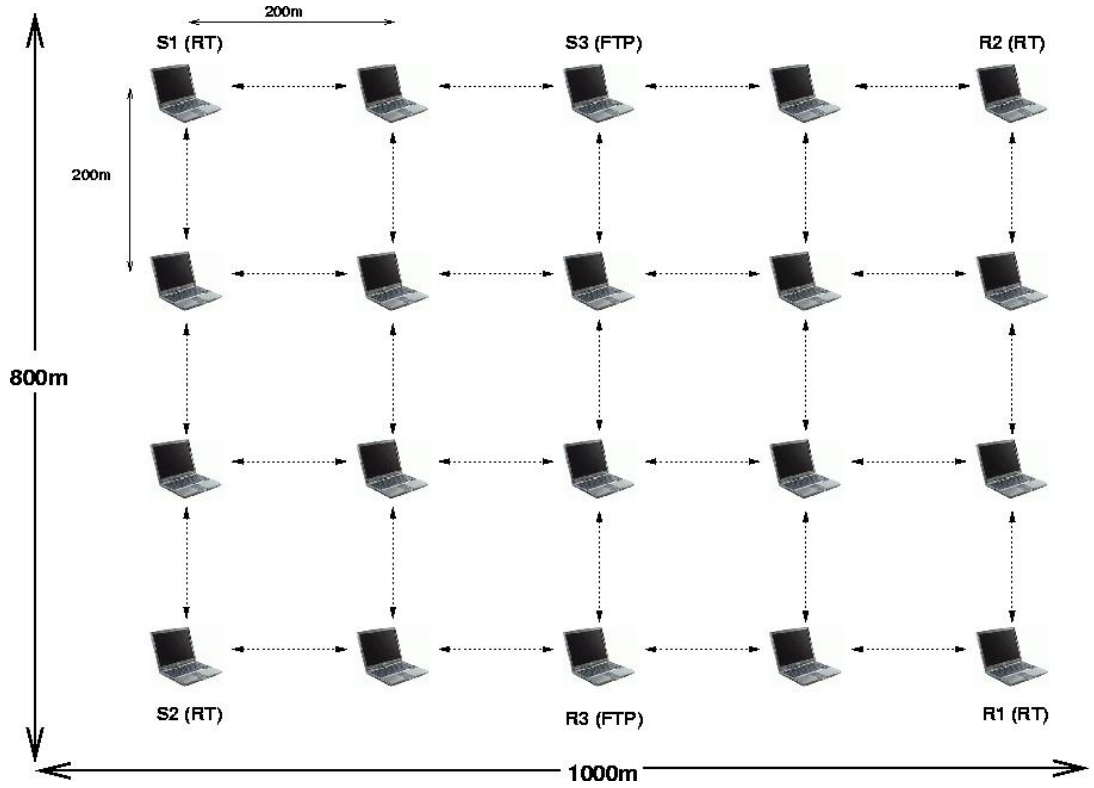



**Figure 8.4: Multi Hop Scenario 2**

When simulating the simulation scenario of MH2, 6 different simulations was carried out with different combinations on aggregating the data traffic. From the previous simulations, we know that Voice and Video traffic have different characteristics. The shortest path from each corner nodes diagonally spaced is minimum over 7 links, through 6 intermediate nodes. In all different combination one FTP flow runs from the upper middle node to the lower middle node. The crossing FTP session intersects all the available paths for CBR sessions (from $S_x$ to $R_x$, or the reverse path) with FTP background traffic.

All the traffic flows is aggregated from the beginning to the end of the simulation period of 100 seconds.

In the first combination, only aggregated video and FTP traffic is aggregated. The upper left node is transmitting video traffic to the lower right node, and the lower left node is transmitting to the upper right node.

In the second combination, video was now left out and replaced by voice traffic. The upper left node was now transmitting voice traffic to the lower right node, and the lower left node was transmitting to the upper right node.

In the third combination of MH2, video traffic flowing was going in both directions of the nodes. Then the upper left node was both sending and receiving traffic from the lower right node. Also, the lower left node was also sending and receiving traffic from the upper right node.

The fourth combination was similar to the third, except that the video traffic was replaced by voice traffic. The FTP traffic was still going from the upper middle node to the lower in the middle.

The fifth traffic scenario was a combination of the first and second aggregation. Here, both video and voice was flowing from the upper left node to the lower right node. Also, both video and voice traffic was flowing from the lower left node to the upper right node. The FTP traffic was still present.

Finally, the last aggregation scenario was a combination of the 3$^{rd}$ and 4$^{th}$ aggregation. Here both voice and video traffic was flowing in both directions of the $S_x/R_x$ pair. In addition to the FTP traffic from the upper to the lower middle node, both video and voice traffic was going to and from the upper left node to the lower right node. In addition, both video and voice traffic was flowing from the lower left node to the upper right node.

These different combinations of the aggregated data traffic in MH2 where pretty similar to each other. Therefore, the result from the simulations, in chapter 9, only presents the most interesting of these simulations.

# 9 Tests and Results

This chapter presents the result from simulations of several different scenarios concerning the performance of the SWAN network module.

## *9.1 Single Shared Channel Simulation*

### 9.1.1 Simulation of SSC1

**Throughput**

From the simulation of SSC1, figure 9.5 shows the average Best-Effort (FTP) throughput of each node, versus the different number of on going FTP sessions. As expected, and also stated in [3] the total throughput of Best-Effort (FTP) traffic in the network, was higher with the SWAN control mechanism turned off, than it was when SWAN in use. This is true no matter how many FTP traffic flows running in the background. (0-12 FTP flows)

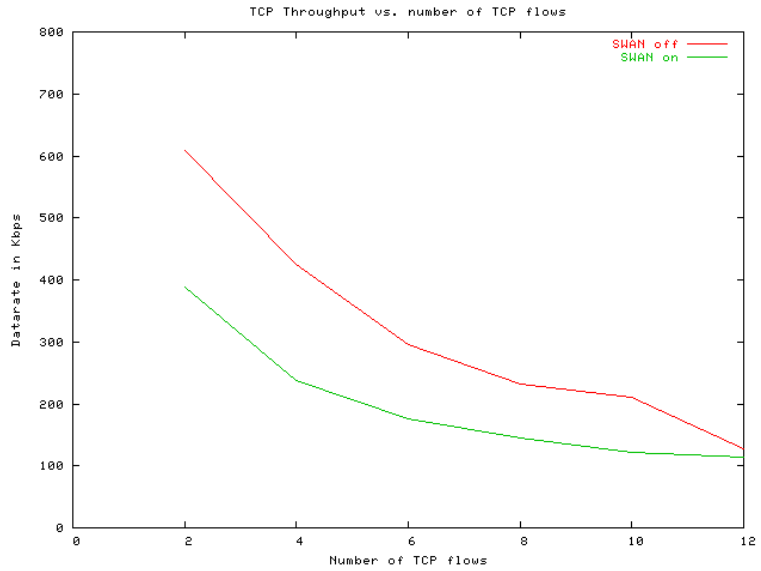



**Figure 9.1: Average TCP throuhtput vs number of TCP flows**

When comparing figure 9.1 to the one posted in the SWAN articles [1] and [3], the average TCP throughput received in this simulation (SSC1) is clearly much below the TCP throughput shown in the SWAN articles (see figure 5.2). Actually, the TCP throughput from SSC1, presented in figure 9.1 is only about one half of the TCP throughput presented in the SWAN articles.

The total throughput of each type of data traffic from the simulations where only two FTP sessions was running, are presented in hence, Figure 9.2 without SWAN, and Figure 9.3 with SWAN. Even thought the throughput of video traffic was slightly more stable without the SWAN mechanism, the overall received throughput of video traffic in these simulations was at the same level as when the SWAN mechanism was used. For voice traffic, the received throughput was also at the same level when the SWAN mechanism was in use, as it was when the SWAN was not used. Though, a more stable throughput for video was now seen in the result from the simulation where the

SWAN mechanism was running. As stated above, the FTP throughput was higher, but the FTP throughput was also more stable in the simulation without the SWAN mechanism.

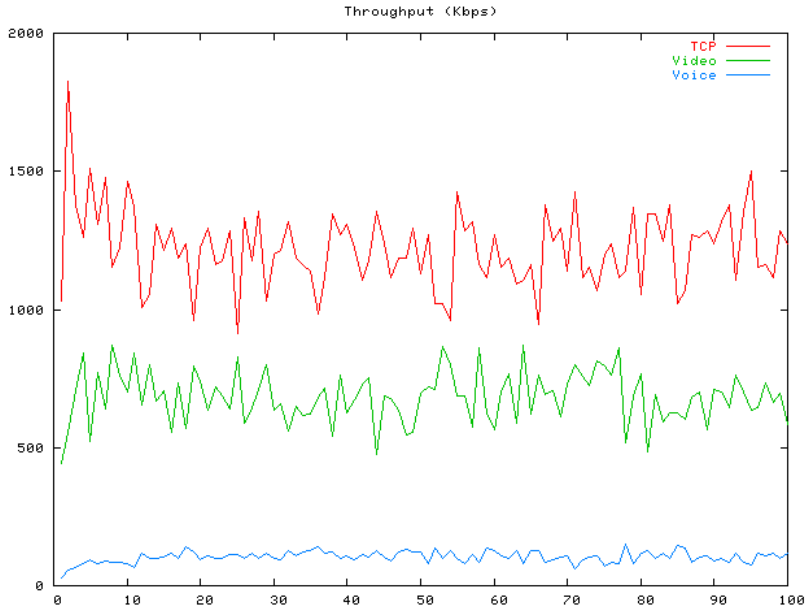



**Figure 9.2: Throughput - SWAN off, 2 TCP flows**

In the simulation without SWAN and where 12 FTP traffic flows where present, both video and voice traffic experienced much lower throughput, than in the simulations with fewer FTP sessions present. When the SWAN mechanism is not used all the data packets is treated equally as Best-Effort packets. At the same time, in this simulation the FTP session are sending even greedier. As a result; the TCP traffic is actually given more of the total bandwidth “cake” and the total TCP throughput in this simulation where higher than in the simulation with only 2 FTP flows.

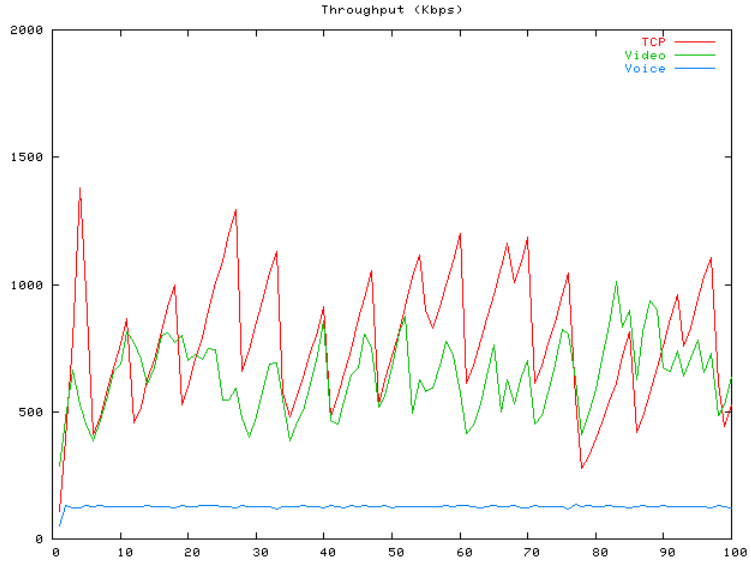



**Figure 9.3: Throughput - SWAN on, 2 TCP flows**

In the simulations where the SWAN mechanism where turned on, the total TCP throughput was decreased from a level around 1500 Kbps, to a level where it oscillate between 1300-1500 Kbps. Video and voice traffic was much higher and more stable in the simulation where the SWAN mechanism was controlling the network. The total video and voice throughput was also closer to the received throughput in the simulations where only two FTP sessions was present. This is due to video and voice traffic being prioritized in the SWAN mechanism.

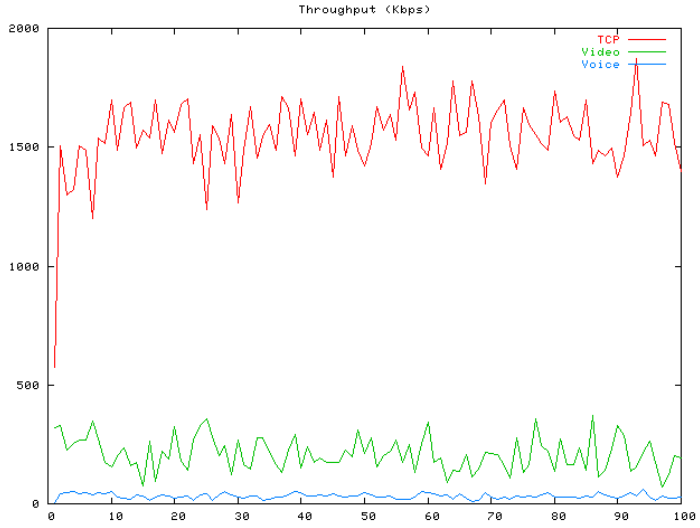



**Figure 9.4: Throughput - SWAN off, 12 TCP flows**

In SSC1, the total throughput of all the data traffic with SWAN off, varies from 2.0 to 2.4 Mbps. The total throughput with SWAN on, varies from 1.5 to 1.9 Mbps. Comparing this results to the results of the available bandwidth, shown in figure 7.1 and figure 7.2; when running the simulation without SWAN, 60-70 percent of the available bandwidth is utilized, while running with SWAN, only 45 to 60 percent of the channel is utilized.

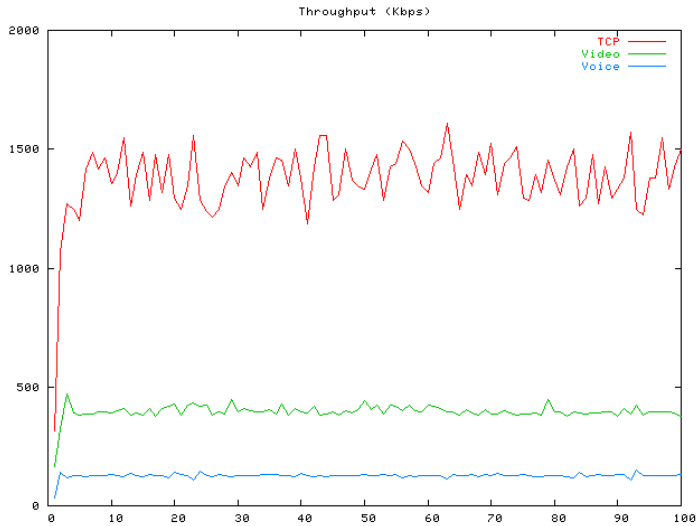



**Figure 9.5: Throughput - SWAN on, 12 TCP flows**

## Delay

When it comes to delay in the simulations of SSC1, the result shows that the SWAN mechanism was able to give a much lower and often, more stable delay for the Real-Time traffic, compared to the delay experienced when SWAN was turned off.

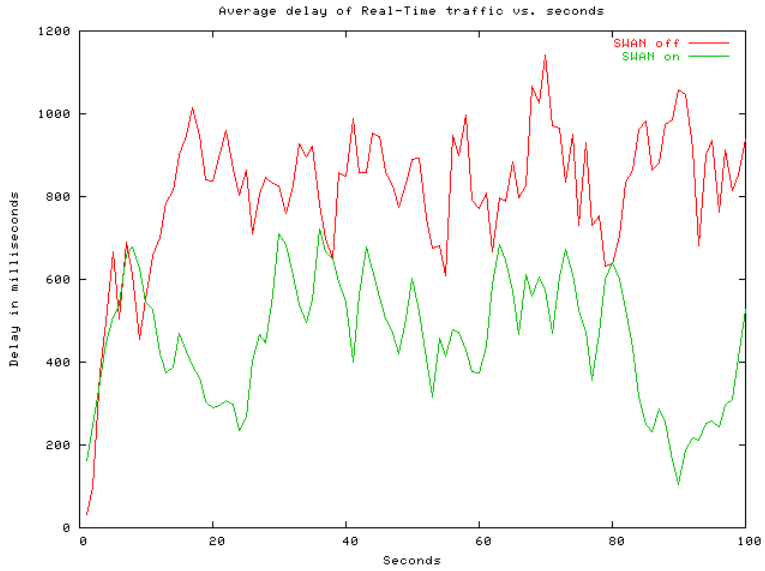



**Figure 9.6: Average RT delay, 2 TCP flows**

In the simulation with 2 FTP background traffic flows, the average RT delay ranges from about 600 milliseconds, up to almost 1200 milliseconds, when SWAN was not used. When the SWAN mechanism was turned, on the delay is noteworthy reduced; the RT delay now ranges from 200-300 milliseconds and only up to 600-700 milliseconds.

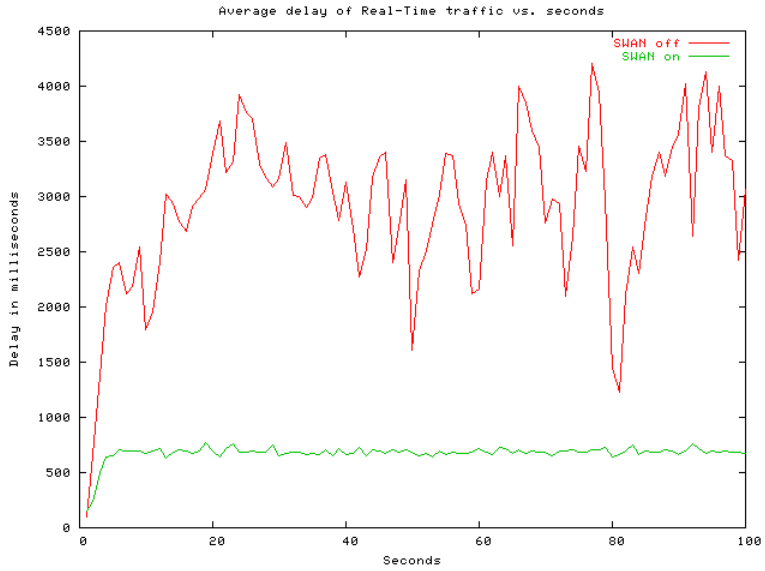



**Figure 9.7: Average RT delay, 12 TCP flows**

In the simulations where 12 FTP sessions where present as background traffic, the divergence between the experienced delay with and without SWAN was much higher than in the simulations with only 2 FTP flows. The RT delay without the SWAN mechanism was extremely high. It ranged from about 1500 milliseconds in the lower end, up to 4000 milliseconds in the upper. When the same simulation was carried out with the SWAN mechanism turned on, the delay laid pretty stable around 600 milliseconds.

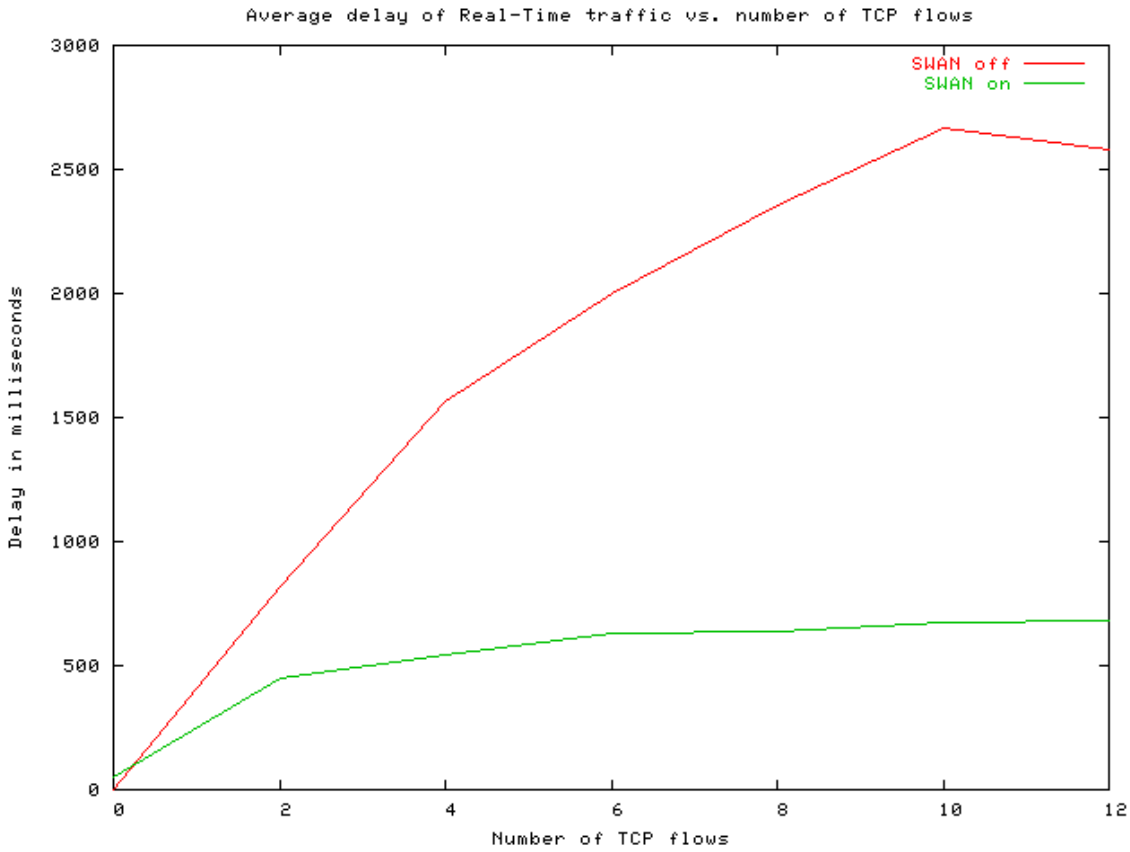



**Figure 9.8: Average end-to end delay (SWAN on / SWAN off) vs. number of TCP flows**

In figure 9.8, the average end-to-end Real-Time delay versus, a raising number of TCP flows present in the simulation area, is shown. In this simulation, the idea was to recreate the same simulation, and then see if similar result as those posted in [1] and [3] could be recreated. When comparing these results (figure 5.1 vs. figure 9.8), the plot of the graphs is very familiar, but still the results are indeed very different. In figure 5.1 the scale of the delay axis is in tens of milliseconds, while my delay results plotted in figure 9.8, is in thousands of milliseconds. The gap between these measurements is tremendous. This is most likely explained by the fact that the delay measured and shown in figure 5.1 is not the end-to-end delay, but indeed the average RT delay measured at the MAC layer.

The average end-to-end delay with the SWAN mechanism was relatively stable around 500 milliseconds, while delay in the simulation without SWAN

grew linearly up to almost 2700 milliseconds. Even though the delay was much lower for the SWAN mechanism, the result from this simulation shows clearly that the end-to-end delay of this simulation is way beyond the maximum delay that can be handled by a Real-Time application. ITU-T's recommendation [16] states an upper limit of delay on 400 milliseconds for any type of applications, while 150 milliseconds is the recommended upper limit of end-to-end delay for a Real-Time application.

### 9.1.2 Simulation of SSC2

## Throughput

From the simulation of SSC2, figure 9.9 shows the average throughput of Best-Effort (FTP) traffic versus a raising number of background FTP sessions. As expected, the simulation of SSC2 shows that the TCP throughput with SWAN was lower than when running simulations without SWAN. The overall average TCP throughput in SSC2 lies close upon the result gained in the simulations of SSC1.

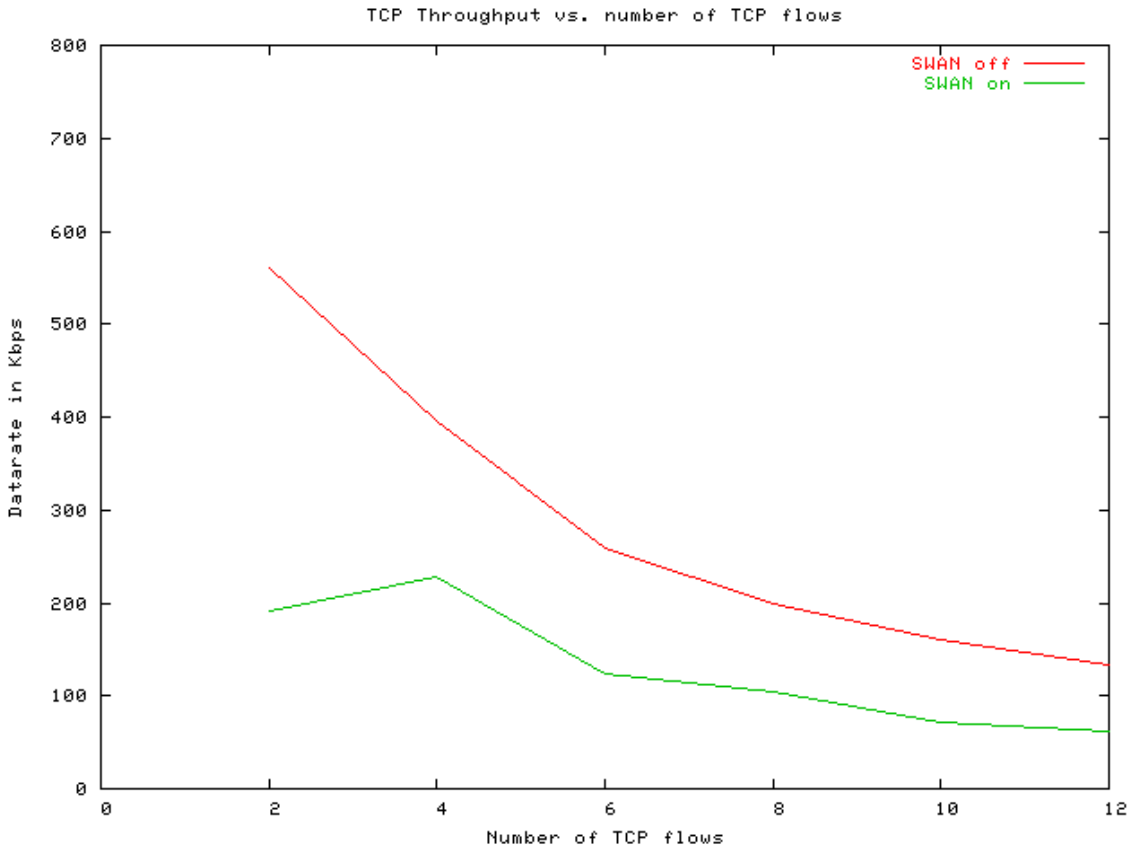



**Figure 9.9: TCP throughput vs. number of TCP flows**

In the simulation of SSC2, when only two FTP sessions was running as background traffic (see figure 9.10 and figure 9.11), the received throughput for voice traffic was almost the same in the simulation without SWAN, as it was in the one with the SWAN mechanism turned on. For video traffic, the throughput was actually to some extend more unstable with SWAN, than it

was without SWAN, but the average throughput was very close in those two simulations. Also, as one can anticipate from the construction of the SWAN network model, the throughput for the TCP traffic was much lower in the simulation when the SWAN mechanism was turned on, compared to the simulations running without SWAN.

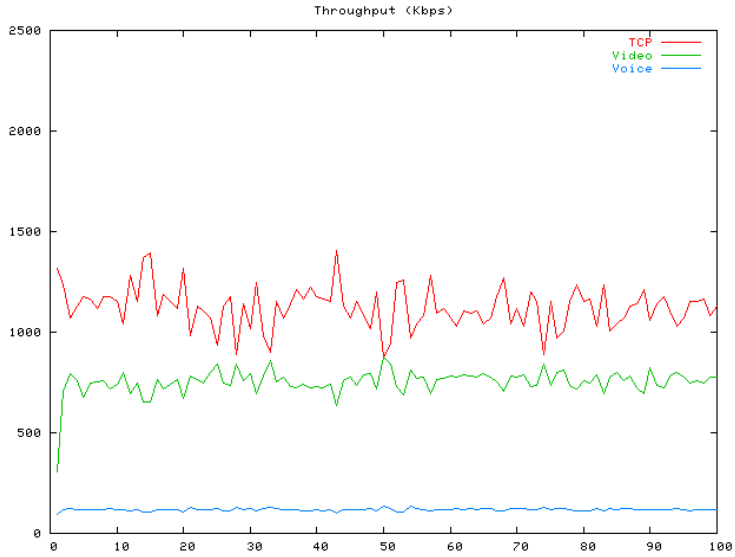



**Figure 9.10: Throughput - SWAN off, 2 TCP flows**

In the simulations where the number of FTP flows was increased to 12 (see figure 9.12 and figure 9.13), the throughput for both voice and video was slightly higher in the simulations where the SWAN mechanism was turned on, compared to the simulation without the SWAN mechanism. Even so, in both simulations the total throughput stayed around 100 Kbps for voice, and 500-600 Kbps for video. When it comes to the throughput for the FTP traffic, the throughput now as in previous simulations was heavily decreased when the SWAN mechanism was switched on. This result is in sharp contrast to the analogous simulations in SSC1, where the TCP throughput lied around 1500 Kbps both with and without SWAN.


**Figure 9.11: Throughput - SWAN on, 2 TCP flows**

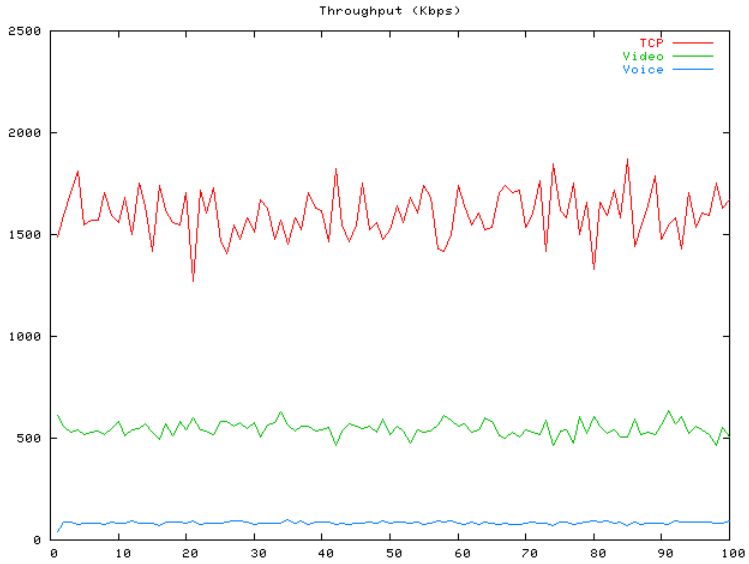



**Figure 9.12: Throughput - SWAN off, 12 TCP flows**

Compared to SSC1, the simulation where the SWAN mechanism was not in use, the average TCP throughput with the all different number of TCP background traffic, ranges from 3 to 50 percent lower than the average TCP throughput in SSC1. At the same time, when running with SWAN, the average TCP throughput of SSC2 ranges from 4 percent higher to 24 lower than in SSC1 (see figure 9.1 and figure 9.9). These results could imply that it might be more difficult to utilize the wireless channel when the density of flows on each node is high, and hence the workload on each node also is high. This was the case when SSC2 was compared to SSC1. It could also imply that SWAN does not handle this higher density very well.

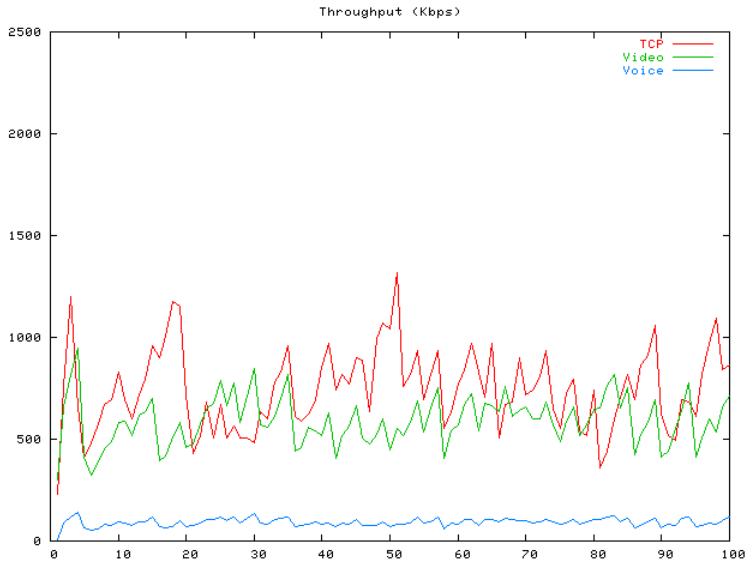



**Figure 9.13: Throughput - SWAN on, 12 TCP flows**

The total average throughput of all traffic in SSC2 ranges from 2.0 to 2.2 Mbps in the simulation without SWAN. With SWAN, this total average throughput ranged from 1.0 to 1.6 Mbps. The total utilization of the channel is then 60-70 percent without SWAN, which is the same as in the simulation of SSC1. When the SWAN mechanism was turned on, the utilization of the bandwidth was down to 30-50 percent, which is less than in SSC1.

## Delay

In the simulation of SSC2 the difference of the RT delay with and without SWAN was not as high as experienced in the SSC1 simulation. Actually, with only a few FTP background traffic flows present, the delay was higher when the SWAN mechanism was used than it was without the SWAN mechanism. In figure 9.14 one can se that the RT delay with SWAN was almost the same as in SSC1 (see figure 9.8). In the simulation of SSC2, the average RT delay without SWAN was heavily decreased compared to SSC1. SWAN was performing slightly better, and the average RT delay experienced in the simulation with the SWAN mechanism was mainly below the delay experienced without SWAN.

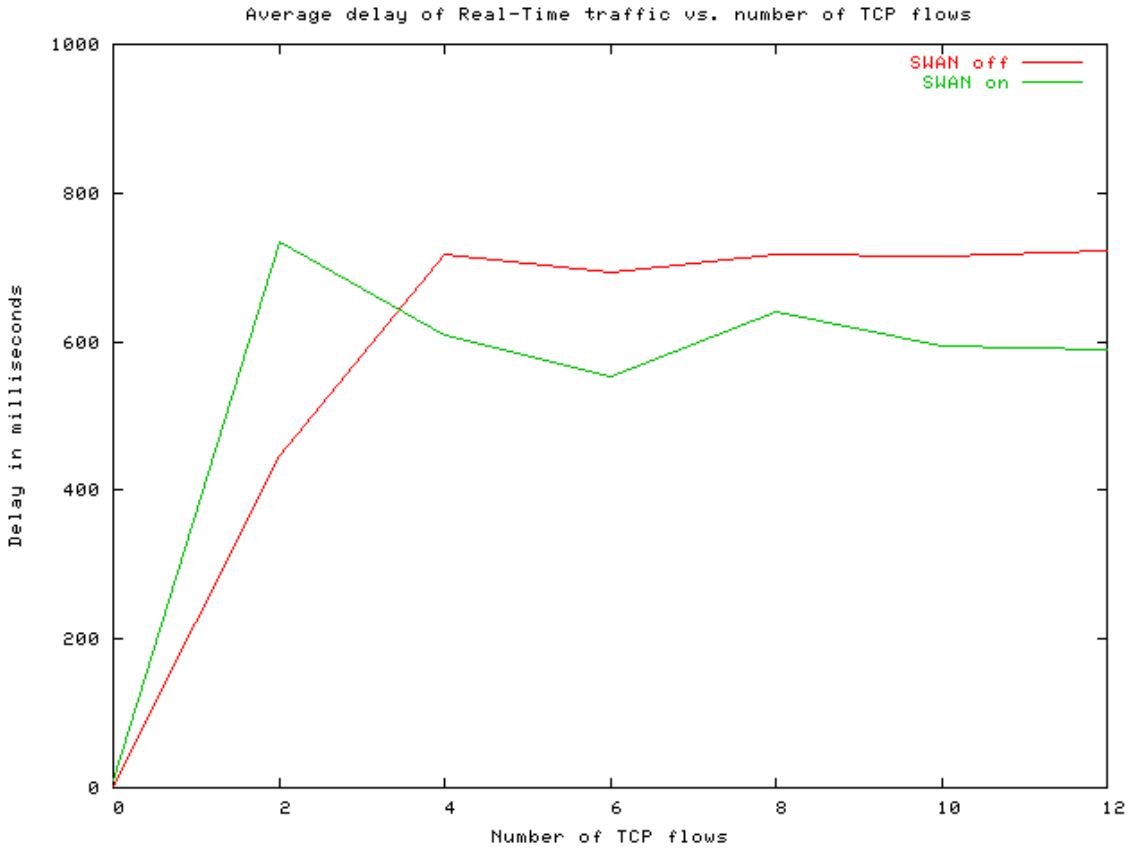



**Figure 9.14: Average end-to end delay (SWAN on/SWAN off) vs. TCP flows**

In the simulations with only 2 FTP background flows (see figure 9.15), the average RT delay without the SWAN mechanism laid mostly around 500 milliseconds, while the delay with the SWAN mechanism turned on ranged from 300 up to 1400 milliseconds. The delay experienced in the simulation with the SWAN mechanism, was very unstable, and most of the time it was higher than the simulation without SWAN.

It seems like that in the simulation scenario of SSC2, the performance of the SWAN mechanism compared to having no QoS mechanism, is higher when the number of background traffic present is increased (compare Figure 9.15 with Figure 9.16). Indeed, this was also the case in the simulation of SSC1. The simulation where 12 FTP sessions was present as background traffic flows can be viewed in figure 9.16. Even thought the RT delay with SWAN ranged from 400 up to 1200 milliseconds, most of the simulation time, the delay laid below the delay experienced without the SWAN mechanism. Without SWAN, the delay laid just below 800 milliseconds except in the very first part of the simulation.

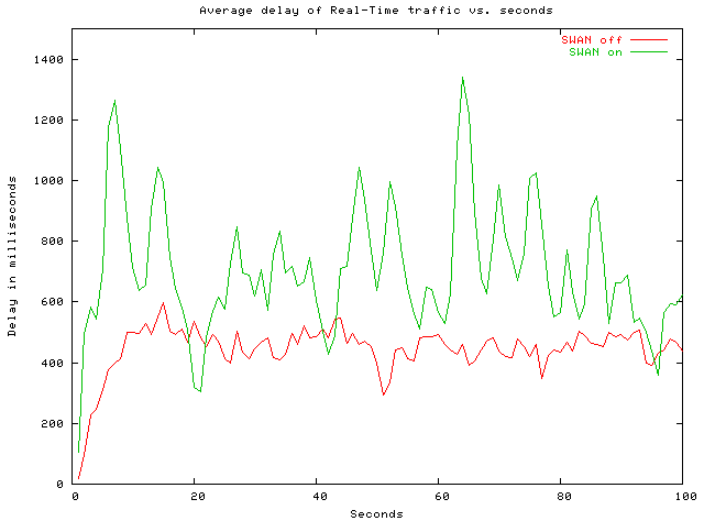



**Figure 9.15: Average RT delay - 2 TCP flows**

These simulations showed that the RT delay for the SWAN mechanism was slightly lower than the delay without SWAN, but the result from the simulation of SSC2 shows that the improvement with SWAN is not enough to properly support Real-Time traffic. The average Real-Time traffic delay for SWAN when 2 or more TCP background traffic is introduced ranges from 550 to 800 milliseconds. This is 3-5 times higher than the upper limit of the ITU-T recommendation.

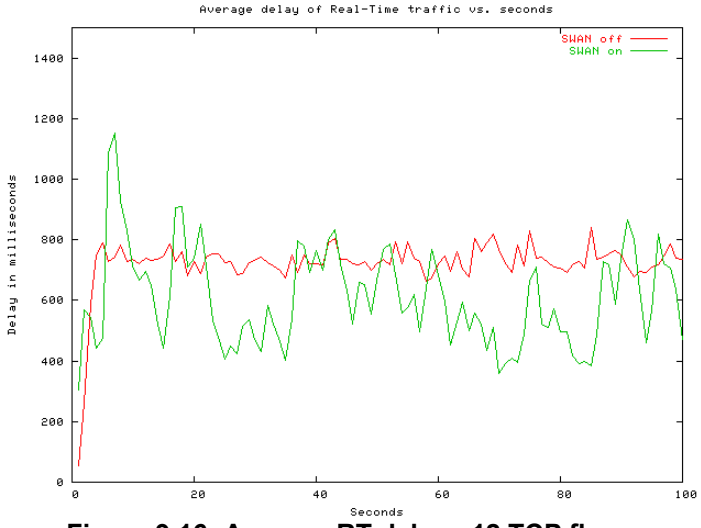



**Figure 9.16: Average RT delay - 12 TCP flows**

Even though the simulation scenario of SSC2 varies a little bit from those in described in the SWAN articles [1] and [3], the results obtained in this simulations has a large aberration from the result posted in the SWAN articles. The experienced delay in both SSC1 and SSC2 are in order of magnitude of 100 of milliseconds, while the result posted in the SWAN articles are in order of magnitude of tens of milliseconds. This further confirms that the delay measured in the SWAN articles are not the end-to-end delay, but the delay experienced at the MAC layer.

## *9.2 Multi hop Simulation*

### 9.2.1 Simulation of MH1

## Throughput

In the simulation of MH1, Figure 9.17 shows the average Best-Effort throughput for FTP traffic of each node, versus an increasing number FTP sessions. Here, the TCP throughput was only slightly higher without SWAN, compared to the simulation where the SWAN mechanism was turned on. The major drawback in the simulation of this scenario was the extremely low TCP throughput received both with and without SWAN. The average TCP throughput ranged from about 40 Kbps up to 120 Kbps. The graph is only slightly decreasing, so the raising number of TCP background traffics, does not seem to impact the throughput in the same manner as in SSC1 (see figure 9.1).

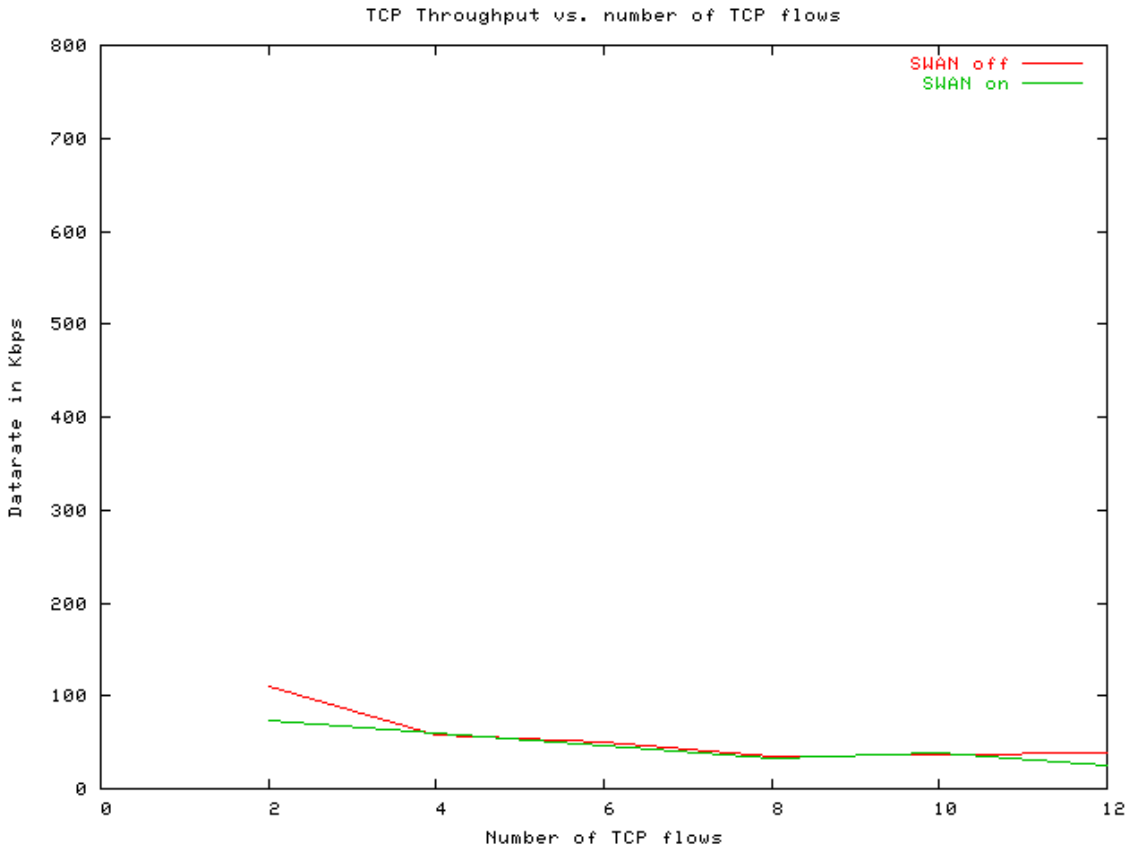



**Figure 9.17: TCP Throughput vs. number of TCP flows**

In the simulations of MH1 with only two FTP flows in the background, video and voice traffic received approximately the same throughput with or without SWAN (Figure 9.18 and figure 9.19). This was approximately the same throughput which also was received in the analogous simulation without multi-hop in SSC1 (see figure 9.2 and figure 9.3). When it comes to the throughput of the FTP traffic, the results show a different behavior with or without the

SWAN mechanism. Without SWAN, the throughput was raised to almost 1000 Kbps for the first five seconds, then it dropped down below 500 Kbps. Throughout the rest of the simulation time, the TCP throughput showed a very unstable behavior. It stayed mostly below 500 Kbps and several times the throughput was almost 0 Kbps. With SWAN the TCP throughput is more stable, but at the same time much lower.

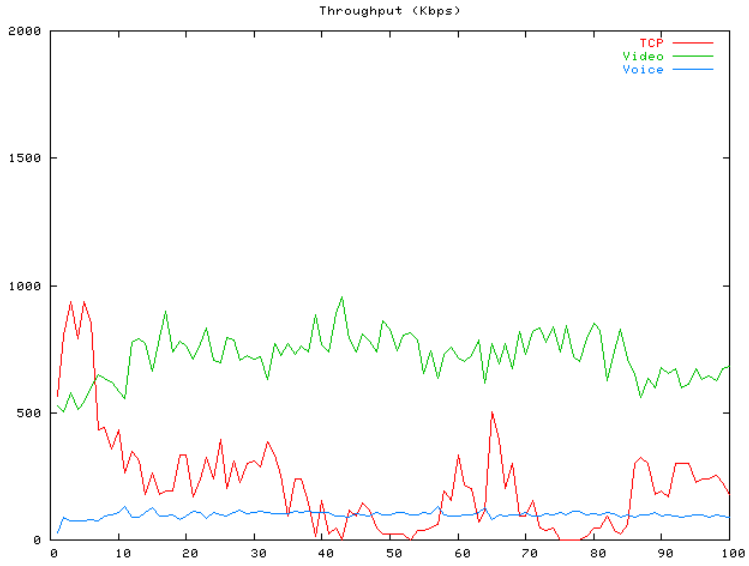



**Figure 9.18: Throughput - SWAN off, 2 TCP flows**

In the simulation where the number of FTP sessions was increased to 12, the throughput for both video and voice was just slightly below the throughput received in the simulations where only two FTP flows were present as background traffic. This is true both with and without the SWAN mechanism. Due to the similarity of these similar results, the graphs from the simulations with 12 FTP sessions are not present in this paper.

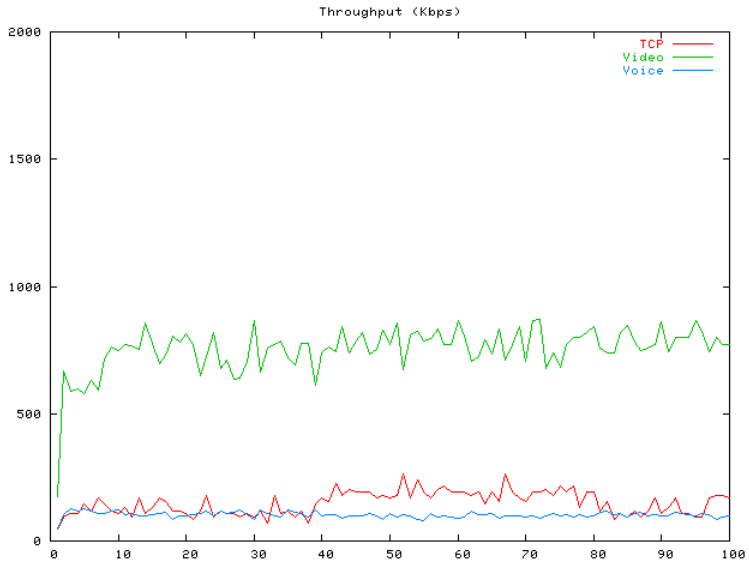



**Figure 9.19: Throughput - SWAN on, 2 TCP flows**

## Delay

From the result gained in the simulations of MH1, it is clear that the experienced Real-Time delay in average was lower when the SWAN mechanism was running compared to the simulation where the SWAN mechanism not was providing any QoS guarantees. The average RT delay when SWAN was not running, was in range of 80 milliseconds below, and up to 800 milliseconds above, the average RT delay experienced when the SWAN mechanism was providing soft Quality of Service guarantees (see figure 9.20).

Compared to the result from SSC1, the average RT delay in MH1 is not growing linearly when the SWAN mechanism not was running, which is the case in SSC1. When 4 or more FTP flows was present as background traffic, the average RT delay in MH1 is less than half of that experienced in the simulations of SSC1.

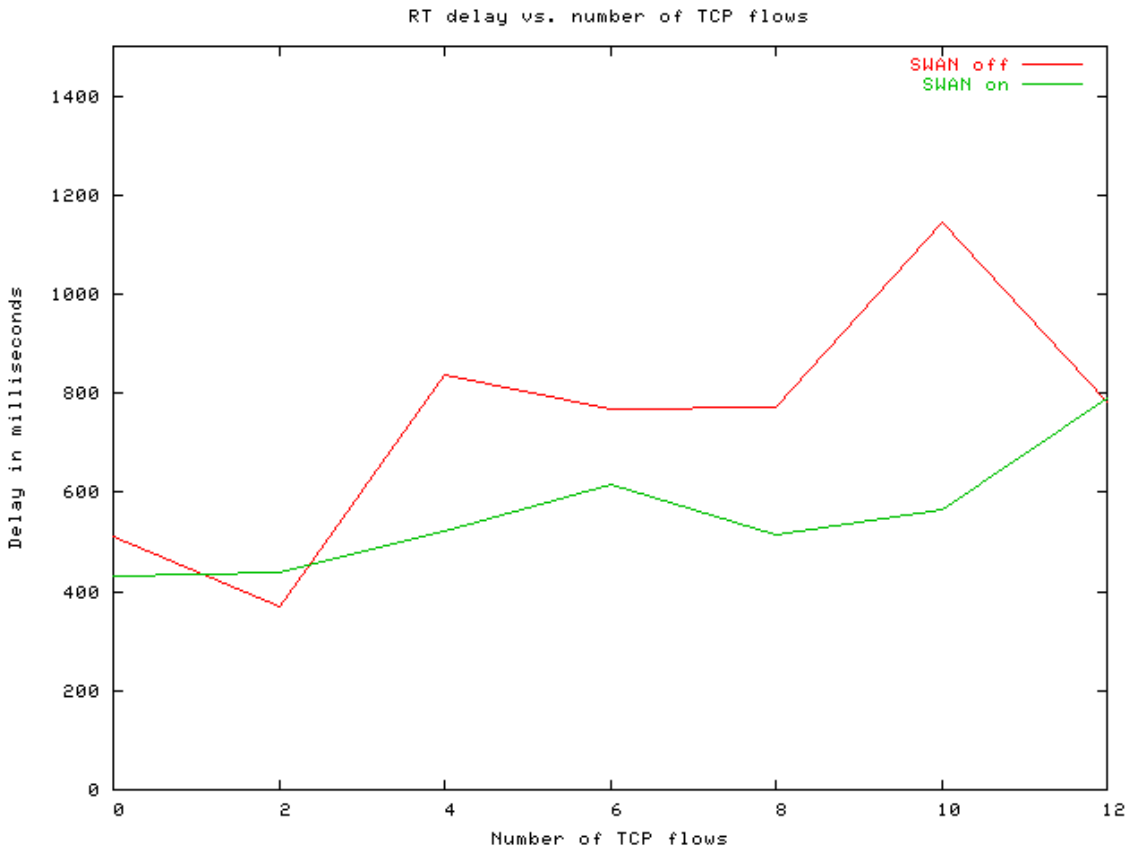



**Figure 9.20: Average end-to-end delay vs TCP flows**

With the SWAN mechanism present, the average RT delay in MH1 was slightly above that delay experienced in SSC1. When no FTP flows was present, the average RT delay was more than 400 milliseconds in MH1, but no substantial delay is experienced with 0 FTP flows in SSC1. For the presents of 2 and up to 10 FTP flows, the average RT delay was

approximately at the same level as in SSC1. When 12 FTP background traffic flows was added to the simulation, the average RT delay was more than 150 milliseconds higher in MH1 than in the analogous simulation in SSC1.

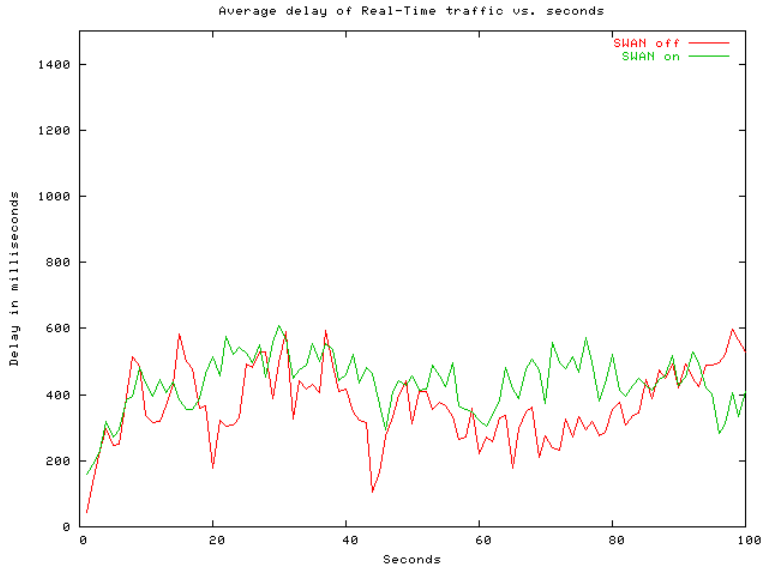



**Figure 9.21: Average RT delay, 2 TCP flows**

Even when the SWAN mechanisms improved performance, having the Real-Time traffics delay in mind, the delay is still too high to give the stable and robust network performance a Real-Time application truly need.

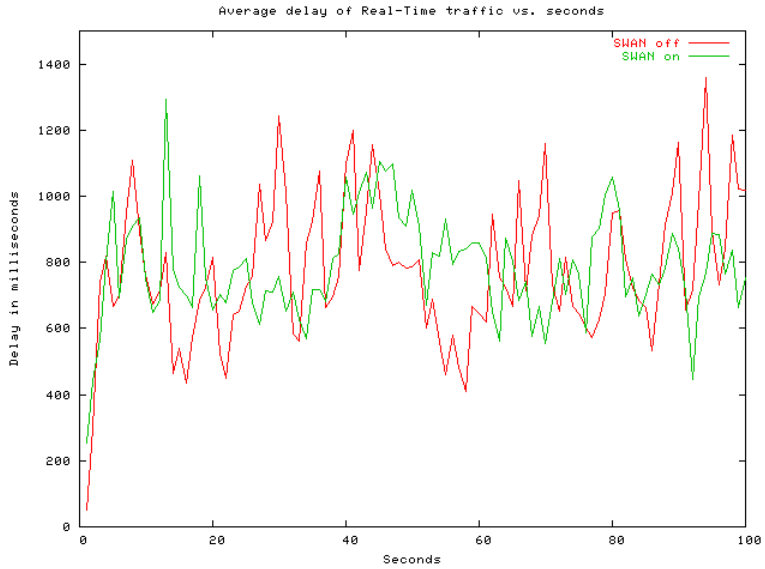



**Figure 9.22: Average RT delay, 12 TCP flows**

When the SWAN mechanism was up and running, the simulations where either 2 or 12 FTP flows was running as additional background traffic, the RT

delay was very close to the delay experienced when not running the SWAN mechanism. In figure 9.21, the graph show the RT delay when 2 FTP flows was running as disturbing background traffic. Here, the delay mainly stayed in the ranges from 300 up to 500 milliseconds.

In figure 9.22, one can se the result of the analogous simulation with 12 FTP flows as disturbing background traffic. Still the delay was pretty much at the same level whether the SWAN mechanism was running or not. But now the delay was much higher than in the simulations displayed in Figure 9.21. The RT delay range from 400 and up to more than 1200 milliseconds.

To get a deeper view of the experienced Real-Time delay, the delay of video and voice is present as individual plots in the graph in Figure 9.23. From this graph, it looks very clear that voice and video traffic have different behavior characteristics. With the SWAN mechanism, the video traffic (blue line) experienced an average delay of 400-500 milliseconds when the number of TCP background traffic flows where 10 or less. With the presents of 12 FTP flows the average delay for video was about 750 milliseconds.

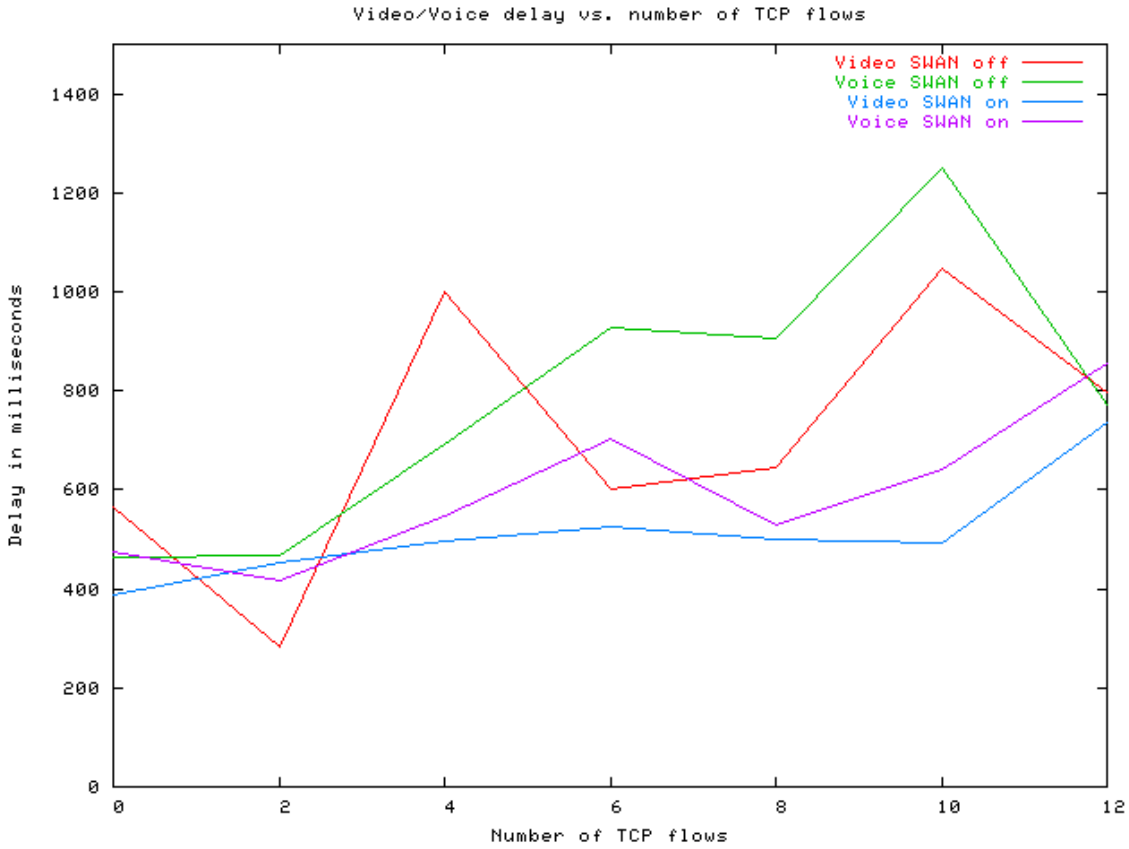



**Figure 9.23: Average end-to-end delay Voice/Video vs. TCP flows**

The average delay of video traffic in the simulation where the SWAN mechanism where not present (red line) was highly variable. When no FTP flows where present, the experienced video delay where almost 600

milliseconds. With a raising number of FTP background traffic flows, the average video delay varies from 300 and up to 1000 milliseconds.

When the SWAN mechanism was not used to assure the Quality of Service, the voice delay (green line) started at about 500 milliseconds, when no FTP flows was present, and up to its highest at 1200 milliseconds, when 10 FTP flows where present. In the analogous simulation where the SWAN mechanism was used, the voice delay (purple line) ranged at a lower level. The delay ranged from 400 milliseconds and up to just above 800 milliseconds at it's highest.

From figure 9.23 we can see that the average delay is higher for voice traffic than for the video traffic, both with and without the SWAN mechanism. This clearly shows that the performance characteristics of voice and video traffic is highly different, and need different treatment in a data network. This is not the case in the resent SWAN network model.

### 9.2.2 Simulation of MH2

In the simulations of the Multi-Hop scenarios, six separate simulations with different combination of traffic aggregation were accomplished. Because some of them are analogous, only four of these simulations are present in this chapter.

#### 9.2.2.1 Delay and Throughput

In the ***first*** simulation of MH2, only video traffic where aggregated in addition to the crossing FTP background traffic flow. The delay observed in this simulation, is present in figure 9.24, and from that graph we can see that the average delay of the two video streams in this simulation was highly variable. Even though the average delay measured each second was around the recommended upper limit of delay for any data traffic, stated by the ITU-T in [16] for most of the simulation period, it was also way beyond that limit several times. Both with and without SWAN, the average delay throughout the 100 seconds simulation period was high above ITU-T's recommended limit of delay for Real-Time applications.

The strange incident in these simulations was that the introduction of the SWAN mechanism did not seem to improve the QoS. Indeed, it seemed that the experienced performance while using the SWAN mechanism, gave us even worse results than the result gained in simulations running without any QoS mechanisms. With the SWAN mechanism, the average Real-Time delay measured once each second, ranged from almost nothing up to 1300 milliseconds. Without the SWAN mechanism, the result where to some extends actually better. Then the average delay ranged from about 50 seconds at it lowest, and up 1150 milliseconds at its highest peak.

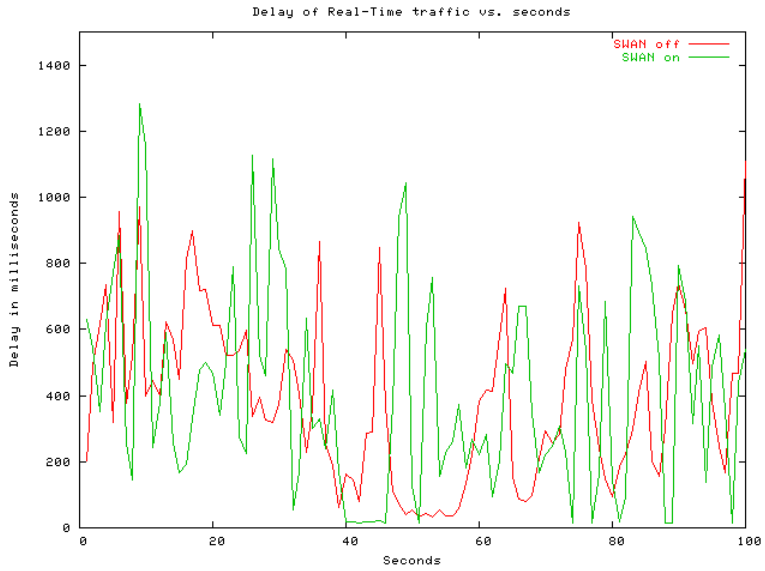



**Figure 9.24: Average delay of Video streams ech second**

As observed in many in the previous simulations with the SWAN mechanism, the start up delay in the very first seconds was very high. The delay of the first packet was actually 5 times higher with the SWAN mechanism than it was without it. Also, the maximum delay of one single packet, experienced in this simulation, run with the SWAN mechanism, was 72 % higher than the delay experienced without the SWAN mechanism.

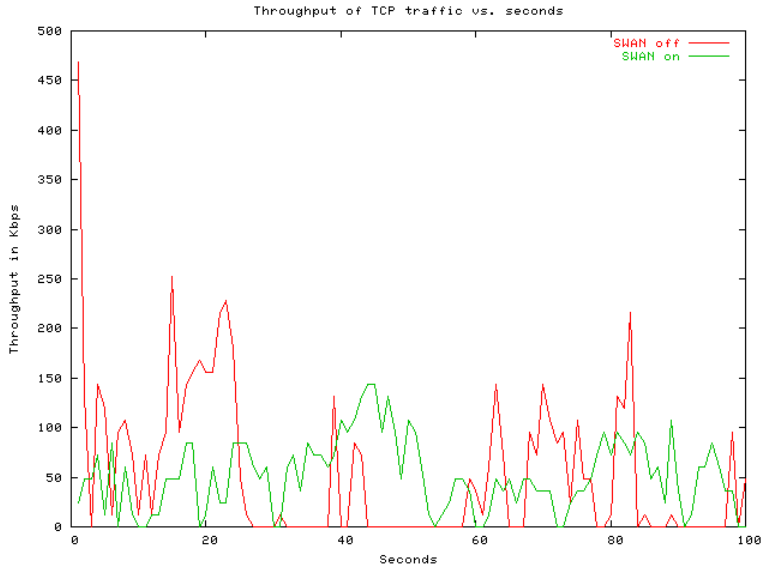



**Figure 9.25: Average TCP throughput**

When it comes to the average delay throughout the simulation period, the divergence in the experienced delay was not as high as experienced with the first packet delay and the maximum packet delay, but still the SWAN mechanism gave higher output values. The average delay of the 100 seconds

simulation period was 389 milliseconds without the SWAN mechanism and 440 milliseconds when the SWAN mechanism was used. This is shown in Table 9.1.

| | First packet delay | Max delay | Average delay | TCP throughput |
|---|---|---|---|---|
| SWAN off | 86.79 ms | 2264.42 ms | 388.59 ms | 5376 Kb |
| SWAN on | 450.44 ms | 3894.96 ms | 440.14 ms | 5124 Kb |

**Table 9.1: Measurement from MH2, simulation 1**

Even though the TCP total throughput was slightly higher without the SWAN mechanism in these simulations, the achieved TCP throughput was approximately the same. Without the SWAN mechanism, the total received TCP throughput was 5376 Kb. With the SWAN mechanism, the TCP throughput received was 5124 Kb. This gave an average TCP throughput on 51-54 Kbps. A graphical presentation of the TCP throughput can be viewed in figure 9.25.

In the ***second*** simulation of MH2, the video traffic was replaced by voice traffic. The FTP background traffic was still present. The results from these simulations, was somehow different from result of the previous simulations. From the graph in figure 9.26, we can observe that the average Voice delay each second, for long periods, was more stable in this simulation, compared to the delay experienced in the previous simulation.

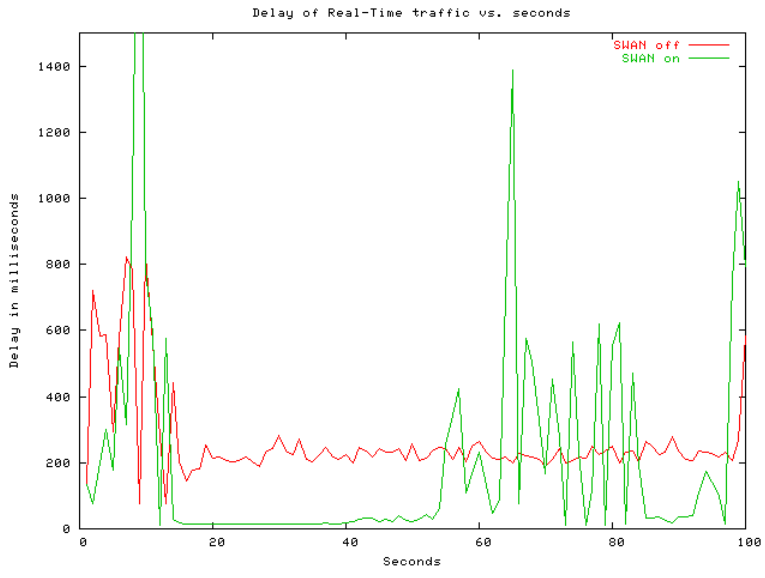



**Figure 9.26: Average delay of Voice streams**

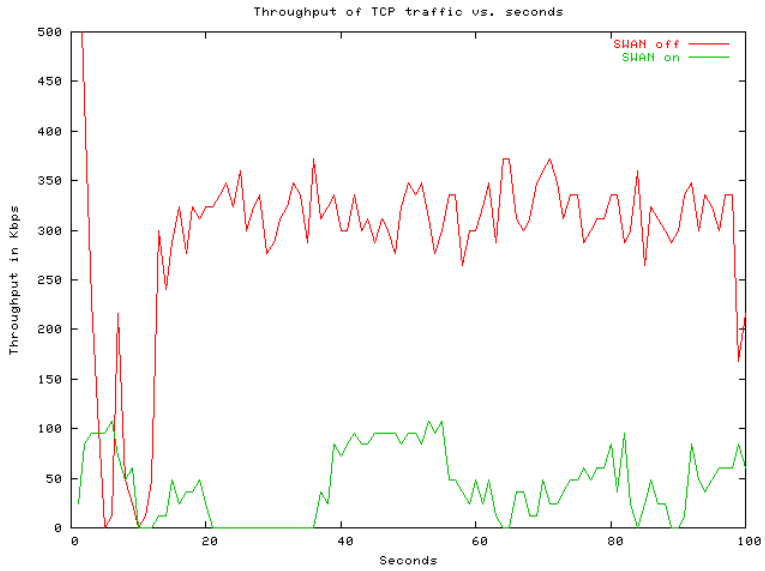



**Figure 9.27: Average TCP throughput**

The average video traffic delay without the SWAN mechanism was highly unstable in the first 15 seconds of the simulation period. In this interval it oscillates between 100 and 800 milliseconds. The average delay of the first packet reached only 82 milliseconds. After that it stayed around 250 milliseconds throughout the simulation period, except for the very last seconds where the delay became excessive and reached almost 600 milliseconds. The highest delay of one single packet without the SWAN mechanism was 1771 milliseconds in this simulation.

With the SWAN mechanism, the delay was also very unstable in the first 15 seconds of the simulation. The delay of the first packet was 190 seconds, and for a very few seconds of this 15 seconds interval, the delay reached an extremely high peak of 7756 milliseconds. For the next 35-40 seconds, the average delay stayed at a very low level at less than 50 milliseconds. Then it started to bounds at a high frequency again. It oscillates from very low, to an extremely high delay, throughout most of the simulation period. With voice traffic the average delay of the 100 seconds simulation period was much lower than in the analogous simulation with video traffic. Without the SWAN mechanism this average delay was 261 milliseconds and even lower with the SWAN mechanism; 217 milliseconds. This is shown in Table 9.2. Even though this is still above the most critical limit of delay for Real-Time streaming, this looks more promising, especially in the stable periods.

| | First packet delay | Max delay | Average delay | TCP throughput |
|---|---|---|---|---|
| SWAN off | 82.20 ms | 1771.47 ms | 261.12 ms | 29460 Kb |
| SWAN on | 190.16 ms | 7456.00 ms | 216.76 ms | 4296 Kb |

**Figure 9.28: Measurement from MH2, simulation 2**

The lower average delay with the SWAN mechanism came with an unreasonably high price to the TCP throughput, see figure 9.27. In these simulations, the TCP throughput was at a pretty fair level without the SWAN mechanism. The total amount of 29460 Kb FTP traffic was received, which gave an average TCP throughput on 295 Kbps. With the SWAN mechanism, only 4296 Kb FTP traffic was transferred in the simulation period, and therefore the average TCP throughput was down to only 43 Kbps.

In the **third** and **forth** simulations of MH2, with video and voice traffic respectively, the Real-Time traffic went in both directions, between the two pair of transmitting/receiving nodes. The FTP session was also crossing the Real-Times flows traffic paths in these simulations. Due to the analogy of these simulations with video and voice traffic, only the results from the simulations with video traffic are present in this chapter.

The average delay of the four video flows, measured once every second, can be viewed in figure 9.28. The average delay was not only highly unstable, but also at an unusually high level, way above the critical limit of 400 milliseconds stated by ITU-T. Both with and without the SWAN mechanism, the average video delay ranged from almost 0 milliseconds in some few periods, up to more than 5000 milliseconds at the highest peaks.

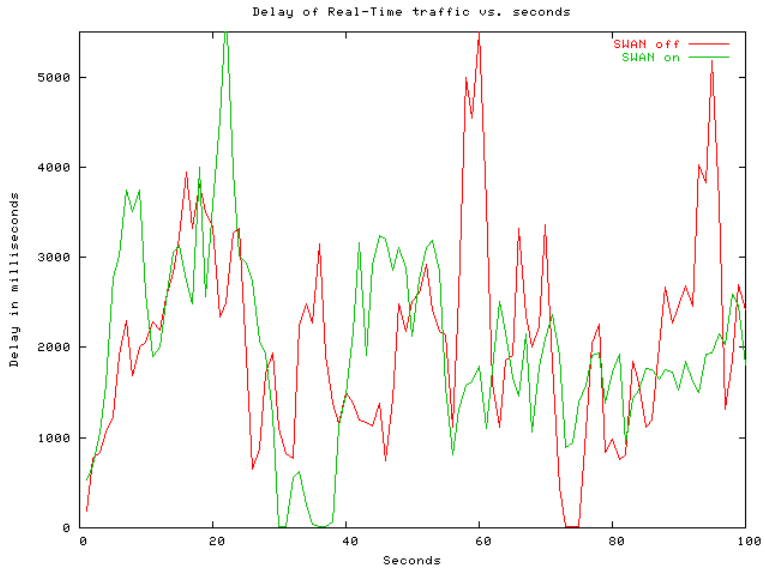



**Figure 9.29: Average delay of Video streams**

In this traffic scenario, the result from the simulation with the SWAN mechanism was slightly better than the one without any QoS mechanisms, except from the start up delay. With the SWAN mechanism, the delay of the first packet was 20 times higher than the simulation without the SWAN mechanism. The plot of the average video delay with the SWAN mechanism is also climbing at a faster rate in the very first second of the simulation, than what is the case without the SWAN mechanism. The maximum delay for one

single video CBR packet in the simulation without SWAN was 12300 milliseconds. With the SWAN mechanism, this maximum delay was also extremely high, but was here reduced to 10285 milliseconds.

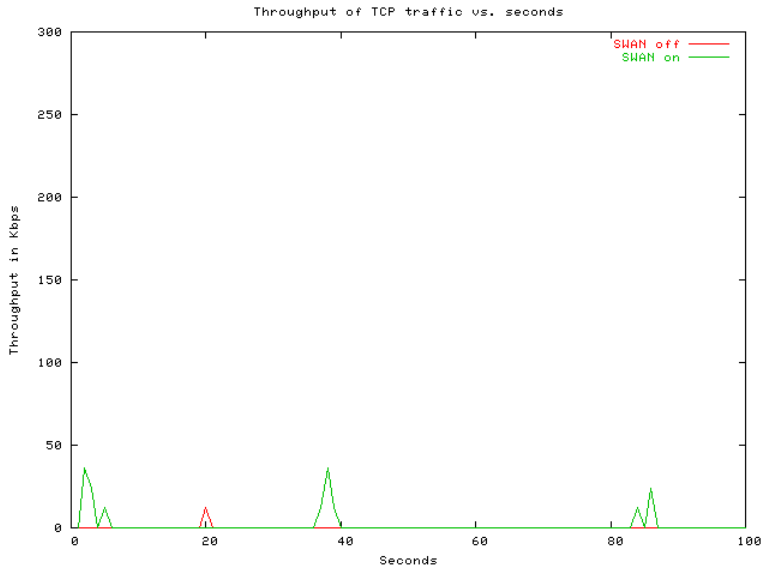



**Figure 9.30: Average TCP throughput**

The average video delay throughout the simulation period of 100 seconds was; 1955 milliseconds without the SWAN mechanism, and 1824 milliseconds with the SWAN mechanism. The average delay is shown in Table 9.3. This average delay is way beyond any limits of being able to support Real-Time streaming.

| | First packet delay | Max delay | Average delay | TCP throughput |
|---|---|---|---|---|
| SWAN off | 21.49 ms | 12299.97 ms | 1954.79 ms | 12 Kb |
| SWAN on | 425.14 ms | 10185.10 ms | 1823.94 ms | 168 Kb |

**Table 9.2: Measurement from MH 2, simulation 3**

In addition to this bad result for the experienced delay, the simulations of this scenario also show terrible results concerning the TCP throughput (see figure 9.29). The total TCP throughput of the simulation period of 100 seconds was only 12 Kb without the SWAN mechanism. Even though the total TCP throughput was 14 times higher in the simulation with the SWAN mechanism it is still too low to be at any acceptable level for the throughput of the Best-Effort service class.

Both the **fifth** and **sixth** traffic scenario of the simulations of MH2 was combinations of the previous simulations of MH2. In these simulations, both video and voice traffic was present simultaneously. The fifth simulation had a

one way transmission of Real-Time traffic. Here the traffic paths of the Real-Time traffic were from the left to the right side of the simulation area, presented in Figure 8.4. In the sixth and last simulation of MH2, the traffic paths of the Real-Time flows where two ways and the video and voice traffic was flowing in both directions between the two diagonal node pairs. Hence, the total number of flows, include the FTP session, was 9 flows. Due to the analogy of the fifth and sixth simulation, only the sixth traffic scenario is present in this chapter. The main difference of the result from the simulations of these traffic scenarios was that the sixth traffic scenario had worse performance than the fifth, due to a more dense traffic situation.

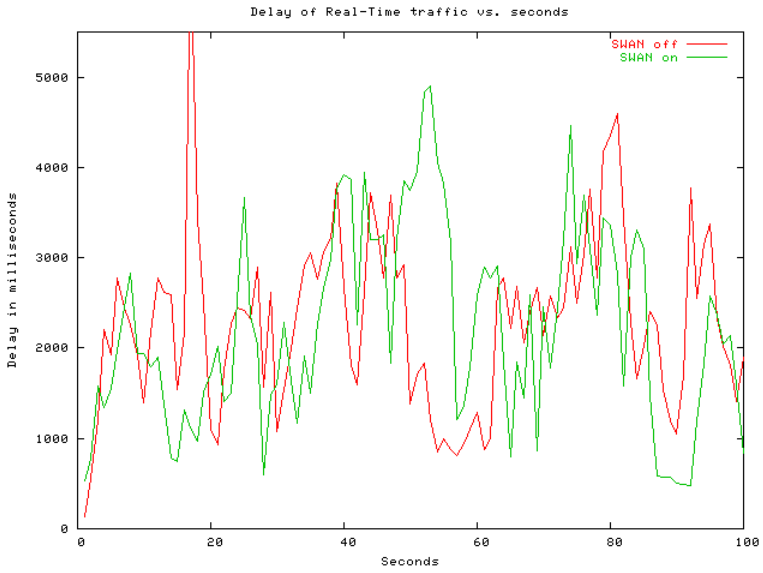



**Figure 9.31: Average delay - Real-Time traffic**

From figure 9.34 we can see that the average delay of the Real-Time traffic for the 8 RT flows. Not only was the Real-Time delay much higher, but also much more unstable than in the previous simulations. It is hard to believe that any RT traffic streams can work properly in such a heavily strained network.

In these simulations, the differences between the simulation with the SWAN mechanism, and the one without SWAN, was not of any convenient matter to make any difference to the user's experience of the Quality of Service. Though the performance was not exactly the same, but the introduction of the SWAN mechanism did hardly made any difference on improving the network performance.

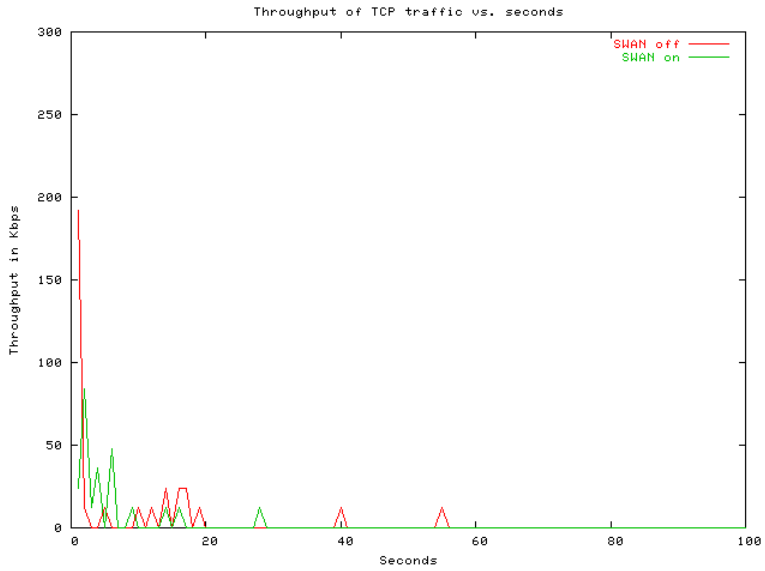



**Figure 9.32: Average TCP throughput**

The average Real-Time traffic delay for the total simulation period, experienced while the SWAN mechanism not was running, was 2169 milliseconds. With the SWAN mechanism the same delay was lower, but still, as high as 1995 milliseconds. The highest maximum delay experienced by a single data packet in these simulations was highest with the SWAN mechanism running; 12740 milliseconds. Without SWAN, this maximum delay for a single data packet reached 11538 milliseconds. Just as experienced many times in previous simulations, the startup delay was very high using SWAN. The delay of the first packet was now 6 times higher in the simulation running SWAN, than in the one without SWAN.

| | First packet delay | Max delay | Average delay | TCP throughput |
|---|---|---|---|---|
| SWAN off | 70.50 ms | 11537.62 ms | 2168.78 ms | 348 Kb |
| SWAN on | 426.23 ms | 12739.76 ms | 1995.25 ms | 252 Kb |

**Table 9.3: Measurement from MH2, simulation 3**

Also as experienced in the previous simulation, the TCP throughput in these simulations was completely terrible, see figure 9.35. The total TCP traffic transferred in the simulation without SWAN was only 348 Kb, and even lower in the one using SWAN; 252 Kb. Hence, this gave a TCP throughput on only 3.5 Kbps without SWAN, and 2.5 Kbps in the simulation running SWAN.

### 9.2.3 Summary of simulations and test results

From the simulations of SSC1 I found out that the total TCP throughput was lower in mine simulations, analogous to the one present in [3]. During these simulations, I also experienced that the TCP throughput was more unstable, when the SWAN mechanism was used. As expected, and also present in [3], the delay experienced by voice or video traffic is mostly lower simulations where the SWAN mechanism is in use. The overall utilization of the available throughput is also inferior in the simulations with the SWAN mechanism. In the benefit of the SWAN mechanism, is the low end-to-end delay for Real-Time traffic. With the SWAN mechanism, and especially in a network with many simultaneous competing traffic flows, the delay is much lover than the delay experienced without any QoS mechanism. The drawback is that the average end-to-end delay still is too high to support Real-Time traffic properly. Another major discovery in the simulation of SSC1 is the gap between the results in [3] and my own result. The result presented in [3] is most likely not the end-to-end delay, but the delay experienced at the MAC layer.

The FTP throughput was lower in the simulations of SSC2 than in SSC1. This could imply that the SWAN mechanism is not so good in network scenarios where the ratio of RT traffic on each node is high. It could also imply that it is more difficult to utilize the wireless channel when the RT traffic distributed on each node is denser. In the simulations of SSC2, the experienced RT delay was lower with SWAN than without SWAN. The drawback in these simulations was that the experienced RT delay with the SWAN mechanism was much higher than in the analogous simulation in SSC1. The result from SSC1 and SSC2 could also imply that the SWAN mechanism is working much better when the number of FTP background traffic is high. The simulations of SSC2 shows us that the network performance due to RT traffic is better with the present of the SWAN mechanism, but the performance is not good enough to meet the requirement from ITU-T [16]. Further, the simulations of SSC2 confirm that the delay present in the SWAN articles is not the end-to-end RT delay, but the RT delay experienced at the MAC layer.

The multi-hop simulations in MH1 show that the TCP throughput is much lower than in the analogous simulations where all the nodes in the network share the same wireless channel, e.g., SSC1. The simulations of MH1 also showed that with the SWAN mechanism, a lower delay was experienced compared to simulations where the SWAN mechanism not was used. Still, the major drawback of these results is that the experienced delay in a multi-hop scenario such as MH1, the delay is too high for Real-Time applications. Another experience of the simulations of MH1 was that the average RT delay was increased, when the number of FTP background traffic flows was increased. One important discovery from the simulations of MH1 was the different performance characteristics for voice and video traffic. The different characteristics of video and voice traffic emphasize that they should be treated differently in a computer network. This is not the case in the today's SWAN model.

The simulations of the multi-hop scenario labeled as MH2, the introduction of the SWAN mechanism does not seem to have any significant impact on

improving the QoS. In the simulations of MH2, SWAN gave a better result in the simulation of voice traffic than in the simulations with video traffic. With voice traffic, the delay was not only lower, but the delay was also for long periods much more stable than it was with video traffic. Thus, SWAN's effort to handle voice traffic better than a network without any QoS mechanism came at a high price to the TCP traffic; the average throughput was 7 times higher without the SWAN mechanism. The introductions of two-way traffic paths for the Real-Time traffic in the simulations of MH2 raised the average delay 5-10 times compared to the previous simulation of MH2, which only had one-way paths for the RT traffic. Even though the delay was very high both with and without the SWAN mechanism, SWAN seems to better handle the situations with an increased number of RT flows, and where RT traffic is going in both directions between the affected nodes. In all the simulations of MH2, the start-up delay when using the SWAN mechanism is always very high. Also, the presents of the SWAN mechanism seems to lower the delay to some extents, but not enough, so it will be of any significant matter from the user's perspective.

# 10 Conclusion

This chapter will conclude the work with this thesis, and presents possible for a future QoS mechanism based on the SWAN network model.

## *10.1 Concluding remarks*

The increased popularity of wireless networks and the widespread usage of multimedia applications call for strictly management of the time variable available wireless channel. State of the art within wireless ad hoc network is not an efficient utilization. Multimedia applications have to rely upon Best-Effort service, which basically is no Quality of Service at all. A supplementary QoS mechanism is most likely to be needed to achieve the goals of reliable multimedia applications in Ad Hoc networks. The SWAN network model is one step towards such a goal.

From the simulations studies I gained knowledge about many of the problems concerning Quality of Service in Ad Hoc network and particularly the problems related to the implementation of the SWAN network model. In this section I will present the most important issues in this context.

- In my early simulations I found problems with the performance when a source node of Real-Time traffic, was given an extra task of an additional session of data transfer with an external node. This could be a problem related to the network simulator used, or it could be a problem in the real world. If the latter is the case, this is a serious problem. Then, action has to be taken upon exhaustive testing and solving this case.
- In most of my simulations the end-to-end delay experienced while the SWAN mechanism was used to ensure the Quality of Service was lower than when no QoS mechanism was used. However, the difference in the experienced delay with or without the SWAN mechanism was not of such significance magnitude that the Quality of Service was considerably improved. Another major drawback was that most of the end-to-end delay measured in the simulations was beyond the critical limit of end-to-end delay of one way transmission for any data applications stated by ITU in [16].
- My suspicions regarding the delay present in the SWAN articles [1] and [3] not being the end-to-end delay, but the delay measured at the MAC layer was correct. After several simulations I found a huge gap between my end-to-end delay and the delay presented in those articles. After sending several e-mails to the people behind the SWAN model, this was confirmed.
- The simulations also revealed that when a Best-Effort traffic session was initiated without any measurement of the MAC delay, the SWAN mechanism in the transmitting node misconceives the situation and believes it has more available bandwidth than it actually has. This issue needs more exhaustive testing and needs to be addressed if it turns out to be a major problem.
- Another drawback of the SWAN implementation discovered in the simulations is the extremely high “start-up” delay experienced in the

very first seconds of a simultaneous initiation of Real-Time traffic. This is due to the probe mechanism in SWAN which needs feedback from the probe message before a node is admitted to transmit such data packet.

## *10.2 Further Studies*

This section presents areas which are open for more research and needs to be addressed in a successful Quality of Service model for the future. First some issues discovered during the simulations studies in this thesis are presented. Second, new proposals to a future QoS model for Ad-Hoc networks gained in theoretical analysis of the SWAN network model are presented.

- The issue concerning the discovery of high rise in the delay, when a RT source is involved in transmission of additional traffic certainly needs further testing either in ns-2 or in a lab network.
- It is of great importance to a Real-Time application of keeping the end-to-end delay below 150 milliseconds. Today, this situation is simply not good enough, and more research is needed to gain additional decrease in the delay and also in having a more stable delay over time.
- The simulation studies also revealed there could be situations where nodes misconceive the available bandwidth due to erroneous measurement of the Mac delay. This needs more testing, and certainly needs to be fixed in the implementation if it turns out to be a serious problem.
- In times of an emerging congestion situation, any computer network will have advantage of a congestion avoidance mechanism, which informs the transmitting node about the upcoming difficulties in the network. Proper action for a transmitting node could be to lower the transfer rate, and hence lower the quality of the multimedia streams. In the resent SWAN model there is no such mechanism, but in a new QoS model for Ad Hoc networks, a congestion avoidance mechanism like this should be incorporated.
- In any computer network a great variety of applications and hence a great variety in the types of data packet using the network is common. These different types of data packets have different characteristics and also different requirements. Some packet may also be more important than others. To satisfy these requirements, an advanced scheduling mechanism which differentiates among the packet types is needed. The resent SWAN model contains only a simple scheduling mechanism which does not fully cover this requirement.
- With the recent year's successful work on increasing the bandwidth capacity in wireless networks, the distance to the wired counterpart have been diminished. It is also believed that IEEE has plans of increasing the bandwidth of the 802.11 standard to 308 Kbps within this year. This bandwidth enhancement could make the need for an admission controller superfluous, if enough resources will always be available. A removal of the Admission control mechanism in SWAN will probably eliminate most of the start-up delay experienced with SWAN.

- A future QoS model must be able to dissolve congestion in a rapid manner. In the SWAN mechanism, measurement shows that it took 4-6 seconds before the Real-Time rate to reach below the Admission control rate. This may be to slow when some applications are delay sensitive in order of milliseconds. Fast congestion recovery has to be addressed in future implementations.
- The addressing of “false admission” is also mention as a problem in the SWAN mechanism. I have suggested an approach where all nodes are informed about congested network situations. In this approach, there is no need for an intelligent mechanism in the network interior, because this task could easily be taken care of by the transmitting nodes, without any large increase of the complexity.

The popularity of wireless network shows that this technology is here to stay. Only the future will show the limitations. However, more effort has to be accomplished in many fields, in order to have a more complete solution. Quality of Service is definitely one area that needs more research and development to satisfy the many requirements. The Comet’s groups Stateless Wireless Ad hoc Network model is one good proposal, but not a complete model. However, it is a giant leap in the right direction.

# 11 REFERENCES

**[1]** G.-S. Ahn, A. T. Campbell, Andras Veres and Li-Hsiang Sun, "Supporting Service Differentiation for Real-Time and Best Effort Traffic in Stateless Wireless Ad Hoc Networks (SWAN)", IEEE Transactions on Mobile Computing, September 2002.

**[2]** A. Veres, A.T. Campbell, M. Barry and L-H. Sun, "Supporting Service Differentiation in Wireless Packet Networks Using Distributed Control", *IEEE Journal of Selected Areas in Communications*, Special Issue on Mobility and Resource Management in Next-Generation Wireless Systems, Vol. 19, No. 10, pp. 2094-2104, October 2001.

**[3]** G.-S. Ahn, A. T. Campbell, Andras Veres and Li-Hsiang Sun, "SWAN: Service Differentiation in Stateless Wireless Ad Hoc Networks", Proc. IEEE INFOCOM'2002, New York, New York, June 2002.

**[4]** Gahng-Seop Ahn, Andrew T. Campbell, Andras Veres and Li-Hsiang Sun, "SWAN", draft-ahn-swan-manet-00.txt, *Work in Progress*, October 2002.

**[5]** Crow, B.P., Widjaja, I, Kim, J.G., and Sakai, P.T.,: "IEEE 802.11 Wireless Local Area Networks", IEEE Commun. Magazine, vol. 35, pp. 116-126, Sept.1997.

**[6]** Mohammad Ilyas: "The Handbook of Ad Hoc Wireless Network", CRC Press.

**[7]** Heegard, C., Coffey, J.T., Gummadi, S., Murphy, P.A., Provencio, R., Rossin, E.J., Schrum, S., and Shoemaker, M.B.: "High-PerformanceWireless Ethernet", IEEE Commun. Magazine, vol. 39, pp. 64-73, Now. 2001.

**[8]** Matthew S. Gast: "802.11 Wireless Networks", O'Reilly

**[9]** Upkar Varshney: "The Status and Future of 802.11 Based WLANs"

**[10]** P. Berthou, T. Gayraud, O. Alphand, C. Prudhommeauz, M. Diaz: "A Multimedia Architecture for 802.11b networks" *WCNC 2003 - IEEE Wireless Communications and Networking Conference,* vol. 4, no. 1, Mar. 2003 pp. 1742-1747

**[11]** P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum, L. Viennot: "Optimized Link State Routing Protocol for Ad Hoc Networks" IEEE INMIC, 2001. Hipercom Project, INRIA Rocquencourt.

**[12]** Ying Ge, Thomas Kunz, Louise Lamont: "Quality of Service Routing in Ad-Hoc Networks Using OLSR" Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03), Track 9, Volume 9.

**[13]** Daqing Gu and Jinyun Zhang: "QoS Enhancement in IEEE802.11 Wireless Local Area Networks", *IEEE Communications Magazine*, pp. 120 – 124, June 2003.

**[14]** Zhigang Wang, MengChu Zhou and Nirwan Ansari: “Ad-hoc Robot Wireless Communication” Systems, Man and Cybernetics, 2003. IEEE International Conference**,** 5-8 Oct. 2003, Volume: 4, On page(s): 4045- 4050 vol.4

**[15]** A. Roche, C. B Westphall and Graf von Mecklenburg: “Quality of Service for Ad Hoc Network” XII International Conference of the Chilean Computer Science Society (SCCC'02**)** , 11 06 - 11, 2002, Copiapó, Atacama, CHILE

**[16]** International Telecommunication Union (ITU), ITU-T Recommendation G.114: “Series G: Transmission systems and media digital systems and networks”

**[17]** G. H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," IEEE Computer, vol. 27, no. 4, 1994

**[18]** D. Cavin et al., "On the accuracy of MANET simulators" Proc. ACM Workshop on Princ. Mobile Computing (POMC'02), Oct. 2002, pp. 38-43.

**[19]** The ns-2 home page: The Network Simulator - ns-2, [http://www.isi.edu/nsnam/ns/]

**[20]** Pawlikowski it al: **“**On credibility of simulation studies of telecommunication networks**”**. *IEEE Communications Magazine* 40

**[21]** J. L. Sobrinho and A. S. Krishnakumar: “Quality-of-Service in ad hoc Carrier Sense Multiple Access Wireless Networks”, IEEE JSAC, vol. 17, no. 8, Aug. 1999, pp.1353-1414.

**[22]** Stefan Mangold, Sunghyun Choi, Peter May, Ole Klein, Guido Hiertz, Lothar Stibor: “IEEE 802.11e Wireless LAN for Quality of Service”, in *Proc. Eur. Wireless'02*, vol. 1, Feb. 2002, pp. 32-39.

**[23]** http://security.zhwin.ch/ITG_WLAN_Theory.pdf “Wireless LAN”

**[24]** Youssef Iraqi and Raouf Boutaba: “Resource Management issues in Future Wireless Multimedia Networks”, The International Journal of High Speed Networking, volume 9 number3 pp. 231-260, 2000.

**[25]** Y. Murat Erten and Emrah Tomur: “A Layered Architecture for Corporate 802.11 Wireless Networks” Wireless Telecommunications Symposium, 2004 Publication Date: 14-15 May 2004, pp. 123- 128.

**[26]** F. Ferreri, M. Bernashi and L. Valcamonici: “Access points vulnerabilities to DoS attacks in 802.11 networks”, Proceedings of WCNC2004, IEEE Wireless Communications and Networking Conference, Atlanta (Georgia-U.S.), 2004.

**[27]** [http://moment.cs.ucsb.edu/AODV/aodv.html#Description]

**[28]** Common Wireless Ad Hoc Network Usage Scenarios - http://www.flarion.com/ans-research/Drafts/draft-irtf-yang-ans-scenarios-00.txt

**[29]** RFC 2501 - Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations.

**[30]** Harpreet S. Arora: "Towards Achieving QoS Guarantees in Mobile Ad Hoc Networks", Masters Thesis, Drexel University, Department of Computer Science, Philadelphia, PA, November 2003.

**[31]** Saurabh Jain and Dharma P. Agrawal: "Wireless community networks", IEEE Computer, Vol. 36, No. 8, August 2003, pp. 90-92.

**[32]** http://planetmath.org/encyclopedia/AdHoc.html

**[33]** http://www.wordiq.com/definition/List_of_Latin_phrases

**[34]** RFC 3917: "Requirements for IP Flow Information Export" (IPFIX)

**[35]** Policy Based Quality of Service - www.csd.uch.gr/~hy536/PB.pdf

**[36]** RFC 1633: Integrated Services in the Internet Architecture: an Overview

**[37]** Constant Gbaguidi, Hans J. Einsiedler, Paul Hurley, Werner Almesberger, and Jean-Pierre Hubaux: "A Survey of Differentiated Services Proposals for the Internet", Technical report No. SSC/1998/020, April 1998.

**[38]** Xipeng Xiao and Lionel M. Ni: "Internet QoS: A Big Picture", *IEEE Network*, vol. 13, no. 2, Mar./Apr. 1999.

**[39]** RFC 2475: An Architecture for Differentiated Services

**[40]** RFC 3086: Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification

**[41]** RFC 2597: Assured Forwarding PHB Group

**[42]** RFC 2598: An Expedited Forwarding PHB

**[43]** H. Dong, I. D. Chakares, C. –H. Lin, A. Gersho, E. Belding-Royer, U. Madhow, J. D. Gibson: "Speech Coding for Mobile Ad Hoc Networks" *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, November 2003.

**[44]** RFC 3819: Advice for Internet Subnetwork Designers

**[45]** IETF Internet-draft: "Using Radius for PE-Based VPN Discovery", draft-heinanen-radius-pe-discovery-03.txt

**[46]** RFC 2026: The Internet Standards Process -- Revision 3

**[47]** Lee, S.B., Ahn, G.S., Campbell, A.T., “Improving UDP and TCP performance in Mobile Ad Hoc Networks with INSIGNIA”, June 2001, pp 156-165, IEEE Communication Magazine

**[48]** M. C. Domingo and D. Remondo: “A Cooperation Model between Ad Hoc Networks and Fixed Networks for Service Differentiation”, *Proceedings of the 4th International IEEE Workshop on Wireless Local Networks (WLN)*. IEEE, p. 692-693.

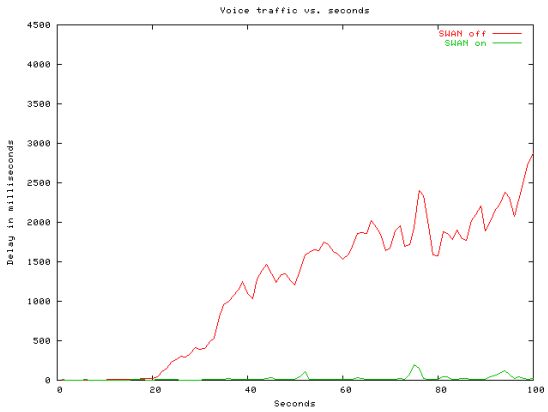
**[49]** Zeinalipour-Yazti Demetrios: “A Glance at Quality of Services in Mobile Ad-Hoc Networks”, Technical report, University of California-Riverside, 2001.

**[50]** S. Chakrabarti and A. Mishra, “QoS Issues in Ad Hoc Wireless Networks”, IEEE Communications Magazine, vol. 39, no. 2, pp. 142-148, February 2001.

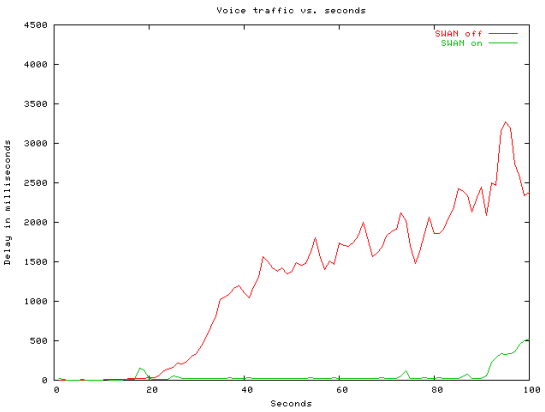
**[51]** Andrew S. Tanenbaum: “Computer Networks”, 4th edition. Pearson Education, Inc.

**[52]** Janusz Gozdecki, Andrezej Jajszczyk, and Rafal Stankiewitz, “Quality of Service Terminology in IP Networks”, IEEE Communications Magazine, pp 153-159, March 2003.

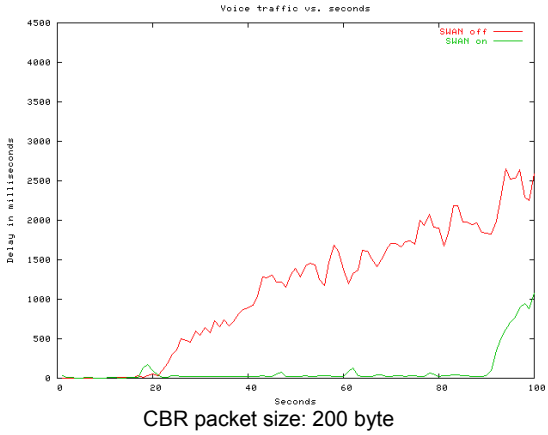
# 12 Appendix A



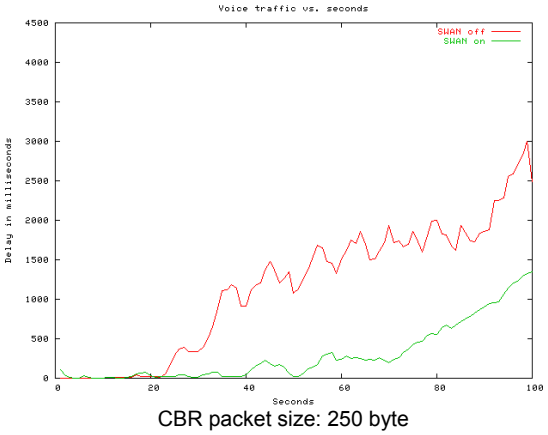

CBR packet size: 80 byte



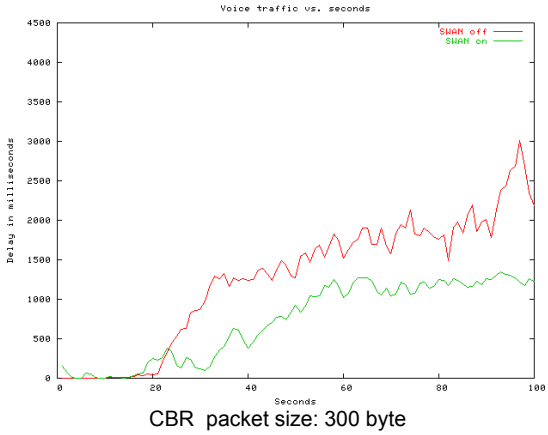

CBR packet size: 150 byte



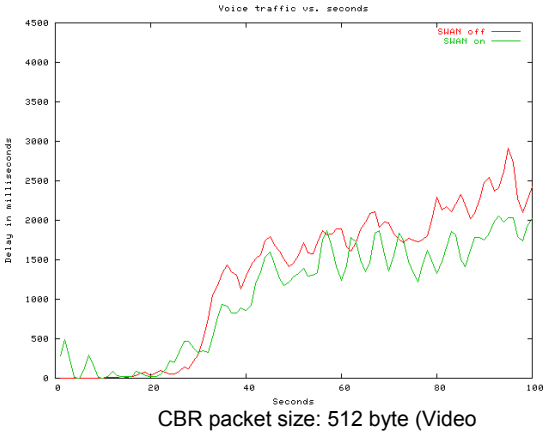

CBR packet size: 200 byte




CBR packet size: 250 byte




CBR packet size: 300 byte




CBR packet size: 512 byte (Video

# 13 Appendix B

| Number of TCP flows | 2 | 4 | 6 | 8 | 10 | 12 |
|---|---|---|---|---|---|---|
| SWAN off | 609 Kbps | 425 Kbps | 295 Kbps | 232 Kbps | 210 Kbps | 128 Kbps |
| SWAN on | 388 Kbps | 237 Kbps | 176 Kbps | 145 Kbps | 121 Kbps | 114 Kbps |

**Table SSC1: Average TCP throughput of each node**

| Number of TCP flows | 2 | 4 | 6 | 8 | 10 | 12 |
|---|---|---|---|---|---|---|
| SWAN off | 561 Kbps | 397 Kbps | 259 Kbps | 199 Kbps | 160 Kbps | 133 Kbps |
| SWAN on | 192 Kbps | 228 Kbps | 124 Kbps | 104 Kbps | 72 Kbps | 61 Kbps |

**Table SSC2: Average TCP throughput of each node**

| Number of TCP flows | 0 | 2 | 4 | 6 | 8 | 10 | 12 |
|---|---|---|---|---|---|---|---|
| SWAN off | 2.25 ms | 818.15 ms | 1562.96 ms | 2003.96 ms | 3357.00 ms | 2665.89 ms | 2893.88 ms |
| SWAN on | 51.28 ms | 452.57 ms | 544.75 ms | 631.13 ms | 638.27 ms | 672.49 ms | 680.46 ms |

**Table SSC1: Average Real-Time delay**

| Number of TCP flows | 0 | 2 | 4 | 6 | 8 | 10 | 12 |
|---|---|---|---|---|---|---|---|
| SWAN off | 2.19 ms | 446.16 ms | 717.89 ms | 693.93 ms | 717.28 ms | 715.37 ms | 721.12 ms |
| SWAN on | 8.94 ms | 734.73 ms | 607.75 ms | 554.17 ms | 639.76 ms | 593.14 ms | 590.26 ms |

**Table SSC2: Average Real-Time delay**

# 14 Appendix C

## Scripting files for SSC2 in ns-2:

```
# ======================================================================================
# Define options
# ======================================================================================
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)           Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)          Phy/WirelessPhy            ;# network interface type
set val(mac)            Mac/802_11                 ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)             LL                         ;# link layer type
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         50                         ;# max packet in ifq
set opt(seed)             0.0                       ;# don't know what this is ???
set val(x)              1000                        ;# x-coord of simulation field
set val(y)              800                        ;# y-coord of simulation field
set val(rp)             AODV                       ;# Routing protocol
set val(energymodel)    EnergyModel                ;#??????????????????????????????
set val(initialenergy)  100                        ;# Initial energy in Joules
set val(progress)       4                          ;# progress markers
set val(nn)             20                         ;# number of mobilenodes
set val(nodes)          "nodes20_1000x800_wireless.tcl"
set val(traffic)        "traffic20_1000x800_wireless.tcl"
set val(stop)           100.0                      ;# stop simulation at this time
# ======================================================================================
# options for SWAN module
# ======================================================================================
set opt(swan_rc)        "ON"            ;# rate controller ON/OFF
set opt(swan_ac)        "ON"            ;# admission controller ON/OFF
set opt(dir)            "result/test"   ;# result directory
set opt(band)           "100Kb"         ;# initial rate
set opt(ssthresh)       "1Mb"           ;# slow start threshold
set opt(segment)        "50Kb"          ;# increment segment (c)
set opt(mdrate)         "50"            ;# decrement rate (r)
set opt(gap)            "1.2"           ;# gap control (g)
set opt(minband)        "100kb"         ;# minimum rate
set opt(acrate)         "2000Kb"        ;# admission control rate
set opt(thrate)         "4000Kb"        ;# threshold rate
# ======================================================================================
# aditional SWAN stuff
# ======================================================================================
set AgentTrace                  OFF
set RouterTrace                 OFF
set MacTrace                    OFF

LL set mindelay_          50us
LL set delay_                   25us    ;# link-level overhead
LL set bandwidth_         0       ;# not used
LL set off_prune_         0       ;# not used
LL set off_CtrMcast_            0        ;# not used

Agent/Null set sport_           0
Agent/Null set dport_           0

Agent/CBR set sport_            0
Agent/CBR set dport_            0

Agent/TCPSink set sport_  0
Agent/TCPSink set dport_  0

Agent/TCP set sport_            0
Agent/TCP set dport_            0
Agent/TCP set packetSize_ 1460

Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1
# ======================================================================================

#############################################################################
# unity gain, omni-directional antennas                                     #
# set up the antennas to be centered in the node and 1.5 meters above it #
#############################################################################
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0
#############################################################################
# Initialize the SharedMedia interface with parameters to make           #
# it work like the 914MHz Lucent WaveLAN DSSS radio interface            #
#############################################################################
Phy/WirelessPhy set CPThresh_ 10.0
```

```
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set Rb_ 11*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0
Phy/WirelessPhy set bandwidth_ 11e6
#======================================================================
# Initialize the 802.11 MAC                                           #
#======================================================================
Mac set bandwidth_ 11e6


#===================================================================================================
# ================================================================================================#
# Main Program                                                                                    #
# ================================================================================================ #
#===================================================================================================

#===================================================================================================
# Initialize Global Variables                                                                     #
#===================================================================================================

# Creating an instance of the simulator
set ns_          [new Simulator]

# set up topography object that keep track of mobile nodes within the topologica boundary
set topo       [new Topography]

#set god_ [new God]

#makedir $opt(dir)
#===================================================================================================
# Settup trace support                                                                            #
#===================================================================================================

# open a files for writing that is going to be used for the nam and trace data
#set tracefd     [open trace40_150x150_wireless.tr w]
#set namtrace    [open out40_150x150_wireless.nam w]

set tracefd     [open trace20_1000x800_wireless.tr w]
set namtrace    [open out20_1000x800_wireless.nam w]

# Provide the topography object with x and y co-ordinates of the boundary
$topo load_flatgrid $val(x) $val(y)

$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

#===================================================================================================
# Create God object (General Operations Director)                                                 #
# used to store global information about the state of                                             #
# the environment, network or nodes                                                               #
#===================================================================================================
create-god $val(nn)

#===================================================================================================
# Create channel #1                                                                               #
#===================================================================================================
set chan_1_ [new $val(chan)]

#===================================================================================================
#  Create the specified number of mobilenodes [$val(nn)] and "attach" them                        #
#  to the channel.                                                                                #
#  Here two nodes are created : node(0) and node(1)                                               #
#===================================================================================================

#===================================================================================================
# configure node                                                                                  #
#===================================================================================================
$ns_ node-config -adhocRouting $val(rp) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                  -channel $chan_1_ \
                 -topoInstance $topo \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace ON \
                 -movementTrace OFF \
                -energyModel $val(energymodel) \
                 -initialEnergy $val(initialenergy)
```

```
#############################################################################################
# Create val(nn) mobile nodes                                                               #
#############################################################################################
for {set i 0} {$i < $val(nn) } {incr i} {
    set node_($i) [$ns_ node]
    $node_($i) random-motion 0             ;# disable random motion
}

#############################################################################################
# Set lossmonitor for the different nodes                                                   #
#############################################################################################
for {set i 0} {$i < $val(nn)} {incr i} {

    set x [expr $i % 2]

    if { $x > 0} {
        puts "putting lossmonitor on node $i"
        set loss_($i) [new Agent/LossMonitor]
    } else {
        # denne skal bort senere
        puts "putting lossmonitor on node $i"
        set loss_($i) [new Agent/LossMonitor]
    }
}

#############################################################################################
# Loading scenario file                                                                     #
#############################################################################################
puts "Loading nodes scenario file..."
source $val(nodes)
puts "Load complete..."

#############################################################################################
# Loading traffic file                                                                      #
#############################################################################################
puts "Loading connection pattern..."
source $val(traffic)
puts "Load complete..."

#############################################################################
# trigger shaper & utilization monitor every second                        #
#############################################################################

for {set i 0} {$i <= $val(stop)} {incr i 1} {
    for {set j 0} {$j < $val(nn) } {incr j} {
        $ns_ at $i "$node_($j) shape 1"
        $ns_ at $i "$node_($j) monitor c1"
    }
}

#############################################################################################
# Define node initial position(size) in nam                                                 #
#############################################################################################
for {set i 0} {$i < $val(nn)} {incr i} {
    # 20 defines the node size in nam, must adjust it according to your scenario
    # The function must be called after mobility model is defined
    $ns_ initial_node_pos $node_($i) 60
}
#############################################################################################
# Tell nodes when the simulation ends                                                       #
#############################################################################################
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).000000001 "$node_($i) reset";
}

#############################################################################################
# stoping procedure                                                                         #
#############################################################################################
proc stop {} {
    global ns_ tracefd namtrace th_put_vi th_put_vo th_put_ftp val recv_ftp
    $ns_ flush-trace

    close $tracefd
    close $namtrace

    exit 0
}
#############################################################################################
# simulation counter                                                                        #
#############################################################################################
for {set i 1} {$i <= $val(progress)} {incr i} {
    set t [expr $i * $val(stop) / ($val(progress) + 1)]
    $ns_ at $t "puts \"completed through $t secs...\""
}

#############################################################################################
# starting procedures                                                                       #
```

```
#############################################################################################
#$ns_ at 0.0 "record_thput"
#$ns_ at 100.0 "stop"
$ns_ at $val(stop) "stop"
$ns_ at $val(stop).00000001 "puts \"NS EXITING...\" ; $ns_ halt"

#############################################################################################
# start the simulation                                                                      #
#############################################################################################
puts "Starting Simulation..."
$ns_ run

#############################################################################################
#                                                                                           #
#                                       END OF THE SCRIPT                                   #
#                                                                                           #
#############################################################################################
```

```
#############################################################################
# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes           #
#############################################################################
set god_ [God instance]

#############################################################################
# Node initial position                                                     #
#############################################################################

# nodes 0-3
$node_(0) set X_ 0.0
$node_(0) set Y_ 0.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 0.0
$node_(1) set Y_ 200.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 0.0
$node_(2) set Y_ 400.0
$node_(2) set Z_ 0.0

$node_(3) set X_ 0.0
$node_(3) set Y_ 600.0
$node_(3) set Z_ 0.0

# nodes 4-7
$node_(4) set X_ 200.0
$node_(4) set Y_ 0.0
$node_(4) set Z_ 0.0

$node_(5) set X_ 200.0
$node_(5) set Y_ 200.0
$node_(5) set Z_ 0.0

$node_(6) set X_ 200.0
$node_(6) set Y_ 400.0
$node_(6) set Z_ 0.0

$node_(7) set X_ 200.0
$node_(7) set Y_ 600.0
$node_(7) set Z_ 0.0

# nodes 8-11
$node_(8) set X_ 400.0
$node_(8) set Y_ 0.0
$node_(8) set Z_ 0.0

$node_(9) set X_ 400.0
$node_(9) set Y_ 200.0
$node_(9) set Z_ 0.0

$node_(10) set X_ 400.0
$node_(10) set Y_ 400.0
$node_(10) set Z_ 0.0

$node_(11) set X_ 400.0
$node_(11) set Y_ 600.0
$node_(11) set Z_ 0.0

# nodes 12-15
$node_(12) set X_ 600.0
$node_(12) set Y_ 0.0
$node_(12) set Z_ 0.0

$node_(13) set X_ 600.0
$node_(13) set Y_ 200.0
$node_(13) set Z_ 0.0

$node_(14) set X_ 600.0
$node_(14) set Y_ 400.0
$node_(14) set Z_ 0.0

$node_(15) set X_ 600.0
$node_(15) set Y_ 600.0
$node_(15) set Z_ 0.0

# nodes 16-19
$node_(16) set X_ 800.0
$node_(16) set Y_ 0.0
$node_(16) set Z_ 0.0

$node_(17) set X_ 800.0
$node_(17) set Y_ 200.0
$node_(17) set Z_ 0.0

$node_(18) set X_ 800.0
```

```
$node_(18) set Y_ 400.0
$node_(18) set Z_ 0.0

$node_(19) set X_ 800.0
$node_(19) set Y_ 600.0
$node_(19) set Z_ 0.0


#############################################################################
# Node movement                                                             #
#############################################################################

#$ns_ at 5.0 "$node_(24) setdest 20.0 120.0 15.0"









# ======================================================================
# Traffic Model
# ======================================================================

set pareto1 [new RandomVariable/Pareto]
$pareto1 set avg_ 10
$pareto1 set shape 1.2

set pareto2 [new RandomVariable/Pareto]
$pareto2 set avg_ 10
$pareto2 set shape 1.2

set uniform [new RandomVariable/Uniform]
$uniform set min_ 0
$uniform set max_ 1

set udp_cnt 0
set tcp_cnt 0


proc create-video-connection { src dst stime} {
    global ns_ node_ val uniform udp_cnt loss_
    set j [expr $stime + [$uniform value] ]

    set udp_($udp_cnt) [new Agent/UDP]
    $ns_ attach-agent $node_($src) $udp_($udp_cnt)

    # set null_($udp_cnt) [new Agent/Null]
    # $ns_ attach-agent $node_($dst) $null_($udp_cnt)
    $ns_ attach-agent $node_($dst) $loss_($dst)

    set cbr_($udp_cnt) [new Application/Traffic/CBR]

    $cbr_($udp_cnt) set packetSize_ 512
    $cbr_($udp_cnt) set interval_ 0.02
    $cbr_($udp_cnt) set random_ 1
    $cbr_($udp_cnt) set maxpkts_ 15000
    $cbr_($udp_cnt) attach-agent $udp_($udp_cnt)

    #$ns_ connect $udp_($udp_cnt) $null_($udp_cnt)
    $ns_ connect $udp_($udp_cnt) $loss_($dst)

    $ns_ at $j "$cbr_($udp_cnt) start"
    $ns_ at 1000 "$cbr_($udp_cnt) stop"

    puts "at $j | SRC($src) | DST($dst) | Video($udp_cnt) | lossMonitor: loss($dst)"
    incr udp_cnt
}

proc create-voice-connection { src dst stime} {
    global ns_ node_ val uniform udp_cnt loss_
    set j [expr $stime + [$uniform value] ]

    # This creates an instance of the UDP agent
    set udp_($udp_cnt) [new Agent/UDP]
```

```
    # This is a common command used to attach any <agent> to a given <node>
    $ns_ attach-agent $node_($src) $udp_($udp_cnt)

    # Creates an instance of the Null agent
    set null_($udp_cnt) [new Agent/Null]

    # Put receiving node on the Null agent
    $ns_ attach-agent $node_($dst) $null_($udp_cnt)

    # Put the receiving node on the LossMonitor agent
    $ns_ attach-agent $node_($dst) $loss_($dst)

    # setup a CBR traffic flow for the udp agent
    set cbr_($udp_cnt) [new Application/Traffic/CBR]

    # Constant size of packets generated
    $cbr_($udp_cnt) set packetSize_ 80

    # Interval between packets
    $cbr_($udp_cnt) set interval_ 0.02

    # Whether or not to introduce random noice in the scheduled departure times
    $cbr_($udp_cnt) set random_ 1

    # Maximum number of packets to send
    $cbr_($udp_cnt) set maxpkts_ 15000

    $cbr_($udp_cnt) attach-agent $udp_($udp_cnt)

    # setup an end-to-end connection between two agents (at the transport layer)
    $ns_ connect $udp_($udp_cnt) $null_($udp_cnt)
    $ns_ connect $udp_($udp_cnt) $loss_($dst)

    # Causes the source to start generating packets at $i sec
    $ns_ at $j "$cbr_($udp_cnt) start"

    # Causes the source to stop generating packets at time 1000 sec
    $ns_ at 1000 "$cbr_($udp_cnt) stop"

    puts "at $j | SRC($src) | DST($dst) | Voice($udp_cnt) | lossMonitor: loss_($dst)"
    incr udp_cnt
}

proc create-ftp-connection { src dst stime} {
    global ns_ node_ val uniform tcp_cnt loss_
    set j [expr $stime + [$uniform value] ]

    # create sender agent
    set tcp_($tcp_cnt) [new Agent/TCP/Reno]

    # Put sender on node $node_($src)
    $ns_ attach-agent $node_($src) $tcp_($tcp_cnt)

    # Create receiver agent
    set null_($dst) [new Agent/TCPSink]

    # Put receiver on node $node_($dest)
    $ns_ attach-agent $node_($dst) $null_($dst)

    # Put the receiving node on the LossMonitor agent
    $ns_ attach-agent $node_($dst) $loss_($dst)

    # Create an FTP source "application"
    set ftp_($tcp_cnt) [new Application/FTP]

    # The size in bytes to use for all packets from this source
    $ftp_($tcp_cnt) set packetSize_ 512

    # The upper bound on the advertised window for the TCP connection
    $ftp_($tcp_cnt) set window_ 32

    # The initial size of the congestion window on slow-start
    $ftp_($tcp_cnt) set windowInit_ 16

    # The maximum number of packets generated by the source
    $ftp_($tcp_cnt) set maxpkts_ 1000000

    # Associate FTP with the TCP sender
    $ftp_($tcp_cnt) attach-agent $tcp_($tcp_cnt)

    # Astablish TCP connection
    $ns_ connect $tcp_($tcp_cnt) $null_($dst)

    # Arrange for FTP to start at time $j sec
    $ns_ at $j "$ftp_($tcp_cnt) start"

    # Arrange for FTP to stop at time 1000 sec
```

```
    $ns_ at 1000 "$ftp_($tcp_cnt) stop"

    puts "at $j |SRC($src) | DST($dst) | FTP($tcp_cnt) | lossMonitor: loss_($dst)"
    incr tcp_cnt
}

# Video
create-video-connection 0 19 0
create-video-connection 3 16 0
create-video-connection 19 0 0
create-video-connection 16 3 0

# Voice
create-voice-connection 0 19 0
create-voice-connection 3 16 0
create-voice-connection 19 0 0
create-voice-connection 16 3 0

# 2 TCP flows
create-ftp-connection 8 11 0
#create-ftp-connection 2 3 0
# 4 TCP flows
#create-ftp-connection 4 5 0
#create-ftp-connection 6 7 0
# 6 TCP flows
#create-ftp-connection 8 9 0
#create-ftp-connection 10 11 0
# 8 TCP flows
#create-ftp-connection 12 13 0
#create-ftp-connection 14 15 0
# 10 TCP flows
#create-ftp-connection 16 17 0
#create-ftp-connection 18 19 0
# 12 TCP flows
#create-ftp-connection 20 21 0
#create-ftp-connection 22 23 0
```

# 15 Appendix D

Python script files for analysis:

```
#!/usr/bin/env python

import sys, re, math, os, shutil

print "script to make plotfile for gnuplot"


# reading infile and outfile
try:
    infilename = sys.argv[1]
    outfilename = sys.argv[2]
    no_tcp_flows = sys.argv[3]
except:
    print 'Usage:', sys,argv[0], 'infile_tcp outfile_tcp
no_of_tcp_flows'
    sys.exit(1)


out_vi = 'vi_cbr_plott.out'
out_vo = 'vo_cbr_plott.out'


# open files
ifile = open( infilename, 'r') # r for reading
ofile = open(outfilename, 'w') # w for writing

vi_ofile = open(out_vi, 'w')
vo_ofile = open(out_vo, 'w')


# initaitions
counter = 0

vi_pac_interval = 0
vo_pac_interval = 0
# tcp
total_pac_interval = 0

total_thput = 0
vi_total = 0
vo_total = 0
case = "tmp1"




# kan putte pÃ¥ en for-lÃ¸kke her som gÃ¥r 6 ganger (2-4-6-8-10-12
tcp flows)

# read line by line:
for line in ifile:

    pattern1 =\
            r"r\s(\d+)\.\d+\s_(\d+)_\s.*"

    pattern2 = \
            r"r\s\d+\.\d+\s_\d+_\sAGT\s{2}---\s\d+\stcp\s(\d+)\s.*"
```

```
    pattern3 = \
            r"r\s\d+\.\d+\s_\d+_\sAGT\s{2}---\s\d+\scbr\s(\d+)\s.*"


    #pattern2 = \
    #        r"s\s\d+\.\d+\s_\d+_\sAGT\s{2}---\s\d+\stcp\s(\d+)\s.*"

    #pattern1 =\
    #         r"s\s(\d+)\.\d+\s.*"


    match1 = re.search(pattern1, line)

    if match1:
        time = float(match1.group(1))
        node = int(match1.group(2))

        if time > counter:
            counter = counter + 1

            total_in_Kbps = total_pac_interval * 8 / 1000
            total_pr_flow = total_in_Kbps / int(no_tcp_flows)

            vo_in_Kbps = vo_pac_interval * 8 / 1000
            vi_in_Kbps = vi_pac_interval * 8 / 1000

            #print "-----------------------------------------------
----------------------------------------------"
            #print "line: %s" % (line)
            #print "tid: %g" % (time)
            #print "%d total average throughput: %d Kbps | average
throughput for each flow: %d Kbps" %
(counter,total_in_Kbps,total_pr_flow)
            #print "voice byte: %d:" % (vo_pac_interval)
            #print "Voice: %d Kbps" % (vo_in_Kbps)
            #print "Video: %d Kbps" % (vi_in_Kbps)

            # write output to plott-file
            ofile.write('%d %d\n' % (counter,total_in_Kbps))
            vi_ofile.write('%d %d\n' % (counter,vi_in_Kbps))
            vo_ofile.write('%d %d\n' % (counter,vo_in_Kbps))

            total_pac_interval = 0
            vo_pac_interval = 0
            vi_pac_interval = 0

        match2 = re.search(pattern2, line)
        if match2:
            #time = float(match.group(1))
            packet = int(match2.group(1))
            # received packet are 20 bytes higher than the one sent
(bug in ns-2)
            packet = packet - 20

            # collect total throughput on tcp
            total_thput = total_thput + packet

            total_pac_interval = total_pac_interval + packet

        match3 = re.search(pattern3, line)
```

```
        if match3:
            #print "line: %s" % (line)
            #node = int(match3.group(1))
            cbr_packet = int(match3.group(1))
            #print "cpr:packet: %d" % (cbr_packet)
            cbr_packet = cbr_packet - 20

            if (node == 33 or node == 35 or node == 37 or node ==
39):
                # voice
                #print "VOICE node: %s | line: %s" % (node,line)
                vo_pac_interval = vo_pac_interval + cbr_packet
                vo_total = vo_total + cbr_packet
                #print "node%d: cbr_packet: %d voice" %
(node,cbr_packet)
            else:
                #print "VIDEO node: %s | line: %s" % (node,line)
                vi_pac_interval = vi_pac_interval + cbr_packet
                vi_total = vi_total + cbr_packet
                #print "node%d: cbr_packet: %d video" %
(node,cbr_packet)

            cbr_packet = 0

counter = counter + 1
total_in_Kbps = total_pac_interval * 8 / 1000
total_pr_flow = total_in_Kbps / int(no_tcp_flows)

vo_in_Kbps = vo_pac_interval * 8 / 1000
vi_in_Kbps = vi_pac_interval * 8 / 1000
#print "-----------------------------------------------------------------
-----------------------------------"
#print "%d total average throughput: %d Kbps | average throughput for
each flow: %d Kbps" % (counter,total_in_Kbps,total_pr_flow)

# write last interval output to plott-file
ofile.write('%d %d\n' % (counter,total_in_Kbps))
vi_ofile.write('%d %d\n' % (counter,vi_in_Kbps))
vo_ofile.write('%d %d\n' % (counter,vo_in_Kbps))

# get the throughput in Kbps
total_thput = total_thput * 8 / 1000 / (counter + 1)
vi_total = vi_total * 8 / 1000 / (counter + 1)
vo_total = vo_total * 8 / 1000 / (counter + 1)
print "-----------------------------------------------------------------
-----------------------------------"
print "total average TCP throughput: %g Kbps" % (total_thput)
print "total average Video throughput: %g Kbps" % (vi_total)
print "total average Voice throughput: %g Kbps" % (vo_total)
# get the average throughput on each flow
total_thput = total_thput / int(no_tcp_flows)
print "average TCP throughput for each flow: %g Kbps" % (total_thput)
print "-----------------------------------------------------------------
-----------------------------------"

ofile.close()
vi_ofile.close()
vo_ofile.close()

# run the outputfile in gnuplot
f = open(case + '.gnuplot', 'w')
```

```
f.write("""
set title 'Throughput (Kbps)';
set yrange [0:2000];
set term png small color;
set output 'MH1_on_th12.png';
""")

f.write("plot '%s' title 'TCP' with lines, '%s' title 'Video' with
lines, '%s' title 'Voice' with lines;\n" %
(outfilename,out_vi,out_vo))
#f.write("plot '%s' title 'TCP' with lines;\n" % (outfilename))
f.close()
cmd = "gnuplot -geometry 800x600 -persist " + case + ".gnuplot"
failure = os.system(cmd)
if failure:
    print "running gnuplot failed"; sys.exit(1)
```

```
#!/usr/bin/env python

import sys, re, math, os, shutil

print "script to make plotfile for gnuplot"

# reading infile and outfile
try:
    #infilename = sys.argv[1]
    no_nodes= sys.argv[1]
    seconds = sys.argv[2]
except:
    print 'Usage:', sys,argv[0], '<number of nodes> <seconds
simulated>'
    sys.exit(1)



SwanOff_infile = 'AGT_SWANoff_'+no_nodes+'_'
SwanOn_infile = 'AGT_SWANon_'+no_nodes+'_'
infile_end = '.tr'
ofile_end = '.out'

#s_list = []
#counter = 0
total_delay = 0


def put_in_list(dsec,seq):
    #s_list[seq] = dsec
    s_list.append(dsec)

def check_sendt(seq):
    return s_list[seq]


# loop for SWAN on/off
for i in range(0,2,1):
    if (i == 0):
        print "j = %d" % (i)
        infilename = SwanOff_infile
    if (i == 1):
        print "j = %d" % (i)
        infilename = SwanOn_infile

    no_flow = 0
    ofile = open(infilename + ofile_end, 'w')
    vi_ofile = open('vi_' + infilename + ofile_end, 'w')
    vo_ofile = open('vo_' + infilename + ofile_end, 'w')

    # loop for diffrent number of TCP flows
    for j in range(7):

        #no_flow = no_flow + 2
        no = no_flow

        s_list = []
        counter = 0
        total_delay = 0
        max_delay = 0
```

```
        vi_tot_delay = 0
        vo_tot_delay = 0
        vi_count = 0
        vo_count = 0

        ifile = open( infilename + str(no) + infile_end, 'r') # r for
reading

        # read line by line:
        for line in ifile:

            pattern0 = \
                     r"s\s(\d+\.\d+)\s_(\d+)_\sAGT\s{2}---
\s(\d+)\s.*"

            pattern1 = \
                     r"r\s(\d+\.\d+)\s_(\d+)_\sAGT\s{2}---
\s(\d+)\scbr\s(\d+)\s.*"

            match0 = re.search(pattern0, line)

            if match0:
                sendt = float(match0.group(1))
                s_node = int(match0.group(2))
                s_seq = int(match0.group(3))

                # call function put_in_list(s_dsec,seq)
                put_in_list(sendt,s_seq)

            match1 = re.search(pattern1, line)

            if match1:
                received = float(match1.group(1))
                r_node = int(match1.group(2))
                r_seq = int(match1.group(3))
                packet = int(match1.group(4))

                # call fuction check_sendt(r_dsec,seq)
                sendt_sec = check_sendt(r_seq)
                delay = received - sendt_sec
                msec_delay = delay * 1000

                if packet < 110:
                    #voice
                    vo_tot_delay = vo_tot_delay + msec_delay
                    vo_count = vo_count + 1
                else:
                    vi_tot_delay = vi_tot_delay + msec_delay
                    vi_count = vi_count + 1


                if msec_delay > max_delay:
                    max_delay = msec_delay

                    #if max_delay > 2700:
                        #print "new max delay: %f" % (max_delay)
                        #print "sendt: %f | received: %f | node: %d |
seq: %d | delay: %f | delay in millisec: %f msec" %
(sendt_sec,float(received),r_node,r_seq,delay,msec_delay)
                total_delay = total_delay + msec_delay
                counter = counter + 1
```

```
        avg_delay = total_delay / counter
        vi_avg_delay = vi_tot_delay / vi_count
        vo_avg_delay = vo_tot_delay / vo_count
        print "------------------------------------------------------
------------------------------------------------------------------
-"
        print "avg delay %s TCP flows: avg_delay: %f msec | max delay
%f | Video: %f | Voice: %f" %
(str(no),avg_delay,max_delay,vi_avg_delay,vo_avg_delay)

        ofile.write('%d %f\n' % (no_flow,avg_delay))
        vi_ofile.write('%d %f\n' % (no_flow,vi_avg_delay))
        vo_ofile.write('%d %f\n' % (no_flow,vo_avg_delay))
        no_flow = no_flow + 2
    ofile.close()
    vi_ofile.close()
    vo_ofile.close()

case = 'avg_rt_delay'
case_vivo = 'avg_vivo_delay'

# run the outputfile in gnuplot
f = open( case + '.gnuplot', 'w')
f.write("""
set title 'RT delay vs. number of TCP flows';
set yrange [0:1500];
set xrange [0:12];
set ylabel 'Delay in milliseconds';
set xlabel 'Number of TCP flows';
set term png small color;
set output 'avg_rt_delay.png';
""")

f.write("plot '%s' title 'SWAN off' with lines, '%s' title 'SWAN on'
with lines;\n" % (SwanOff_infile+ofile_end,SwanOn_infile+ofile_end))

f.close()
cmd = "gnuplot -geometry 800x600 -persist " + case + ".gnuplot"
failure = os.system(cmd)
if failure:
    print "running gnuplot failed"; sys.exit(1)

# second plot
vivo = open( case_vivo + '.gnuplot', 'w')
vivo.write("""
set title 'Video/Voice delay vs. number of TCP flows';
set yrange [0:1500];
set xrange [0:12];
set ylabel 'Delay in milliseconds';
set xlabel 'Number of TCP flows';
set term png small color;
set output 'avg_vivo_delay.png';
""")

vivo.write("plot '%s' title 'Video SWAN off' with lines, '%s' title
'Voice SWAN off' with lines, '%s' title 'Video SWAN on' with lines,
'%s' title 'Voice SWAN on' with lines;\n" %
('vi_'+SwanOff_infile+ofile_end,'vo_'+SwanOff_infile+ofile_end,'vi_'+
SwanOn_infile+ofile_end,'vo_'+SwanOn_infile+ofile_end))
```

```
vivo.close()
cmd2 = "gnuplot -geometry 800x600 -persist " + case_vivo + ".gnuplot"
failure = os.system(cmd2)
if failure:
    print "running gnuplot failed"; sys.exit(1)
```