On the implementation and performance of the (α, t) protocol on Linux

Anandha Gopalan
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, U.S.A
Email: axgopala@cs.pitt.edu

Sanjeev Dwivedi *
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, U.S.A
Email: sanjeev@cc.gatech.edu

Taieb Znati
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, U.S.A
Email: znati@cs.pitt.edu

Bruce McDonald †
Electrical and Computer Engineering Department
Northeastern University
Boston, MA 02115, U.S.A
Email: mcdonald@ece.neu.edu

Abstract

This paper details the design and implementation of the (α, t) protocol for ad-hoc networks on Linux. The (α, t) protocol utilizes adaptive clustering to organize nodes into clusters in which the probability of path failure due to node movement can be bounded over time. Based on the (α, t) cluster scheme, routes within clusters are maintained on a proactive basis while routing between clusters is maintained on a reactive basis.

1. Introduction

With the advent of cellular communication and its prominence, wireless technology is slowly taking over the role played by LANs in yester years. Increasingly, users want to be able to access their information while on the move, thus increasing the demand for wireless networks. This rapid expansion demands the increasing of the existing infrastructure which can get quite unwielding. Wireless networks solve these problems, and hence are the choice of next generation networks.

Even though wireless connectivity is extremely useful when infrastructure is present, connectivity in absence of infrastructure is desirable. A large mass of literature is present in this area (named ad-hoc networks) and there has been demand from the military towards such architectures but no significant commercial applications had been present so far.

Currently wired computing offers a better platform for users because the Internet is designed based on it. Even though mobile computing is convenient, it still does not compare to the wired platform for users. It is our job to make sure that the Internet is changed/modified to accommodate mobile computing in such a way that the inherent convenience in the paradigm of mobile computing is not obscured by inefficient and inadequate system design.

Often it is argued that in the present scenario, ad-hoc networks do not offer significant advantages because we do not have applications that can utilize them. [2] counters the argument by stating that in the absence of infrastructure, the wireless devices themselves take on the functions yielded by them.

This scenario is changing slowly. Approaches towards infrastructure-less network solutions from large vendors like Apple's rendezvous protocol [6] and IETF's zeroconf protocols [15] are very important steps in this direction.

This necessitates giving working solutions for protocols that have been proposed in the past. Most of the work in the ad-hoc domain has remained dormant and has been forgotten over time for lack of implementation. For any protocol to be acceptable to the commercial/internet community, a real life working implementation that has been tested needs to be shown.

There are two competing objectives when designing an ad-hoc network protocol. The first one is to make sure that each node is able to synchronize itself with the network rapidly and correctly as the underlying network topology changes. Also, it is important to make this adaptation process consume the least amount of resources. These goals do not compliment each other because if a node tries to

*The work was done while this author was a student at the University of Pittsburgh

†The (α, t) protocol was designed while this author was a student at the University of Pittsburgh

adapt itself with the network rapidly, it consumes a lot of resources (processing, transmission and storage capacity). Since both of the factors are important to our purpose, we need to make a compromise somewhere. Clustering provides us with such a compromise. Clustering allows us to club together elements of the ad-hoc network that are highly mobile and hence apt to consume a lot of resources. By restricting the number of nodes that can potentially lead to dissemination of a large number of packets (traffic) we can restrict the network traffic significantly. Using clustering, we convert a highly mobile network of nodes into a comparably slow network of clusters. We assign one or more controllers that represent the cluster and help achieve stability and efficiency within a cluster.

The second factor is that of scalability. Since we do not have a fixed infrastructure, the query process to search for a node would involve a large number of nodes. This would lead to excessive use of resources and strain the network. Also, it would significantly increase the amount of storage required at each node. Hence we come across the problem of reducing the network diameter so that number of messages passed for queries become small. It also leads us to the problem of reducing the information required to keep the elements of network in synchronization. With clustering, instead of maintaining routes to all the nodes in the network, each node needs to maintain the routes of its own cluster. This decreases the storage required at each node significantly. Clusters only need to maintain information about adjacent clusters, as opposed to each node maintaining information regarding the whole network. Hence, maintaining the information needed to keep the network in synchronization abstracts to only maintaining the information as to the number of clusters present and which clusters are adjacent to each other. This abstraction of information allows cluster based protocols to scale well.

Having discussed both cluster-based and clusterless protocols, we believe that cluster-based protocols would perform better than clusterless protocols in the construct of ad-hoc networks. Even though theoretically cluster-based protocols appear to be better, they have not been much investigated and as far as we know, no implementations of a cluster-based ad-hoc routing protocol is in existence. In the course of this research we have investigated a cluster-based protocol called the (α, t) protocol. This protocol has been designed with the features of both cluster-based and clusterless protocols and tries to avoid their shortcomings. At the same time, it introduces a new metric for cluster formation which allows intra-cluster routing to be more efficient. In this research we have investigated, redesigned, implemented and analyzed the (α, t) protocol.

The (α, t) protocol might have far reaching significance towards QoS in ad-hoc networks, which has been largely non-existent from other implementations (schemes). This work is the first step towards the implementation of the (α, t) protocol and tries to build a proof of concept.

2. Requirements and Related work

A couple of requirements/system capabilities are required/called for in an efficient (and easy) implementation (all implementations thus far have implemented the following) [8]:

1. Finding out when a route is needed
2. Initiating a request
3. Queuing packets for an outstanding request
4. Re-injection of outstanding packet in the stream
5. Refreshing timers/validating routes

Ad-hoc networks have emerged in response to advances in hardware systems, availability of unlicensed radio spectrum, and frustrations over the costs and limitations of infrastructured wireless networks. As public cellular wireless system move into their third and fourth generations, wireless LANs have become important components of many corporate information infrastructures. Efforts have been underway to address many of the limitations of these emerging systems. Specification of the wireless MAC-layer protocol standard, IEEE 802.11, and the charter of the IETF MANET working group have reinforced the need for more flexible wireless networks, and thus a growing sub-field of wireless communications has taken hold, namely, wireless ad-hoc networking.

From the earliest adaptations of traditional distance vector routing proposed for the DSDV protocol [13] to sophisticated techniques that use information gathered from GPS to report and estimate node position information for the purpose of efficiently building on-demand routes [9], the published work displays a wealth of varied and interesting techniques and ideas; however, a gap remains to be filled. Specifically, none of the schemes that have been proposed have been shown to perform well enough over a wide range of environments. Consequently, the question remains as to how to efficiently support routing that is responsive to a wide range of mobility patterns, that is scalable and that can form the nucleus of a strategy capable of supporting QoS requirements in terms of throughput and delay?

The problem of routing in wireless ad-hoc networks has motivated researchers and protocol designers to re-examine the basic tenets of adaptive routing as they have evolved over the past several decades. Challenges that were faced by early routing protocol designers, including limited bandwidth and unreliable communications links are being faced once again in the context of ad-hoc communications. However, in some ways the ad-hoc routing problem is more difficult. In particular, node mobility, asymmetric channel

characteristics, and power constraints are added difficulties which must be addressed in order to implement a truly effective and commercially acceptable network architecture.

The structure of the Internet suggests that hierarchical routing is essential to achieve scalability. In ad-hoc networks, maintaining hierarchy (clusters) becomes more difficult due to the dynamic nature of the network. We believe that clustering can increase the scalability of ad-hoc networks by dividing the pro-active and reactive parts of the network into intra-cluster and inter-cluster domains.

Most of the literature on ad-hoc routing deals with reactive schemes. However, reactive schemes become extremely inefficient when the network is subject to heavy traffic loads and high mobility. This leads us to the pro-active schemes. The main arguments against pro-active schemes are: periodic updates that requires bandwidth and processing, frequently using scarce resources to maintain routes that are seldom used.

As a result of the shortcomings present in both the reactive and pro-active protocols, it is apparent that a hybrid scheme is needed. Hybrid schemes contain the features of both these methods and hence can use a pro-active scheme for high mobility elements of the network while relatively immobile elements can communicate using reactive schemes.

The Destination sequenced distance vector DSDV routing protocol [13] is a pro-active routing protocol, where each routing entry is assigned a sequence number. This helps nodes to easily distinguish between old routes (one that is no longer valid) and a new routes. Zone routing protocol (ZRP) [5] is a hybrid routing protocol that divides the network topology into overlapping zones. Routing inside a zone uses the intra-zone routing protocol (IARP) and routing between zones uses a inter-zone routing protocol (IERP). Advanced on demand distance vector protocol (AODV) [12] is a reactive protocol, wherein routes are created and maintained as and when needed. When a source requests a route to a destination, the source broadcasts a route request message (RREQ). This request is re-broadcast by the other nodes until it reaches the destination. The destination on receipt of the RREQ message replies using a request reply (RREP) message, which is sent back to the sender using the reverse path that the RREQ took. Dynamic source routing (DSR) [7] is another reactive routing protocol. This protocol is very similar to AODV, but instead of re-broadcasting the request, nodes do a limited broadcast. A limited broadcast is when a node does not broadcast a request, but discards it if it has already processed the request.

3. Architecture

The $(\alpha, t)-$Cluster framework supports a scalable routing infrastructure that is able to adapt to a changing network topology by dynamically organizing nodes into clusters and hence bounding the impact of routing overhead.

The $(\alpha, t)-$Cluster framework introduces a probabilistic metric to provide a bound on the availability of paths inside a cluster. This metric allows for the dynamic balancing of the trade offs according to temporal and spatial dynamics of the network.

Intra cluster routing is done on a pro-active basis using a table driven pro-active routing algorithm. The $(\alpha, t)-$Cluster framework is flexible and independent of the specific intra-cluster routing algorithm and hence, any pro-active routing algorithm designed for ad-hoc networks can be used for routing within a cluster.

Inter cluster routing strategy tries to take advantage of the cluster topology and the intra-cluster routing tables. The Inter Cluster Routing Protocol (ICRP) is a fully reactive cluster based routing protocol that discovers and maintains routes on an on-demand basis. In ICRP, the *parent* nodes (central coordinator for each cluster) of each cluster cooperate to control the route query process to avoid flooding the network.

The Distributed Dynamic Clustering Algorithm (DDCA) is an event driven algorithm which monitors the status of each node in order to maintain its cluster affiliation and current state. Each node can be in one of five states, namely, *inactive*, *unclustered*, *orphan*, *child* and *parent*. The algorithm runs continuously and asynchronously on each active node in the ad-hoc network and forms the platform on which the intra-cluster protocol operates.

The DDCA controls the cluster formation and using the set of states mentioned above, it provides the means for distributed control over the clustering process. A node cannot participate in routing until it is affiliated with a cluster and hence, as soon as any node becomes active, it tries to become part of a cluster. Once the node has associated itself with a cluster, the association is maintained until the node gets disconnected. A disconnected node tries to locate a feasible cluster; failing which, it forms a cluster of its own.

The set of events that invoke clustering decisions and other actions in DDCA are described briefly below. Each un-clustered node seeks a feasible cluster by broadcasting a *join-request* message. If it receives no responses it creates a new cluster in which it is the only member, this type of a node is called an *orphan* node. To prevent adjacent un-clustered nodes from each creating new clusters, simultaneous requests are handled by forcing nodes with higher identifiers to back-off and try again. A node that receives at least one join-response message joins the maximum strength cluster from which a response was received. A node joins a cluster by changing its state, setting its cluster identifier (CID) and initiating an intra-cluster routing exchange with its neighbors. As a child, each node must process and respond to join-request messages and detect if it

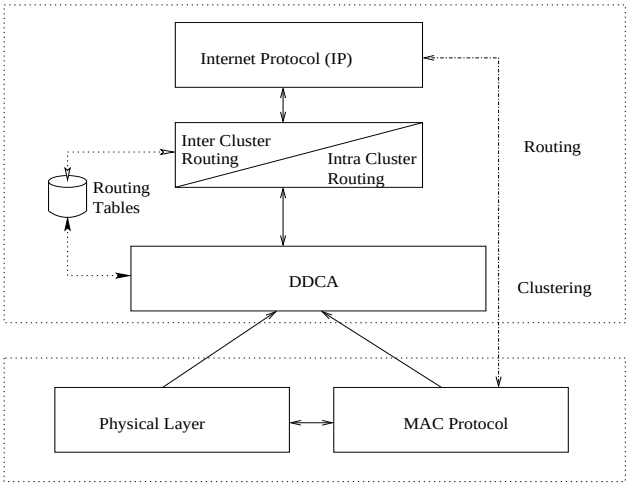

Figure 1. Overview of the design

has become disconnected from the cluster, or if a cluster partition has occurred. The parent of every cluster is initially an orphan. Each orphan node periodically attempts to join an adjacent cluster until it detects that at least one child has joined its cluster. This can be detected by the reception of routing information and the subsequent increase in size of the intra-cluster routing table. Each parent node must process and respond to join-request messages and detect if it has become disconnected from the its children.

The DDCA guarantees that each node in a given cluster knows the address of all nodes currently affiliated within the same cluster and the address of each external border node (also called a gateway node) of the cluster. The *border node* of a cluster is a node that can listen to the messages from another cluster.

ICRP constructs routes on-demand and maintains them. Each node involved in routing maintains a cache of the nodes that it can reach via either intra-cluster or inter-cluster routing. The Inter Cluster route construction and maintenance protocol has four phases:

- Route Search
- Query Dissemination
- Route Setup
- Route Maintenance

3.1. Route Search

Search phase involves query initiation by a node that requires a route to a destination that is neither in its cluster nor in its inter-cluster destination cache. The query messages are forwarded to all the gateway nodes of the cluster.

3.2. Query Dissemination

Once a gateway node to a cluster receives a copy of the query, it first checks if it is a duplicate query. Duplicate queries are checked by first forwarding the query to the parent node and waiting for a reply. If the query is not a duplicate, it is again forwarded to all the gateway nodes of the cluster who again forward it to the other adjacent clusters. A cache is maintained regarding the reception of this query. Once a gateway node finds that the entry being requested belongs to its cluster it forwards the query directly to that node or if it itself is the object of the query, it starts processing the query.

3.3. Route Setup

Once the destination has been reached, the query terminates and no further queries are generated. The destination then updates its routing table, generates a query reply packet and sends it back to the node from which it received the query. This node in turn forwards the query back to the node from which it received the query. This continues until the originator of the query is reached. Once the query reply has been received by the originator, it updates its routing tables and the setup phase comes to an end. In case no reply is received within a timeout interval, the query is discarded from the cache.

3.4. Route Maintenance

In this phase each inter-cluster destination is checked periodically. Once a path remains inactive for time greater than a timeout value, the route is deleted. If a route is lost because the next-hop node is not available, a query is again initiated for that destination and a new path is setup if possible, from the point of disconnection.

4. Implementation

There are many approaches to protocol implementation. The simple design on which the implementation is based is shown in Figure 2. Following sub-sections explain the function of each component and the interactions between different components.

4.1. Applications

Applications are the user level applications (usually applications built on top of the transport layer protocols like *telnet, ftp*) that want to initiate a connection and transfer data. These applications are unaware of the routing protocol or the underlying infrastructure that is being used. The interaction between the application and Netfilter (see section

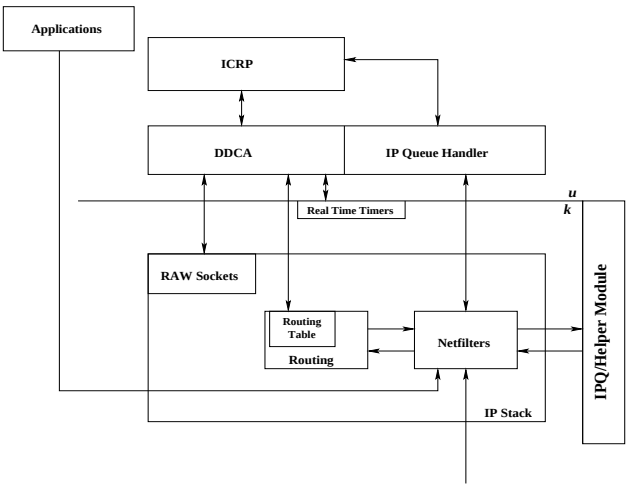

Figure 2. Architecture *(u = user level, k = kernel level)*

4.5) is transparent to the application. Netfilter processes all the packets being generated by the application.

4.2. RAW Socket Interface

The raw socket interface provides the upper layer modules (DDCA and ICRP) with a platform to create and inject special type of packets into the network by bypassing the transport layer protocols (and to some extent, network layer protocols as well.)

4.3. Real Time Timers

The Real Time Timers module provides the protocol with a framework by which we can define multiple iterating Real-Time timers. These timers are required for various book-keeping functions of DDCA.

4.4. DDCA

Distributed Dynamic Clustering Algorithm (DDCA) is the clustering algorithm on top of which the Inter Cluster Routing Protocol (ICRP) operates. DDCA creates the cluster and lets ICRP access the elements of the cluster through various interfaces that it exports. The interaction between ICRP and DDCA is limited to addition and removal of routing table entries from the kernel and ICRP receiving routing packets from DDCA.

DDCA uses RAW sockets to define a new protocol type (*IPPROTO_DDCA*) [1] [10] since it needs to create the IP packets including the header by itself. DDCA also uses with the Real-Time Timers [11] provided by the kernel for various timer based functions in the clustering algorithm.

4.5. Netfilter

Netfilter is an architecture inside the kernel to filter packets based on various criteria. It exports functions through which kernel modules can have access to packets passing through the network layer protocol. The kernel module can then perform various operations on the packet before letting it pass further through the protocol stack. Additionally, it can also specify a verdict for the packet which can be one of *accept*, *reject*, *queue* or *steal*. If the kernel module has specified the verdict as *queue*, the IP packet queues up for processing by the IP Queue Handler module. Stealing the packet means that the kernel need not bother about the packet anymore, it will be managed by the module that registered to process it.

4.6. IPQ Helper Module

This kernel module is responsible for deciding which packets will be routed or dropped or forwarded. This module is inserted as part of the kernel before the (α, t) protocol starts to execute.

4.7. Routing Table Entries

After reading the implementation of *ospfd* in [14], it was decided to develop an interface to add and delete routing table entries from user space. A user level copy of the kernel level routing table is maintained as a queue. Any change made to the user level routing table is reflected in the kernel routing table. The advantage of this scheme is that the kernel routing table need not be queried every time, only updates need to be sent to the kernel routing table.

4.8. Packet Formats

This section lists the various packet formats used in this implementation.

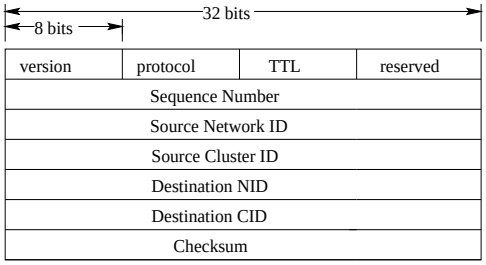

Figure 3. Format of the DDCA Header packet

Figure 3 shows the format of the DDCA header packet. The important fields in this packet are:

- Version: Contains the version number of the protocol.
- Protocol: Contains the type of the protocol that IP would use to identify the correct protocol stack to hand the packet over to. In our case, since there is no registered handler, IP looks for a RAW socket which has registered to receive a packet of this protocol type.
- Source Network ID: Contains the ID of the node that initiated this packet.
- Source Cluster ID: Contains the cluster ID of the node that initiated this packet.
- Destination NID: Contains the ID of the node for whom this packet is intended.
- Destination CID: Contains the cluster ID of the node for whom this packet is intended.

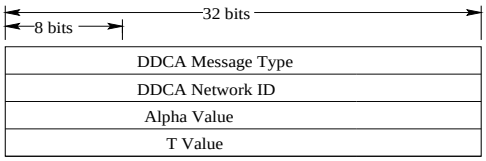

Figure 4. Format of the DDCA Body packet

Figure 4 shows the format of the DDCA Body packet. The important fields in this packet are:

- DDCA Message Type: Contains the message type to be sent.
- DDCA Network ID: Contains the ID of the node that initiated this packet.
- Alpha Value: Contains the value of α, the parameter used along with the value of t to test the link strength.
- T Value: Contains the value of t, the parameter used along with the value of α to test the link strength.

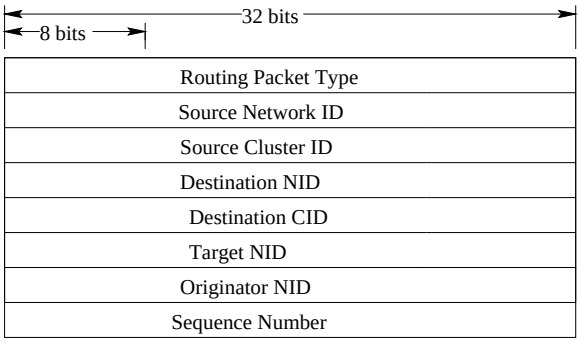

Figure 5. Format of the Query Body packet

Figure 5 shows the format of the Query Body packet. The important fields in this packet are:

- Routing Packet Type: Contains the type of the query, either RREQ (Route Request) or RREP (Route Reply).
- Source Network ID: Contains the ID of the node that initiated this packet.
- Source Cluster ID: Contains the cluster ID of the node that initiated this packet.
- Destination NID: Contains the ID of the node for whom this packet is intended.
- Destination CID: Contains the cluster ID of the node for whom this packet is intended.
- Target NID: Contains the NID of the node for which we need to discover a route.
- Originator NID: Contains the NID of the node who originated this packet.
- Sequence Number: Contains the sequence number of the query.

5. Results

To test a hierarchical protocol like the $(\alpha, t)-$Cluster, we have to test both aspects of the protocol, the intra-cluster as well as the inter-cluster part of it. The experimental setup consisted of three nodes which were used to form a cluster (for intra-cluster testing) and two nodes used to form a cluster and another node acting as an orphan (for inter-cluster testing). Three different mobility models were studied.

- Static: The nodes are stationary.
- Group Mobility: The nodes move in groups, the rate of disconnections is not very high.
- High Mobility: The nodes are constantly on the move, leading to a very high rate of disconnections.

Experiments were conducted for the above three mobility models and two types of data analysis were studied.

- Goodput Analysis: This study measures the amount of data that had to be re-transmitted in order for the whole data to go through (e.g: A file).
- Data Rate Analysis: This study measures the sustained throughput that is achieved in the network.

The experiments were conducted with one node being stationary, while the other two nodes were moved around based on the mobility model (the mobility that was used was random mobility, where the nodes move as they want to, for example, laptops used in the experiment, were just carried around the department by the researchers conducting the experiment). The breaking point of the network was found (this was the point where inter and intra-cluster routes were being broken), and this knowledge was used in the group and high mobility models. During the experiments, the nodes were assigned different IP addresses that

belonged to different domains. This was to make sure that the protocol worked irrespective of whether the nodes belonged to the same network or not (this is to mirror real life scenarios wherein, ad-hoc networks consist of nodes belonging to different networks).

The results were obtained by letting the nodes cluster and then by moving them around physically according to the mobility model. A file transfer was initiated between two nodes, one designated as the *sender*, the node responsible for sending the file, and the other designated as the *receiver*, the node responsible for receiving the file and also for logging the statistics. The size of the file transferred was varied from a very small file to a large file. The file to be transferred was broken down into packets of size 1500 bytes each.

The *sender* node is equipped with a *server* program, which is responsible for sending the file across. This program has a timer associated with it to calculate the time taken to send the file. The file is transferred continuously without any timeouts (regardless of the state of the connection). The *receiver* node is equipped with a *client* program, which is responsible for receiving the file. This program is also responsible for maintaining a count of the number of packets received and also the size of the packets received (this is to ensure that packets were not corrupted and wrong packets not received). The *client* does not sent acknowledgments for the packets that are received. Due to the frequent disconnections that can occur in ad-hoc networks, both the *server* and the *client* programs were written using RAW sockets. This does not use the services provided by the transport layer like acknowledgments and timeouts. This ensures that even if the connectivity between nodes is lost, the application does not time out.

5.1. Intra-Cluster evaluation

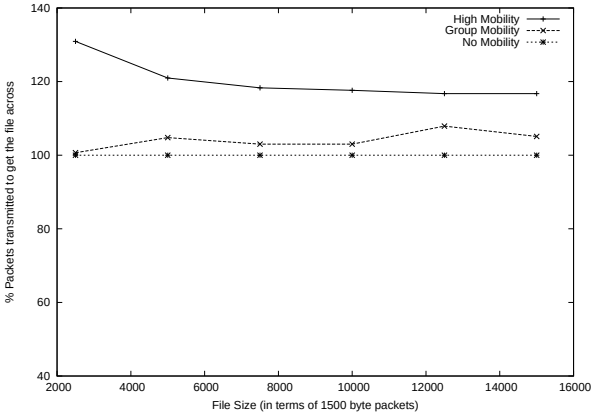

Figure 6. Intra-Cluster: Goodput Analysis

From figure 6, we can see that the goodput in the static case is 100%. This is to be expected as the nodes are stationary and hence there is continuous connectivity. We can also conclude that the protocol is very tolerant to disconnections. Even as the file size increases (in the order of tens of megabytes), the protocol is able to maintain a good performance by ensuring that connections are restored quickly after a disconnection. The high mobility of the nodes helps in re-forming routes that are broken quickly due to the presence of other nodes that take the place of the previously disconnected node.

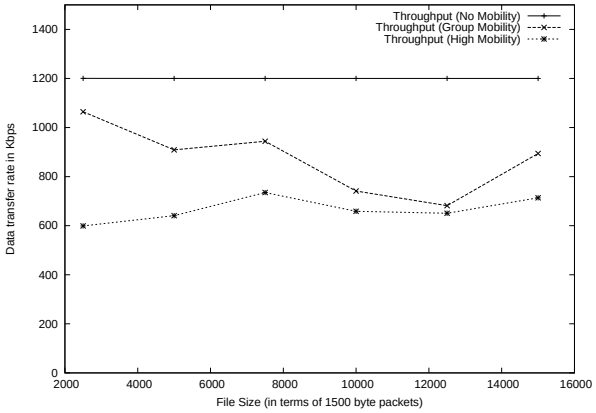

Figure 7. Intra-Cluster: Throughput Analysis

From figure 7, we can conclude that the throughput remains constant in the static case. This is because the nodes are stationary and connectivity is maintained throughout. The throughput falls significantly once disconnections are introduced into the network. There is a significant drop in the throughput in the high mobility case, but due to the fact that the protocol handles disconnections quickly, the throughput remains stable across different file sizes. This is very useful for applications that require a steady throughput (even though it is much lower than in the static case).

5.2. Inter-Cluster evaluation

From figure 8, we conclude that the goodput in the static case is 100%. This does not come as a surprise as the nodes are stationary and hence have connectivity at all times. The goodput in the high mobility case has gone down considerable due to the increase in the number of disconnections, but we can see that the the protocol tolerates disconnections well and restores the connection soon. Even though the file sizes increase, thus giving a larger window for disconnections to take place and hence lower the goodput, the protocol maintains a steady goodput.

From figure 9, we can conclude that the throughput for the static case averages well. The small dips and peaks can be attributed to transient environmental factors. When group mobility is introduced, we see that the data rate falls

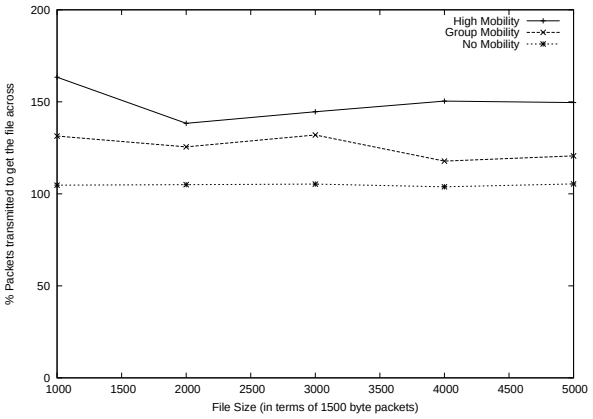

Figure 8. Inter-Cluster: Goodput Analysis

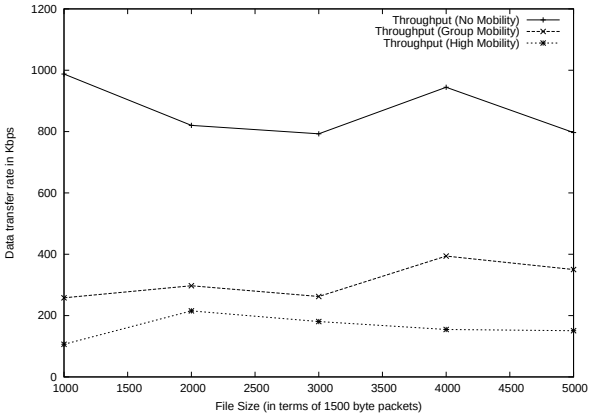

Figure 9. Inter-Cluster: Throughput Analysis

significantly, but the protocol handles these disconnections well, as can be seen from the fact that the data rate is quite stable. In the high mobility case, the data rate falls significantly due to a high number of disconnections, but overall the data rate remains stable.

6. Conclusions

The $(\alpha, t)-$Cluster framework was implemented on Linux and it was tested and evaluated. We believe this to be one of the few protocols for ad-hoc networks to have a proof-of-concept work. There is much work that still needs to be done in terms of tests, modifications and optimizations to make this protocol worthy of industry standards. The challenge is to be able to deploy these kinds of protocols in the industry so as to be able to test them out more rigorously which would help in improving the protocol.

We have to determine the overhead caused because of running DDCA continuously on these nodes. Careful analysis could determine how this protocol could be optimized and help reduce the number of packets transmitted, and also fix the optimum timeouts for the timers used.

Hybrid routing must be looked at in a whole new light when we talk about routing in Ultra-large scale networks and wireless sensor networks. Better models of routing need to be studied, models like data diffusion, content-based routing and information dissemination could be implemented and analyzed.

7. Acknowledgments

This work has been supported in part by the National Science Foundation (under grant no: 000073972).

References

[1] A.B. McDonald, T. Znati, A. Gopalan. (α, t) protocol specification. *Internet Draft*, August 2001.

[2] Charles E. Perkins. Ad-hoc networks. *Addison Wesley*, 2001.

[3] S. Dwivedi. Implementation and analysis of the (α, t) routing protocolon linux. *Masters Thesis, University of Pittsburgh*, 2002.

[4] A. Gopalan. Implementation and analysis of the (α, t) clustering protocol on linux. *Masters Thesis, University of Pittsburgh*, 2002.

[5] Z. J. Haas and M. Pearlman. The Zone Routing Protocol (ZRP) for Ad Hoc Networks. *Internet Draft*, August 1998.

[6] A. C. Inc. Rendezvou's Developer Page. *http://developer.apple.com/macosx/rendezvous/*, 2002.

[7] D. J. J. Broch and D. Maltz. The Dynamic Source Routing Protocol for Mobile Ad-Hoc Networks. *Internet Draft*, Mar. 1998.

[8] V. Kawadia, Y. Zhang, and B. Gupta. System services for implementing ad hoc routing protocols. *International Workshop on Ad Hoc Networking*, 2002.

[9] Y. Ko and N. Vaidya. Location-Aided Routing (LAR) in Mobile Ad-Hoc Networks. In *Proc. ACM/IEEE MOBICOM*, Oct. 1998.

[10] A. McDonald. A Mobility Based Framework for Adaptive Dynamic Cluster-Based Hybrid Routing in Wireless Ad Hoc Networks. *PhD. Thesis, University of Pittsburgh*, December 2000.

[11] S. Pather. POSIX 1003.1b timer patches for Linux. *http://www.rhdv.cistron.nl/posix.html*, August 2001.

[12] C. Perkins and E. Royer. Ad Hoc On Demand Distance Vector (AODV) Routing. *Internet-Draft*, Nov. 1998.

[13] C. R. Perkins and P. Bhagwat. Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for Mobile Computers. In *ACM SIGCOMM*, pages 234–244, Oct. 1994.

[14] M. J. T. OSPF Complete Implementation. *Addison Wesley*, September 2000.

[15] A. Williams. Zero Configuration Networking. *Internet Draft*, September 2002.