A PROTOCOL STACK FOR ADAPTABLE AND SECURE COMMUNICATIONS

By

MADHUKAR ALLI

Master of Science in Computer Science Dept

Oklahoma State University

Stillwater, Oklahoma

2006

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May 2006

A PROTOCOL STACK FOR ADAPTABLE AND

SECURE COMMUNICATIONS

Thesis Approved:

Dr. JOHNSON P THOMAS

Thesis Advisor

Dr. JOHN P CHANDLER

Dr. DEBAO CHEN

Dr. GORDON EMSLIE

Dean of the Graduate College

ACKNOWLEDGEMENTS

At this point I would like to express my sincere appreciation to my advisor, Dr. Johnson P. Thomas for his continuous supervision, constructive guidance, inspiration, motivation throughout the course of my thesis work. I would also like to express my gratitude to Dr. Chandler, Dr. Debao Chen and Dr. GE Hedrick for their invaluable suggestions and support.

I thank Oklahoma State University for providing all the facilities needed for the research. I would like to express my sincere thanks to the System Managers and the Information Technology Division Team for their timely cooperation. I thank all my friends here at Oklahoma State University and elsewhere for their support and encouragement when I needed them the most.

I am ever thankful to my parents and my family for their continuous support and encouragement.

TABLE OF CONTENTS

Chapter **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

CHAPTER I

INTRODUCTION

This proposal presents a multiple protocol stack approach to the problem of meeting QoS and security requirements and utilizing the network conditions to the maximum. The main objective of this approach is to demonstrate the improvement in QoS and/or security in having a protocol stack configured dynamically from a set of protocols as opposed to a static stack which does not change during the session while transferring packets over a congested network.

Using a single protocol stack for the entire communication session is the process most widely used in networking by user level applications. The protocol stack is generally decided statically by the application depending on its requirements and it continues to use it for the entire session. A common assumption traditionally made, while selecting a network protocol by an application is that:

The network conditions support the selected protocol for the entire duration of the communication session and hence it will be well suited and that there is no significant over or under utilization of the bandwidth in using that protocol.

However a network may have:

- Highly dynamic network topology: The network is highly dynamic. The conditions of network keep changing due to the addition or deletion of new nodes and servers, the neighbor relationship is only for a short time and is renewed after every few time frames.

- Varying network Load: Presence of various nodes each with varying traffic requirements cause the load on the network to be highly fluctuating.

Hence a statically selected protocol may not be best suited given the varying network conditions.

1.1 Variables

Some of the basic variables involved in a network communication are as follows:

- **Bandwidth**: It is the amount of data that can be transmitted in a fixed amount of time over the network. For digital devices and in networking, the bandwidth is usually expressed in bits per second (bps) or bytes per second. For analog devices, the bandwidth is expressed in cycles per second, or Hertz (Hz).

  Measurement of bandwidth:

  The following algorithms have been used in determining the bandwidth:

  a) Pathchar Algorithm: This algorithm works by sending packets of various sizes starting from 64 bytes to the MTU (maximum transfer unit) of the link layer and measuring their round trip times, correlating the round trip times (RTT) with packet sizes to calculate bandwidth. It increases the packet size by units of 32 bytes.

  b) Packet Pair Algorithm: This algorithm relies on the fact that if two packets are queued next to each other at a bottleneck link, then they will exit the link with a separation time called “bottleneck separation”. The algorithm uses the two packets of the same size as different sizes may have different speeds. By measuring the bottleneck separation the algorithm calculates the available bandwidth.

Other possible algorithms used are Packet Pair Filtering and Receiver and Sender Based Packet Pair algorithms.

- **Packet Loss**: It indicates the number of packets being dropped or lost by the network which is kept track of by attaching a sequence number at the server side to each packet being relayed and by checking the same at the client side. The Client keeps track of the sequence number of the previous packet $\mathrm{Seq}_{\mathrm{prev}}$ and compares it with the current packet received. Hence the total packet loss is given by:

  $PktLoss = \sum_{i=0toN} Seq_{i=cur} - Seq_{i=prev} - 1$, where N is the total number of packets sent by the server, cur and prev denote sequence number of the latest and previous packet received respectively.

- **Latency**: In a packet switched network it indicates the delay in either a one way trip (source sending a packet and it being received at the receiving end) or a round trip (that is the sender sending a packet till it receives an acknowledgement back for the packet).

- **QoS parameters**: These include the following:

  a) Security provided: We have provided with three levels of security to the network communications, in terms of the strength of security provided. The RSA Security algorithm has been used with 32 bit encryption, and has been implemented in two schemes – Shared key and Public / Private key.

b) Reliability: It indicates how much guarantee we provide for the packets to reach the destination, within a time frame. It has been categorized in to three levels again, as follows: No guarantee – when using UDP protocol, acknowledgments and guaranteed transfer using TCP and best guarantee provided by SCTP (Stream control transfer protocol).

c) QoP: Quality of perception is a term used to measure how the user at client side perceives a video or an audio transmission. For better QoP it is observed that usage of UDP (connection less) protocol over TCP and SCTP gives better perception. Hence the selection of a protocol from the stack is also in part determined by the type of the file being transferred.

1.2 Advantages of adaptability - Protocol Stack

The most important goal in a network communication is to achieve the maximum throughput possible while maintaining the minimum QoS and/or security requirement for that session. While a statically selected protocol may well be suited for the initial conditions of the network that prevail like available bandwidth, less traffic and henceforth, but it might not be the best choice for delivering the required quality of service as the network conditions change dynamically. As a result switching over to a protocol from the stack is preferable, such that the required quality of service is maintained and the new protocol causes less traffic if the network is congested or provides better QoS if the network bandwidth is underutilized.

To study the effect of such an adaptation we have done a real implementation using a network of about four computers connected via a switch and the results were obtained by shaping the traffic in that network as required. We have considered delay of the network and the network bandwidth as our main parameters which play an important role in determining the state of any network. These two parameters are measured from the network using ping command and a network bandwidth measurement tool called Iperf. We realize that different types of file have different transmission requirements and hence have considered the three major file types: Text files, Audio files, and Video files. There is a different adaptation policy for each type of file, suitable to their transfer using a different neural network. Provision for giving security as a QoS or as a definite requirement has been provided.

Following this introduction, is chapter 3 that explains the preliminaries and review of the previous work related to the use of protocol stacks in different areas. Chapter 4 explains our approach including the protocols used, adaptation issues. Chapter 5 provides implementation details which include the operating environment, libraries used, client server synchronization, feedback from the network and finally the overheads involved in the system. Chapter 6 discusses the results obtained and Chapter 7 summarizes the main conclusion of this research and scope present for future work in this area.

CHAPTER II

PRELIMINARIES AND REVIEW

The problem of bandwidth utilization has been addressed in many ways, most popular of which are by developing multiple variations of TCP, i.e. TCP – Vegas, TCP – Reno, TCP – SACK. A detailed modeling technique to mimic the structure of TCP source and the interactions a source has with the network has been proposed in [3] using network simulator software – ns2. Although the variants of the TCP exist, most of the previous work has been done using only one of the variants for the entire transmission session.

2.1 Configuration Methods

The method to generate a configurable protocol has a significant effect on its performance and its flexibility. The generation methods are classified as online or offline methods. The online configuration occurs during the execution of an application. It may be performed manually through an Application Processing Interface – API or automatically. This approach requires negotiating of protocol functionality between the participating hosts and server.

Offline configuration occurs before the execution of the application starts. Since generation takes place before the application is run, the future network conditions cannot be taken in to account. It can be put in to effect by having predetermined time slots for each protocol and making a switch as the slot ends. The approach would have to guess when the network might get congested or not and switch accordingly.

Unlike the offline approach, the online configuration is a time critical event, and the time required to send a request for a connection and its processing should not be time consuming and may not pose any obstructive overheads. In both the cases the manual specification occurs under the direct control of an application or the user, through an interface which can be a command line utility or a GUI based interaction. This proposal concentrates on automatic online switching, which is performed without the interaction of the user. Such a system would accept a set of QoS requirements and an appropriate protocol is derived dynamically, using a heuristics approach which could be an analytical approach or a trained neural network. Apart from QoS requirements the selection of the functionality is also effected by available host resources and existing network conditions.

2.2 Control of adaptation

Many aspects of a communication system may be tuned at runtime. These include parameters used internally by the protocol's mechanism such as scheduling of data in transmission queues, window sizes, maximum transfer units etc. Parameters effecting how the protocol works will be controlled by its mechanism itself, although there exists a scope for some user level customization. These include maximum transfer units (MTU), various flow controls and number of connections such as multicast or unicast addresses and their binding.

Manual control is the most straightforward approach to exert control over the protocol selection, though this has a major drawback. It implies that end user has an up to date feedback of current network conditions and also has a detailed knowledge of the underlying functions calls or APIs to make the desired change. Also since the configuration takes place at the user end, the user or the application is aware of the appropriate function calls. Generally most users would not want to be concerned with such specification and manipulation of these details. Given these facts we have chosen to make the protocol configuration automated controlled by a selection engine. The runtime system makes the decision based on a set of QoS parameters, which identify the user requirements, and network characteristics. Some heuristic control mechanism is then used to perform the reconfiguration. We have chosen a multi-layer neural network based approach as our heuristic mechanism. The neural network unit takes feedback from the network and required QoS as input and determines the configuration upon being called by the selection engine.

2.3 Run time Environment

The execution environment can reside in either user space or kernel space, which has been a source of debate. Embedding protocols and reconfiguration capabilities in kernel space will lead us to what is known as a Monolithic Kernel, which includes all operating system functionality and other features embedded in to single large executable kernel. The other approach is that of Micro kernel, which is a core functional kernel on to which other services can be added.

Most modern operating systems such as Windows, Solaris, fall between these two approaches with moderate size kernels into which extra features can be added with the help of libraries. The different approaches to implementing protocol in user or kernel space have been depicted in the below shown figure:

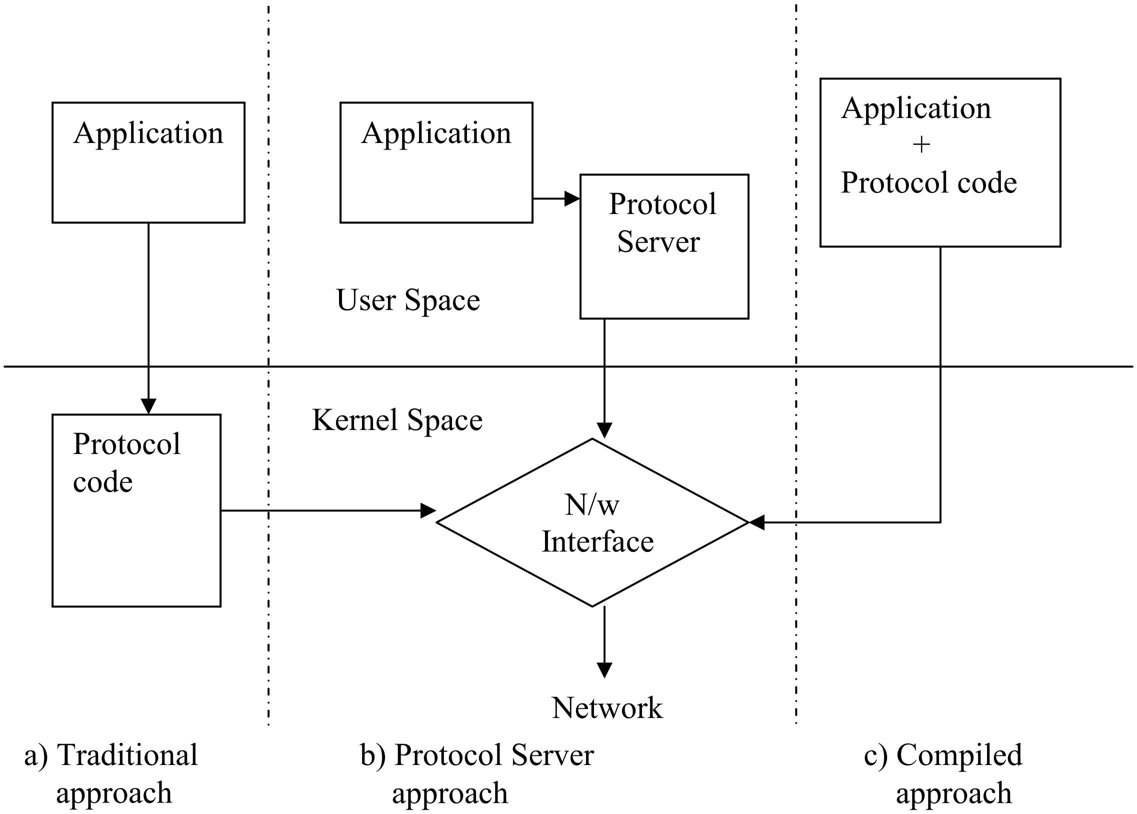

Figure 1 Different Execution environments.

2.4 Related Work

Some of the work related to configurable communication systems:

2.4.1 System V STREAM

This architecture was conceptualized to support the composition of various UNIX terminal drivers [4]. It was further extended to include inter-process communication (IPC) via multiplexer drivers. The functional components of the STREAM consist of: a) STREAM drivers: The driver accepts incoming messages from the network and passes them upstream to the head of the STREAM. Also the drivers pass the segments received from the STREAM module to the network.

b) STREAM heads: The head breaks the application data into discrete messages that are passed down to the one or more STREAM modules to the drivers.

c) STREAM modules: Objects in STREAM are linked to others by a pair of queues used to exchange control information and data. A module performs processing on the data in the queues forwards the results on to the next module.

Modules and multiplexers can be inserted or removed between the head and the driver by offering Push and Pop operations to the application. The STREAM approach was not found to be the best choice for high entropy configurations which required many Push and Pop operations from the stack. Further work on the STREAMs approach has been done which include implementation of shared memory and multiprocessing techniques.

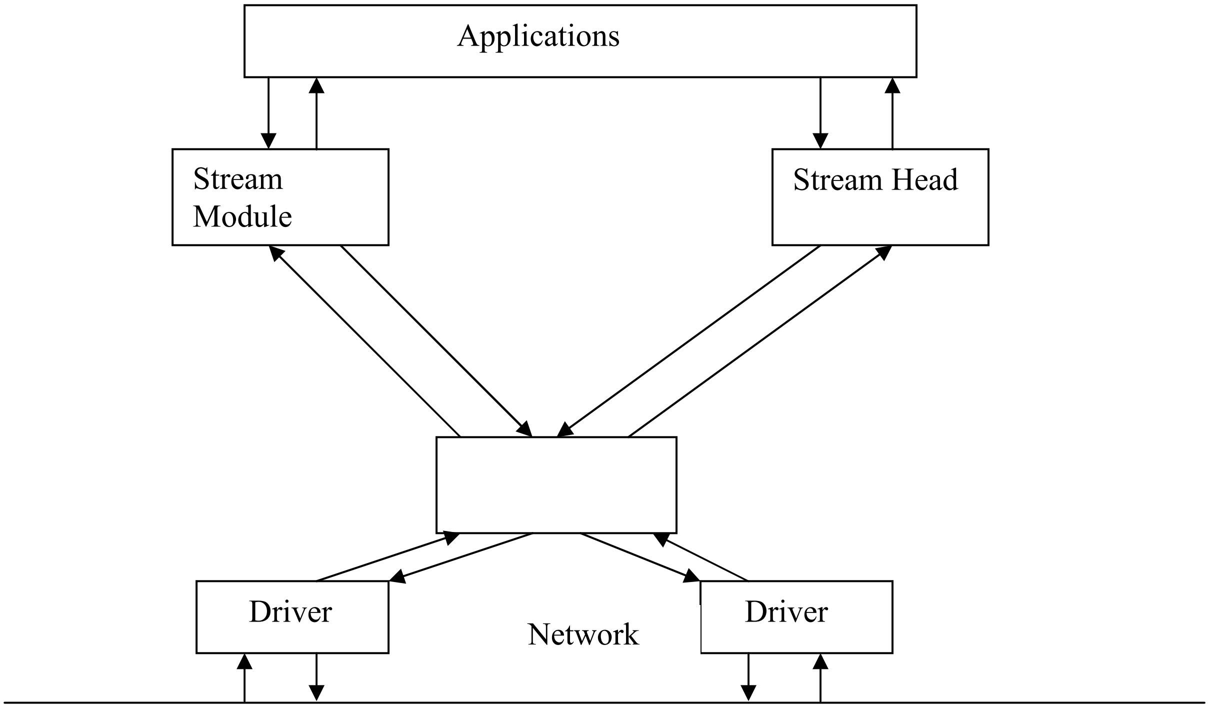

Figure 2 System V STREAMS

2.4.2 X-Kernel

The *x*-kernel is an object-based framework for implementing network protocols. It defines an interface that protocols use to invoke operations on one another (i.e., to send a message to and receive a message from an adjacent protocol) and a collection of libraries for manipulating messages, participant addresses, events, associative memory tables (maps), threads, and so on [7]. Within this model, a communication system is constructed from functional units called microprotocols. These are mechanisms which implement entire protocol layers such as TCP, UDP, and IP. Since these protocols are embedded in to the kernel a kernel programmer configures an instance of x-kernel by specifying a protocol graph using a graphical editor which defines the component functionality and relations among the selected protocol layers. Like traditional protocols these micro protocols communicate among themselves by adding headers to messages and de-multiplexing them at the receiver end.

Since the kernel and the protocols exist as a core program, once the protocols are configured no modifications can be made to its functionality. Though the presence of objects called virtual microprotocols enables customization to a certain limited level. Virtual protocols do not have any headers and in most cases do not have a peer at the other end of the communication. An interaction between microprotocols and virtual protocols has been shown in Figure 3.

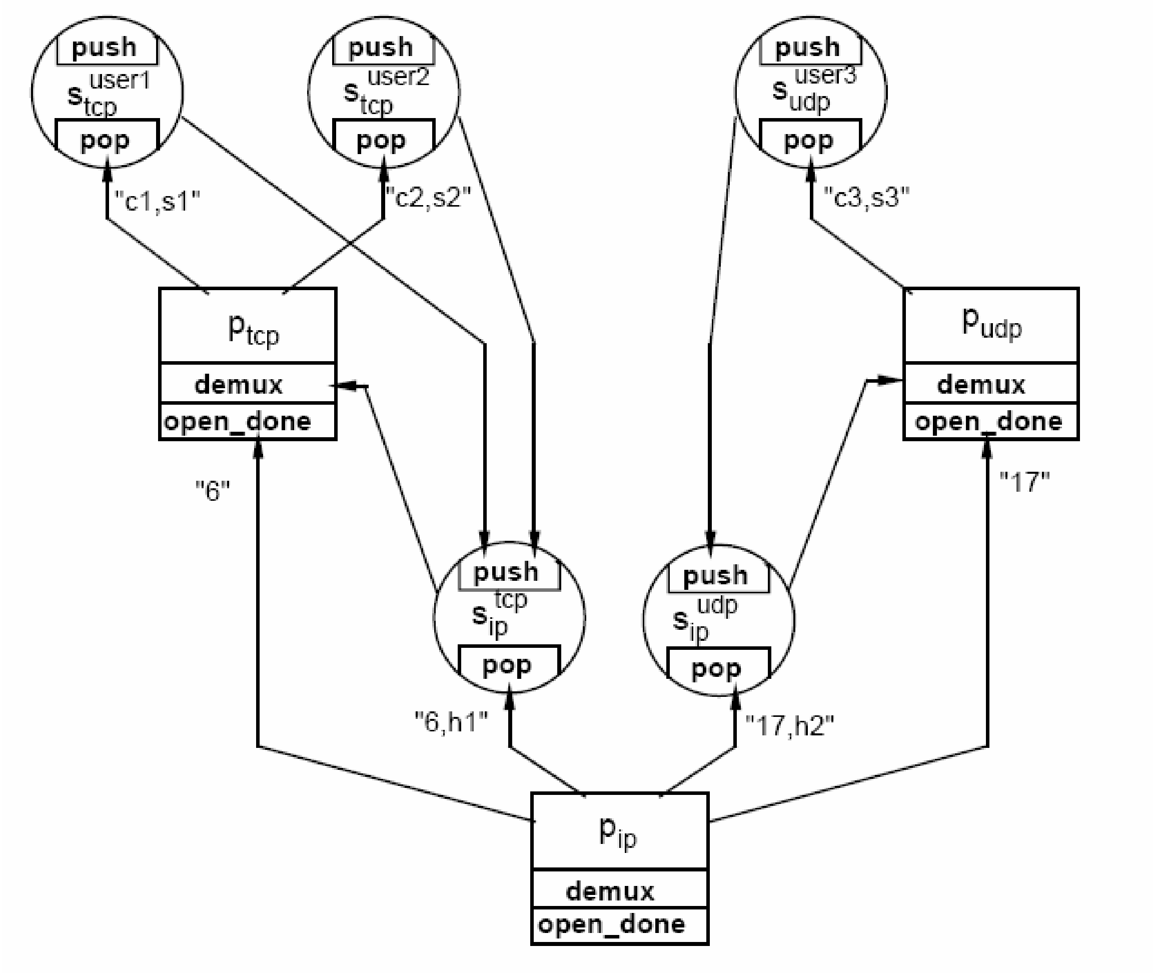

Figure 3 An Example Protocol graph and Session Objects [8]

X-kernel may be run in the two modes:

a) Operating system on its own: In this mode the X-kernel supports message based process architecture, which means that it associates processes with messages, and allows them to escort the incoming or outgoing messages through the appropriate protocol graph. The kernel process is handed back to the kernel process pool, where it may be reused to escort another incoming message.

b) User space protocol: In this mode the X-kernel adheres to the process architecture of the host operating system deploying it.

2.4.3 Adaptive Service Execution (ASX)

The ASX aims at utilizing structural parallelism to reduce communication overheads in shared memory multiprocessor platforms. The framework has been tested in distributed applications that contain multiple network services executing concurrently in one or more processor threads. This framework consists of architecturally independent collection of communication components created in an object oriented methodology, giving significance to inheritance. The protocol generation phase takes care of protocol configuration and is divided into the following phases:

a) Protocol machine specification: This phase deals with the collection of QoS requirements which might be passed on to the system during transport system boot time or at run time as requests.

b) Protocol machine configuration: The configuration phase converts the QoS requirements into a machine independent protocol graph.

c) Protocol machine instantiation: This phase converts the machine independent configurations to machine dependent executables. The machine as mentioned earlier might be a shared memory multiprocessor.

While ASX defines the framework for configurable protocols the internals of ASX require either System V Streams or a X-kernel in order to be implemented. In the event of reconfiguration the end points are notified of configuration changes through separate signaling channels, which enables ASX to experiment with different protocol architectures.

2.4.4 The DaCaPo Project

This Dynamic Configuration of Protocols projects tries to configure the optimal network protocol depending on the offered network services, the requirements of the application and the available processing power. The project divides the communication system in to three discrete sub systems which are Application layer, Communication layer and Transport layer designated as A, C and T respectively and the detailed interactions are given in [6].

Application exists at layer A, a transport system at level T, and a communication system at level C as shown in figure 4.

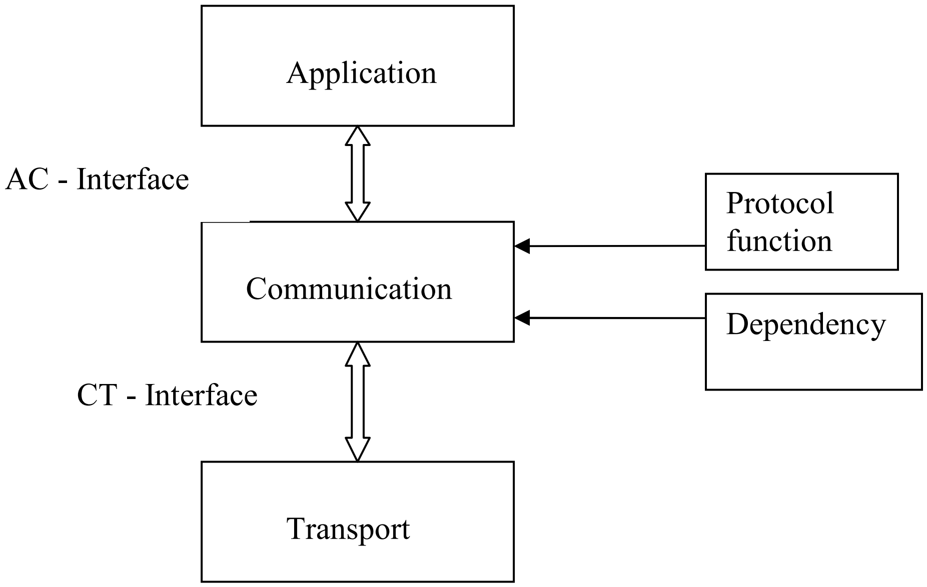

Figure 4 Example Protocol graph of the three layers in DaCaPo

Protocol Configuration in DaCaPo: A configuration process selects the most suitable functionality for the communication layer C, depending on the application requirements and available services in transport layer T. The project has a proprietary language L, used to specify the QoS parameters, the services provided by transport layer and the user requirements. The DaCaPo protocols are more tailored to application needs and may be adapted to dynamically changing network and have potential for handling high entropy situations. Application requirements are specified in terms of tuples using the language L. Initial and run time configurations at all the endpoints of the communication are managed by a system called CoRA

Each DaCaPo application which includes all configured protocols is mapped on to a single process and is implemented in message based protocol architecture. Additionally the run time system is distinguished in three stages: Work, Lift, Monitor corresponding to work done during configuration of the protocol, escorting any packets arriving at AC or CT boundaries and monitoring the properties of the protocols.

2.4.5 DRoPS

This framework provides an adaptable communication system based on a suite of micro protocols collectively called Reading Adaptable Protocol (RAP) [5]. It has a kernel based architecture where the protocols and the adaptation system are implemented in the kernel space. Hence the applications are forced to call these services through a system call interface. In our approach we are implementing the automated adaptation mechanism at application level, so that if in case the application needs to, it can modify the adaptation mechanism and experiment for better results.

CHAPTER III

APPROACH

The thesis proposes constructing a framework that is optimized for performance and is not based on a heavily restricted protocol model, without suffering from poor performance due to the operating environment. The idea here is to have a set of protocols in the kernel level considered as a stack of protocols and be able to dynamically select the best suited protocol from it depending on the current network conditions and security requirements, which is done by continuously measuring the network characteristics. The dynamic selection of the protocol (or adaptation) is done with the help of a neural network which is trained to make the best choice in similar situations.

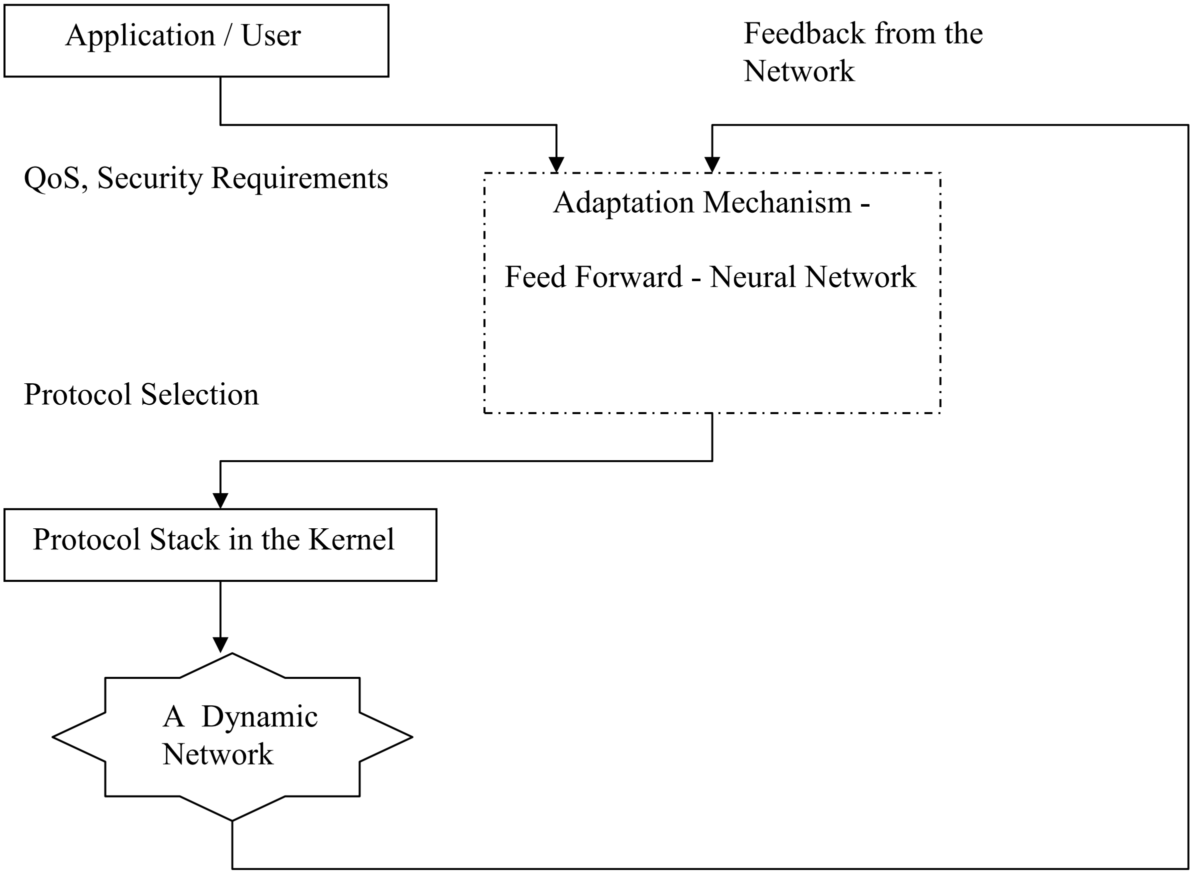
Application / User
Feedback from the Network
QoS, Security Requirements
Adaptation Mechanism -
Feed Forward - Neural Network
Protocol Selection
Protocol Stack in the Kernel
A Dynamic Network

Figure 5 Our Approach

The advantages of this approach are:

a) Integration of protocol mechanism and operating system

The research done by current techniques are based on application or protocol server mechanisms. However we aim to have a system with certain level of integration of the protocols in to the operating systems kernel. The advantages of such integration are low latency communication, compatibility, and less overhead.

b) Easily manageable protocol stack

We are developing a flexible protocol stack where new mechanisms may be enabled, disabled or removed. Since the management of protocols in the stack, is done at application level we have a flexible system which can be modified without interrupting the system.

c) Portability

We have used the standard BSD socket library based bi-directional data stream sockets and hence the code at the application can be ported to any of the flavors of UNIX / LINUX operating systems having the standard protocols suite libraries. The SCTP library is an add-on which can be added as a kernel module before initiating the framework. The protocols are embedded in the kernel including the SCTP kernel patch and the management of the protocols is done the user level.

d) Automation of protocol adaptation

Previous attempts at automation of the protocol adaptation policies have been done in DRoPS[5] and in the DaCaPo project[6]. The DaCaPo project considers a search based model and the DRoPS project has used neural network for the adaptation. The overheads involved in the search based model consume a lot of processing power and hence prove to be not a suitable method. The DRoPS project uses a Protocol specification language APSL and an APSL parser which parses over the protocol configuration and resources. In this thesis we realize that the protocol configuration data is already available internally to the system, and the network parameters can be provided and measured at run time directly. Hence, the need of a protocol/resource configuration language and the parser is not required in our approach thus reducing the processing time and the time in which an adaptation decision can be made.

3.1 The transport layer protocols

In this section we will look at the transport layer protocols which have been used in our stack model. The protocols selected were:

3.1.1 User Datagram Protocol – UDP

It is most widely used non–connection oriented protocol in today's networks. It offers a minimal transport service – a non guaranteed datagram delivery. UDP is almost a null protocol as the only services it provides over IP are checksum, and hence if the application if needs be has to take care of issues such as retransmission for reliable delivery, packetization and reassembly, flow control, congestion avoidance.

3.1.2 Transmission Control Protocol – TCP

It is a connection oriented, reliable-delivery byte-stream transport layer communication protocol, which is the most widely used protocol on the internet today. It uses three-way handshake to establish connections with a host and keeps a count of each packet by having a sequence number for them and by expecting an acknowledgement for each packet sent. No matter what the particular application, TCP almost always operates full-duplex. All the scheduling and stream control algorithms operate in both directions, in an almost completely independent manner. It is more useful to think of a TCP session as two independent byte streams, traveling in opposite directions.

3.1.3 Stream Control Transmission Protocol – SCTP

SCTP is a reliable transport protocol operating on top of a connectionless packet network such as IP much like TCP. It was designed by the IETF Signaling Transport

(SIGTRAN) working group [7]. Though it is similar to TCP, in terms of ensuring reliable, in-sequence delivery of messages with congestion control, while TCP is stream or byte oriented SCTP deals with framed messages. The need for the development of SCTP was realized by the SIGTRAN group whose aim was to create an IP complement to the telephone switching's SS7 network. SCTP was designed to address the following two major issues:

i) Head-of-line blocking – This is a problem where sending independent messages over an order-preserving TCP connection causes delivery of messages sent later to be delayed within a receiver's transport layer buffers until an earlier lost or delayed message is retransmitted and arrives. These delayed messages often establish telephone calls and such delay of critical messages caused critical call control timers to expire thus resulting in undesirable call setup failures.

ii) Multihoming – It is the capability where a host can have multiple points of attachment to the Internet, for redundancy purposes. The protocol mechanism then does not want to wait for a routing convergence (often on the order of minutes) to communicate critical messages to its peer communication endpoint. For call control signaling, such delay is unacceptable when an alternate available path exists. A TCP connection only binds a single point of attachment at either end point.

SCTP incorporates both these mechanisms in its routing policies. Also because of its multihoming capabilities, four-way handshake connection establishment technique (compared to three-way used by TCP), and 32 bit checksums against TCP’s 16 bit, SCTP offers a more reliable and robust communication when compared to TCP.

3.1.4 Added security layer

Apart from the above three protocols, we have added an optional layer of security to all the three protocols to have a wider range of protocols in the stack, meeting the security requirement of the user or the application. Encryption and decryption of the data is done using DES algorithm, which are present as a built in function call in the crypt.h library.

3.2 Adaptation issues

The term adaptation here refers to selection and configuration of a more suitable protocol than the present, given the current network conditions and user security requirements. As discussed earlier the two ways to approach this issue is either by manual adaptation or by automating it. Manual adaptation allows the application the complete control over the functionality of the communication system. This implies that the application needs to be aware of protocol specific details and of the API calls to do so. This will put an extra burden and requires every application to take care of the adaptation policies which increases the complexity of the application and reduces its efficiency. It should also be noted that the application developers might not have hands on information about the adaptation strategies. Given these drawbacks the automation of the adaptation strategy using a heuristic approach is more suitable.

3.2.1 Artificial Neural network as the heuristic in the adaptation strategy

An Artificial Neural Network (ANN) is an information processing paradigm which is inspired and designed in the way the biological nervous systems in the brain processes information and learns. Our brain is composed of a large number of highly interconnected processing elements (neurons) in the range of 10^{10}, working in unison to solve specific problems. ANNs like people, learn by example and need training to take any correct decision. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. The most fundamental difference between brain and computer is that today's computers operate primarily sequentially, or with a small amount of parallelism while human brains are massively parallel. The three key concepts involved here are inputs to the neuron, weights present at each connection which determine how much effect the input will have on that connection, and the activation function which indicates how an artificial neuron responds to an input. The activation function limits the amplitude of output signal based on the sum of all the weighted inputs arriving at the neuron.

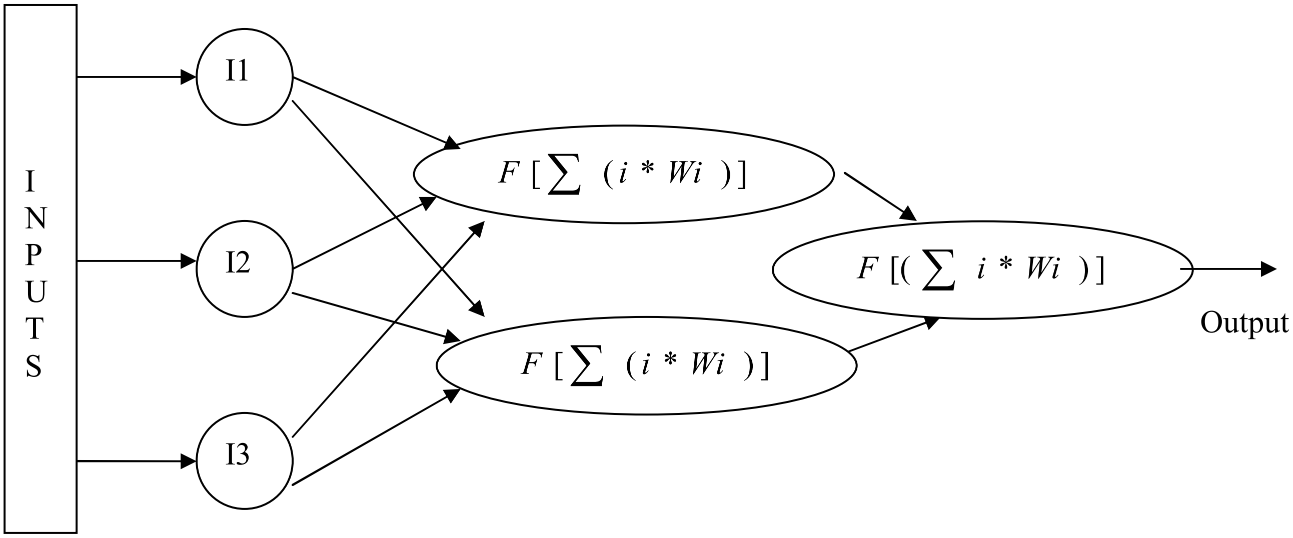

Figure 6 An example Feed Forward neural network.

3.2.2 Activation function: This function's primary goal is to introduce nonlinearity in to the system. The activation function is useful when the neural network contains some hidden units. Since we are incorporating a hidden layer in our design, choosing a good activation function is important which makes it more powerful than a single perceptron with linear inputs and outputs. For backpropagation, the activation function must be differentiable that is it should be continuous as we need to adjust weights at each input proportional to the effect the input had in the output generated. Sigmoidal functions such as tangent-hyperbolic and the Gaussian function are the most common choices.

A simple continuous function is shown below in the figure:

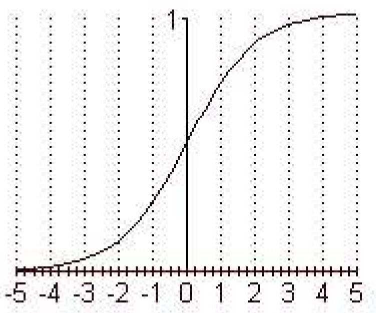

Figure 7 Sigmoid Function

The Sigmoid function we have used is:

Y = 1 / (1+ e^{-x}) where x is the net input to the perceptron and Y is the output of the unit, which is fed as input to the next layer neurons.

3.2.3 Multi Layer Perceptron (MLP) model

Perceptron Learning Rule: In a single-layer network, each neuron adjusts its weights according to what output was expected of it and the weight adjustment of a neuron i is given by:

$\Delta W_i = x_i \delta$, where δ = *desired output – actual output*, W is an array of weights and x is an array of inputs at each neuron.

The Perceptron Learning Rule is not useful when we extend the network to multiple layers to account for non-linearly separable problems. This is because adjusting a weight at one neuron will have an effect on the output of all the subsequent neurons and hence on the overall output of the system. So when we use a sigmoid activation function the error function is linked to the derivative of the activation function.

Weight adjustment in a neuron i is mathematically given as follows:

$$\Delta Wi = \eta / P \sum_{p=1}^{P} x_{ip} \delta_p$$

This equation is made more iteratively friendly, when it is broken up as follows:

$\Delta W_i = \eta x_i \delta$, where $\delta = y_p(1 - y_p)(d_p - y_p)$

here, η is the learning rate, y_p and d_p corresponds to the actual output and the desired output. This δ is for the output layer, but in order to adjust weights in the hidden layer we need to calculate δ for the hidden layers also, which is determined using back propagation as follows.

For a neuron q in hidden layer p, its δ is given as:

$\delta_p(q) = x_p(q)[1 - x_p(q)]\sum W_{p+1}(q,i)\delta_{p+1}(i)$

Each delta value for hidden layers requires that the delta value for the layer after it be calculated, hence the delta of layer (p+1) is being used in the above formula. W_{p+1}(q,i) represents the weights of all the neurons in layer p+1, connected with the neuron q.

In this thesis we have used a 10 neuron - 3 layer architecture (2-2-6 ANN), where the first layer takes the parameters from the application as the input, the next layer is an intermediate- hidden layer and the third layer consists of the six neurons each representing a particular protocol, is the output for the model. Among the output layer the neuron whose value is closest to one represents the protocol to be used for the given inputs. Training of the neural networks was done using a training data which was composed of three parameters: bandwidth of the network, delay in the network followed by the desired protocol for that combination. Each neural network is trained by repeatedly calling a train function certain number of times with the training data, so that the weights of the neural network connections are adjusted to such a value so that when called by the program after training with the current parameters the desired protocol is selected.

The training data was obtained by observing which protocol performs the best (in terms of throughput and packet loss) when the two input parameters are set to a different values and the network is shaped accordingly, covering the possible ranges. The best performing protocol then was used as the desired protocol in the neural network training data for the corresponding values of bandwidth and delay.

3.3 Contributions of this thesis

a) Run-time selection and switching of protocols has been proposed while most of the current systems choose to have statically pre-determined protocol for the entire communication session.

b) We have added dynamic adaptation control to the communication system such that the adaptation lies in the user level and the protocols themselves lie in the kernel level. This allows the user / application to take part in the decision making part while selecting a protocol.

c) With larger sets of QoS parameters taken from the network and user requirements the neural networks will be able to take intelligent decisions even if not trained for that particular scenario.

CHAPTER IV

IMPLEMENTATION

4.1 Operating Environment and Libraries

We have used the LINUX – Fedora Core 3 environment, running 2.6.9 version of the kernel to write and implement the code. The code was written in C and compiled using GNU compiler gcc. Header files for the TCP and UDP protocols are provided bundled in the packet library packet.h. Since SCTP is a relatively new protocol it was installed as a kernel module provided by the - The Linux Kernel Stream Control Transmission Protocol (LKSCTP) project and the version used was lksctp-2.6.14-1.0.4. The corresponding kernel module is loaded every time the system is started using a kernel module program called - modprobe. More information regarding the installing the SCTP protocol can be found in [11].

Our system does integration between the user level code and the kernel level functions using various system calls provided by the Linux environment present in the header files. While the neural network, selection and reading of the file to be transferred, creating sequence and protocol numbers etc are done at the user level, the actual socket creation, send and receive packet functions for different protocols are implemented in the kernel space which are accessed using system call interfaces.

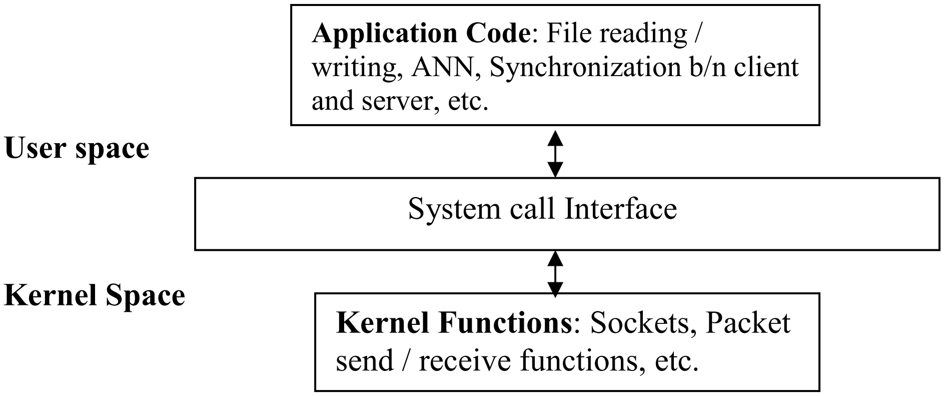
Application Code: File reading / writing, ANN, Synchronization b/n client and server, etc.
User space
System call Interface
Kernel Space
Kernel Functions: Sockets, Packet send / receive functions, etc.

Figure 8 Interface with kernel functions.

To provide a security layer over the protocols, a built in library function was used which does DES encryption on the passed string. The DES algorithm for encryption and decryption is present as a header file - crypt.h in the Linux environment. With the addition of the security layer we have a total of six protocols to choose from, as shown in following figure.

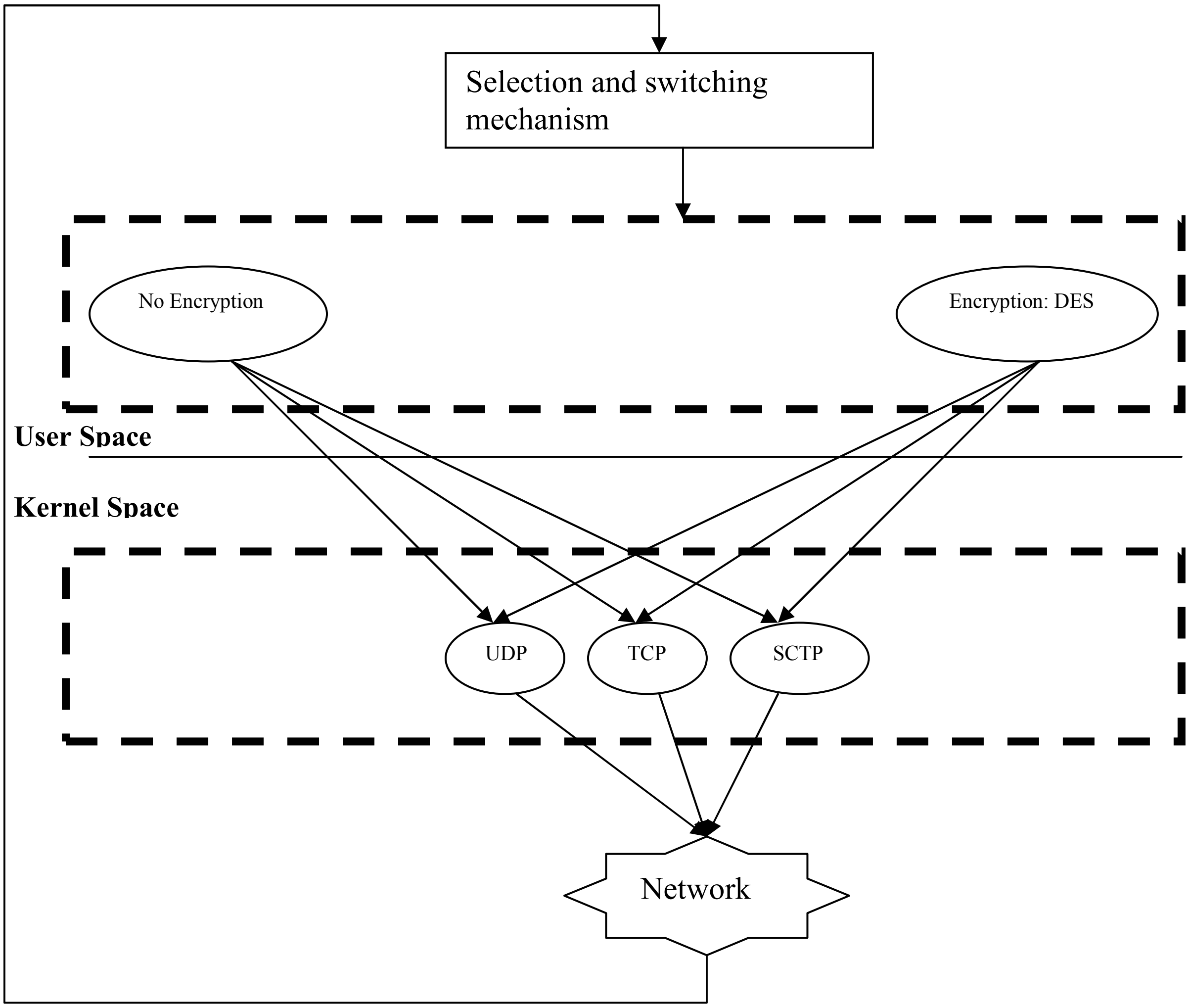

Figure 9 Overview of the system.

4.2. Adaptation

The neural network is trained to adapt based on prevailing network conditions. Our training set currently consists of three parameters which are the available bandwidth and delay reading from the network and the desired protocol.

The feed forward network uses the sigmoid activation function which is $Y = 1 / (1+ e^{-x})$.

The error propagation corresponds to the derivative of the sigmoid function and is given as follows:

a) At the output layer: $\delta_{out} =$ out (1-out) (d_{out} – out)

b) At the first neuron in the hidden layer: δ_{h1} = i_4*(1-i_4)*(W[2][2])*(δ_{out})

c) At the second neuron in the hidden layer: δ_{h2} = i_3*(1-i_3)*(W[1][2])*(δ_{out})

Where, i_4 and i_3 represent the net inputs to the first and second neuron in the hidden layer, d_{out} is the desired output and W [i, l] array represent the weight at i^{th} neuron in the l layer. The ANN was trained using the data collected by shaping the network conditions to different bandwidth and delay values and noting the performance of each of the protocol in such conditions. The best performing protocol was used as the desired protocol for such conditions.

Adaptation can be done using various techniques. A straight forward adaptation technique is to use an analytical formula based on the input parameters from the network for determining the required protocol for such conditions. Another technique employed by a colleague working under Dr. Thomas was using an Analytical Hierarchical Process for the decision process. Analytical Hierarchy Process is an effective multi-criteria decision making process that aids in making a good decision by using set priorities among all the alternatives, involving several levels of dependent and independent information.

4.3 Traffic Shaping and Feedback

Measuring the bandwidth available in the network is a complex task. While much software exists in the market today, most of them provide us with only an approximation of the actual bandwidth. We had chosen to use an open-source software called – Iperf, which gives us an approximation of the achievable bandwidth rather than the measure of total available bandwidth of the network and hence provides a more accurate bandwidth measurement. Iperf uses no internal bandwidth determination algorithm but rather opens up multiple socket connections and pumps a lot of packets in to the network and thus measures the total achievable bandwidth. For shaping the bandwidth we needed to run multiple instances of iperf on a single or more systems, depending on how much bandwidth was desired to be shaped.

Iperf generates the bandwidth readings in an interval for which it is set, ranging from a single second to multiple seconds. To read and input this bandwidth in to our program we have redirected the output of iperf in to a text file and the program parses the text file to obtain the last entry among all of the bandwidth values. Also to continuously loop iperf for updated readings we have made use of the command line option through which we can specify the number of times we can run it.

For measuring the current delay in the network we are using the system ping command which reports the delay value in micro seconds. The ping reading is input into the program by redirecting its results to a text file and then parsing that file, similar to the method of obtaining bandwidth readings.

4.4 File type and Security requirements

We have considered three different file types: Video, Audio, and Text files. Each of the file type has a different QoS requirement and the corresponding file transfer performs better when using specific protocols under certain network conditions; which differ for each file type under the same network conditions. Hence we trained three neural networks separately to handle each file type. Transfer of video files requires the highest throughput which is generally achieved by UDP under most network conditions. It should be noted that UDP starts dropping packets as the network conditions worsen. In case of text files, it is required to prevent any packet loss from occurring and requiring a high throughput / bandwidth is not a necessity. Hence transfer of text files is generally done using connection oriented and reliable TCP protocol or in some cases using HTTP. But the very slow speed, at which HTTP works, does not make it a good choice for transferring files. Throughput requirements for Audio files fall in between that of Video and Text files.

The security requirement is taken from the user at the start of the file transfer and if required all packets are encrypted before sending them across the network to the client side. The client determines the security requirement from extracting the protocol number from the packet and hence does the corresponding decryption of the packet before writing it in to the file. Provision of security given is to show that the adaptation policy also works in with variety of protocols, secure or non-secure.

4.5 Synchronization between the Client and the Server

Switching to a different protocol requires both the client and the server to do so at the same instance which otherwise will lead to client to timeout waiting for packets of a protocol other than what server is sending. This requires that both client and the server keep notifying each other of the current protocol they are expecting and the also the type of the next packet to be transferred. This synchronization was achieved by including about two bytes of extra data over the packet, which represented the protocol to be used for the next packet to be sent and also the packet sequence.

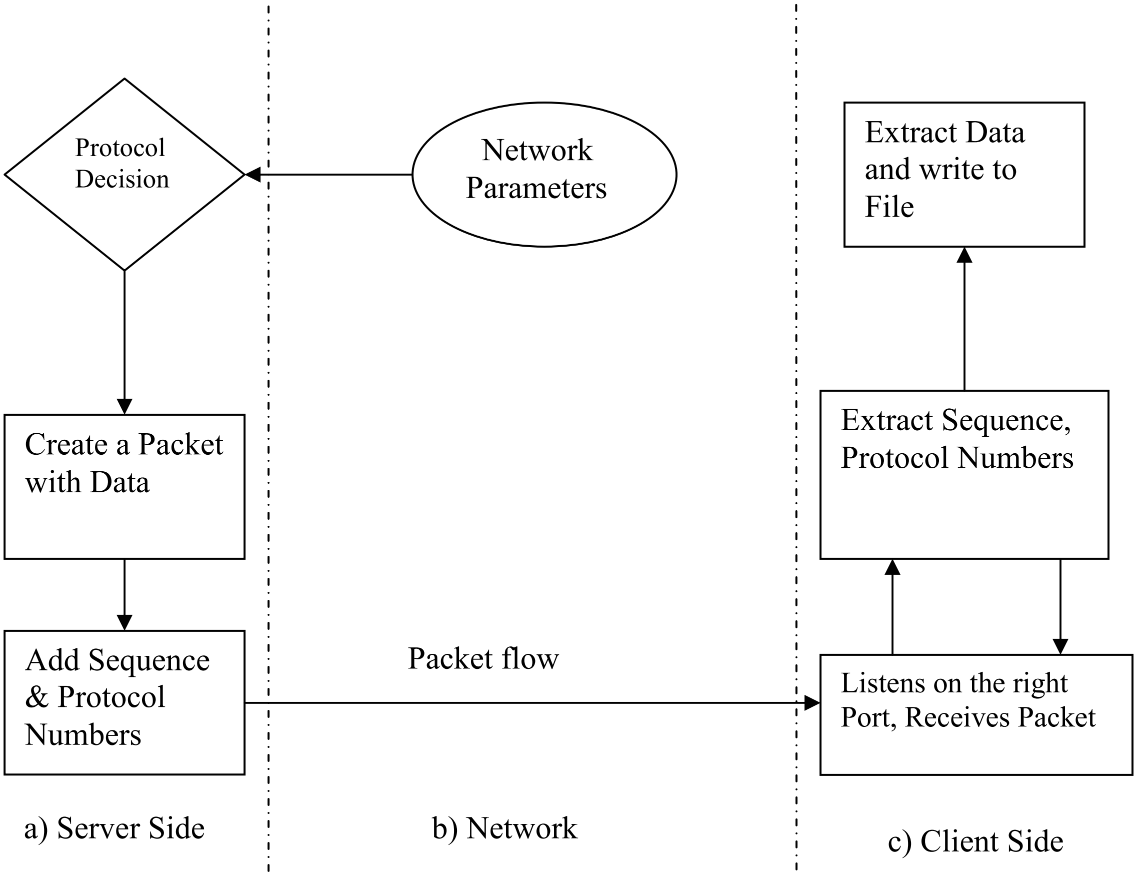
Protocol Decision
Network Parameters
Extract Data and write to File
Create a Packet with Data
Extract Sequence, Protocol Numbers
Add Sequence & Protocol Numbers
Packet flow
Listens on the right Port, Receives Packet
a) Server Side
b) Network
c) Client Side

Figure 10 Client – Server Synchronization.

This figure explains the client – server synchronization, and shows the main events concerning the protocol synchronization. The port corresponding to each protocol is created initially and is sustained till the end of each complete file transfer session. If the client recognizes that there is change of protocol for the next packet onwards it switches over to the port corresponding to that particular protocol and listens on that port for incoming packets.

4.5.1 Packet Sequence number

We have added a sequence number on top of every packet exclusively for tracking every transmitted packet at the application layer at client side. This sequence number allowed us to calculate the number of packets received or dropped. In case of UDP and UDP with security any loss of packets was covered up for writing the last received packet repeatedly in to the file at the client side till the corresponding loss of data was compensated for. This provision allowed us to be able to open the files received even though some packets are lost, which in normal case is not generally not possible as the file fails to match the integrity checks for file size or just become corrupt.

Also packets are checked for arrival for a limited amount of period in the client side – timeout period, and if no packet is received within this period the client stops listening for anymore packets from the server and terminates that particular file transfer.

At the end of a file transfer, the client checks for consistency with the file size it has received and the actual file size by checking with number of packets lost during the transfer if any or if any time-outs have occurred.

4.6 Network Monitoring

The primary monitoring tool used in this thesis was the Ethereal network analyzer [15]. This tool allowed for monitoring the actual packet flow, identifies the packet type, and also source and destination address. We have chosen to run Ethereal in a promiscuous mode so as to enable packet capture not only destined for its source on which it is running but also the packets flowing through the interface on which it is listening. At the end of every session it gives a total summary of number packets of each type which were transferred in the network.

4.7 Overheads involved in the system

Some of the overheads are:

a) Continuous Network monitoring: Since the mechanism requires a constant feedback from the network, there is a need to measure the bandwidth and the delay in the network in regular intervals of time. Running the bandwidth measurement algorithm is an overhead for the system in terms of CPU utilization.

b) Neural Network Training: We need to train the neural network prior to running the system, which is an initial overhead. Neural network adaptation can be achieved in real time and therefore it does not incur any considerable overhead while the system is making a decision. The training overhead is offline and therefore does not affect the online performance of the system in the file transfer process.

c) Security overhead: The use of DES algorithm to encrypt each message at the server side and decrypting the same at the client side is a processor overhead. The delay it presents is a trade off as the throughput reduces, but considering security requirements by the user this may be tolerated.

CHAPTER V

RESULTS

The results corresponding to our system are marked as X, in the graphs shown below. We have taken results for each type of file and compared the results of file transfer of our system against those obtained when using a single protocol (as in traditional systems) for the file transfer. Firstly, we have shown the results in the scenario where the network conditions are optimum to start with and they deteriorate with time. In the other case, we have presented our results in a scenario where network conditions fluctuate randomly.

For streaming of video files, it is essential that there is constant flow of data at the maximum rate possible. Use of UDP allows for a constant flow but at the cost of certain packets which are lost. Traditionally Video file transfer in most cases is done using either UDP or RTP. Real Time Protocol (RTP) is a derivative of UDP, and hence as unreliable as UDP. RTP includes a time-stamp and sequence number added to the packet header which allows it to detect packets lost.

In case of Video and Audio files, comparisons have been done against a system using just UDP and SCTP protocols. This is done because UDP and SCTP deliver high throughput in comparison to TCP and as mentioned are a requirement for streaming video and/or audio files. In case of text files, we have compared with a system using TCP and SCTP and not UDP, as TCP and SCTP are reliable transport protocols whereas UDP is not and may lead to loss of packets and hence the valuable information in text files might be lost.

5.1 Throughput measurement

5.1.1 Under deteriorating network conditions

	Time (s)	UDP (KBps)	X (KBps)	SCTP (KBps)
Optimum n/w conditions	1	195	193	190
	10	197	196	192
Average n/w conditions	11	182	181	175
	20	184	182	176
Unfavorable n/w conditions	21	159	171	172
	29	158	169	171

Table 1 Throughput readings for deteriorating network conditions for Video Files.

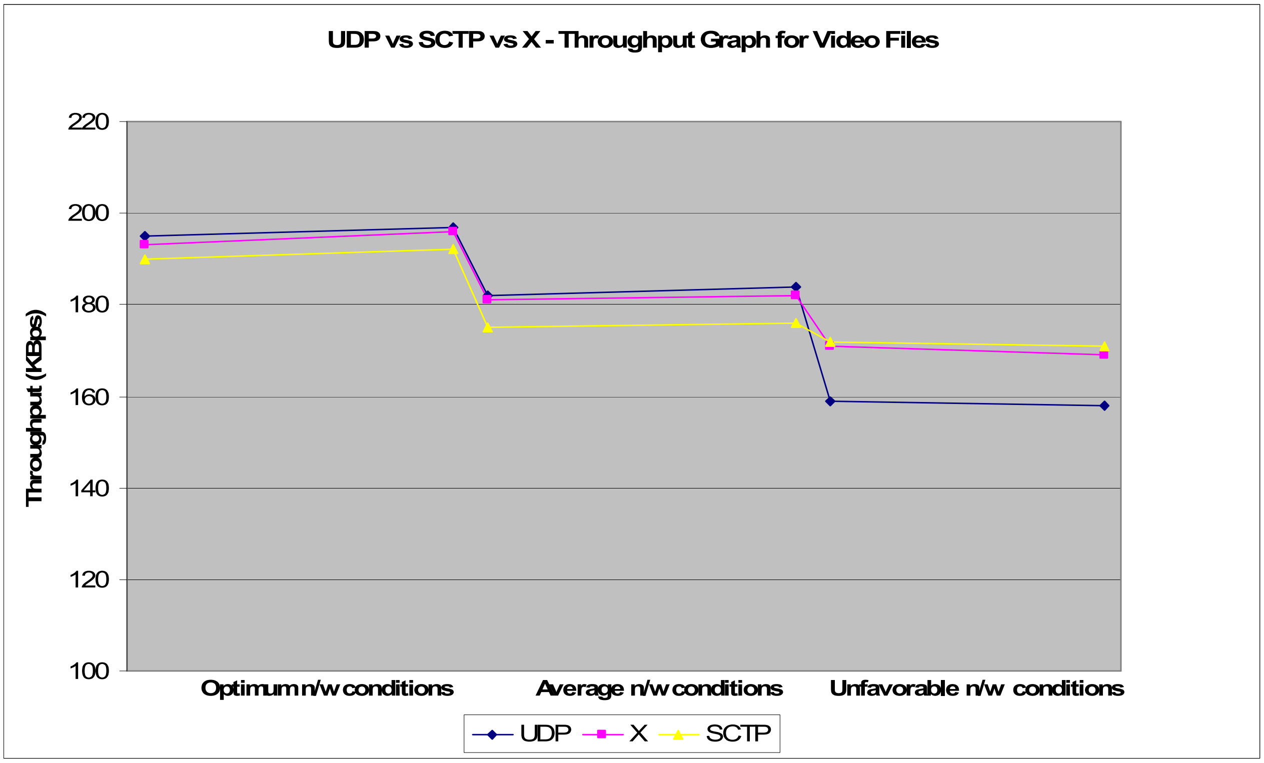

Figure 11 Throughput graph for deteriorating network conditions for Video Files.

In the graph we have shown throughput of UDP, SCTP and our system (X) in the following three scenarios:

a) Optimum conditions (High bandwidth, Less Delay): The bandwidth reading from running a single instance of iperf in this scenario ranged from about 50 to 75 Mbps and delay reading range from anywhere between 0.1ms to 4.5ms.

b) Average conditions (Medium bandwidth, Medium Delay): In this scenario we restricted the bandwidth to the range of 25Mbps to about 50Mbps using at least 2 instances of iperf server on the file server and the iperf clients on the file receiving client. Delay values were restricted to the range of 5ms to 15ms.

c) Unfavorable conditions (Less bandwidth, High Delay): Here the bandwidth is drastically reduced to the range from 25 Mbps to less than 1Mbps, using at least 4 instances of iperf servers and clients. A third party machine was also used to clog the network by pumping dummy packets in it. High delay in the network was produced by pinging the client with large sized packets specified in the ping command.

It is observed that UDP has better throughput than SCTP and also TCP (not shown) from the graph when the network conditions are favorable. In our approach the adaptation mechanism chooses to use UDP in both the best case and the average case scenarios while the adaptation selects the SCTP protocol when the network conditions worsen. It is seen that switching to SCTP when the network produces high delay and has clogged bandwidth is an advantage as SCTP is able to provide more throughput than UDP, a necessity for video files.

Audio Files:

Time (s)	Time (s)	UDP (KBps)	X (KBps)	SCTP (KBps)
Optimum n/w conditions	1	195	193	190
	10	197	195	192
Average n/w conditions	11	181	174	175
	20	183	175	175
Unfavorable n/w conditions	21	158	168	172
	29	160	170	171

Table 2 Throughput readings for deteriorating network conditions for Audio Files.

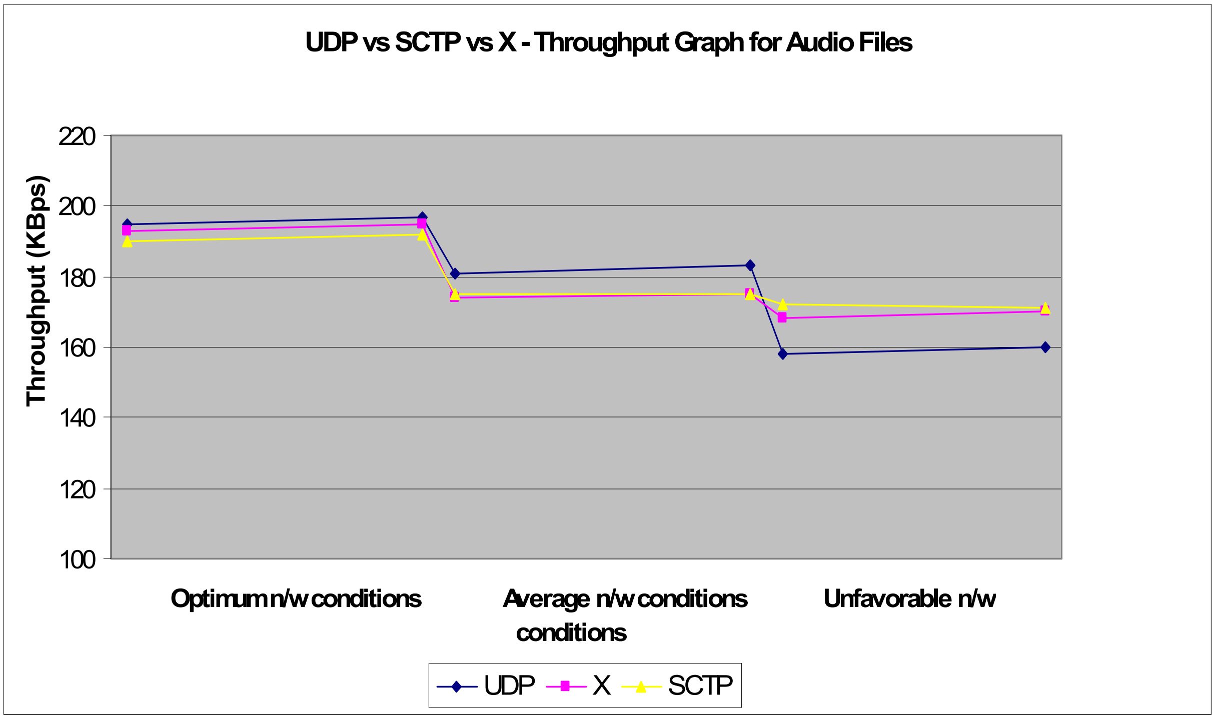

Figure 12 Throughput graph for deteriorating network conditions for Audio Files.

In case of Audio files, our system adapts to UDP under favorable conditions and switches to SCTP during average and unfavorable network conditions. SCTP was chosen even when network conditions are mediocre to prevent few packet losses which is experienced by UDP and is a trade off for not using its higher bandwidth capacity. Here a little reduction in bandwidth is traded off for achieving a better quality in sound by not loosing data packets.

Text Files:

	Time (s)	TCP (KBps)	X (KBps)	SCTP (KBps)
Optimum n/w conditions	1	186	183	190
	10	184	182	192
Avg. n/w conditions	11	161	177	175
	20	161	176	176
Unfavorable n/w conditions	21	146	171	170
	29	147	167	171

Table 3 Throughput readings for deteriorating network conditions for Text Files.

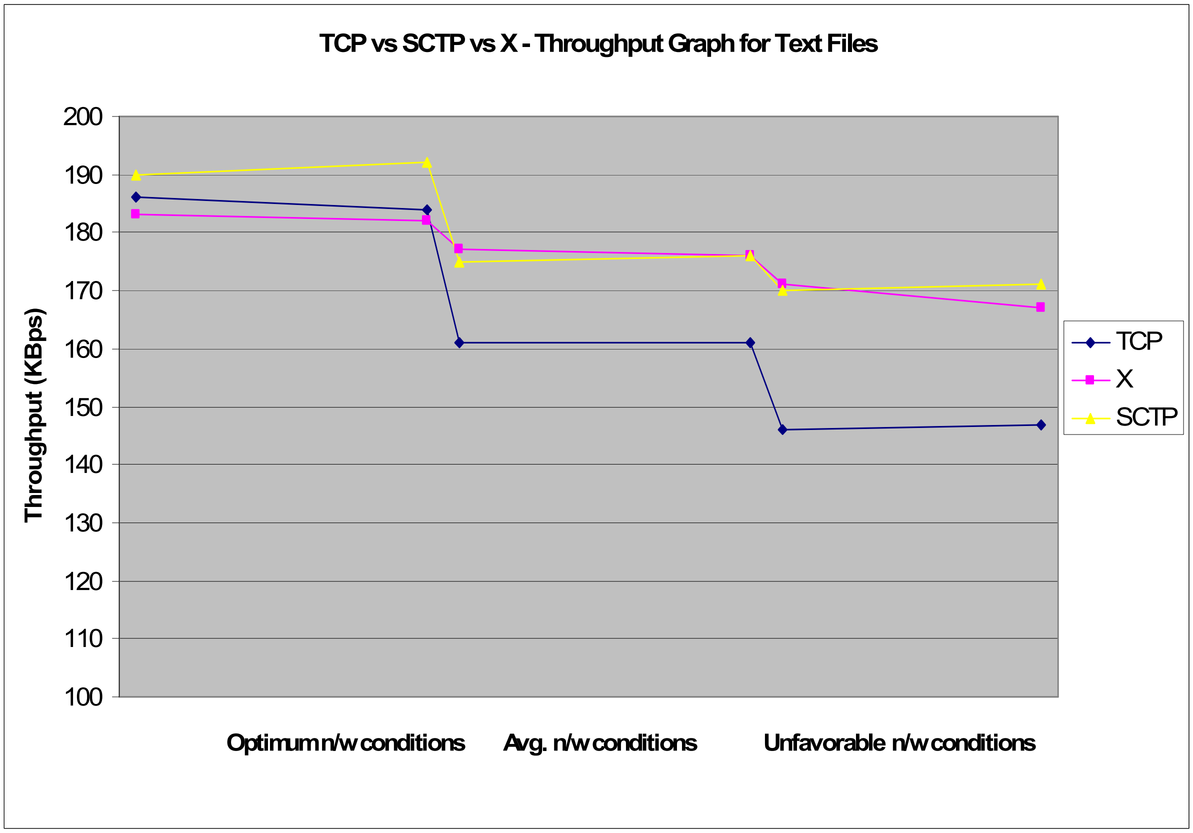

Figure 13 Throughput graph for deteriorating network conditions for Audio Files.

For text files, it is important to not loose any data packets during the file transfer as each packet represents a critical part of the information contained in the file. Hence our system does not to select UDP as its choice in any of the network conditions. As seen in the graph we start with TCP when the network conditions are very favorable but as is seen the throughput of TCP falls as the network experiences delay or clogging. Our system switches to SCTP protocol in all the conditions of the network except the favorable one for achieving higher throughput than TCP. It is also noted that adaptation may not be needed as in this case, if a high performance protocol is supported to begin with (SCTP achieves higher or as much throughput as TCP) in all the network conditions. But since TCP is the standard transmission protocol at present for the transfer of data files, we can start with TCP and only switch to SCTP if at all needed that is only if the network conditions worsen (provided the client, server and the intermediate nodes can understand and support SCTP sockets and packets).

5.1.2 Under Random network conditions

Since the original network conditions are always changing, either deteriorating or improving we have tested our results by producing such conditions and running our system. To denote the conditions we have used D for Delay in the network and B for Bandwidth, and H, M, L stand for High, Medium, and Low values. The actual range of values are as follows BH – 50 to 80 Mbps; BM - 50 to 20Mbps; BL – 20 to 0 Mbps; DH > 15 ms; DM – 5 to 15ms; DL – 5 to 0ms.

Video:

N/w Conditions	UDP (KBps)	SCTP (KBps)	X (KBps)
DH, BM	165	172	175
DM, BM	180	178	180
DM, BL	178	173	176
DL, BH	195	190	195
DL, BM	195	189	193
DH, BL	160	170	171
DL, BL	192	189	192
DM, BH	182	181	181
DH, BH	170	172	171

Table 4 Throughput readings for random network conditions for Video Files.

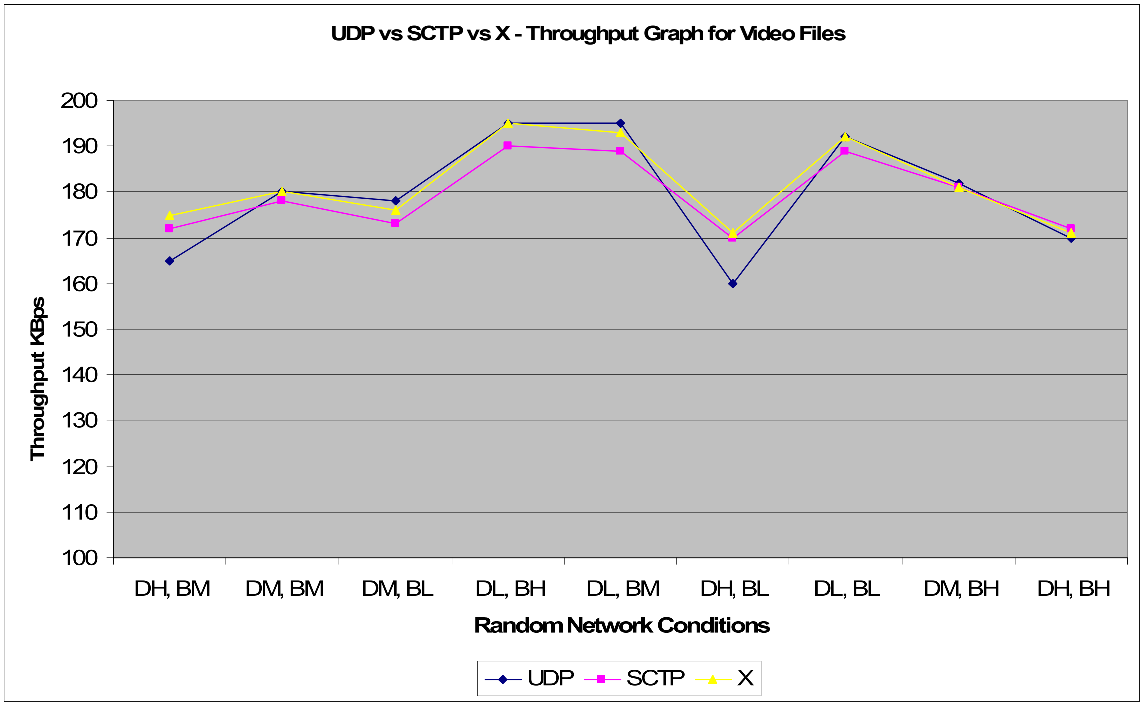

Figure 14 Throughput graph for deteriorating network conditions for Video Files.

Here we see that our system always switches to the protocol which has higher throughput between UDP and SCTP and TCP. In most cases X chooses to follow UDP, and switches to SCTP only when the delay is high, as in those cases UDP's throughput drops by a big margin.

Audio:

N/w Conditions	UDP (KBps)	SCTP (KBps)	X (KBps)
DH, BM	165	172	173
DM, BM	180	178	179
DM, BL	178	173	172
DL, BM	195	189	194
DL, BH	195	190	196
DH, BL	160	170	172
DL, BL	192	189	191
DM, BH	182	181	180
DH, BH	170	172	172

Table 5 Throughput readings for random network conditions for Audio Files.

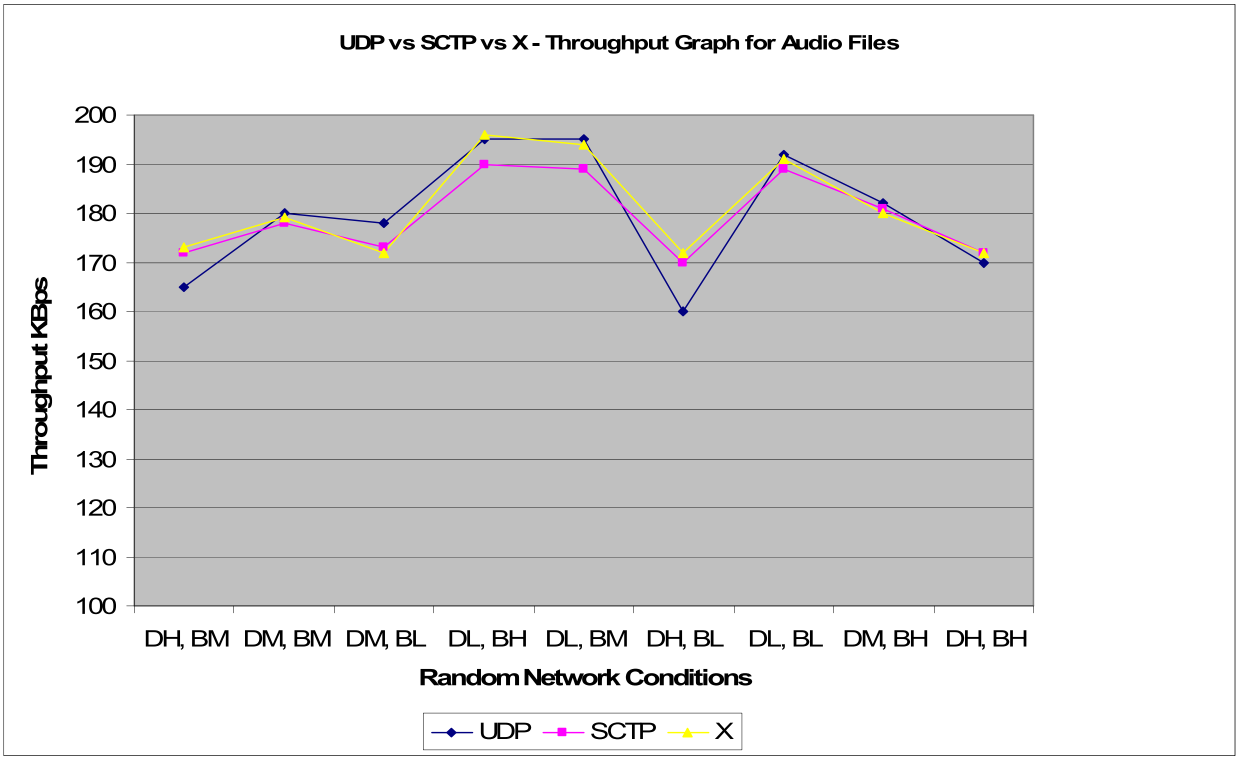

Figure 15 Throughput graph for deteriorating network conditions for Audio Files.

Here it is seen that X follows UDP only in good conditions and switches to SCTP when the network conditions are average or worse.

Text:

N/w Conditions	SCTP (KBps)	TCP (KBps)	X (KBps)
DH, BM	172	147	174
DM, BM	178	160	179
DM, BL	173	157	171
DL, BH	190	186	186
DL, BM	189	181	180
DH, BL	170	145	173
DL, BL	189	178	177
DM, BH	181	166	182
DH, BH	172	152	172

Table 6 Throughput readings for random network conditions for Text Files.

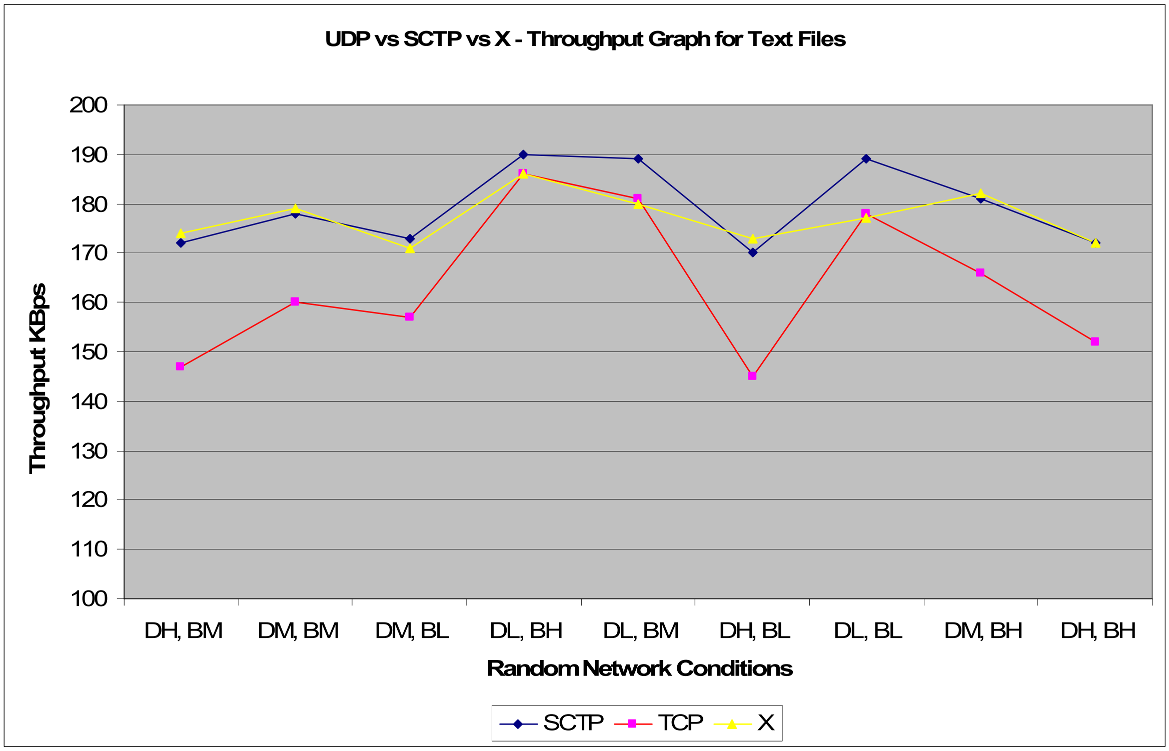

Figure 16 Throughput graph for random network conditions for Text Files.

It is seen that we switch to SCTP from TCP when packet flow starts to experience a delay in the network. Using SCTP in such conditions allows for higher throughput through the network. Again it can be observed that adaptation may not be required in this case as SCTP has higher performance than TCP but as mentioned earlier we can continue to use TCP, switching to SCTP can be avoided and only done only when there is delay in the network.

5.1.3 With Security

The adaptation can be such that when there is an available scope to provide an extra QoS parameter such as security it can be done. The ANN was trained in this scenario to select a secure protocol when the network conditions are best and to select non-secure protocols otherwise. This is shown in case of transfer of text files as an example.

N/w Conditions	SCTP (KBps)	TCP (KBps)	X (KBps)
DH, BM	172	147	174
DM, BM	178	160	179
DM, BL	173	157	171
DL, BH	190	186	142
DL, BM	189	181	180
DH, BL	170	145	173
DL, BL	189	178	177
DM, BH	181	166	182
DH, BH	172	152	172

Table 7 Readings for random network conditions for a Text file with conditional security provided.

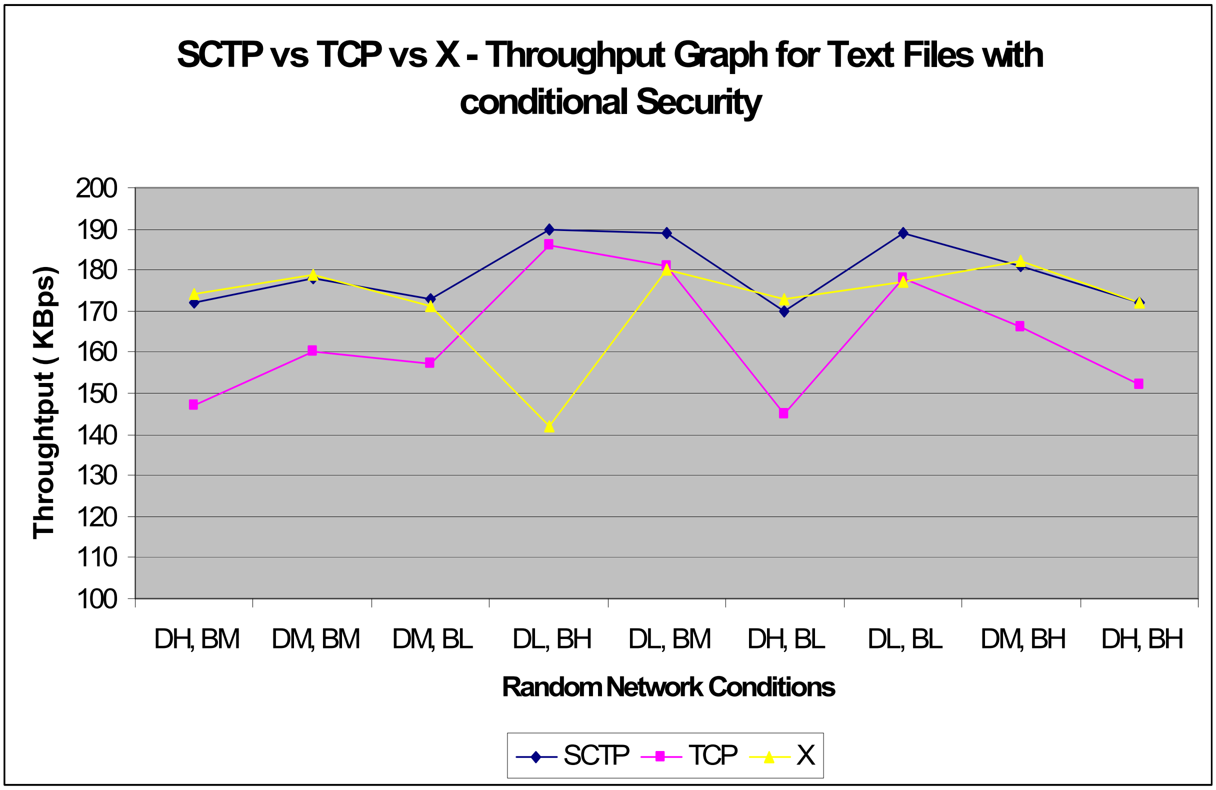

Figure 17 Throughput graph for random network conditions for a Text file with conditional security.

The selection mechanism chooses secure TCP when delay is least and there is maximum available bandwidth. In all other cases it chooses non secure protocols. With security additionally provided we see that there is a considerable drop in throughput in that case, which is a trade off for the security provided. The time overhead is of about 7 seconds during the transfer of a 5Mb file with security provided all throughout its transmission. Also, security is provided when the user specifically asks for it, in this case we choose a secure version of the protocol selected. The following graph shows the result of providing security throughout the session in case of video files, and the comparison has been done with secure version of TCP and SCTP.

N/w Conditions	UDP (KBps)	SCTP (KBps)	X: Video (KBps)
DH, BM	118	127	129
DM, BM	135	130	133
DM, BL	129	127	130
DL, BH	149	143	151
DL, BM	150	144	147
DH, BL	114	122	125
DL, BH	144	140	147
DM, BH	133	136	134
DH, BH	126	127	125

Table 8 Readings for random n/w conditions for Video files with security provided throughout the session.

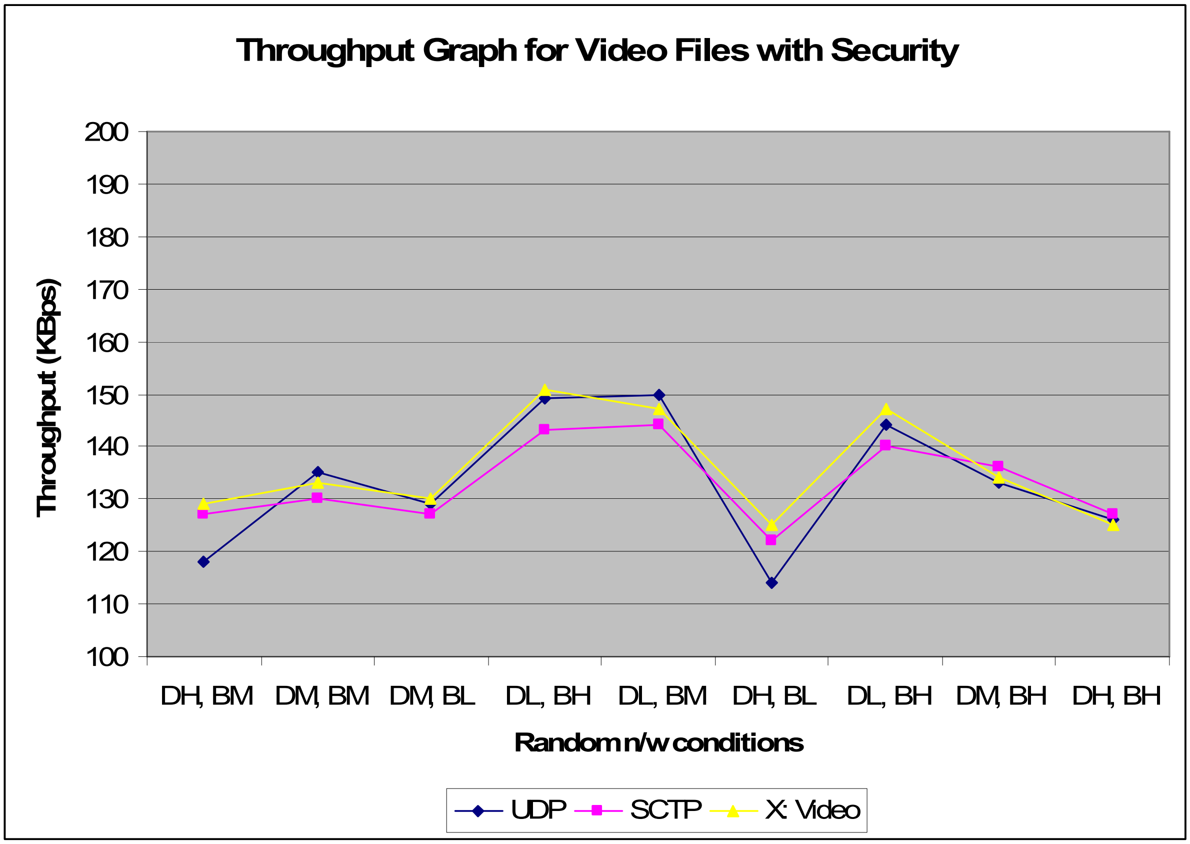

Figure 18 Throughput graph for random network conditions for Video with security provided throughout the session.

Similar results were obtained in the case of using audio and text files. There is large drop in throughput achieved by the additional security provided. This is because of the overhead involved in encrypting the packets just before they are sent across the network and decrypting the secure packets at the client side.

5.2 Packet loss readings

In this section we have provided the packet loss data when using UDP, in the case of video and audio files. Since SCTP and TCP do are connection oriented protocols and do not experience any packet loss, comparison is only done with UDP.

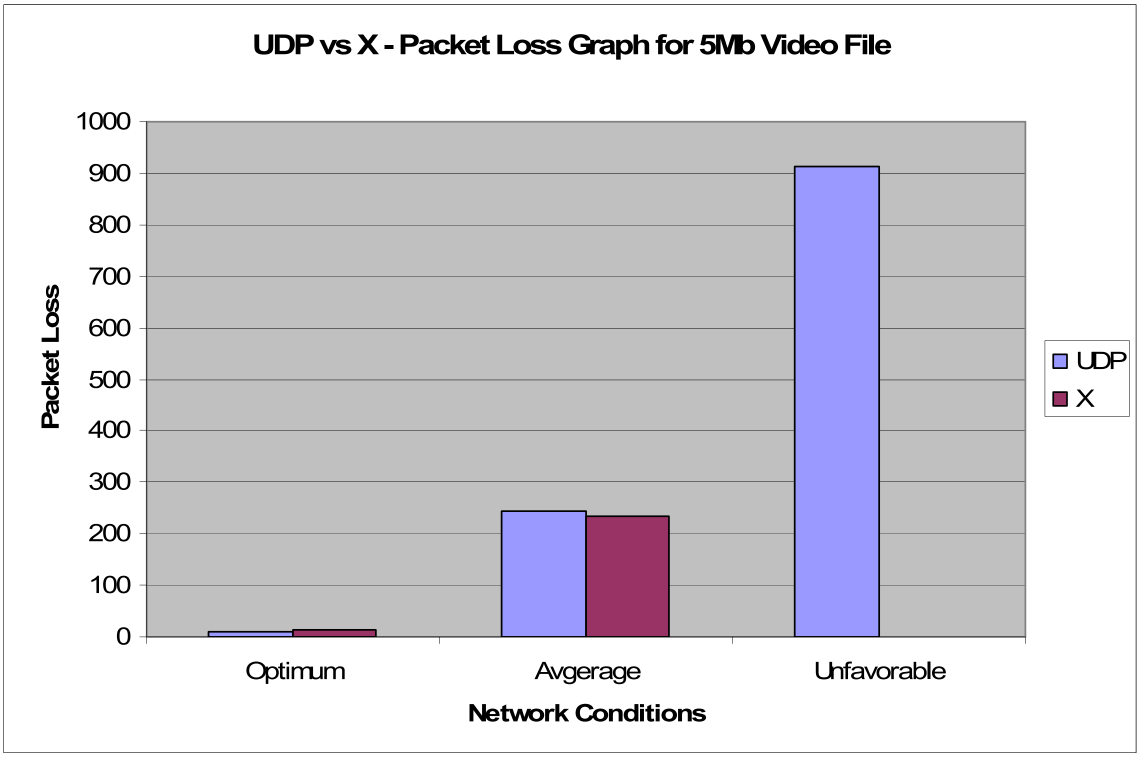

Figure 19 Packet Loss graph for a Video File.

Here we have shown the packet loss value when a 5Mb video file is transferred between the server and the client in three scenarios when using standard UDP and when using our system which switches to SCTP to avoid packet loss only in the last case. High packet loss is the reason for huge drop in UDP's throughput in such conditions.

The following graph shows the same parameter for an audio file transfer in which it is seen that since our system switches to SCTP whenever medium or worst case network conditions arise so the packet loss in case of an audio file transfer is negligibly low when compared to UDP.

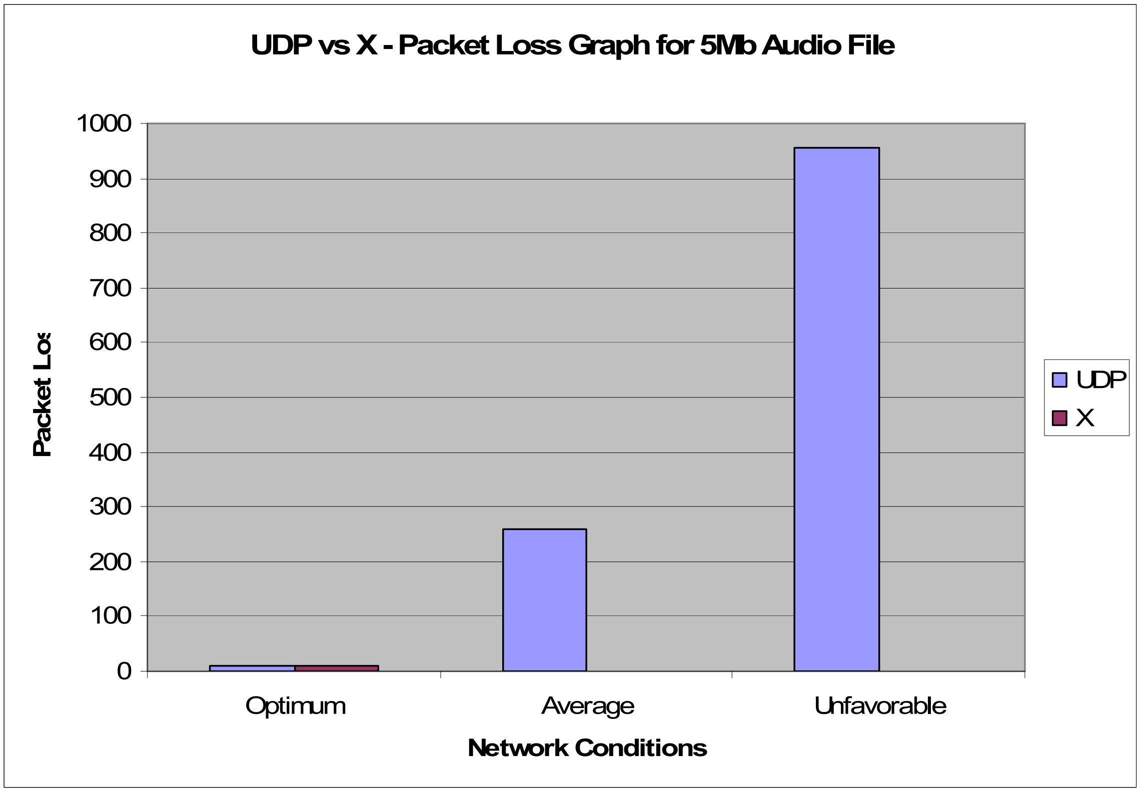

Figure 20 Packet loss graph for an Audio File.

X does not loose any packets in the Average and worst conditions as it is using one of the connection oriented protocol.

In case of text files since we are not using non-connection oriented protocols, hence there is no packet loss during their transmission.

CHAPTER VI

CONCLUSION

In case of all the three type of files the throughput achieved by our system was equivalent or greater than the throughput achieved for the file transfer using a single protocol, under most conditions. The throughput and packet loss graphs show that there is considerable increase in file transfer rate and better utilization of the network bandwidth. It is also shown that adaptation system can be used with a variety of protocols, such as all secure protocols, or all non secure protocols or a mix of both. Adaptation policy here takes in to account different QoS parameters and human perception in to consideration for each type of file. While Video files would benefit if a protocol providing higher bandwidth is used, Text file transfer need preference be given to those protocols which are reliable. Audio files requirements may lie intermediate to the Video and Text files.

Having a set of pre-configured protocols negotiated by the client and the server would hence allow the communication to take place between them using a suitable protocol and not be restricted to a single protocol as configuration of connection parameters can be done on the fly. Adaptation decision is very small overhead though there is an initial overhead to train the neural network but being completely offline it does not affect the performance of the system. It is also noted that our adaptation technique may not be always beneficial as is in a case where a certain protocol clearly performs better than the remaining protocols under all network conditions.

6.1 Future Work

Further research associated with this thesis may focus on the following areas:

a) Testing the effect of such adaptation in a multicast scenario: Multicasting refers to streaming data (generally multimedia) to more than a single host simultaneously. In this scenario it is observed that each of the individual clients will have separate network conditions and might need the server to use a more suitable protocol according to the network conditions prevailing in their link. Here the adaptation mechanism needs to take into account the prevailing network conditions at the server, the client requests and also provide the pre-set QoS requirements.

b) Increasing the QoS input parameters to the neural network: Currently we are using two parameters but more input parameters such as jitter in network would allow the mechanism to make better decisions. If the structure of the neural network is increased and hence their capability to take more number of inputs, the network can be more efficiently trained for selecting a more appropriate protocol.

c) Hardware support: The neural network adaptation mechanism might be suited to be implemented in the hardware level, to provide a much faster adaptation mechanism then compared to the software implementation, if its structure grows in size. The Field Programmable Gate Arrays – FPGA technology is an available option for such research.

REFERENCES

[1] R. Fish, G. Ghinea, and J. Thomas: “Mapping Quality of Perception to quality of service for a runtime adaptable communication system”, *Proc. SPIE ACM SIGMultimedia conference on Multimedia Computing and Networks*, Vol. 3654, San Jose, January 1999.

[2] Adam Wierman, Takayuki Osogami, and Olsen Olsen: "A Unified Framework for Modeling TCP-Vegas, TCP-SACK, and TCP-Reno," *mascots*, p. 269, *11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems* (MASCOTS'03), 2003.

[3] Bandwidth measurement algorithms, http://mosquitonet.stanford.edu/~laik/projects /nettimer/ publications/infocom1999/html/node4.html [Last accessed: 20th September 2005].

[4] D. Ritchey: “A Stream Input Output System”, *AT&T Bell Labs Technical Journal*, pp.311-324, Vol. 63(8), October 1984.

[5] Robert Simon Zachary Fish: “An Integrated Framework for Runtime Adaptable Communication Systems”, Ph.D. thesis, Department of Computer Science, University of Reading, 2001.

[6] B. Stiller, D. Bauer, G. Caronni, C. Class, C. Conrad, B. Plattner, M. Vogt and M. Waldvogel: DaCaPo++: “Communication Support for Distributed Applications”, Technical report, Computer Engineering and Networks Laboratory (TIK), ETH Zurich.

[7] N. Hutchinson and L. Peterson: “The x-kernel: An architecture for implementing network protocols”, *IEEE Transactions On Software Engineering*, Vol. 17(1), pp. 64-76, January 1991.

[8] X-kernel research project, University of Arizona, http://www.cs.arizona.edu/xkernel/ [Last accessed: 24th September 2005].

[9] D. Schmidt: “An Object-Oriented Framework for Experimenting with Alternative Process Architectures for Parallelizing Communication Subsystems”, Ph.D. thesis Department of Information and Computer Science, University of California, Irvine, 1995.

[10] Back Propagation in Neural networks, http://www.speech.sri.com/people/anand/771/html/node37.html [Last accessed: 20th September 2005].

[11] IETF, SIGTRAN working group, http://www.ietf.org/html.charters/sigtran-charter.html [Last accessed: 30th September 2005].

[12] Lksctp tools package installation notes, http://lksctp.sourceforge.net/INSTALL [last accessed: 15th September 2005].

[13] S. Saxena, J. Peacock, F. Yang, V. Verma and M. Krishnan: “Pitfalls in Multithreading SVR4 STREAMS and other weightless processes”, *Proc. Winter USENIX Conference*, pp. 85-106, San Diego, CA, January 1993.

[14] Ethereal network monitoring tool, www.ethereal.com [Last accessed: 20th September 2005].

[15] Wei Lian, and Wenye Wang: “An analytical study on the impact of authentication in wireless local area networks, *Proc. IEEE ICCCN*, 2004.

VITA

Madhukar Alli

Candidate for the Degree of

Master of Science

Thesis: A PROTOCOL STACK FOR ADAPTABLE AND SECURE COMMUNICATIONS

Major Field: Computer Science

Biographical:

Personal Data: Born in Jaggayapet, Andhra Pradesh, India, on August 30, 1982, the son of Mr. A.V. Brahmam and Mrs. A Jyothi.

Education: Obtained Senior High School Diploma from Andhra Education Society, Delhi, India in May 1999. Received Bachelor of Engineering in Computer Science & Engineering from M.S.Ramaiah Institute of Technology - Viveswaraiah Technological University, Bangalore, India in August 2003. Completed the requirements for the Master of Science Degree at Oklahoma State University in May 2006.

Experience: Teaching Assistant, CS department OSU, Spring 2005. Part-time lab monitor at HES computer labs, Oklahoma State University.

Name: Madhukar Alli Date of Degree: May 2006

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A PROTOCOL STACK FOR ADAPTABLE AND SECURE COMMUNICATIONS

Pages in Study: 54 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study:

Modern applications using network communications demand a large set of transport service requirements. Although these requirements might be initially met at the server's end, traffic variations at intermediate nodes effect the overall connection requirements thus resulting in non-optimal use of available network resources. Our thesis proposes building a protocol stack so as to be able to have flexibility in selecting a more suited protocol according to the current network conditions and deliver better communication in terms of bandwidth utilization and QoS provided.

Approach, Findings and Conclusions:

In our approach we have implemented a set of protocols where each protocol is a slight variation of the other in terms of one or more QoS parameters and security requirements. The protocol to be used is dynamically decided based on traffic conditions such as delay; bandwidth and also security requirements, in other words the system adapts itself depending on the current traffic conditions and user preferences. The adaptation, which involved configuring the protocol from the stack, was achieved by training based learning and adaptation using neural networks. The neural network used is a feed forward network which forwards the inputs to the output layer and the error is back propagated to the initial layer till the desired output is received. When the system is initiated, the network is measured for current conditions which are used as parameters in the decision making process and the client and server connections are switched simultaneously to the protocol selected. This automation enables the system to be self configuring and to adapt to the changing network conditions without human intervention.

For our results we have used the system for different type of files which differ in terms of QoS requirement. Our results demonstrate that adapting to a better protocol as the network experiences fluctuations, result in higher utilization of the available bandwidth, less packet loss and hence help to provide better QoS for the connection.

Advisor's Approval: Dr. Johnson P Thomas