# 1 Introduction

## 1.1 Background

The client/server paradigm can be employed to describe the Inter Vehicular Communication platform effectively. The basic unit for update and query is a data item. In order to read the most recent copy of a data item, the nodes issue only simple requests. On a node, there may be one or more processes running, referred to as clients. The base station needs to communicate with the database server to retrieve the data items so as to serve a request sent from a client.

An effective technique to improve performance in a vehicle communication is to cache the frequently accessed data items on the client side. Average data access latency is reduced as several data access requests can be satisfied from the local cache , thereby obviating the need for data transmission over the scarce wireless links. However, frequent disconnections and mobility of the clients make cache consistency a challenging problem Classical cache management strategies may not be suitable for mobile environments due to the disconnection and mobility of the mobile clients.

Caching plays a key role in inter vehicle communication because Of, its ability to alleviate the performance and availability limitations of weakly-connected and disconnected operation. But evaluating alternative caching strategies for vehicular communication is problematic. Cache management in mobile environment, in general, includes the following issues to be addressed:

1. The cache discovery algorithm that is used to efficiently discover, select, and deliver the requested data item(s) from neighbouring nodes.

2. Cache admission control - this is to decide on what data items can be cached to improve the performance of the caching system.

A Combined and Complete Cache Management (CCCM) Architecture was proposed for mobile hosts. The goal of this architecture is to reduce the caching overhead and provide optimal consistency and replacement. It aims to improve the network utilization, reduce the search latency, bandwidth and energy consumption.

The architecture comprises of the following algorithms:

- Cache placement algorithm
- Cache discovery algorithm

Wireless communication between vehicle clients is becoming more popular than ever before. This due to recent technological advances in laptop computers and wireless data communication devices, such as wireless modems and wireless LAN s. This has lead to lower prices and higher data rates, which are the two main reasons why vehicle communication continues to enjoy rapid growth.

There are two distinct approaches for enabling wireless communication between two hosts. The first approach is to let the existing inter vehicular communication network infrastructure carry data as well as voice. The major problems include the problem of hand-off, which tries to handle the situation when a connection should be smoothly handed over from one base station to another base station without noticeable delay or packet loss. Another problem is that networks based on the vehicular infrastructure are limited to places where there exists such a inter vehicular network infrastructure.

The second approach is to form an ad-hoc network among all users wanting to communicate with each other. This means that all users participating in the ad-hoc network must be willing to forward data packets to make sure that the packets are delivered from source to destination. This form of networking is limited in range by the individual nodes transmission ranges and is typically smaller compared to the range of cellular systems. This does not mean that the cellular approach is better than the ad-hoc approach. Ad-hoc networks have several advantages compared to traditional cellular systems.

These advantages include:

- On demand set-up
- Fault tolerance
- Unconstrained connectivity

Ad-hoc networks do not rely on any pre-established infrastructure and can therefore be deployed in places with no infrastructure. This is useful in disaster recovery situations and places with non-existing or damaged communication infrastructure where rapid deployment of a communication network is needed. Ad-hoc networks can also be useful on conferences where people participating in the conference can form a temporary network without engaging the services of any pre-existing network.

Because nodes are forwarding packets for each other, some sort of routing protocol is necessary to make the routing decisions. Currently there does not exist any standard for a routing protocol for ad-hoc networks, instead this is work in progress.

## 1.2 Problem description

The objective for this master thesis was to design Cache management Architecture for Inter Vehicular Communication. This design should be done through simulation. At the beginning of this master thesis, no implementation of the Architecture had been released, so the first main task was to implement some of the Algorithms.

The thesis also included the goal to generate a simulation environment that could be used as a platform for further studies within the area of Inter-Vehicular networks.

The goal of this master thesis was to:

- Get a general understanding of mobile networks.
- Get a general understanding of Inter-Vehicular networks.
- Generate a simulation environment that could be used for further studies.
- Implement some of the proposed communication Algorithms for Mobile Computing Environments.
- Analyse the Algorithms theoretically and through simulation.

## 1.3 Related work

D. Barbara, and T. Imielinski have proposed a cache solution which is suitable for mobile environments. In their approach, the server periodically broadcasts an invalidation report (IR) in which the changed data items are indicated. Rather than querying the server directly regarding the validation of cached copies, the clients can listen to these Irs over the wireless channel, and use them to validate their local cache. The IR-based solution is attractive because it can scale to any number of clients who listen to the IR. However, the IR-based solution has some major drawbacks such as long query latency and low bandwidth utilization. Guohong Cao has addressed an UIR-based approach. In his approach, a small fraction of the essential information (called updated invalidation report (UIR)) related to cache invalidation is replicated several times within an IR interval, and hence the client can answer a query without waiting until the next JR. However, if there is a cache miss, the client still needs to wait for the data to be delivered. Guohong Cao has proposed a power-aware cache management. Based on a novel pref-etch access ratio concept, the proposed scheme can dynamically optimize performance or power based on the available resources and performance requirements. Simulation results have shown that their solution not only improves the cache hit ratio, the throughput, and the bandwidth utilization, but also reduces the query delay and the power consumption. Miguel Castro et al., have proposed a new hybrid adaptive caching technique, which combines page and object caching to reduce the miss rate in client caches dramatically. Athena Vakali has presented a study of applying a history based approach to the Web-based proxy cache replacement process. Trace-driven simulation was employed to evaluate and comment on the performance of the proposed cache replacement techniques.

Ismail Ari Et al. have proposed the use of machine learning algorithms to rate and select the current best policies or mixtures of policies via weight updates based on their recent success, allowing each adaptive cache node to tune itself based on the workload it observes. ACME is used to manage the replacement policies within distributed caches to further improve the hit rates over static caching techniques.

Bin Tang et al. have developed a paradigm of data caching techniques to support effective data access in ad-hoc networks. In particular, they have considered memory capacity constraint of the network nodes, and developed efficient algorithms to determine near-optimal cache placements to maximize reduction in overall access cost.

## 1.5 Disposition

This report consists of 8 chapters. Chapters 1 and 2 explain the concept of Inter Vehicular Networks and routing in general. Chapter 3 describes the different IVC Algorithms. Chapters 4 and 5 describe the proposed Architecture and the simulations that were made. Chapter 6 describes the implementation study. Chapter 7 concludes the whole report and chapter 8 is the references that we have used.

# 2 General Concepts

## 2.1 Wireless ad-hoc networks

### 2.1.1 General

### 2.1.2 Usage

### 2.1.3 Characteristics

## 2.2 Routing

### 2.2.1 Conventional protocols

### 2.2.2 Link State

### 2.2.3 Distance Vector

### 2.2.4 Source Routing

### 2.2.5 Flooding

### 2.2.6 Classification

## 2.1 Wireless ad-hoc networks

### 2.1.1 General

A wireless ad-hoc network is a collection of mobile/semi-mobile nodes with no pre-established infrastructure, forming a temporary network. Each of the nodes has a wireless interface and communicate with each other over either radio or infra-red. Laptop computers and personal digital assistants that communicate directly with each other are some examples of nodes in an ad-hoc network. Nodes in the ad-hoc network are often mobile, but can also consist of stationary nodes, such as access points to the Internet.

Semi mobile nodes can be used to deploy relay points in areas where relay points might be needed temporarily. Figure 1 shows a simple ad-hoc network with three nodes. The outermost nodes are not within transmitter range of each other. However the middle node can be used to forward packets between the outermost nodes. The middle node is acting as a router and the three nodes have formed an ad-hoc network.

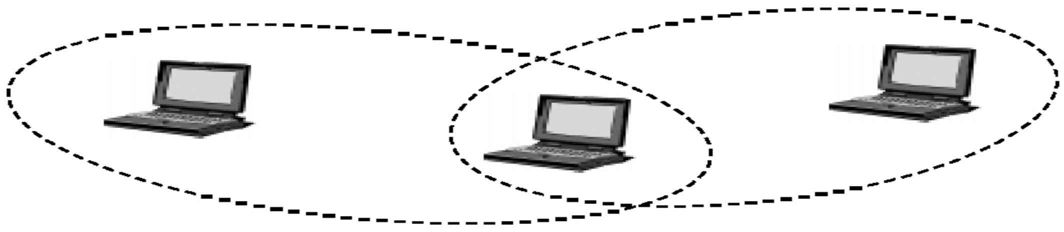


**Figure 1:** Example of a simple ad-hoc network with three participating nodes.

An ad-hoc network uses no centralized administration. This is to be sure that the network wont collapse just because one of the mobile nodes moves out of transmitter range of the others. Nodes should be able to enter/leave the network as they wish. Because of the limited transmitter range of the nodes, multiple hops maybe needed to reach other nodes. Every node wishing to participate in an ad-hoc network must be willing to forward packets for other nodes. Thus every node acts both as a host and as a router. A node can be viewed as an abstract entity consisting of a router and a set of affiliated mobile hosts (Figure 2). A router is an entity, which, among other things runs a routing protocol. A mobile host is simply an IP-addressable host/entity in the traditional sense.

Ad-hoc networks are also capable of handling topology changes and malfunctions in nodes. It is fixed through network reconfiguration. For instance, if a node leaves the network and causes link breakages, affected nodes can easily request new routes and the problem will be solved. This will slightly increase the delay, but the network will still be operational.

Wireless ad-hoc networks take advantage of the nature of the wireless communication medium. In other words, in a wired network the physical cabling is done a priori restricting the connection topology of the nodes. This restriction is not present in the wireless domain and, provided that two nodes are within transmitter range of each other, an instantaneous link between them may form.

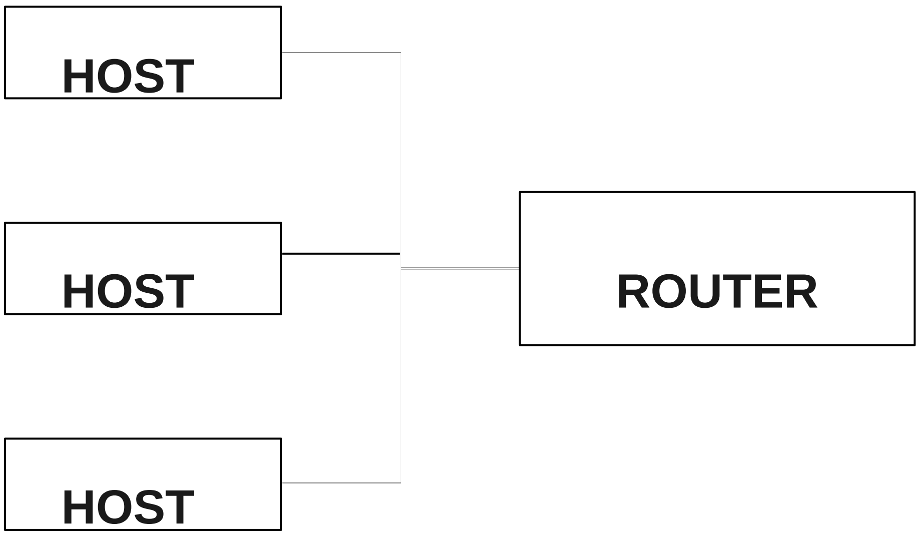


Figure 2:  Block diagram of a mobile node acting both as hosts and as router.

### 2.1.2 Usage

There is no clear picture of what these kinds of networks will be used for. The suggestions vary from document sharing at conferences to infrastructure enhancements and military applications. In areas where no infrastructure such as the Internet is available an ad-hoc network could be used by a group of wireless mobile hosts. This can be the case in areas where a network infrastructure may be undesirable due to reasons such as cost or convenience. Examples of such situations include disaster recovery personnel or military troops in cases where the normal infrastructure is either unavailable or destroyed.

Other examples include business associates wishing to share files in an airport terminal, or a class of students needing to interact during a lecture. If each mobile host wishing to communicate is equipped with a wireless local area network interface, the group of mobile hosts may form an ad-hoc network. Access to the Internet and access to resources in networks such as printers are features that probably also will be supported.

### 2.1.3 Characteristics

Ad-hoc networks are often characterized by a dynamic topology due to the fact that nodes change their physical location by moving around. This favours routing protocols that dynamically discover routes over conventional routing algorithms like distant vector and link state [23]. Another characteristic is that a host/node have very limited CPU capacity, storage capacity, battery power and bandwidth, also referred to as a “thin client”. This means that the power usage must be limited thus leading to a limited transmitter range.

The access media, the radio environment, also has special characteristics that must be considered when designing protocols for ad-hoc networks. One example of this may be unidirectional links. These links arise when for example two nodes have different strength on their transmitters, allowing only one of the host to hear the other, but can also arise from disturbances from the surroundings. Multi-hop in a radio environment may result in an overall transmit capacity gain and power gain, due to the squared relation between coverage and required output power. By using multi-hop, nodes can transmit the packets with a much lower output power.

## 2.2 Routing

Because of the fact that it may be necessary to hop several hops (multi-hop) before a packet reaches the destination, a routing protocol is needed. The routing protocol has two main functions, selection of routes for various source-destination pairs and the delivery of messages to their correct destination. The second function is conceptually straightforward using a variety of protocols and data structures (routing tables). This report is focused on selecting and finding routes.

### 2.2.1 Conventional protocols

If a routing protocol is needed, why not use a conventional routing protocol like link state or distance vector? They are well tested and most computer communications people are familiar with them. The main problem with link-state and distance vector is that they are designed for a static topology, which means that they would have problems to converge to a steady state in an ad-hoc network with a very frequently changing topology.

Link state and distance vector would probably work very well in an ad-hoc network with low mobility, i.e. a network where the topology is not changing very often. The problem that still remains is that link-state and distance-vector are highly dependent on periodic control messages. As the number of network nodes can be large, the potential number of destinations is also large. This requires large and frequent exchange of data among the network nodes. This is in contradiction with the fact that all updates in a wireless interconnected ad hoc network are transmitted over the air and thus are costly in resources such as bandwidth, battery power and CPU. Because both link-state and distance vector tries to maintain routes to all reachable destinations, it is necessary to maintain these routes and this also wastes resources for the same reason as above.

Another characteristic for conventional protocols are that they assume bi-directional links, e.g. that the transmission between two hosts works equally well in both directions. In the wireless radio environment this is not always the case. Because many of the proposed ad-hoc routing protocols have a traditional routing protocol as underlying algorithm, it is necessary to understand the basic operation for conventional protocols like distance vector, link state and source routing.

### 2.2.2 Link State

In link-state routing [23], each node maintains a view of the complete topology with a cost for each link. To keep these costs consistent; each node periodically broadcasts the link costs of its outgoing links to all other nodes using flooding. As each node receives this information, it updates its view of the network and applies a shortest path algorithm to choose the next-hop for each destination.

Some link costs in a node view can be incorrect because of long propagation delays, partitioned networks, etc. Such inconsistent network topology views can lead to formation of routing-loops. These loops are however short-lived, because they disappear in the time it takes a message to traverse the diameter of the network.

### 2.2.3 Distance Vector

In distance vector [23] each node only monitors the cost of its outgoing links, but instead of broadcasting this information to all nodes, it periodically broadcasts to each of its neighbours an estimate of the shortest distance to every other node in the network. The receiving nodes then use this information to recalculate the routing tables, by using a shortest path algorithm.

Compared to link-state, distance vector is more computation efficient, easier to implement and requires much less storage space. However, it is well known that distance vector can cause the formation of both short-lived and long-lived routing loops. The primary cause for this is that the nodes choose their next-hops in a completely distributed manner based on information that can be stale.

### 2.2.4 Source Routing

Source routing [23] means that each packet must carry the complete path that the packet should take through the network. The routing decision is therefore made at the source. The advantage with this approach is that it is very easy to avoid routing loops. The disadvantage is that each packet requires a slight overhead.

### 2.2.5 Flooding

Many routing protocols uses broadcast to distribute control information, that is, send the control information from an origin node to all other nodes. A widely used form of broadcasting is flooding [23] and operates as follows. The origin node sends its information to its neighbours (in the wireless case, this means all nodes that are within transmitter range). The neighbours relay it to their neighbours and so on, until the packet has reached all nodes in the network. A node will only relay a packet once and to ensure this some sort of sequence number can be used. This sequence number is increased for each new packet a node sends.

### 2.2.6 Classification

Routing protocols can be classified [1] into different categories depending on their properties.

- Centralized vs. Distributed
- Static vs. Adaptive
- Reactive vs. Proactive

One way to categorize the routing protocols is to divide them into centralized and distributed algorithms. In centralized algorithms, all route choices are made at a central node, while in distributed algorithms, the computation of routes is shared among the network nodes.

Another classification of routing protocols relates to whether they change routes in response to the traffic input patterns. In static algorithms, the route used by source-destination pairs is fixed regardless of traffic conditions. It can only change in response to a node or link failure. This type of algorithm cannot achieve high throughput under a broad variety of traffic input patterns. Most major packet networks uses some form of adaptive routing where the routes used to route between source-destination pairs may change in response to congestion

A third classification that is more related to ad-hoc networks is to classify the routing algorithms as either proactive or reactive. Proactive protocols attempt to continuously evaluate the routes within the network, so that when a packet needs to be forwarded, the route is already known and can be immediately used. The family of Distance-Vector protocols is an example of a proactive scheme. Reactive protocols, on the other hand, invoke a route determination procedure on demand only. Thus, when a route is needed, some sort of global search procedure is employed. The family of classical flooding algorithms belongs to the

reactive group. Proactive schemes have the advantage that when a route is needed, the delay before actual Packets can be sent is very small. On the other side proactive schemes needs time to converge to a steady state. This can cause problems if the topology is changing frequently.

# 3 IVC Algorithms

This chapter describes the different Inter Vehicular Communication Networking routing protocols .

## 3.1 ADAPTIVE RATE CONTROL (ARC)

With ARC algorithm, it works like a gate to control the  arriving packets. When packets arrive at the node and if a packet drop is not present, the packet is transmitted without delay. If packet drop is present, the dropped packet will be dropped blocked in a packet queue (Qc) and waiting to retransmit when ARC finishes adjusting the new windows size in order to overcome the packets drop. At the same time to maintain quality of service (QoS), the maximum packet delay time has been defined as DT that means the packets that have been waiting in the packet queue longer than DT will be discarded finally.  In case that the arrival traffic (average arrival packet rate or traffic ( ka ) is less than the packet drop rate (kp) and packet drop is not yet present. ARC will set windows size equal one On the other hand, if ( ka ) is larger than (kp) , ARC will adjust the window size with reference to packet drop rate (kp) and arrival traffic rate (ka ).In our assumption, ARC will adjust the window size between. One to five.

ARC Algorithm is described as follows :

```
Start:

    Find number of predecessor node (P_all)

    Find current allocate rate for each predecessor node

    P(id)(Ai)

    Find current window size (Wi)

    Do
    {
        IF packet drop THEN

        for(i=O and i<P_all so i = i+1)
        {
            Find New allocate rate for P(id)(Ai)

            current allocate rate for P(id)(Ai)
             =New
            allocate rate P(id)(Ai
            )
        }
        Find New Window Size
        current window
        size = New Window size
    }
    ELSE
        {
            for(i=O and i < P_all so i = i + 1)
                {
                    current allocate rate P(id)(Ai)
                     = current
                    allocate rate P(id)(Ai)

                }
            current window size = current window size
        }
        While(Packet is transmitting)

End
```

## 3.2 Priority Based Inter-Vehicle Communication in Vehicular Ad-Hoc Networks using IEEE 802.11e

The focus is on providing differential service to different priority safety messages in a Vehicular Ad-hoc Network (VANET) with the following characteristics:

- i) all vehicles move fast, thus, it can cause high bit error rates;
- ii) all vehicles move in the same direction, following the road topology. Therefore, the relative velocity of all nodes is relatively low.
- iii) the communication safety messages have a few hundred bytes and have different priorities .

It relaid on broadcast communication with no protection mechanism, such as RTS/CTS and acknowledgement, the reliability of the network can be low. In order to alleviate this issue, it was decided to transmit messages repetitively. The number of repetitions is assigned based on the priority of a message. To increase the probability of a successful transmission for high priority messages, a high priority message is transmitted more times than a lower priority one.

In general, EDCA provides different services for different Access Categories (ACs). Each priority level of safety messages is mapped to a different traffic class. In all vehicles, there will be four different queues for each priority with a virtual collision handler, which handles the internal collision. Each priority i has different values of the following parameters: minimum contention window size (CWmin[i]), maximum contention widow size (CWmax[i]), arbitrary inter-frame space (AIFS[i]), transmission opportunity (TXOP[i]), and number of repetitions (No_Repetition[i]).

In [31] the authors use Enhanced Distributed Channel Access (EDCA) in IEEE 802.11e to provide a priority scheme. However, the authors did not consider contention among high priority vehicles. This unavoidable contention can lead to significant message errors, causing low communication reliability. In [32], the authors proposed priority CSMA (P-CSMA) and polling P-CSMA (PP-CSMA). However, the protocol is not compatible with the IEEE 802.11 and IEEE 802.11e standards. In addition, it is well known that the polling mechanism is not robust to channel errors and dynamic network topologies. However, VANETs have relatively high Bit Error Rates (BER) due to fast mobility, as well as a highly dynamic network topology. Therefore, polling messages may face a high error rate significantly decreasing the normalized channel throughput. In our previous work [33], we introduced a priority scheme. However, it works only in a hierarchical ad-hoc network with support of a  cluster head.

IEEE 802.11e EDCA MAC [34, 35] protocol uses Arbitrary Inter-Frame Space (AIFS) and Contention Window (CW) to provide QoS for different categories of traffic, such as Voice, Video, Best Effort and Background. IEEE 802.11e EDCA MAC protocol is used in conjunction with repetitive transmissions. This approach provides proportional service differentiation in VANETs in terms of delay and reliability to prioritized messages. Higher priority messages will wait for shorter AIFS and CW leading to faster transmission and higher probability of channel access.

Furthermore, the use of broadcast communication with no feedback mechanism, such as RTS/CTS and acknowledgement, can cause low reliability. Therefore, it is decided to adopt repetitive transmissions mechanism, which can significantly enhance the transmission reliability. Higher priority messages will be transmitted more times than lower priority messages, resulting in higher reliability for higher priority messages.

## 3.3 Vehicular Information Broadcasting Relay (VIBROR)

### 3.3.1 Overview of VIBROR protocol

The deal is with information reconstruction scheme to realize effective multi-hop transmission on broadcasting type IVC. There are some differences between IVC broadcasting and the ordinary multi-hop scheme. In IVC, some information missing is not so serious. Especially, the information from distant vehicle is not so important, and not necessary to update in short interval each other periodically. In IVC environment, it is quite essential to relay important urgent information first and some information discarding is also acceptable. To fit the property mentioned above, VIBROR protocol is proposed.

VIBROR protocol has 3 major features. The first is packet structure. All information packets consists of some sub-packets. Each sub-packet has significant information .It is also possible to apply other information such as web, e-mail, still picture, voice and so on. Vehicle can extracts all included sub-packets from a packet and can construct a new packet from some sub-packets. The second is buffering management. At every time vehicle receives a packet successfully, vehicle extracts all included sub-packets and stores them into its own receiving buffer. At this time vehicle decide sub-packets priority.

Finally, the third is packet construction management. When a vehicle tries to transmit a packet it assemble some stored sub-packet that has higher priority. A sub-packet of its own information is also prepared. A transmission packet is reconstructed from these sub-packets.

### 3.3.2. General Scheme Flow

In this protocol an information packet consists of some sub-packets. Each sub-packet has the source vehicle ID, birth time, number of hops, and information. At every time a vehicle receives a packet, it extracts all included sub-packets from the packet and stored them into its own receiving buffer. At this time, priority of each information sub-packet is decided. These sub-packets are sorted in the receiving buffer by the priority. When the vehicle is going to transmit a new information packet, it assembles some sub-packets. A sub-packet of its own information is also prepared. A transmission packet is reconstructed from these sub-packets and will be transmitted immediately. The packet size is fixed, and a packet consists of 3 sub-packets. Here, I omitted the backward transmission (i.e. packet from vehicle 4 to vehicle 2) for simplicity. The number in the sub-packet indicates the source vehicle ID of the sub-packet. Here, the number of hops decides the priority of sub-packet. Vehicle 1 has no other vehicle information in its receiving buffer. So vehicle 1 transmits only its own information and fills the rest of the packet null sub-packets. Vehicle 2 and 3 receive a packet from vehicle 1. They transmit their own information and vehicle 1 information. Since vehicle 4 receives packets from vehicle 2 and 3, vehicle 4 knows the information of vehicle 1,2,3 and 4. It is impossible for vehicle 4 to transmit all sub-packets, and lower priority sub-packet about vehicle 1 is discarded. Vehicle 4 transmits its own information with vehicle 3 and 2 information.

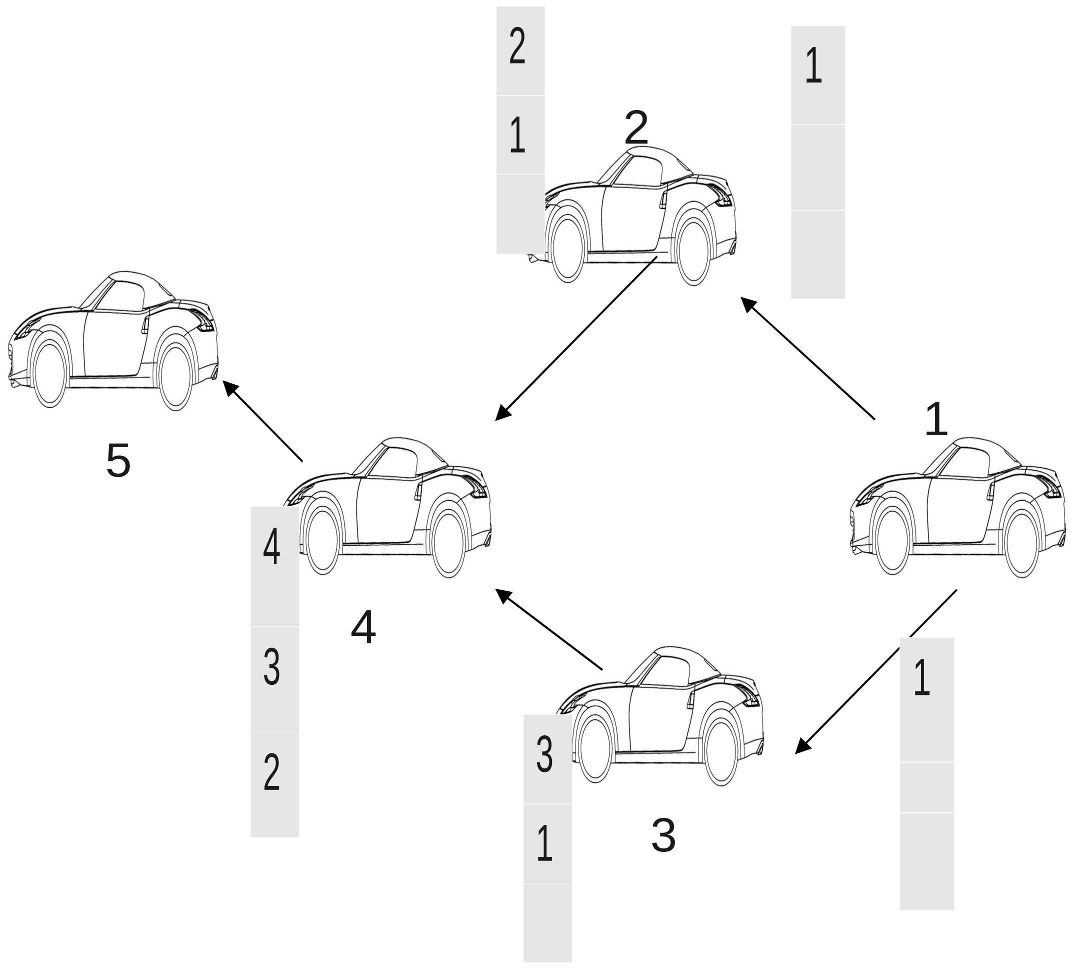


Fig2 : Example of scheme

### 3.3.3. Receiving algorithm

Next, the detail of the receiving algorithm of the receiving buffer. When a vehicle receives a packet, it extracts all included sub-packets from the packet and overwrites the sub-packets at the bottom of receiving buffer. A vehicle decides priority of each sub-packet for next transmission. We can consider a number of algorithms to decide the priority. There are many kinds of element to define priority (i.e. number of hop, delay, distance, etc.) The suitable algorithm to each segment is different from each other. To combine two or more different types of algorithm to each sub-packet is also available. Subsequently, sub-packets are sorted by the priority. So some lower priority sub-packets are discarded. Each vehicle repeats the process mentioned above at every time it receives a packet.

### 3.3.4. Transmission Algorithm

When a vehicle needs to transmit a packet, it constructs a packet from some sub-packets. A sub-packet of its own information is attached. Some sub-packets with higher priority in receiving buffer are also attached. To attach two or more own vehicle information (sub-packets) at once is also acceptable. After the transmission, the vehicle lowers the priority of transmitted sub-packet in the received buffer. So if the transmitted sub-packets are not discarded, they might be retransmitted later. As a result, even if density of vehicles increases, the increase of packet will be restrained moderately.

## 3.4 Dynamic Source Routing - DSR

### 3.4.1 Description

Dynamic Source Routing (DSR) [3][12][13] also belongs to the class of reactive protocols and allows nodes to dynamically discover a route across multiple network hops to any destination. Source routing means that each packet in its header carries the complete ordered list of nodes through which the packet must pass. DSR uses no periodic routing messages (e.g. no router advertisements), thereby reducing network bandwidth overhead, conserving battery power and avoiding large routing updates throughout the ad-hoc network.

Instead DSR relies on support from the MAC layer (the MAC layer should inform the routing protocol about link failures). The two basic modes of operation in DSR are route discovery and route maintenance. Route discovery Route discovery is the mechanism whereby a node X wishing to send a packet to Y, obtains the source route to Y. Node X requests a route by broadcasting a Route Request (RREQ) packet. Every node receiving this RREQ searches through its route cache for a route to the requested destination. DSR stores all known routes in its route cache. If no route is found, it forwards the RREQ further and adds its own address to the recorded hop sequence. This request propagates through the network until either the destination or a node with a route to the destination is reached. When this happen a Route Reply (RREP) is unicast back to the originator.

This RREP packet contains the sequence of network hops through which it may reach the target. In Route Discovery, a node first sends a RREQ with the maximum propagation limit (hop limit) set to zero, prohibiting its neighbours from rebroadcasting it. At the cost of a single broadcast packet, this mechanism allows a node to query the route caches of all its neighbours.

Nodes can also operate their network interface in promiscuous mode, disabling the interface address filtering and causing the network protocol to receive all packets that the interface overhears. These packets are scanned for useful source routes or route error messages and then discarded. The route back to the originator can be retrieved in several ways. The simplest way is to reverse the hop record in the packet. However this assumes symmetrical links. To deal with this, DSR checks the route cache of the replying node. If a route is found, it is used instead. Another way is to piggyback the reply on a RREQ targeted at the originator. This means that DSR can compute correct routes in the presence of asymmetric (unidirectional) links. Once a route is found, it is stored in the cache with a time stamp and the route maintenance phase begins. Route maintenance is the mechanism by which a packet sender S detects if the network topology has changed so that it can no longer use its route to the destination D. This might happen because a host listed in a source route, move out of wireless transmission range or is turned off making the route unusable. A failed link is detected by either actively monitoring acknowledgements or passively by running in promiscuous mode, overhearing that a packet is forwarded by a neighbouring node.

When route maintenance detects a problem with a route in use, a route error packet is sent back to the source node. When this error packet is received, the hop in error is removed from this hosts route cache, and all routes that contain this hop are truncated at this point.

### 3.4.2 Properties

DSR uses the key advantage of source routing. Intermediate nodes do not need to maintain up-to-date routing information in order to route the packets they forward. There is also no need for periodic routing advertisement messages, which will lead to reduce network bandwidth overhead, particularly during periods when little or no significant host movement is taking place. Battery power is also conserved on the mobile hosts, both by not sending the advertisements and by not needing to receive them, a host could go down to sleep instead.

This protocol has the advantage of learning routes by scanning for information in packets that are received. A route from A to C through B means that A learns the route to C, but also that it will learn the route to B. The source route will also mean that B learns the route to A and C and that C learns the route to A and B. This form of active learning is very good and reduces overhead in the network.

However, each packet carries a slight overhead containing the source route of the packet. This overhead grows when the packet has to go through more hops to reach the destination. So the packets sent will be slightly bigger, because of the overhead. Running the interfaces in promiscuous mode is a serious security issue. Since the address filtering of the interface is turned off and all packets are scanned for information. A potential intruder could listen to all packets and scan them for useful information such as passwords and credit card numbers. Applications have to provide the security by encrypting their data packets before transmission. The routing protocols are prime targets for impersonation attacks and must therefore also be encrypted. One way to achieve this is to use IP sec. DSR also has support for unidirectional links by the use of piggybacking the source route a new request. This can increase the performance in scenarios where we have a lot of unidirectional links.

## 3.5 Cluster Based Routing Protocol - CBRP

### 3.5.1 Description

The idea behind CBRP [11] is to divide the nodes of an ad-hoc network into a number of overlapping or disjoint clusters. One node is elected as cluster head for each cluster. This cluster head maintains the membership information for the cluster. Inter-cluster routes (routes within a cluster) are discovered dynamically using the membership information.

CBRP is based on source routing, similar to DSR. This means that intra cluster routes (routes between clusters) are found by flooding the network with Route Requests (RREQ). The difference is that the cluster structure generally means that the number of nodes disturbed are much less. Flat routing protocols, i.e. Only one level of hierarchy, might suffer from excessive overhead when scaled up. CBRP is like the other protocols fully distributed. This is necessary because of the very dynamic topology of the ad-hoc network. Furthermore, the protocol takes into consideration the existence of unidirectional links.

**Link sensing**

Each node in CBRP knows its bi-directional links to its neighbours as well as unidirectional links from its neighbours to itself.

Each node periodically broadcasts its neighbour table in a hello message. The hello message contains the node ID, the nodes role (cluster head, cluster member or undecided) and the neighbour table. The hello messages are used to update the neighbour tables at each node. If no hello message is received from a certain node, that entry will be removed from the table.

**Clusters**

The cluster formation algorithm is very simple, the node with lowest node ID is elected as the cluster head. The nodes use the information in the hello messages to decide whether or not they are the cluster heads. The cluster head regards all nodes it has bi-directional links to as its member nodes. A node regards itself as a member node to a particular cluster if it has a bi-directional link to the cluster head. It is possible for a node to belong to several clusters.

Clusters are identified by their respective cluster heads, which means that the cluster head must change as infrequently as possible. The algorithm is therefore not a strict "lowest ID" clustering algorithm. A non-cluster head never challenges the status of an existing cluster head. Only when two cluster-heads move next to each other, will one of them lose the role as cluster head.

**Routing**

Routing in CBRP is based on source routing and the route discovery is done, by flooding the network with Route Requests (RREQ). The clustering approach however means that fewer nodes are disturbed. This, because only the cluster heads are flooded. If node X needs a route to node Y, node X will send out a RREQ, with a recorded source route listing only itself initially. Any node forwarding this packet will add its own ID in this RREQ. Each node forwards a RREQ only once and it never forwards it to node that already appears in the recorded route.

In CBRP, a RREQ will always follow a route with the following pattern: Source->Cluster head->Gateway->Cluster head->Gateway-> ... ->Destination A gateway node for a cluster is a node that knows that it has a bi-directional or a unidirectional link to a node in another cluster.

The source uni casts the RREQ to its cluster head. Each cluster-head uni-casts the RREQ to each of its bi-directionally linked neighbour clusters, which has not already appeared in the recorded route through the corresponding gateway. There does not necessarily have to be an actual bi-directional link to a bi-directional linked neighbour cluster. This procedure continues until the target is found or another node can supply the route. When the RREQ reaches the target, the target may chose to memorize the reversed route to the source. It then copies the recorded route to a Route Reply packet and sends it back to the source.

### 3.5.2 Properties

This protocol has a lot of common features with the earlier discussed protocols. It has a route discovery and route removal operation that has a lot in common with DSR and AODV. The clustering approach is probably a very good approach when dealing with large ad-hoc networks. The solution is more scalable than the other protocols, because it uses the clustering approach that limits the number of messages that need to be sent. CBRP also has the advantage that it utilizes unidirectional links. One remaining question is however how large each cluster should be. This parameter is critical to how the protocol will behave.

## 3.6 An Efficient Fully Ad-Hoc Multi-Hop Broadcast Protocol

### 3.6.1 OVERVIEW OF THE UMB PROTOCOL

UMB is an efficient IEEE 802.11 based Urban Multi-hop Broadcast protocol for inter-vehicular networks with infrastructure support. It is designed to address (i) broadcast storm, (ii) hidden node,and (iii) reliability problems of multi-hop broadcast in urban-canyons. Unlike flooding based broadcast protocols, UMB assigns the function of forwarding and acknowledging the packet to only the furthest node without any apriori topology information. At the intersections where the communication among incident road segments are blocked by buildings, UMB employs repeaters installed at intersections to forward the packet to all road segments.

It was assumed that each vehicle is equipped with a GPS receiver and an electronic road map. Since the vehicle mobility is high and vehicles leave and enter the network frequently, the topology of this network changes fast. Therefore, the UMB protocol is designed to operate without exchanging location information among neighbouring nodes.

**A. Directional Broadcast**

1) RTB/CTB Handshake: To mitigate the hidden node problem while minimizing the overhead, sender vehicles engage
in RTS/CTS like handshake with only one of the recipients among the sender’s neighbours. If we can select the furthest away node with the handshake then other nodes in between can overhear the transmission as well. To pick this vehicle, the protocol divides the road portion

inside the transmission range into segments. Note that these segments are created only in the direction of dissemination. If there is more than one node in the furthest non-empty segment, this segment is divided iteratively into sub-segments with smaller widths. If these segment based iterations are not sufficient to pick only one node, the nodes in the last sub-segment enter to a random phase.

RTS and CTS are referred to as Request to Broadcast (RTB) and Clear to Broadcast (CTB), respectively. It cannot use the original RTS/CTS handshake because a broadcast packet has more than one destination and there is not any explicit broadcast support in IEEE 802.11 protocol. In an RTB packet, in addition to the transmission duration, the source node includes its position and intended broadcast direction. If the source wants to disseminate the message in more than one direction, a new RTB packet should be generated for each direction.

**a) First RTB attempt:** The source vehicle obeys all IEEE 802.11 transmission rules (CSMA/CA) while attempting to send an RTB packet. When the nodes in the direction of the dissemination receive this RTB packet, they compute their distance to the source node. Based on this distance, they send an energy burst (channel jamming signal) called black-burst. The black-burst method was proposed to provide guaranteed access delays to rate-limited packet traffic. In these proposals, the length of the original black-burst is proportional to the time that the node has been waiting for channel access. In our directional broadcast, we use the black-burst to select the furthest node by letting receivers sending black-burst signals proportional to their distance to the source. Since the position information of all nodes are unique, using the position information to determine the length of the black-burst gives us the capability of selecting the furthest node. The duration of the black-burst signal in the first iteration is computed as follows:

$$L1=[d.Nmax]/R. \text{ Slot-time}, \qquad (1)$$

where L1 is the black-burst duration in the first iteration, d is the distance between the source and the vehicle, R is the transmission range, Nmax is the number of segments created, R and Slot-time is the length of one slot. Note that Nmax is Nmax the segment width and d ∗ R is the number of slots the black-burst will keep busy. As a result of this computation, the furthest node sends the longest black-burst. Nodes send their black-burst in the shortest possible time (SIFS) after they hear the RTB packet. At the end of the black- burst, nodes turn around and listen to the channel. If they find the channel empty, it means that their black-burst was the longest and they are now responsible for replying with a CTB packet after a duration.

**b) Collision among CTB packets:** When there is more than one vehicle in the furthest non-empty segment, they all find the channel empty after sending their black-bursts and continue to send CTB packets. However, since all vehicles start sending the CTB packets at the same time, their CTB packets will collide. When the source node detects a transmission but cannot decode the CTB packet, it detects the collision and repeats the RTB packet after SIFS time. This time, only the nodes which have sent CTB packets join the collision resolution. To pick only one node, the furthest non-empty segment is divided into Nmax sub-segments. This process continues iteratively until a successful CTB packet is received by the source or Dmax iterations are completed.

Note that in an RTB packet, the source only indicates that there has been a collision: It is the receiver nodes' responsibility to choose the segment to be split. Only nodes who have sent the longest black-burst in the previous (i − 1)th iteration can join to the current (ith ) iteration. As a result, Longest is the black-burst length of these nodes in the previous i−1 iteration and Longest + 1 is the segment to be split. i−1 If the segment based black-burst cannot resolve the collision after the Dmax iteration, the vehicles that have sent the CTB response in the last iteration choose the black-burst length randomly. The segment based iterations decrease the segment to a very short strip and only a small number of nodes will be left at the beginning of the random phase increasing the success probability of this phase.

**c) No black-burst response:** Detecting a free channel after sending the RTB packet, the source node assumes that nobody has received its RTB packet. In this case, source node goes back to the first segment based iteration after a random amount of time. Details of this back-off procedure are the same as those of the IEEE 802.11 standard when no CTS is received by the sender.

**2) Transmission of DATA and ACK**:

After receiving a successful CTB, the source node sends its broadcast packet . In this broadcast packet, the source node includes the ID of the node which has successfully sent the CTB. This node is referred to as the corresponding node of the source. This node is now responsible for forwarding the broadcast packet and sending an ACK to the source. This ACK packet ensures the reliability of packet dissemination in the desired direction. Although all other nodes between the source and the ACK sender receive the broadcast packet, they do not rebroadcast or acknowledge it. If the ACK packet is not received by the source before the ACK time-out, the source goes back to the first segment based iteration after a random amount of time. Details of this back-off procedure are the same as those of the IEEE 802.11 standard when ACK is not received. Note that there is a maximum number of times (RETmax ) that the source node can go back to the first iteration.

**B. Intersection Broadcast**

When a node is selected to forward a packet and it is outside the transmission range of a repeater, it continues with the directional broadcast protocol. On the other hand, if the node is inside the transmission range of a repeater, the node sends the packet directly to the repeater using the point-to-point IEEE 802.11 protocol. Note that using the GPS and digital road map, each node knows the locations of itself, intersections, and repeaters. According to our protocol, the node inside the transmission range of a repeater sends RTS packet to the repeater and only the repeater replies with the CTS packet. Upon receiving the CTS packet from the repeater, the node sends the DATA packet and the transmission ends when it receives an ACK packet from the repeater. After receiving the broadcast packet, the repeater initiates new directional broadcasts in all road segments other than the road segments.

### 3.6.2 Fully Ad-Hoc Intersection handling with the AMB protocol

The UMB protocol is an effective protocol for urban canyons with repeaters installed at the intersections. It is, however, not necessary to install repeaters at the intersections when the line-of-sight path exists among road segments. A fully ad-hoc extension to the UMB protocol was proposed which handles intersections without infrastructure support when there is line of sight among all road segments. In the AMB protocol, the directional broadcast mechanism of the UMB protocol is employed; however, a new intersection broadcast mechanism is proposed where vehicles find the best candidate among themselves to branch the packet dissemination to other road segments. The vehicle closest to the intersection is a good candidate for this function because it is likely that vehicles closer to the intersection have a better coverage of the other road segments. The new intersection broadcast mechanism is composed of two phases. The first phase is choosing a HUNTER vehicle which tries to select the closest vehicle to the intersection. For this purpose, we will define an intersection region. In the second phase, the HUNTER vehicle initiates a search to find the closest vehicle and in response to this search, vehicles reply with a black-burst according to their distance to the intersection. Once the HUNTER vehicle selects the closest vehicle to the intersection, this vehicle becomes responsible for branching the message to the other road segments.

1) Selecting the HUNTER vehicle: In the directional broadcast protocol , the dissemination of messages is controlled by a subset of vehicles in the network. These vehicles are assigned the function of forwarding the message after the RTB/CTB handshake. Since each of these vehicles choose a new vehicle in the transmission range (R) to forward the message, at least one vehicle is chosen in every R m. Keeping this fact in mind, we have defined an intersection region around each intersection starting at R/2 m before and extending to R/2 m beyond the intersection. Note that at least one vehicle is chosen inside this region during the directional broadcast when the intersection region length is at least R. The first vehicle chosen in the intersection region becomes the HUNTER vehicle. Another reason to choose the intersection

region starting at R/2 m before the intersection is as follows: Since the HUNTER vehicle tries to select a closer vehicle than itself, its transmission range should cover the points closer to the intersection than itself. When we use a transmission region with the proposed borders, in the worst case the HUNTER vehicle is R/2 away from intersection and it can cover the points up to R/2 away at the other side of the intersection.

2) Selecting a vehicle for branching the packet dissemination: Having being selected inside the intersection region, the HUNTER vehicle sends an RTB packet different than regular RTB . This new type of RTB packet which is employed to select the closest vehicle to the intersection is called Intersection RTB (I-RTB).

The black burst response to an I-RTB is different from the response to a regular RTB employed in the directional broadcast i.e., when vehicles receive a regular RTB, the furthest vehicle from the source sends the longest black-burst. On the other hand, when vehicles receive an I-RTB, the vehicle closest to the intersection sends the longest black-burst.

The black-burst length for the I-RTB (L) is calculated using Eq. 1 and Eq. 2. Note that these equations give the L values for the regular RTB response. While determining the black-burst response length for an I-RTB, vehicles use a different d parameter (d) which is the distance from the intersection instead of the distance from the source.

$d=(Xn - Xint )2 + (Yn - Yint )2$ , (3)

where (Xn , Yn ) is the position of the node which sends the black-burst and (Xint , Yint ) is the position of the intersection. Once this Li value is found by using d, L is computed as follows:

$Li (d) = (Nmax - 1) - Li (d)$ (4)

The vehicle which sends the longest black-burst finds the channel idle and sends the CTB packet. After the transmission of DATA and ACK packets, the selection process is finished. This selected vehicle becomes responsible for initiating directional broadcasts in all road directions except the direction where it received the packet from.

# 4 Cache Management in Inter-Vehicular Networks

## 4.1 System Model

### 4.1.1 Network Model

In an inter-vehicle communication environment, the geographical area is divided into small regions, called cells. Each cell has a base station (BS) and a number of mobile terminals (MTs). Inter cell and intra-cell communications are managed by the Bss. The MTs communicate with the BS by wireless links. An MT can move within a cell or between cells while retaining its network connection. An MT can either connect to a BS through a wireless communication channel or disconnect from the BS by operating in the doze (power save) mode . Consider a mobile environment with n cells CI, C2. C.

For each cell Ci, DSi is the database server that can keep pieces of information that may be accessed by other systems. We assume that the database is updated only by the server. A client is a system, which invokes queries for data. Each cell Ci contains a set of clients SI, 2 .....Sm

Each client S1 of the cell C1 can issue the query through the base station Bsi which is directly connected to the database server1DS. A database server (simply server hereafter) can contain more than one database and can indirectly communicate with all mobile clients in the same cell through the Base station BS . A database can be cached in one or more clients in a cell.

Our architecture has a tree structure with multiple levels which consist of a number of distributed databases (DS). Communications between these databases are through their root database DSO. This multi-level tree architecture is more efficient than the hierarchical architecture. Each MN's location profile is stored in one of the root databases according to its current location. Jn this architecture, when the root database crashes, recovery of the failed root database can be performed easily, without disturbing the operation of other databases.

The different DSs may be considered as mobile networks owned by different service providers. They are interconnected together through a PSTN or ATM network. Communication between DSs takes place only through their root databases. Each cell is controlled by a DS2. The user can roam freely within the cell, without the need for registrations. Each DS2 is co located with a BS, which performs query processing on a query arrival. A number of DS2s are clustered into one DS1 and several DSIs are connected to a single DSO. The DSO maintains a location profile for each mobile client currently residing in its service area. It consists of a record for each client in the entire mobile system. The entry contains either a pointer to another DSO where the client is residing or a pointer to the client record that contains a pointer to the DS 1 with which the client is currently associated. Each DS 1 has an entry for every currently residing client, storing a pointer to the DS2 the user is currently visiting. Every DS2 has a copy of the location profiles of the users currently roaming within its area. With this architecture, the frequency of queries to the higher level databases is greatly reduced due to the locality of calling and mobility patterns.

### 4.4.2 Organizations of Databases

**Structure of DSO:** DSO consists an index file. The index file consists of pointers to all the DS1s. When the Vehicle is in the current DS area, the pointer will point the vehicle's location profile stored in the data file. The client location profile contains a pointer to the DS 1 where the client is visiting. All entries in the index file are allocated the same amount of memory. They are stored in the ascending order of client's node ids.

**Structure of DS1:** Each DS1 consists of an index file which contains the record for DS2, residing in the service area of DS 1. Each record in the index file consists of a pointer to the DS2.

**Structure of DS2:** Each database DS2 consists of an index file and data file. The index file consists of pointers to the vehicle clients, residing in DS2. When the client is in the current DS2 area, the pointer will point the client's location profile stored in the data file.

## 4.4.3 Location Update Procedure

Now the description of the location update procedure for a mobile client that moves to another cell.

1. When a mobile vehicle Cl enters into the cell of DS2 from DS1, a request message is sent to the associated DS2 . DS2 then forwards this message to the DS1 of that area.

2. If Cl E DS1, Then the new DS2 is within the same service area DS1 of old DS2 Then the client's entry in DS1 is updated as P(old DS2) =P(new DS2) DS 1 sends a cancellation message to the old DS2. The new DS2 has moved to the new DS1. Add P(new DS2) in new DS1. In DS0, update the entry of DS1 as Update P(old DS1) = P(new DS1). DSO sends a cancellation message to the old DS 1. Old DS1 removes the record of C1. Old DS 1 sends a cancellation message to the old DS2.

3. Old DS2 sends the cache profile of Cl to new DS2 Old DS2 removes the record and cache profile of Cl.

4. New DS2 adds one record and store the received cache profile of Cl

## 4.2 A DISTRIBUTED CACHE MANAGEMENT ARCHITECTURE (DCMA)

### 4.2.1 Cache Placement Algorithm

This algorithm places data caches into some clients on the basis of their weight vector which comprises the following parameters:

- Available Bandwidth
- CPU Speed
- Access Latency
- Cache Hit Ratio

Owing to the low cost for communication among the active nodes belonging to the neighbourhood of a given client both in terms of energy consumption and message exchanges, they form a cooperative cache system for this client. In case of a data miss in the local cache, the client initially searches the data item in its zone before the request is forwarded to the next client that lies on a path towards server.

The algorithm is presented below in detail:

For each client of the cell Ci, j = 1,2 ....... n, let

BW - Available bandwidth (Maximum to the vehicle entering first.)

SPi - CPU speed (Kept constant for all vehicles)

AL - Access Latency (Kept constant)

CRi - Cache Hit Ratio (Kept constant)

where i = 1,2 .....m

1. The weight of the client can be calculated as

W = (SPi +BW + CRi) / AL, (1)

2. Form the vector W = {Si, Wi }, which denotes the client ids and their corresponding weight values, sorted on the descending order.

3. Denote the set of active nodes Sk, (0 <= k < i), which satisfies the following condition Wk > ,6, where /3? is the minimum threshold value for the weight.

4. Each database server DS1 caches the databases into the active nodes set Sk

### 4.5.2 Cache Discovery

This algorithm first looks for the data item in its own cache, when a data request is initiated at a client. In case of a local cache miss, request packet will be broadcast by the client to the set of active clients. When an active client receives the request and has the data item in its local cache, an ack packet will be sent to the requester so as to acknowledge that it has the data item Subsequently a cooperative cache system is formed by the mobile clients belonging to the active node set intended for other clients, owing to the low cost for communicating between the mobile clients belonging to the active node set, both in terms of energy consumption and message exchange.

For each request, one of the following three cases holds:

**Case 1:** Local hit- In this case, a copy of the requested data item is stored in the cache of the requester. If the data item is valid, it is retrieved to serve the query and does not necessitate cooperation.

**Case 2:** Active hit- In this case, the requested data item is stored in the cache of one or more active node neighbours of the requester.

**Case3:** Global hit- In this case, the data item is retrieved from the database server.

1) Cache Discovery Algorithm: A cache discovery algorithm is necessary to determine where the requested item is cached when the destination is not known by the requester.

(i) After the formation of a set of active clients, the vector Sk, dkj } is broadcast by the server to all clients,
where dkj , j = 1,2 .......... is the index of the cached items placed in the active client Sk, k=1,2......

ii) When a data request is initiated at a client, it initially looks for the data item in its own cache (local hit). In case of local cache miss, a request packet is broadcast by the client to the set of active clients.

(iii) When an active client receives the request packet and has the data item in its local cache (i.e., a active hit), an ack packet will be sent to the requester to acknowledge that it has the data item. The ack packet will comprise of two fields, time stamp Ts, which helps in choosing the latest copy of the searched item and weight value W, which helps in choosing the best client node.

(iv)When the ack packet is received by the query client from the active clients, the best active client Sbest with max (T , W) is selected and a confirm packet is sent to the client best. The query client discards the ack packets for the same items received from other clients .
(v) When a confirm packet is received by the client Sbest, it responds with the actual data value to the requested query node.

# 5 Simulation Environment & Study

## 5.1 C Language

### 5.1.1 Introduction

**C** is a general-purpose computer programming language developed between 1969 and 1973 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix Operating System Although C was designed for implementing system software it is also widely used for developing portable application software. C is one of the most popular programming languages of all time and there are very few computer architectures for which a C compiler does not exist. C has greatly influenced many other popular programming languages, most notably C++, which began as an extension to C.

Like most imperative languages in the ALGOL tradition, C has facilities for structured programming and allows lexical variable scope and recursion, while a static type system prevents many unintended operations. In C, all executable code is contained within functions. Function parameters are always passed by value. Pass-by-reference is simulated in C by explicitly passing pointer values. Heterogeneous aggregate data types (struct) allow related data elements to be combined and manipulated as a unit. C program source text is free-format, using the semicolon as a statement terminator.

C also exhibits the following more specific characteristics:

- Partially weak typing; for instance, characters can be used as integers
- Low-level access to computer memory by converting machine addresses to typed pointers
- Function and data pointers supporting ad-hoc run-time polymorphism
- array indexing as a secondary notion, defined in terms of pointer arithmetic
- A preprocessor for macro definition, source code file inclusion, and conditional

compilation

- Complex functionality such as I/O, string manipulation, and mathematical functions consistently delegated to library routines
- A large number of compound operators, such as +=, -=, *=, ++, etc.

### 5.1.2 Arrays

Arrays in C act to store related data under a single variable name with an index, also known as a *subscript*. It is easiest to think of an array as simply a list or ordered grouping for variables of the same type. As such, arrays often help a programmer organize collections of data efficiently and intuitively.

Later we will consider the concept of a *pointer*, fundamental to C, which extends the nature of the array (array can be termed as a constant pointer). For now, we will consider just their declaration and their use.

Arrays in C are indexed starting at 0, as opposed to starting at 1. The first element of the array above is point[0]. The index to the last value in the array is the array size minus one. In the example above the subscripts run from 0 through 5. C does not guarantee bounds checking on array accesses.

During program execution, an out of bounds array access does not always cause a run time error. Your program may happily continue after retrieving a value from point[-1]. To alleviate indexing problems, the sizeof() expression is commonly used when coding loops that process arrays.

C also supports multi dimensional arrays (or, rather, arrays of arrays). The simplest type is a two dimensional array. This creates a rectangular array - each row has the same number of columns.

Array types in C are traditionally of a fixed, static size specified at compile time. (The more recent C99 standard also allows a form of variable-length arrays.) However, it is also possible to allocate a block of memory (of arbitrary size) at run-time, using the standard library's malloc function, and treat it as an array. C's unification of arrays and pointers (see below) means that true arrays and these dynamically-allocated, simulated arrays are virtually interchangeable. Since arrays are always accessed (in effect) via pointers, array accesses are typically *not* checked against the underlying array size, although the compiler may provide bounds checking as an option. Array bounds violations are therefore possible and rather common in carelessly written code, and can lead to various repercussions, including illegal memory accesses, corruption of data, buffer overruns, and run-time exceptions.

C does not have a special provision for declaring multidimensional arrays, but rather relies on recursion within the type system to declare arrays of arrays, which effectively accomplishes the same thing. The index values of the resulting "multidimensional array" can be thought of as increasing in row-major order. Multidimensional arrays are commonly used in numerical algorithms (mainly from applied linear algebra) to store matrices. The structure of the C array is well suited to this particular task. However, since arrays are passed merely as pointers, the bounds of the array must be known fixed values or else explicitly passed to any subroutine that requires them, and dynamically sized arrays of arrays cannot be accessed using double indexing.

### 5.1.3 Pointers

In computer science , a pointer is a programming language data type whose value refers directly to (or "points to") another value stored elsewhere in the computer memory using its address. For high-level programming languages, pointers effectively take the place of general purpose registers in low-level languages such as assembly language or machine code, but may be in available memory. A pointer references a location in memory, and obtaining the value at the location a pointer refers to is known as dereferencing the pointer. A pointer is a simple, more concrete implementation of the more abstract reference data type. Several languages support some type of pointer, although some have more restrictions on their use than others.

Pointers to data significantly improve performance for repetitive operations such as traversing strings, lookup tables, control tables and tree structures. In particular, it is often much cheaper in time and space to copy and dereference pointers than it is to copy and access the data to which the pointers point.

Pointers are also used to hold the addresses of entry points for called subroutines in procedural programming and for run-time linking to dynamic link libraries (DLLs). In object-oriented programming, pointers to functions are used for binding methods, often using what are called virtual method tables.

While "pointer" has been used to refer to references in general, it more properly applies to data structures whose interface explicitly allows the pointer to be manipulated (arithmetically via *pointer arithmetic*) as a memory address, as opposed to a magic cookie or capability where this is not possible.

C supports the use of pointers, a very simple type of reference that records, in effect, the address or location of an object or function in memory. Pointers can be *dereferenced* to access data stored at the address pointed to, or to invoke a pointed-to function. Pointers can be manipulated using assignment and also pointer arithmetic. The run-time representation of a pointer value is typically a raw memory address (perhaps augmented by an offset-within-word field), but since a pointer's type includes the type of the thing pointed to, expressions including pointers can be type-checked at compile time. Pointer arithmetic is automatically scaled by the size of the pointed-to data type. (See Array-pointer interchangeability below.) Pointers are used for many different purposes in C. Text strings are commonly manipulated using pointers into arrays of characters. Dynamic memory allocation, which is described below, is performed using pointers. Many data types, such as trees, are commonly implemented as dynamically allocated struct objects linked together using pointers. Pointers to functions are useful for callbacks from event handlers.

A *null pointer* is a pointer that explicitly points to no valid location. Even though it is created by setting a pointer to literal zero '0', it is not necessarily zero. Dereferencing a null pointer is therefore undefined, typically resulting in a run-time error. Null pointers are useful for indicating special cases such as no *next* pointer in the final node of a linked list, or as an error indication from functions returning pointers. In code, null pointers are usually represented by 0 or NULL, and logically evaluate to false.

Void pointers (void *) point to objects of unknown type, and can therefore be used as "generic" data pointers. Since the size and type of the pointed-to object is not known, void pointers cannot be dereferenced, nor is pointer arithmetic on them allowed, although they can easily be (and in many contexts implicitly are) converted to and from any other object pointer type.

Careless use of pointers is potentially dangerous. Because they are typically unchecked, a pointer variable can be made to point to any arbitrary location, which can cause undesirable effects. Although properly-used pointers point to safe places, they can be made to point to unsafe places by using invalid pointer arithmetic; the objects they point to may be deallocated and reused (dangling pointers); they may be used without having been initialized (wild pointers); or they may be directly assigned an unsafe value using a cast, union, or through another corrupt pointer. In general, C is permissive in allowing manipulation of and conversion between pointer types, although compilers typically provide options for various levels of checking. Some other programming languages address.

## 5.1.4 Recursion

**Recursion** is the process of repeating items in a self-similar way. For instance, when the surfaces of two mirrors are exactly parallel with each other the nested images that occur are a form of infinite recursion. The term has a variety of meanings specific to a variety of disciplines ranging from linguistics to logic. The most common application of recursion is in mathematics and computer science, in which it refers to a method of defining functions in which the function being defined is applied within its own definition. Specifically this defines an infinite number of instances (function values), using a finite expression that for some instances may refer to other instances, but in such a way that no loop or infinite chain of references can occur. The term is also used more generally to describe a process of repeating objects in a self-similar way.

**Formal definitions of recursion**

Recursion in a screen recording program, where the smaller window contains a snapshot of the entire screen.

In mathematics and computer science, a class of objects or methods exhibit recursive behaviour when they can be defined by two properties:

1. A simple base case (or cases), and
2. A set of rules which reduce all other cases toward the base case.

For example, the following is a recursive definition of a person's ancestors:

- One's parents are one's ancestors (*base case*).
- The parents of one's ancestors are also one's ancestors (*recursion step*).

The Fibonacci sequence is a classic example of recursion:

- Fib(0) is 0 [base case]
- Fib(1) is 1 [base case]
- For all integers n > 1: Fib(n) is (Fib(n-1) + Fib(n-2)) [recursive definition]

Many mathematical axioms are based upon recursive rules. For example, the formal definition of the natural numbers in set theory follows: 1 is a natural number, and each natural number has a successor, which is also a natural number. By this base case and recursive rule, one can generate the set of all natural numbers.

A more humorous illustration goes: "*To understand recursion, you must first understand* *recursion.*" Or perhaps more accurate is the following, from Andrew Plotkin: "*If you already know what recursion is, just remember the answer. Otherwise, find someone who is standing closer to Douglas Hofstadter than you are; then ask him or her what recursion is.*"

Recursively defined mathematical objects include functions, sets, and especially fractals.

**Recursion in language**

Linguist Noam Chomsky theorizes that unlimited extension of a language such as English is possible using the recursive device of embedding phrases within sentences. Thus, a chatty person may say, "*Dorothy, who met the wicked Witch of the West in Munchkin Land where her wicked Witch sister was killed, liquidated her with a pail of water.*" Clearly, two simple sentences—"*Dorothy met the Wicked Witch of the West in Munchkin Land*" and "*Her sister was killed in Munchkin Land*"—can be embedded in a third sentence, "*Dorothy liquidated her with a pail of water,*" to obtain a very verbose sentence.

The idea that recursion is an essential property of human language (as Chomsky suggests) is challenged by linguist Daniel Everett in his work *Cultural Constraints on Grammar and Cognition in Pirahã: Another Look at the Design Features of Human Language*, in which he hypothesizes that cultural factors made recursion unnecessary in the development of the Pirahã language. This concept, which challenges Chomsky's idea that recursion is the only trait which differentiates human and animal communication, is currently under debate. Andrew Nevins, David Pesetsky and Cilene Rodrigues provide a debate against this proposal.[1] Indirect proof that Everett's ideas are wrong comes from works in neuro linguistics where it appears that all human beings are endowed with the very same neurobiological structures to manage

with all and only recursive languages. For a review, see Kaan et al. (2002)Recursion in linguistics enables 'discrete infinity' by embedding phrases within phrases of the same type in a hierarchical structure. Without recursion, language does not have 'discrete infinity' and cannot embed sentences into infinity (with a 'Russian nesting doll' effect). Everett contests that language must have discrete infinity, and that the Pirahã language - which he claims lacks recursion - is in fact finite. He likens it to the finite game of chess, which has a finite number of moves but is nevertheless very productive, with novel moves being discovered throughout history.

**Recursion in plain English**

Recursion is the process a procedure goes through when one of the steps of the procedure involves invoking the procedure itself. A procedure that goes through recursion is said to be 'recursive'.

To understand recursion, one must recognize the distinction between a procedure and the running of a procedure. A procedure is a set of steps that are to be taken based on a set of rules. The running of a procedure involves actually following the rules and performing the steps. An analogy might be that a procedure is like a cookbook in that it is the possible steps, while running a procedure is actually preparing the meal.

Recursion is related to, but not the same as, a reference within the specification of a procedure to the execution of some other procedure. For instance, a recipe might refer to cooking vegetables, which is another procedure that in turn requires heating water, and so forth. However, a recursive procedure is special in that (at least) one of its steps calls for a new instance of the very same procedure. This of course immediately creates the danger of an endless loop; recursion can only be properly used in a definition if the step in question is skipped in certain cases so that the procedure can complete. Even if properly defined, a recursive procedure is not easy for humans to perform, as it requires distinguishing the new from the old (partially executed) invocation of the procedure; this requires some administration of how far various simultaneous instances of the procedures have progressed. For this reason recursive definitions are very rare in everyday situations. An example could be the following procedure to find a way through a maze. Proceed forward until reaching either an exit or a

branching point (a dead end is considered a branching point with 0 branches). If the point reached is an exit, terminate. Otherwise try each branch in turn, using the procedure recursively; if every trial fails by reaching only dead ends, return on the path that led to this branching point and report failure. Whether this actually defines a terminating procedure depends on the nature of the maze: it must not allow loops. In any case, executing the procedure requires carefully recording all currently explored branching points, and which of their branches have already been exhaustively tried.

## 5.2 Data structure

In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.

Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, B-trees are particularly well-suited for implementation of databases, while compiler implementations usually use hash tables to look up identifiers.

Data structures are used in almost every program or software system. Data structures provide a means to manage huge amounts of data efficiently, such as large databases and internet indexing services. Usually, efficient data structures are a key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design.

**Basic principles**

Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by an address — a bit string that can be itself stored in memory and manipulated by the program. Thus the record and array data structures are based on computing the addresses of data items with arithmetic operations; while the linked data structures are based on storing addresses of data items within the structure itself. Many data structures use both principles, sometimes combined in non-trivial ways (as in XOR linking).

The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure. The efficiency of a data structure cannot be analysed separately from those operations. This observation motivates the theoretical concept

of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations (including their space and time cost).

## Common data structures

Common data structures include: array, linked list, hash-table, heap, B-tree, red-black tree,, stack, and queue.

## Language support

Most Assembly languages and some low-level languages, such as BCPL, generally lack support for data structures. Many high-level programming languages, and some higher-level assembly languages, such as MASM, on the other hand, have special syntax or other built-in support for certain data structures, such as vectors (one-dimensional arrays) in the C language and multi-dimensional arrays in Pascal.

Most programming languages feature some sorts of library mechanism that allows data structure implementations to be reused by different programs. Modern languages usually come with standard libraries that implement the most common data structures. Examples are the C++ Standard Template Library, the Java Collections Framework, and Microsoft's .NET Framework.

Modern languages also generally support modular programming, the separation between the interface of a library module and its implementation. Some provide opaque data types that allow clients to hide implementation details. Object-oriented programming languages, such as C++, Java and .NET Framework use classes for this purpose.

Many known data structures have concurrent versions that allow multiple computing threads to access the data structure simultaneously.

**Structure Tables and Functions**

Data structures are not only represented in the memory space allocated to a program. They are also embedded in relational database designs. These are referred to as Structure Tables. They are a generic way of storing the data of tree information, often found in modern commercial planning and On-line Analytical Processing software products. (e.g. Arthur planning system, the Merchandise Planning System from I2, Pyramid, and Cubes from Microsoft and Oracle.) In the structure of these products, Variable Measures are stored at the intersections of Member objects found in the Levels of tree Hierarchies of multiple Dimensions. User Defined Attributes can be associated with the members in a dimension. This allows dynamic analysis and aggregation at desired level of the data.

A great advantage of storing the structures and measures data in a relational database, is that it can quickly and easily be maintained by writing programs in the native SQL and SQL extension protocol/language of the native database by maintenance programmers. Since these tools are easy to use and there is an ANSI standard for the SQL, database portability can also be facilitated. An API can be developed to support navigation of the Structures and Cubes.

**Structure functions** most probably required are as follows: First_dimension() Next_dimension() Dimension_name() First_hierarchy() Next_hierarchy() Hierarchy_name() Top_level() Next_level() Bottom_level() Parent_level() Child_level() Top_member() Parent_member() First_child_member() Next_child_member() Ancestor_member_in_level() Descendant_member_in_level() First_peer_member() Next_peer_member() First_attribute_of_member() Next_attribute_of_member() etc.

**Cube functions** most probably required are as follows: Value_of_member_at_intersection_of cube() Sum_of_children_of_member_for_measure()

Sum_of_children_of_member_in_dimension_for_measure()

Contribution_of_member_for_measure() Portion_of_parent_member_for_measure() etc.

These functions can be implemented in the relational database as stored procedures and can also be implemented for use in-memory. If both are implemented, then a suit of functions for storing and loading cubes must be implemented. In a three tier client server design, the cube resides in the memory of a Cube service. A presentation layer service connects to the Cube service. A database service connects the Cube service and the database engine service and converts from the in-memory format to the in-database format of the specific database and database-language required.

## 5.3 Linked Lists

In computer science, a **linked list** (or more clearly, "singly linked list") is a data structure that consists of a sequence of nodes each of which contains a reference (i.e., a *link*) to the next node in the sequence.

Linked lists are among the simplest and most common data structures. They can be used to implement several other common abstract data structures, including stacks, queues, associative arrays, and symbolic expressions, though it is not uncommon to implement the other data structures directly without using a list as the basis of implementation.

The principal benefit of a linked list over a conventional array is that the list elements can easily be added or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk. Linked lists allow insertion and removal of nodes at any point in the list, and can do so with a constant number of operations if the link previous to the link being added or removed is maintained during list traversal.

On the other hand, simple linked lists by themselves do not allow random access to the data other than the first node's data, or any form of efficient indexing. Thus, many basic operations — such as obtaining the last node of the list (assuming that the last node is not maintained as separate node reference in the list structure), or finding a node that contains a given datum, or locating the place where a new node should be inserted — may require scanning most or all of the list elements.

### Basic concepts and nomenclature

Each record of a linked list is often called an **element** or **node**.

The field of each node that contains the address of the next node is usually called the ***next***

**link** or ***next* pointer**. The remaining fields are known as the **data**, **information**, **value**, **cargo**, or **payload** fields.

The **head** of a list is its first node, and the **tail** is the last node (or a pointer thereto). In Lisp and some derived languages, the tail may be called the **CDR** (pronounced *could-er*) of the list, while the payload of the head node may be called the **CAR**.

Bob (bottom) has the key to box 201, which contains the first half of the book and a key to box 102, which contains the rest of the book.

## Post office box analogy

The concept of a linked list can be explained by a simple analogy to real-world post office boxes. Suppose Alice is a spy who wishes to give a codebook to Bob by putting it in a post office box and then giving him the key. However, the book is too thick to fit in a single post office box, so instead she divides the book into two halves and purchases two post office boxes. In the first box, she puts the first half of the book and a key to the second box, and in the second box she puts the second half of the book. She then gives Bob a key to the first box. No matter how large the book is, this scheme can be extended to any number of boxes by always putting the key to each box in the previous box.

In this analogy, the boxes correspond to *elements* or *nodes*, the keys correspond to *pointers*, and the book itself is the *data*. The key given to Bob is the *head* pointer, while those stored in the boxes are *next* pointers. The scheme as described above is a *singly-linked list* (see below).

## Linear and circular lists

In the last node of a list, the link field often contains a **null** reference, a special value that is interpreted by programs as meaning "there is no such node". A less common convention is to make it point to the first node of the list; in that case the list is said to be **circular** or **circularly linked**; otherwise it is said to be **open** or **linear**.

## Singly, doubly, and multiply linked lists

Singly linked lists contain nodes which have a data field as well as a *next* field, which points to the next node in the linked list.

In a **doubly linked list**, each node contains, besides the next-node link, a second link field pointing to the *previous* node in the sequence. The two links may be called **forward**(**s**) and **backwards**, or **next** and **previous**.

The technique known as XOR-linking allows a doubly linked list to be implemented using a single link field in each node. However, this technique requires the ability to do bit operations on addresses, and therefore may not be available in some high-level languages.

In a **multiply linked list**, each node contains two or more link fields, each field being used to connect the same set of data records in a different order (e.g., by name, by department, by date of birth, etc.). (While doubly linked lists can be seen as special cases of multiply linked list, the fact that the two orders are opposite to each other leads to simpler and more efficient algorithms, so they are usually treated as a separate case.)

In the case of a doubly circular linked list, the only change that occurs is the end, or "tail" of the said list is linked back to the front, "head", of the list and vice versa.

## Sentinel nodes

In some implementations, an extra **sentinel** or **dummy** node may be added before the first data record and/or after the last one. This convention simplifies and accelerates some list-handling algorithms, by ensuring that all links can be safely dereferenced and that every list (even one that contains no data elements) always has a "first" and "last" node.

## Empty lists

An **empty list** is a list that contains no data records. This is usually the same as saying that it has zero nodes. If sentinel nodes are being used, the list is usually said to be empty when it has only sentinel nodes.

### Hash linking

The link fields need not be physically part of the nodes. If the data records are stored in an array and referenced by their indices, the link field may be stored in a separate array with the same indices as the data records.

### List handles

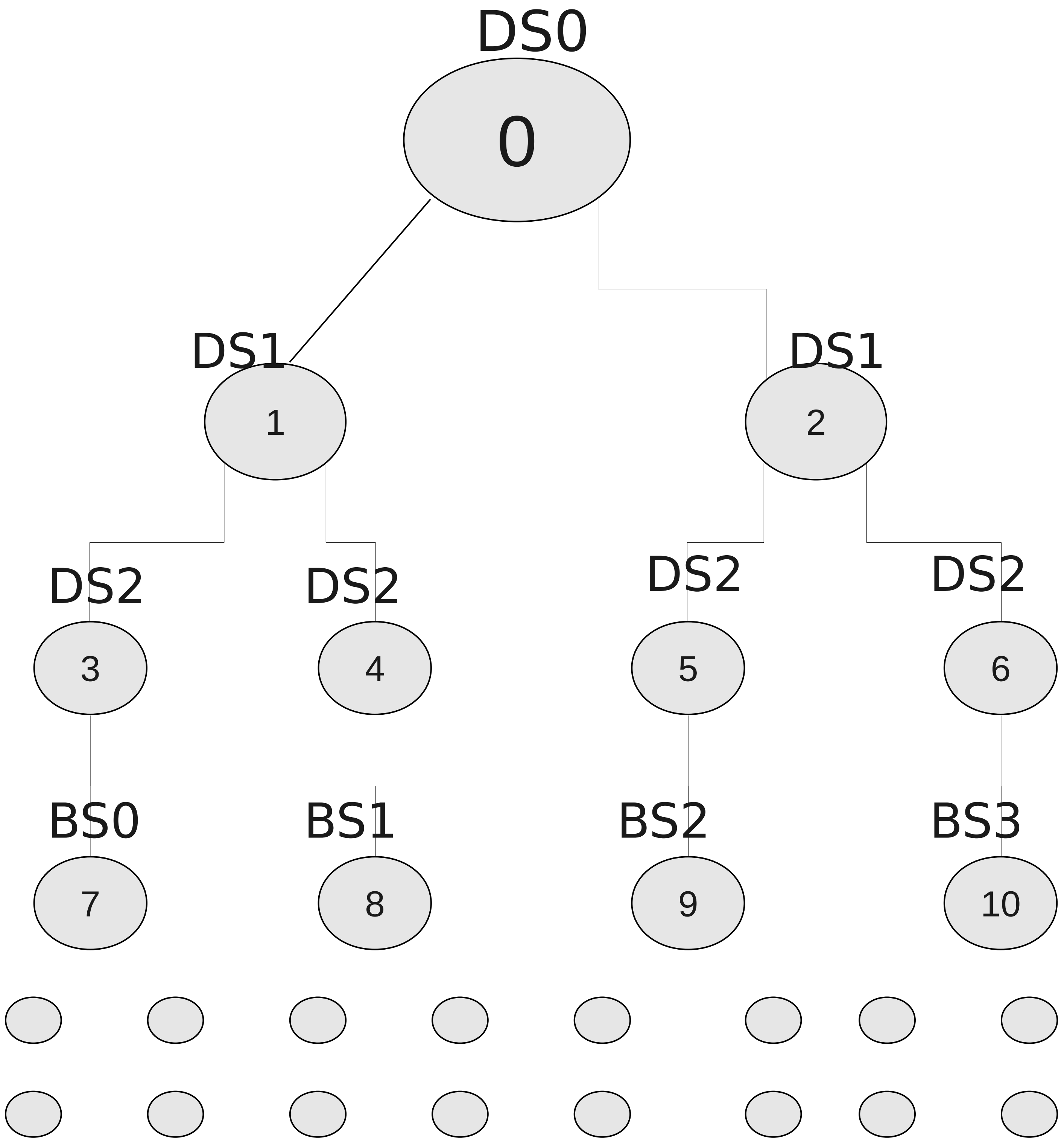
Since a reference to the first node gives access to the whole list, that reference is often called the **address**, **pointer**, or **handle** of the latter. Algorithms that manipulate linked lists usually get such handles to the input lists and return the handles to the resulting lists. In fact, in the context of such algorithms, the word "list" often means "list handle". In some situations, however, it may be convenient to refer to a list by a handle that consists of two links, pointing to its first and last nodes.

### Combining alternatives

The alternatives listed above may be arbitrarily combined in almost every way, so one may have circular doubly linked lists without sentinels, circular singly linked lists with sentinels, etc.

# 6. Implementation Study

## 6.1 Structure



## 6.2 Linked Lists

### 6.2.1 Mobile Clients

Sixteen Mobile clients were used so a list was designed to store all the necessary details of mobiles in the structure.

```
struct mc
{
        char *loc,*ds2,*ds1,*ds;
        float BW,SP,AL,CR,k;
        float W;
        int aa,bb,cc,dd;
        int mid,ds2id,ds1id,dsid,lid;
        struct mc *mnext;
};
```

### 6.2.2 Data Server 2

Four Data Server were used to store their respective mobile clients in a particular location.

```
struct ds2
{       int count;
        struct mc *A[16];
        struct ds2 *d2next;


};
```

### 6.2.3 Data Server 1

Two Data Server 1 were used to store the Data Server 2 information and their Index

```
struct ds1
{

        struct ds2 *B,*C;
        struct ds1 *d1next;
};
```

### 6.2.4 Data Server

One Data Server  was used to store the Data Server 1 information and their Index

```
struct ds
{

        struct ds1 *D,*E;
        struct ds *dnext;
};
```

## 6.3 Organization of Database

A total of four different database were maintained , Viz.

1 A database to store 16 mobile clients

2 A database to store 4 Data Server 2

3 A database to store 2 Data Server 1

4 A database for Data Server

## 6.4 Functions

### 6.4.1 Function for Mobile Clients

```
	struct mc*funm(struct mc **q,int j,char loca[4],int ds2ida,int ds1ida,int dsida,char ds2a[4],char ds1a[4],char dsa[4],int llm)
	{
		int i;
		struct mc *temp,*r;
		temp = *q;
		if(*q==NULL)
			{
				temp=malloc(sizeof(struct mc));
				temp->mid=j;
				temp->lid=llm;
				temp->ds2id=ds2ida;
				temp->ds1id=ds1ida;
				temp->dsid=dsida;
				temp->loc=loca;
				temp->ds2=ds2a;
				temp->ds1=ds1a;
				temp->ds=dsa;


				temp->mnext=NULL;
				*q=temp;
				return temp;
			}
		else
```

```
        {
                temp = (*q);
                while(temp->mnext!=NULL)
                {
                        temp = temp->mnext;
                }
                r = malloc(sizeof(struct mc));
                r->mid=j;
                r->lid=llm;
                r->ds2id=ds2ida;
                r->ds1id=ds1ida;
                r->dsid=dsida;
                r->loc=loca;
                r->ds2=ds2a;
                r->ds1=ds1a;
                r->ds=dsa;

                r->mnext=NULL;
                temp->mnext=r;
                return r;
        }

}
```

### 6.4.2 Function for DS2

```
struct ds2 *funds2(struct ds2 **q,struct mc *j[],int x)
    {
        int i;
        struct ds2 *temp,*r;
        temp = *q;
        if(*q==NULL)
            {
                temp=malloc(sizeof(struct ds2));
                for(i=0;i<x;i++)
                    {
                        temp->A[i]=j[i];

                    }

                temp->d2next=NULL;
                *q=temp;
                return temp;
            }
        else
            {
                temp=*q;
                while(temp->d2next!=NULL)
                {
                    temp = temp->d2next;
                }
```

```
                                        r = malloc(sizeof(struct ds2));
                                        for(i=0;i<x;i++)
                                              {
                                                    r->A[i]=j[i];


                                              }
                                        r->d2next=NULL;
                                        temp->d2next=r;


                                        return r;
                                  }
                     }
```

### 6.4.3 Function for DS1

```
      struct ds1 *funds1(struct ds1 **q,struct ds2 *j,struct ds2 *k)
      {
      struct ds1 *temp,*r;
      temp = *q;
      if(*q==NULL)
                  {
                        temp=malloc(sizeof(struct ds1));
                        temp->B=j;
                        temp->C=k;
                        temp->d1next=NULL;
                        *q=temp;
                        return temp;
                  }
      else
                  {
                        temp=*q;
```

```
                while(temp->d1next!=NULL)
                {
                        temp = temp->d1next;
                }
                r = malloc(sizeof(struct ds1));
                r->B=j;
                r->C=k;
                r->d1next=NULL;
                temp->d1next=r;
                return r;
        }
}
```

### 6.4.4 Function for DS

```
void funds(struct ds **q,struct ds1 *j,struct ds1 *k)
{
struct ds *temp,*r;
temp = *q;
if(*q==NULL)
        {
                temp=malloc(sizeof(struct ds));
                temp->D=j;
                temp->E=k;
                temp->dnext=NULL;
                *q=temp;

        }
else
```

```
                {
                        temp=*q;
                        while(temp->dnext!=NULL)
                        {
                                temp = temp->dnext;
                        }
                        r = malloc(sizeof(struct ds));
                        r->D=j;
                        r->E=k;
                        r->dnext=NULL;
                        temp->dnext=r;

                }
        }
```

### 6.4.5 Funtion for Cache Placement

```
        void cacheplacement(struct mc *q,float BW,float SP,float AL,float CR,float k,float W,int
aa,int bb,int   cc,int dd)
        {
                struct mc * temp;
                temp=q;
                q->BW=BW;
                q->SP=SP;
                q->AL=AL;
                q->CR=CR;
                q->k=k;
                q->W=W;
```

```
if(W==9.000000)
        {
                q->aa=aa;
                q->bb=bb;
                q->cc=cc;
                q->dd=dd;
        }
if(W==8.500000)
        {
                q->aa=bb;
                q->bb=cc;
                q->cc=dd;
                q->dd=0;
        }
if(W==8.000000)
        {
                q->aa=cc;
                q->bb=dd;
                q->cc=0;
                q->dd=0;
        }
if(W!=8.000000 && W!=8.500000 && W!=9.000000 )
        {
                q->aa=dd;
                q->bb=0;
                q->cc=0;
                q->dd=0;
        }
```

```
}
```

### 6.4.6 Function for Cache Discovery

```
struct mc *cachediscovery(struct mc *q,int cdid,int g)
{
        struct mc *temp;
        int te;
        temp=q;
        if(temp->mid==cdid)
                {



                        return temp;

                }
        else
                {
                        q=temp->mnext;
                        cachediscovery(q,cdid,g);
                }
}
```

### 6.4.8 Functions for Cache search

```
int finalsearch(struct mc **q,struct mc **r,struct mc **s,struct mc **t,int cdid,int g)
{
        struct mc *o,*p,*m,*n;
        o=*q;p=*r;m=*s;n=*t;
        if(o->aa == g || o->bb==g || o->cc==g || o->dd==g)
                {
                        printf("\nGlobal server hit\n");
                        printf("Data requested is at DS server\n");
                        printf("Client Id is mid[%d]\n",o->mid);
                        return 1;
                }
        if(p->aa == g || p->bb==g || p->cc==g || p->dd==g)
                {
                        printf("\nGlobal server hit \n");
                        printf("Client requested is at DS server\n");
                        printf("His Id is mid[%d]\n",p->mid);
                        return 1;
                }
        if(m->aa == g || m->bb==g || m->cc==g || m->dd==g)
                {
                        printf("\nGlobal server hit \n");
                        printf("Data requested is at DS server\n");
                        printf("Client Id is mid[%d]\n",m->mid);
                        return 1;
                }
                if(n->aa == g || n->bb==g || n->cc==g || n->dd==g)
                        {
                                printf("\nGlobal server hit \n");
                                printf("Data requested is at DS server\n");
                                printf("Client Id is mid[%d]\n",n->mid);
                                return 1;
                        }

                else
                        {
                                printf("Sorry this data is not at DS service area too\n");
                                return 2;

                        }
        }
int globalsearch(struct mc **q,struct mc **r,int cdid,int g)
        {
                struct mc *o,*p;
                o=*q;p=*r;
                if(o->aa == g || o->bb==g || o->cc==g || o->dd==g)
                        {
                                printf("\nGlobal hit dude!\n");
                                printf("Data requested is at global server\n");
                                printf("Client Id is mid[%d]\n",o->mid);
                                return 1;
                        }
                if(p->aa == g || p->bb==g || p->cc==g || p->dd==g)
                        {
                                printf("\nGlobal hit dude!\n");
```

```
					printf("Data requested is at global server\n");

					printf("Client Id is mid[%d]\n",p->mid);

					return 1;

				}


			else

				{


					return 2;

					printf("Sorry this data is not at global service area \n");

				}


		}

int localsearch(struct mc **q,int mid,int g)

		{

			struct mc *temp;

			temp=*q;

			if(temp->aa==g)

				{

					printf("\nLocal hit\n");

					return 1;

				}

			else return 2;

		}

int cachefind(struct mc **q,int cdid,int g)

		{

			struct mc *temp;

			int a;

			temp = *q;
```

```
        a=temp->mid;

        if(temp->aa == g || temp->bb==g || temp->cc==g || temp->dd==g)

                {

                        printf("\nActive hit \n");

                        printf("Data requested is at your local server\n");

                        printf("His Id is mid[%d]\n",temp->mid);

                        return 1;

                }


        else

                {


                        return 2;

                }


}
```

# 7 Conclusion

## 7.1 Conclusion

## 7.2 Further Studies

## 7.1 Conclusion

Cache Managements in mobile environments, in general, includes database organization, location update, cache placement and cache discovery. In this thesis, we have designed distributed cache management architecture (DCMA) for vehicular hosts which include all the above techniques. The architecture presented effective technique for location update, in case of moving vehicle clients. The architecture improved the network utilization, achieves lower latency and packet loss, reduced network bandwidth consumption and reduced data server workload.

## 7.2 Further Studies

Inter Vehicular Communication networking is a rather hot concept in computer communications. This means that there is much research going on and many issues that remains to be solved. I have focused on the Cache Management Architecture.

Furthermore, Cache Consistency and Cache Replacement algorithms form the distributed cache management architecture for mobile computing environments can also be implemented in the IVC environment to higher consistency .

# 8 References

[1] Dimitri Bertsekas and Robert Gallager, “Data Networks - 2nd ed”. Prentice Hall, New Jersey, ISBN 0-13-200916-1.

[2] Bommaiah, McAuley and Talpade. AMRoute, “Adhoc Multicast Routing Protocol”, Internet draft, draft- talpade-manet-amroute-00.txt, August 1998. Work in progress.

[3] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu and Jorjeta Jetcheva, “A performance Comparison of Multi-hop Wireless Ad Hoc Network Routing Protocols”. Mobicom'98, Dallas Texas, 25–30 October, 1998.

[4] Josh Broch, David B. Johnsson, David A. Maltz, “The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks”. Internet Draft, draft-ietf-manet-dsr-00.txt, March 1998. Work in progress.

[5] G.Anandharaj , Dr. R.Anitha.”A distributed Cache Management Architecture for Mobile Computing Environments”, 2009 IEEE International Advance Computing Conference (IACC 2009), Patiala, India 6-7 March 2009.

[6] M.Scott Corson, S. Papademetriou, Philip Papadopolous, Vincent D. Park and Amir Qayyum, “An Internet MANET Encapsulation Protocol (IMEP) Specification”. Internet draft, draft-ietf-manet-imep- spec01.txt, August 1998. Work in progress.

[8] Zygmunt J. Haas and Marc R. Pearlman, “The Zone Routing Protocol (ZRP) for Ad Hoc Networks”, Internet draft, draft-ietf-manet-zone-zrp-01.txt, August 1998. Work in progress.

[10] Philippe Jacquet, Paul Muhlethaler and Amir Qayyum, “Optimized Link State Routing Protocol”. Internet draft, draft-ietf-manet-olsr-00.txt, November 1998. Work in progress.

[11] Mingliang Jiang, Jinyang Li and Yong Chiang Tay, “Cluster Based Routing Protocol (CBRP) Functional

specification”. Internet draft, draft-ietf-manet-cbrp-spec-00.txt, August 1998. Work in progress.

[12] David B. Johnson and David A.Maltz, “Dynamic source routing in ad hoc wireless networks”. In Mobile Computing, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages

153-181. Kluwer Academic Publishers.

[13] David B. Johnson and David A. Maltz, “Protocols for adaptive wireless and mobile computing”. In IEEE Personal Communications, 3(1), February 1996.

[16] Vincent D. Park and M. Scott Corson, “Temporally-Ordered Routing Algorithm (TORA) Version 1:Functional specification”. Internet draft, draft-ietf-manet-tora-spec-01.txt, August 1998. Work in progress.

[19] Charles E. Perkins, “Ad Hoc On Demand Distance Vector (AODV) Routing”. Internet draft, draft-ietf-manet-aodv-01.txt, August 1998. Work in progress.

[22] Charles E. Perkins and Pravin Bhagwat, “Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers”. In Proceedings of the SIGCOM '94 Conference on Communications Architecture, protocols and Applications, pages234-244, August 1994. A revised version of the paper is available from http://www.cs.umd.edu/projects/mcml/papers/Sigcomm94.ps. (1998-11-29)

[23] Larry L. Peterson and Bruce S. Davie, “Computer Networks - A Systems Approach”. San Francisco, Morgan Kaufmann Publishers Inc. ISBN 1-55860-368-9.

[26] Raghupathy Sivakumar, Prasun Sinha and Vaduvur Bharghavan, “Core Extraction Distributed Ad hoc Routing (CEDAR) Specification”, Internet draft, draft-ietf-manet-cedar-spec-00.txt, October 1998. Work in progress.

[27] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, “The broadcast [28] W. Lou and J. Wu, “On reducing broadcast redundancy in ad hoc pp. 111–122, April-June 2002.

[29] I. Stojmenovic, S. Seddigh, and J. Zunic, “Dominating sets and neighbour Jan. 2002.

[30] L. Briesemeister and G. Hommel, “Role-based multicast in highly mo-pp. 45–50, Aug. 2000.

[31]M. Torrent-Moreno, D. Jiang, and H. Hartenstein, “Broadcast Reception Rates and Effects of Priority Access in 802.11-Based Vehicular Ad-Hoc Networks”, The 1st ACM international workshop on Vehicular ad hoc networks (VANET04), Oct. 2004.

[32]S. Yang, H. Refai, and M. Xiaomin, “CSMA based Inter-Vehicle Communication Using Distributed and Polling Coordination”, The 8th International IEEE Conference on Intelligent Transportation Systems (ITSC05), Sep. 2005.

[33]C. Suthaputchakun, and A. Ganz, “Military Inter-Vehicle Communication with message

Priority using IEEE 802.11e”, The 25th International IEEE Military Communications Conference (MILCOM06), Oct. 2006

[34]IEEE802.11 WG, IEEE Std 802.11, “International Standard for Information Technology. Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks. Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, Sep. 1999.

[35] 802.11 WG, IEEE Std 802.11e/D10.0, “Draft Amendment to Standard [for] Information Technology — Telecommunications and Information Exchange Between Systems — LAN/MAN Specific Requirements —Part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Quality of Service (QoS) Enhancements”, Sep. 2004.