Load Balancing On Destruction Hybrid Wireless Network Using DTRP For High Bandwidth Data Transmission In Mobile Sensor Network

Raghavendiran N
Jayam College of Engineering and Technology

I. INTRODUCTION

1.1 AN OVERVIEW

In the hybrid WSN, node of energy consumption is important for every sensor node because it extends hybrid WSN life. The Wireless sensor network is a collection of all sensors which spread over huge geographic area. As sensors are spread in large area and huge in number, the occurrences of faults in the network are also find. Hence to find out the fault node and to replace the fault node an algorithm is proposed. This paper proposes different algorithm to increase the lifetime of a hybrid wireless sensor networks when some of the sensor nodes fail down using the algorithm can result in some replacements of sensor nodes and used routing path. Thus, the algorithm enhances the hybrid WSN lifetime and reduces the change of the sensor nodes. A hybrid wireless network combination of a mobile ad-hoc network and an infrastructure wireless network and finally enhances the capacity of a wide area wireless network. Routing protocol is an important component that affects the strength of a wireless network in data transmission. Routing path in hybrid wireless networks combination of the cellular Transmission Mode (BSTransmission Mode) in Ad-Hoc transmission mode and infrastructure wireless networks the in mobile ad- hoc networks.

1. Load Balancing Algorithm: It propose a load balancing scheme called iCAR for cellular networks, which places ad hoc relay nodes at strategic locations to relay traffic from congested cells to noncongested ones.
2. Wireless Network with RRP algorithm: It consider the Multistage Multiplane Clos-Network based switch by Chao et a. It is designed IN five stages of switch modules with top-level architecture same as to external input or output ports. The first and final stages Clos are contain of input De-Multiplexers and output multiplexers, having similar internal structures and various wireless sensors. This algorithm generates the grade number and routing table, a set of acquaintance nodes and payload value each sensor node.

1.2 MODULES

- Server
- Client
- DTR
- Load Balncing

1.2.1 MODULE DESCRIPTION:

1. Sever

Server is the souce mechine. it is usused to send the data from the admin. those datss datas are sending through acknlodgemnt. Server can monitoring the client data through routing path. Since BSes are connected with a wired backbone, we assume that there are no bandwidth and power constraints on transmissions between BSes. We use intermediate nodes to denote relay nodes that function as gateways connecting an infrastructure wireless network and a mobile ad-hoc network.We assume every mobile node is dual-mode; that is, it has ad-hoc network interface such as a WLAN radio interface and infrastructure network interface. DTR aims to shift the routing burden from the adhoc network to the infrastructure network by taking advantage of widespread base stations in a hybrid wireless network. Rather than using one multi-hop path to forward a message to one BS, DTR uses at most two hops to relay the segments of a message to different BSes in a distributed manner, and relies on BSes to combine the segments.

2. Client

Client is a destination mechine,to receive the server data sequentially. It receives the data through router form of packets. In this module, we develop it in Router. When a source node wants to transmit a message stream to a destination node, it divides the message stream into a number of partial streams called segments and transmits each segment to a neighbor node. Upon receiving a segment from the source node, a neighbor node locally decides between direct transmission and relay transmission based on the QoS requirement of the application. The neighbor nodes forward these segments in a distributed manner to nearby BSes. Relying on the infrastructure network routing, the BSes further transmit the segments to the BS where the destination node resides. The final BS rearranges the segments into the original order and forwards the segments to the destination. It uses the cellular IP transmission method to send segments to the destination if the destination moves to another BS during segment transmission.

A long routing path will lead to high overhead, hot spots and low reliability. Thus, DTR tries to limit the path length. It uses one hop to forward the segments of a message in a distributed manner and uses another hop to find high-capacity forwarder for high performance routing.

3. DTR

DTR is a Distributed Three-hop Routing protocol, it is used to splitting the data stream into segments and transmits the segments in a distributed manner. It also reduces overhead due to short path lengths and the elimination of route discovery and maintenance. As a result, DTR limits the path length of uplink routing to two hops in order to avoid the problems of long-path multi-hop routing in the ad-hoc networks. Specifically, in the uplink routing, a source node initially divides its message stream into a number of segments, then transmits the segments to its neighbor nodes. The neighbor nodes forward segments to BSes, which will forward the segments to the BS where the destination resides.Below, we first explain how to define capacity, then introduce the way for a node to collect the capacity information from its neighbors, and finally present the details of the DTR routing algorithm.

4. Load balncing

This is last module in this project for balancing the load using conjestion control algorithm. these divides the huge load between the base stations for send the data very fast and efficient manner. We propose a congestion control algorithm to avoid overloading BSes in uplink transmission and downlink transmission, respectively. In order to reduce the broadcasting overhead, a mobile node residing in the region of a BS not close to the destination BS drops the query. The nodes can determine their approximate relative positions to BSes by sensing the signal strengths from different BSes. Each node adds the strength of its received signal into its beacon message that is periodically exchanged between neighbor nodes so that the nodes can identify their relative positions to each other. Only those mobile nodes that stay farther than the query forwarder from the forwarder’s BS forward the queries in the direction of the destination BS. In this way, the query can be forwarded to the destination BS faster.

1.3 OBJECTIVES

- DTR also has a congestion control algorithm to balance the traffic load between the nearby BSes in order to avoid traffic congestion at BSes.
- Using self-adaptive and distributed routing with high-speed and short-path ad-hoc transmission, DTR significantly increases the throughput capacity and scalability of hybrid wireless networks
- This project proposed using ad-hoc relay stations to dynamically relay traffic from one cell to another in order to avoid traffic congestion in BSes.
- We use intermediate n- odes to denote relay nodes that function as gateways connecting an infrastructure wireless network and a mobile ad-hoc network.

II. SYSTEM STUDY

2.1 EXISTING SYSTEM

A hybrid wireless network synergistically combines an infrastructure wireless network and a mobile adhoc network to leverage their advantages and overcome their shortcomings, and finally increases the throughput capacity of a wide-area wireless network. A routing protocol is a critical component that affects the throughput capacity of a wireless network in data transmission. Most current routing protocols in hybrid wireless networks simply combine the cellular transmission mode (i.e. BS transmission mode) in infrastructure wireless networks and the ad-hoc transmission mode in mobile ad-hoc networks.
The protocols use the multi-hop routing to forward a message to the mobile gateway nodes that are closest to the BSes or have the highest bandwidth to the BSes. The bandwidth of a channel is the maximum throughput (i.e., transmission rate in bits/s) that can be achieved. The mobile gateway nodes then forward the messages to the BSes, functioning as bridges to connect the ad-hoc network and the infrastructure network.

2.1.1 DRAWBACKS OF EXISTING SYSTEM

- Direct combination of the two transmission modes inherits the following problems that are rooted in the ad-hoc transmission mode.
- High overhead: Route discovery and maintenance incur high overhead. The wireless random access medium access control (MAC) required in mobile ad-hoc networks, which utilizes control handshaking and a back-off mechanism, further increases overhead.
- Hot spots: The mobile gateway nodes can easily become hot spots. The RTS-CTS random access, in

which most traffic goes through the same gateway, and the flooding employed in mobile ad-hoc routing to discover routes may exacerbate the hot spot problem. In addition, mobile nodes only use the channel resources in their route direction, which may generate hot spots while leave resources in other directions under-utilized. Hot spots lead to low transmission rates, severe network congestion, and high data dropping rates.

- Low reliability: Dynamic and long routing paths lead to unreliable routing. Noise interference and neighbor interference during the multi-hop transmission process cause a high data drop rate. Long routing paths increase the probability of the occurrence of path breakdown due to the highly dynamic nature of wireless ad-hoc networks.

2.2 PROPOSED SYSTEM

Considering the widespread BSes, the mobile nodes have a high probability of encountering a BS while moving. Taking advantage of this feature, we propose a Distributed Three-hop Data Routing protocol (DTR). In DTR a source node divides a message stream into a number of segments. Each segment is sent to a neighbor mobile node. Based on the QoS requirement, these mobile relay nodes choose between direct transmission or relay transmission to the BS. In relay transmission, a segment is forwarded to another mobile node with higher capacity to a BS than the current node. In direct transmission, a segment is directly forwarded to a BS.

In the infrastructure, the segments are rearranged in their original order and sent to the destination. The number of routing hops in DTR is confined to three, including at most two hops in the ad-hoc transmission mode and one hop in the cellular transmission mode. To overcome the aforementioned shortcomings, DTR tries to limit the number of hops. The first hop forwarding distributes the segments of a message in different directions to fully utilize the resources, and the possible second hop forwarding ensures the high capacity of the forwarder. DTR also has a congestion control algorithm to balance the traffic load between the nearby BSes in order to avoid traffic congestion at BSes.

2.2.1 ADVANTAGES OF PROPOSED SYSTEM

- Using self-adaptive and distributed routing with high speed and short-path ad-hoc transmission, DTR significantly increases the throughput capacity and scalability of hybrid wireless networks by overcoming the three shortcomings of the previous routing algorithms. It has the following features:
- Low overhead: It eliminates overhead caused by route discovery and maintenance in the ad-hoc transmission mode, especially in a dynamic environment.
- Hot spot reduction: It alleviates traffic congestion at mobile gateway nodes while makes full use of channel resources through a distributed multi-path relay.
- High reliability: Because of its small hop path length with a short physical distance in each step, it alleviates noise and neighbor interference and avoids the adverse effect of route breakdown during data transmission. Thus, it reduces the packet drop rate and makes full use of spacial reuse, in which several source and destination nodes can communicate simultaneously without interference.

III. SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS

• System	:	Core i5
• Hard Disk	:	500 GB
• Monitor	:	17 LED
• Mouse	:	Optical
• Ram	:	4GB

3.2 SOFTWARE REQUIREMENTS

• Operating system	:	Windows XP/7.
• Coding Language	:	C#.net
• Tool	:	Visual Studio 2010

IV. SOFTEWRE ANALYSIS

4.1. Net framework

The .NET Framework is many things, but it is worthwhile listing its most important aspects. A Platform designed from the start for writing Internet-aware and Internet-enabled applications that embrace and adopt open standards such as XML, HTTP, and SOAP.

A Platform that provides a number of very rich and powerful application development technologies, such as Windows Forms, used to build classic GUI applications, and of course ASP.NET, used to build web applications.

A Platform with an extensive class library that provides extensive support for date access (relational and

XML), director services, message queuing, and much more. A platform that has a base class library that contains hundreds of classes for performing common tasks such as file manipulation, registry access, security, threading, and searching of var char using regular expressions.

A platform that doesn't forget its origins, and has great interoperability support for existing components that you or third parties have written, using COM or standard DLLs.

4.1.1 Interoperability

Because computer systems commonly require interaction between newer and older applications, .NET Framework provides means to access functionality implemented in newer and older programs that execute outside .NET environment. Access to COM components is provided in System.Runtime.InteropServices andSystem.EnterpriseServices namespaces of the framework; access to other functionality is achieved using the P/Invoke feature.

4.1.2 Common Language Runtime engine

Common Language Runtime (CLR) serves as the execution engine of .NET Framework. All .NET programs execute under the supervision of CLR, guaranteeing certain properties and behaviors in the areas of memory management, security, and exception handling.

4.1.3 Language independence

.NET Framework introduces a Common Type System, or CTS. CTS specification defines all possible datatypes and programming constructs supported by CLR and how they may or may not interact with each other conforming to Common Language Infrastructure (CLI) specification. Because of this feature, .NET Framework supports the exchange of types and object instances between libraries and applications written using any conforming .NET language.

4.1.4Framework Class Library

Framework Class Library (FCL) is a library of functionality available to all languages using .NET Framework. FCL provides classes that encapsulate a number of common functions, including file reading and writing, graphicrendering, database interaction, XML document manipulation, and so on. It consists of classes, interfaces of reusable types that integrates CLR.

4.1.5 Simplified deployment

.NET Framework includes design features and tools which help manage the installation of computer software to ensure that it does not interfere with previously installed software, and that it conforms to security requirements.

4.2 Common Language Runtime (CLR)

The CLR is described as the "execution engine" of .NET. It provides the environment within which programs run. The most important features are

- Conversion from a low-level assembler-style language, called Intermediate Language (IL), into code native to the platform being executed on.
- Memory management, notably including garbage collection.
- Checking and enforcing security restrictions on the running code.
- Loading and executing programs, with version control and other such features.
- The following features of the .NET framework are also worth description:

4.21 Managed Code

The code that targets .NET, and which contains certain extraInformation - "metadata" - to describe itself. Whilst both managed and unmanaged code can run in the runtime, only managed code contains the information that allows the CLR to guarantee, for instance, safe execution and interoperability.

4.2.2 Managed Data

With Managed Code comes Managed Data. CLR provides memory allocation and Deal location facilities, and garbage collection. Some .NET languages use Managed Data by default, such as C#, Visual Basic.NET and JScript.NET, whereas others, namely C++, do not. Targeting CLR can, depending on the language you're using, impose certain constraints on the features available. As with managed and unmanaged code, one can have both managed and unmanaged data in .NET applications - data that doesn't get garbage collected but instead is looked after by unmanaged code.

4.2.3 Common Type System

The CLR uses something called the Common Type System (CTS) to strictly enforce type-safety. This ensures that all classes are compatible with each other, by describing types in a common way. CTS define how types work within the runtime, which enables types in one language to interoperate with types in another language, including cross-language exception handling. As well as ensuring that types are only used in appropriate ways, the runtime also ensures that code doesn't attempt to access memory that hasn't been allocated to it.

4.2.4 Common Language Specification

The CLR provides built-in support for language interoperability. To ensure that you can develop managed code that can be fully used by developers using any programming language, a set of language features and rules for using them called the Common Language Specification (CLS) has been defined. Components that follow these rules and expose only CLS features are considered CLS-compliant.

4.2.5 Class Library

.NET provides a single-rooted hierarchy of classes, containing over 7000 types. The root of the namespace is called System; this contains basic types like Byte, Double, Boolean, and String, as well as Object. All objects derive from System. Object. As well as objects, there are value types. Value types can be allocated on the stack, which can provide useful flexibility. There are also efficient means of converting value types to object types if and when necessary.

The set of classes is pretty comprehensive, providing collections, file, screen, and network I/O, threading, and so on, as well as XML and database connectivity.

The class library is subdivided into a number of sets (or namespaces), each providing distinct areas of functionality, with dependencies between the namespaces kept to a minimum.

4.3 C# Introduction

C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. You can use C# to create Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much, much more. Visual C# provides an advanced code editor, convenient user interface designers, integrated debugger, and many other tools to make it easier to develop applications based on the C# language and the .NET Framework.

C# syntax is highly expressive, yet it is also simple and easy to learn. The curly-brace syntax of C# will be instantly recognizable to anyone familiar with C, C++ or Java. Developers who know any of these languages are typically able to begin to work productively in C# within a very short time. C# syntax simplifies many of the complexities of C++ and provides powerful features such as nullable value types, enumerations, delegates, lambda expressions and direct memory access, which are not found in Java. C# supports generic methods and types, which provide increased type safety and performance, and iterators, which enable implementers of collection classes to define custom iteration behaviors that are simple to use by client code. Language-Integrated Query (LINQ) expressions make the strongly-typed query a first-class language construct.

As an object-oriented language, C# supports the concepts of encapsulation, inheritance, and polymorphism. All variables and methods, including the Main method, the application's entry point, are encapsulated within class definitions. A class may inherit directly from one parent class, but it may implement any number of interfaces. Methods that override virtual methods in a parent class require the **override** keyword as a way to avoid accidental redefinition. In C#, a struct is like a lightweight class; it is a stack-allocated type that can implement interfaces but does not support inheritance.

In addition to these basic object-oriented principles, C# makes it easy to develop software components through several innovative language constructs, including the following:

- Encapsulated method signatures called delegates, which enable type-safe event notifications.
- Properties, which serve as accessors for private member variables.
- Attributes, which provide declarative metadata about types at run time.
- Inline XML documentation comments.
- Language-Integrated Query (LINQ) which provides built-in query capabilities across a variety of data sources.

If you have to interact with other Windows software such as COM objects or native Win32 DLLs, you can do this in C# through a process called "Interop." Interop enables C# programs to do almost anything that a native C++ application can do. C# even supports pointers and the concept of "unsafe" code for those cases in which direct memory access is absolutely critical.

The C# build process is simple compared to C and C++ and more flexible than in Java. There are no separate header files, and no requirement that methods and types be declared in a particular order. A C# source file may define any number of classes, structs, interfaces, and events.

The following are additional C# resources:

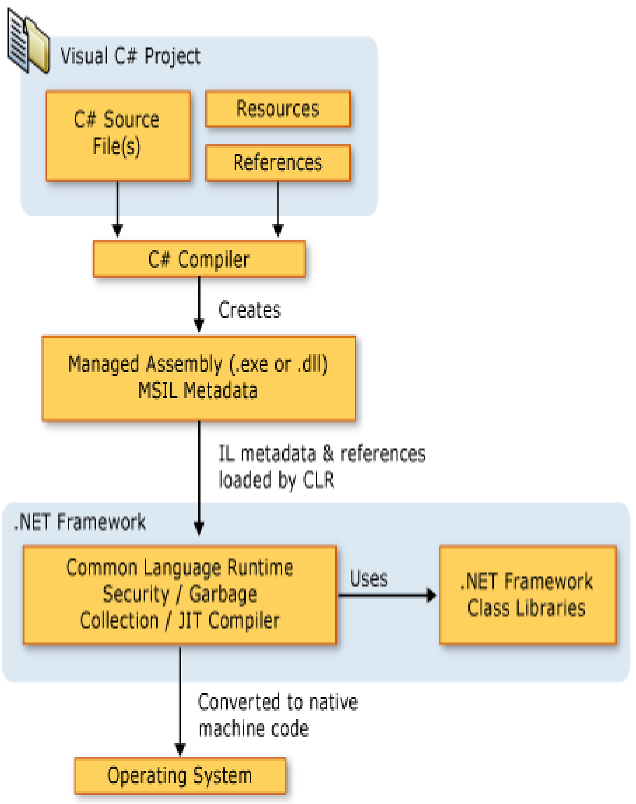
- For a good general introduction to the language, see Chapter 1 of the C# Language Specification.
- For detailed information about specific aspects of the C# language, see the C# Reference.
- For more information about LINQ, see LINQ (Language-Integrated Query).
- To find the latest articles and resources from the Visual C# team, see the Visual C# Developer Center.

5.4.1 .NET Framework Platform Architecture

C# programs run on the .NET Framework, an integral component of Windows that includes a virtual execution system called the common language runtime (CLR) and a unified set of class libraries. The CLR is the commercial implementation by Microsoft of the common language infrastructure (CLI), an international standard that is the basis for creating execution and development environments in which languages and libraries work together seamlessly.

Source code written in C# is compiled into an intermediate language (IL) that conforms to the CLI specification. The IL code and resources, such as bitmaps and strings, are stored on disk in an executable file called an assembly, typically with an extension of .exe or .dll. An assembly contains a manifest that provides information about the assembly's types, version, culture, and security requirements.

When the C# program is executed, the assembly is loaded into the CLR, which might take various actions based on the information in the manifest. Then, if the security requirements are met, the CLR performs just in time (JIT) compilation to convert the IL code to native machine instructions. The CLR also provides other services related to automatic garbage collection, exception handling, and resource management. Code that is executed by the CLR is sometimes referred to as "managed code," in contrast to "unmanaged code" which is compiled into native machine language that targets a specific system. The following diagram illustrates the compile-time and run-time relationships of C# source code files, the .NET Framework class libraries, assemblies, and the CLR.


Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

Language interoperability is a key feature of the .NET Framework. Because the IL code produced by the C# compiler conforms to the Common Type Specification (CTS), IL code generated from C# can interact with code that was generated from the .NET versions of Visual Basic, Visual C++, or any of more than 20 other CTS-compliant languages. A single assembly may contain multiple modules written in different .NET languages, and the types can reference each other just as if they were written in the same language.

In addition to the run time services, the .NET Framework also includes an extensive library of over 4000 classes organized into namespaces that provide a wide variety of useful functionality for everything from file input and output to string manipulation to XML parsing, to Windows Forms controls. The typical C# application uses the .NET Framework class library extensively to handle common "plumbing" chores.
For more information about the .NET Framework, see Overview of the Microsoft .NET Framework.

5.4.2 C# (C Sharp) Meaning

C# is a general object-oriented programming (OOP) language for networking and Web development. C# is specified as a common language infrastructure (CLI) language.

In January 1999, Dutch software engineer Anders Hejlsberg formed a team to develop C# as a complement to Microsoft’s NET framework. Initially, C# was developed as C-Like Object Oriented Language (Cool). The actual name was changed to avert potential trademark issues. In January 2000, NET was released as C#. Its NET framework promotes multiple Web technologies.

C# and .NET Programming demonstrates a very simple C# program that prints the text string "Hello World!" to the console screen and provides a line-by-line analysis of that program. However, even that simple program was complex enough that we had to skip some of the details. In this chapter, we'll begin an in-depth exploration of the syntax and structure of the C# language. The syntax of a language is the order of the keywords, where you put semicolons, and so forth. The semantics are what you are expressing in the code, and how your code fits together. Syntax is trivial and unimportant, but because compilers are absolute sticklers for correct syntax, novice programmers pay a lot of attention to syntax until they are comfortable. Fortunately, Visual Studio 2008 makes managing syntax much easier so that you can focus on semantics, which are far more important.

In this chapter, we'll introduce statements and expressions, the building blocks of any program. You'll learn about variables and constants, which let you store values for use in your program. We'll also begin an explanation of types, and we'll take a look at strings, which you saw briefly in the Hello World program. This is all very basic stuff, but it's the foundation you need to start getting fancy. Without variables, your applications can't actually process any data. All variables need types, and variables are used in expressions. You'll see how neatly this all fits together.

C# is a modern language created by Microsoft as part of its .NET platform of languages. .NET is a layer of software that makes it easier for you to write programs that can communicate with the operating system (in this case, Windows). As the name implies, C# has its roots in C++, but over three versions, it has evolved its own techniques and elements that make it distinct. Most important, C# has the backing of the .NET Framework behind it, which we'll get into shortly. We're not going to assume that you have any C++ experience, so we won't frame our discussions of C# in terms of C++, or any other programming language. What you need to know right now is that you can write applications in C# that will do just about anything you need to do. You can write applications to manage your company's inventory (interacting with a database); you can write applications to analyze documents; you can write games; you can create an entire operating system in C# if you have a mind to. The .NET Framework allows C# to operate seamlessly with Windows, and take advantage of the familiar Windows features that users all over the world already know. You can also create C# applications that you can use on the Web, in much the same way.

To be completely honest, most modern object-oriented languages are rather similar underneath. The choice of one over the other is usually just a matter of personal preference. C# and Visual Basic have the advantage of the .NET Framework, but third-party languages can interact with the framework, too. C#'s similarity to C++ and Java makes it easy to learn for programmers familiar with those languages, but it's also easy to learn as your first language. Once you have the basics of C# down, you'll find it much easier to learn any other language you want to.

Unless we specifically say otherwise, when we refer to C# in this book, we mean C# 3.0; when we refer to .NET, we mean the .NET 3.5 Framework; and when we refer to Visual Studio, we mean Visual Studio 2008. We could spend some time telling you about the cool new features of C# 3.0 over its predecessors-and we're pretty excited about them-but if you're new to the language, it's all new to you, so there's little point in calling attention to specific features.

Finally, when we refer to using Visual Studio 2008, you may be using Visual C# 2008 Express Edition instead. C# Express is the free version of Visual Studio, designed for students and home users, but that doesn't mean it's a toy. In fact, the examples in this book were written and tested using C# Express. C# Express has the same compiler and libraries as Visual Studio, and within the examples in this book, you won't find any significant differences. There are some small differences in look and feel, or in feature names, and any time those come up, we'll mention them.

C#, as mentioned earlier, is one of the languages you can use to create applications that will run in the .NET CLR. It is an evolution of the C and C++ languages and has been created by Microsoft specifically to work with the .NET platform. The C# language has been designed to incorporate many of the best features from other languages, while clearing up their problems.

Developing applications using C# is simpler than using C++, because the language syntax is simpler. Still, C# is a powerful language, and there is little you might want to do in C++ that you can't do in C#. Having said that, those features of C# that parallel the more advanced features of C++, such as directly accessing and manipulating system memory, can be carried out only by using code marked as unsafe. This advanced programmatic technique is potentially dangerous (hence its name) because it is possible to overwrite system-critical blocks of memory with potentially catastrophic results. For this reason, and others, this book does not cover that topic. At times, C# code is slightly more verbose than C++. This is a consequence of C# being a type-safe language (unlike C++). In layperson's terms, this means that once some data has been assigned to a type, it cannot subsequently transform itself into another unrelated type. Consequently, strict rules must be adhered to when converting between types, which means you will often need to write more code to carry out the same task in C# than you might write in C++. However, you get two benefits: the code is more robust and debugging is simpler, and .NET can always track the type of a piece of data at any time. In C#, you therefore may not be able to do things such as ″take the region of memory 4 bytes into this data and 10 bytes long and interpret it as X,″ but that's not necessarily a bad thing.

C# is just one of the languages available for .NET development, but it is certainly the best. It has the advantage of being the only language designed from the ground up for the .NET Framework and is the principal language used in versions of .NET that are ported to other operating systems. To keep languages such as the .NET version of Visual Basic as similar as possible to their predecessors yet compliant with the CLR, certain features of the .NET code library are not fully supported, or at least require unusual syntax. By contrast, C# can make use of every feature that the .NET Framework code library has to offer. The latest version of .NET includes several additions to the C# language, partly in response to requests from developers, making it even more powerful.

Applications You Can Write with C#

The .NET Framework has no restrictions on the types of applications that are possible, as discussed earlier. C# uses the framework and therefore has no restrictions on possible applications. However, here are a few of the more common application types:

- Windows applications. Applications, such as Microsoft Office, that have a familiar Windows look and feel about them. This is made simple by using the Windows Forms module of the .NET Framework, which is a library of controls (such as buttons, toolbars, menus, and so on) that you can use to build a Windows user interface (UI). Alternatively, you can use Windows Presentation Foundation (WPF) to build Windows applications, which gives you much greater flexibility and power.
- Web applications. Web pages such as those that might be viewed through any Web browser. The .NET Framework includes a powerful system for generating Web content dynamically, enabling personalization, security, and much more. This system is called ASP.NET (Active Server Pages .NET), and you can use C# to create ASP.NET applications using Web Forms. You can also write applications that run inside the browser with Silverlight.
- Web services. An exciting way to create versatile distributed applications. Using Web services you can exchange virtually any data over the Internet, using the same simple syntax regardless of the language used to create a Web service or the system on which it resides. For more advanced capabilities, you can also create Windows Communication Foundation (WCF) services.

4.4 Feasibility Analysis

An important objective of conducting the system analysis is the determination of the feasibility. All projects are feasible if given unlimited resources and infinite time. But our systems have shortage of resources. It is necessary to evaluate the feasibility of a project at the earliest possible time. Feasibility and risk analysis are related in many ways. If project risk is great, the feasibility of producing quality software is reduced. During system engineering, however, we concentrate our attention on three primary areas of interest

4.4.1 Technical feasibility

It is a study of function, performance and constraints that may affect the ability to achieve an acceptable system. The proposed system is web technology based system and all the technical requirements are available with the organization and the internet technology is available for most of the people in this world. So the proposed system will definitely work with the current equipment, existing software technology, and available personnel. The proposed system is developed in such a way that, it is simple enough to understand and manipulate.

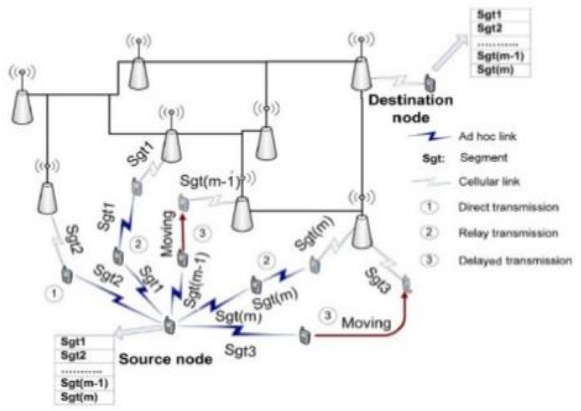
4.4.2 Operational feasibility

Will the system be used if it is developed and implemented? Will there be resistance from users that will undermine the possible application benefits? Since household items and all other consumer products are being used worldwide for human purposes there will be worldwide interest in the information about these items. Hence the system will be definitely used by the users. By considering the various factors, the proposed system gives high performance, it is reliable, maintainable. Hence the proposed system is feasible.

4.4.3 Economical feasibility

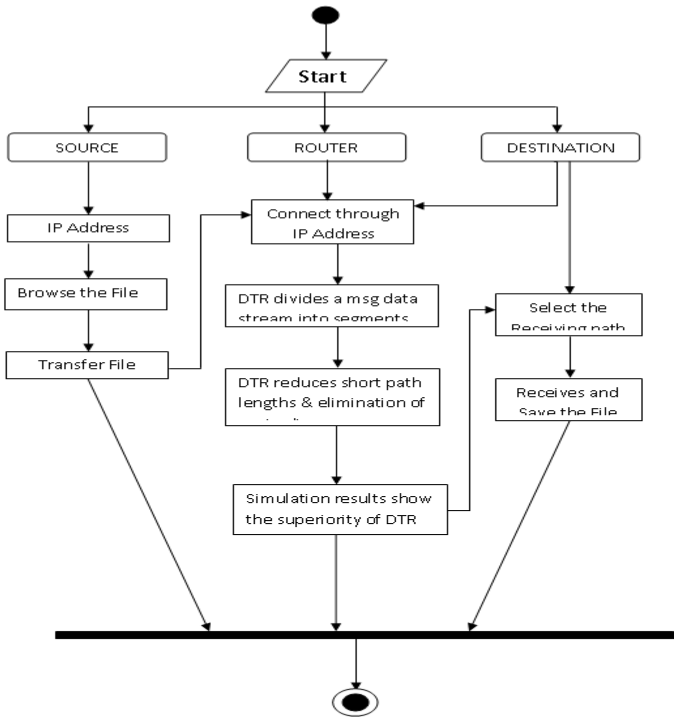
It is an evaluation of development of cost weighed against the ultimate income or benefit derived from the developed system. Economic justification includes a broad range of concerns that include cost–benefit analysis, long-term corporate income strategies, cost of resources needed for development.

V. SYSTEM ARCHITECTURE

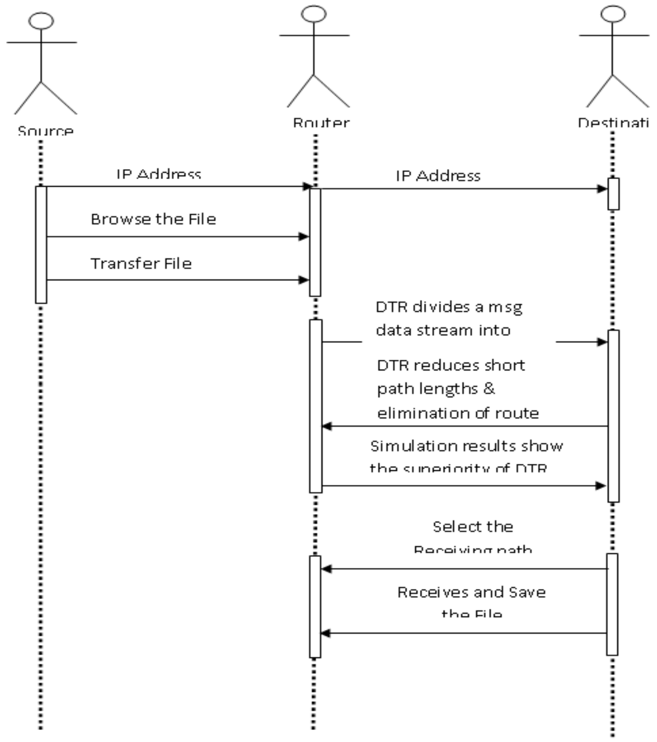


VI. SYSTEM DESIGN

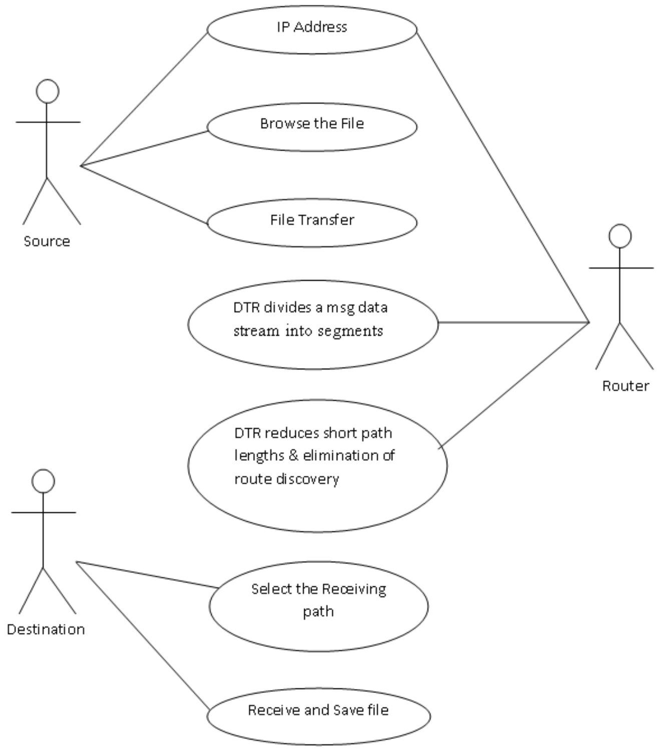
6.1 ACTIVITY DIAGRAM:


Start
SOURCE
ROUTER
DESTINATION
IP Address
Browse the File
Transfer File
Connect through IP Address
DTR divides a msg data stream into segments
DTR reduces short path lengths & elimination of
Simulation results show the superiority of DTR
Select the Receiving path
Receives and Save the File

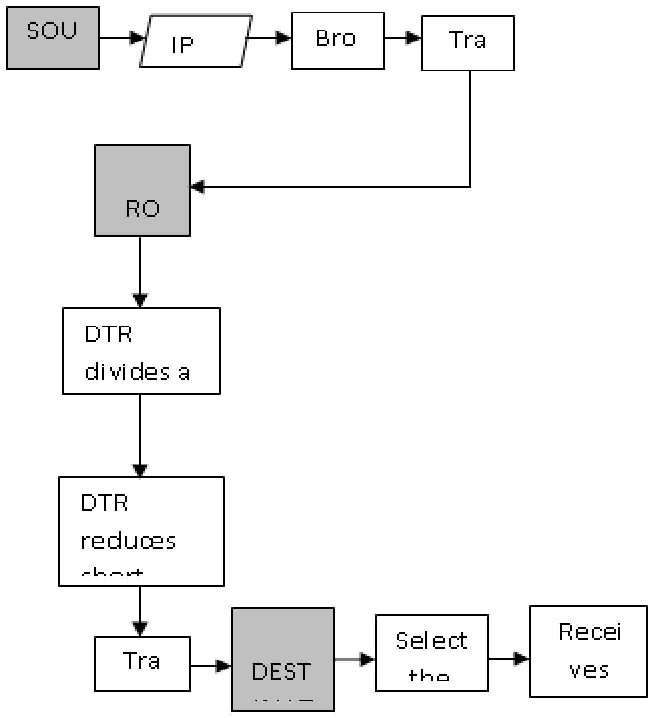
6.2 SEQUENCE DIAGRAM:


Source
Router
Destinati
IP Address
IP Address
Browse the File
Transfer File
DTR divides a msg data stream into
DTR reduces short path lengths & elimination of route
Simulation results show the superiority of DTR
Select the Receiving path
Receives and Save the File

6.3 USE CASE DIAGRAM:


IP Address
Browse the File
File Transfer
Source
DTR divides a msg data stream into segments
Router
DTR reduces short path lengths & elimination of route discovery
Destination
Select the Receiving path
Receive and Save file

6.4 SYSTEM FLOW DIAGRAM:



4. SCREEN SHOTS

Server Form

VII. SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

7.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

7.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

7.3 OUTPUT TESTING

Various outputs have been generated by the system. The system generated output and the desk-calculated values have been compared. All the output is perfect as the company desires. It begins with low volumes of transactions based on live tone. The volume is increased until the maximum level for each transaction type is reached. The total system is also tested for recovery and fallback, after various major failures to ensure that no data are lost during the emergency time.

VIII. LITERATURE SURVEY

1) Efficient resource allocation in hybrid wireless networks
AUTHORS: B. Bengfort, W. Zhang, and X. Du.

In this paper, we study an emerging type of wireless network - Hybrid Wireless Networks (HWNs). A HWN consists of an infrastructure wireless network (e.g., a cellular network) and several ad hoc nodes (such as a Mobile ad hoc network). Forming a HWN is a very cost-effective way to improve wireless coverage and the available bandwidth to users. Specifically, in this work we investigate the issue of bandwidth allocation in multi-hop HWNs. We propose three efficient bandwidth allocation schemes for HWNs: top-down, bottom-up, and auction-based allocation schemes. In order to evaluate the bandwidth allocation schemes, we develop a simulated HWN environment. Our simulation results show that the proposed schemes achieve good performance: the schemes can achieve maximum revenue/utility in many cases, while also providing fairness. We also show that each of the schemes has merit in different application scenarios.

2) Interference aware resource allocation for hybrid hierarchical wireless networks
AUTHORS: P. Thulasiraman and X. Shen.

This paper addresses the problem of interference aware resource allocation for OFDMA based hybrid hierarchical wireless networks. We develop two resource allocation algorithms considering the impact of wireless interference constraints using a weighted SINR conflict graph to quantify the interference among the various nodes: (1) interference aware routing using maximum concurrent flow optimization; and (2) rate adaptive joint subcarrier and power allocation algorithm under interference and QoS constraints. We exploit spatial reuse to allocate subcarriers in the network and show that an intelligent reuse of resources can improve throughput while mitigating interference. We provide a sub-optimal heuristic to solve the rate adaptive resource allocation problem. We demonstrate that aggressive spatial reuse and fine tuned-interference modeling garner advantages in terms of throughput, end-to-end delay and power distribution.

3) A hybrid network model for wireless packet data networks
AUTHORS: H. Y. Hsieh and R. Sivakumar

We propose a hybrid network model called Sphinx for cellular wireless packet data networks. Sphinx uses a peer-to-peer network model in tandem with the cellular network model to achieve higher throughput and lower-power consumption. At the same time, Sphinx avoids the typical pitfalls of the pure peer-to-peer network model including unfair resource allocation, and throughput degradation due to mobility and traffic locality. We present simulation results showing that Sphinx outperforms the cellular network model in terms of throughput and power consumption, and achieves better fairness and resilience to mobility than the peer-to-peer network model.

4) Multihop cellular networks: Technology and economics
AUTHORS: X. J. Li, B. C. Seet, and P. H. J. Chong

Recently, multihop cellular networks (MCNs) were proposed to preserve the advantages of traditional single-hop cellular networks with multihop ad hoc relaying networks, while minimizing the drawbacks that they involved. In this way, MCNs enhance the performance of both the existing cellular networks and ad hoc networks. Consequently, MCN-type system is considered as a promising candidate of fourth generation (4G) wireless network for future mobile communications. This paper surveys a number of MCN-type architectures in literature through a comprehensive comparison and discussion among the proposed architectures. The discussion is divided into two phases. In the first phase, we review the concept of MCN and compare the selected MCN-type architectures from a technology perspective. In the second phase, we further compare and discuss the economic perspective on the deployment of MCNs. Specifically, we focus on the economic considerations for deploying relays in MCN-type systems.

5) Dynamic source routing in ad hoc wireless networks
AUTHORS: D. B. Johnson and D. A. Maltz

An ad hoc network is a collection of wireless mobile hosts forming a temporary network without the aid of any established infrastructure or centralized administration. In such an environment, it may be necessary for one mobile host to enlist the aid of other hosts in forwarding a packet to its destination, due to the limited range of each mobile host’s wireless transmissions. This paper presents a protocol for routing in ad hoc networks that uses dynamic source routing. The protocol adapts quickly to routing changes when host movement is frequent, yet requires little or no overhead during periods in which hosts move less frequently. Based on results from a packet-level simulation of mobile hosts operating in an ad hoc network, the protocol performs well over a variety of environmental conditions such as host density and movement rates. For all but the highest rates of host movement simulated, the overhead of the protocol is quite low, falling to just 1% of total data packets transmitted for moderate movement rates in a network of 24 mobile hosts. In all cases, the difference in length between the routes used and the optimal route lengths is negligible, and in most cases, route lengths are on average within a factor of 1.01 of optimal.

IX. CONCLUSION

Hybrid wireless networks have been receiving increasing attention in recent years. A hybrid wireless network combining an infrastructure wireless network and a mobile ad-hoc network leverages their advantages to increase the throughput capacity of the system. However, current hybrid wireless networks simply combine the routing protocols in the two types of networks data transmission, which prevents them from achieving higher system capacity. In this paper, we propose a Distributed Three-hop Routing (DTR) data routing protocol that integrates the dual features of hybrid wireless networks in the data transmission process. In DTR, a source node divides a message stream into segments and transmits them to its mobile neighbors, which further forward the segments to their destination through an infrastructure network. DTR limits the routing path length to three, and always arranges for high-capacity nodes to forward data.