COMPARATIVE STUDY OF CAN, PASTRY, KADEMLIA AND CHORD DHTS

Lacine KABRE[1] and Telesphore TIENDREBEOGO[2]

[1] Departement of Mathematics and Computer Engineering Joseph Ki-Zerbo University, Ouagadougou, Burkina Faso
[2] Departement of Mathematics and Computer Engineering Nazi Boni University, Bobo-Dioulasso, Burkina Faso

ABSTRACT.

Peer-to-Peer (P2P) systems allow decentralization, sharing of all the resources of a network with direct communication and collaboration between nodes. There are three main families of P2P networks: the centralized architecture, the decentralized architecture that can be structured or unstructured and the hybrid architecture. Today, there are several implementations for structured decentralized architectures. This implies that the insertion and search algorithms are different. Among them we have; Chord, Pastry, Kademlia, CAN(Content Addressable Network) . The choice of these DHTs (Distributed Hash Table) for an application is made on the basis of their performances. Studies of each of these DHTs mentioned have been done, proving their performance. But a comparative study of the four DHTs Chord, Pastry, CAN, Kademlia has not been clearly addressed by previous works. In this paper, we have conducted a comparative theoretical study of the DHTs Chord, Pastry, CAN, Kademlia. Then, by simulation, we have evaluated the performances in terms of latency, number of hops and number of transmitted messages. Our study clearly shows the differences between mathematically established performance and actual performance in an environment with less restriction. This analysis was made from the data obtained by using the simple network layer of the PeerfactSim simulator. This simulator abstracts the different network layers, which gives the advantage of testing the performances with reasonable accuracy. The use of the single network layer can be considered an ideal case because the node searches are done locally.

KEYWORDS

P2P, DHT, peerfactSim, CAN, Pastry, Kademlia, Chord.

1. INTRODUCTION

With the arrival of concept of P2P, people have changed the way they use Internet. It allows users to create an application based on a virtual network in which data can be exchanged, even with the restrictions of the underlying networks of the Internet, such as firewalls, dynamic IP address assignment (1). This new generation of file sharing systems has been completely
decentralized in terms of storage and resource sharing. It has many advantages that break the limits of the client/server model. We have among others:

- the absence of a central element allowing to increase considerably the fault tolerance of the services, because if the disappearance of a server generates the unavailability of the offered service, the disappearance of a peer is without consequence;
- the fair distribution of data and executed tasks balances the traffic of the underlying network and the load assigned to each participant;

- the potential aggregation of the computing power and storage space of millions of machines offers users a power that exceeds any existing centralized infrastructure;
- the use of machines belonging to different owners allows to reduce the costs related to the purchase and maintenance of equipment.

In the absence of centralized and fixed coordination, P2P networks, for more scalability, allow interaction between the nodes that constitute them. This scalability is made even more robust by the introduction of Distributed Hash Tables (DHTs) in P2P systems. When a peer-to-peer network is equipped with this indexing distributed technique, over all the nodes, it is then referred to as a structured system. The most known are CAN [21], Kademlia [6], Pastry [20] and Chord [11]. The principle of these structured networks is to organize the peers according to a structured logical network. The routing protocols in P2P networks using DHTs are simply called DHTs. A Distributed Hash Table is a data structure that associates a key with an object by hashing and create an index. A key is the result of a hash function (e.g. SHA-1: Secure Hash Algorithm) applied to an element of the object. For example, the key of a peer is the result of applying a hash function to the IP address of its node, while the key of an object is the hash of its name. The particularity of distributed hash tables is the new distribution and localization technique it proposes. In a DHT, peers and objects are identified in the same namespace. Each node is responsible for a portion of the keys in the system and each object is stored in the node whose identifier is closest to its key according to the distance metric used. These structured P2P networks give several types of routing topology that will change the performance of peer-to peer systems. The trend today is to use these DHTs in all peer-to-peer applications.

In most of the DHT comparison work, a study of the four protocols that are the subject of our study is missing. The objective of our work is to make a comparison of the theoretical and practical performances of the CAN, pastry, Kademlia and chord protocols. We are interested in metrics such as the number of hops, the latency, the success rate of the messages sent. Our contribution is to compare in terms of hop count, latency and transmitted messages both theoretical results and simulation results of CAN, Pastry, Kademlia and chord protocols by using the peerfactSim [18] simulator. This simulator abstracts the different network layers which gives the advantage to test the performances with a good accuracy.

Our article is subdivided in three main parts. First, we analyze the operating principles of the four DHTs that are the subject of our study. Secondly, we proceed to their simulation and finish with a comparative analyse.

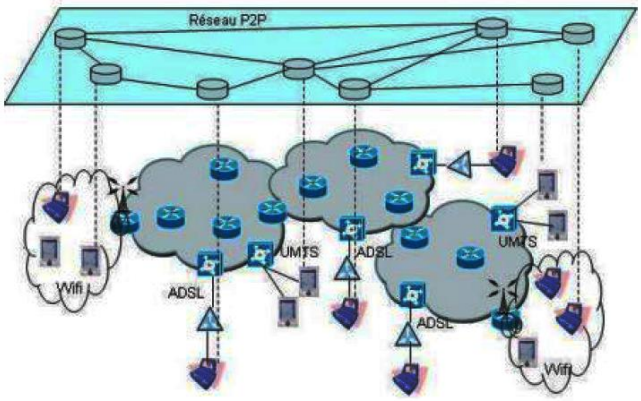

Fig 1:Achitecture of a P2P overlay network, from [9].

2. RELATED WORKS

2.1. Principle of the Kademlia DHT

2.1.1. Distance between Two Nodes With XOR Metric

The notion of distance in the Kademlia protocol is based on the "or-exclusive bit by bit" operator. For two given identifiers (id), the distance between them is:

$$d(x; y) = XOR(binary(x); binary(y)) \quad (1)$$

The mathematical properties of the XOR [3] allow us to say that its result is always positive or null (only if $x = y$), $d(x;y) = d(y; x)$, which guarantees the symmetry of this metric. Thus we obtain the distance between two id.

2.1.2. The Kademlia protocol works

Kademlia is a network protocol based on distributed hash tables and the XOR metric. It uses the principle of distributed hash tables to prioritize and organize the network into a binary tree. The complexity being in Log(n) during a search (Fig:2). One of its main advantages compared to other DHT systems is the reduced number of configuration messages to send. In terms of searching for its neighbors, a node must be able to know the network topology to be able to transmit its requests with a short latency. This latency is kept to a minimum by using the UDP [10] protocol to send all configuration and communication messages within the network. Similarly, requests are sent in parallel and not synchronized to speed up transmission times and avoid waiting times from faulty nodes. Each node in the network has its own identifier (id) which is a 160 bit key. The entries of the hash table are also 160 bit keys, and are associated with a value, which in our case will be a structure, which gives the pair (key-value) as element of the table. A node in the network stores in its table only those (key-value) pairs whose key is nearest to its id. The other main advantage of Kademlia is the new *XOR* metric used. Since it is a symmetric metric, it allows Kademlia's nodes to receive query responses from the same set of nodes as contained in its table (the set of nodes that corresponds to the set of nodes contacted). Indeed, the nodes of the Kademlia network update their routing table with each UDP message received, and store, if necessary, the information relating to the sending node. Each node "A" in Kademlia maintains a routing table composed of a set of nodes called buckets. The nodes in the buckets are ranked in order of distance according to the *XOR* metric. The average degree of a node is therefore $O(k*log^b{}_2N)$. All the nodes of the same bucket are at the same distance from node "A". Each bucket corresponds to a sub-tree (Fig:2) and contains k neighbors classified according to their seniority in the system.

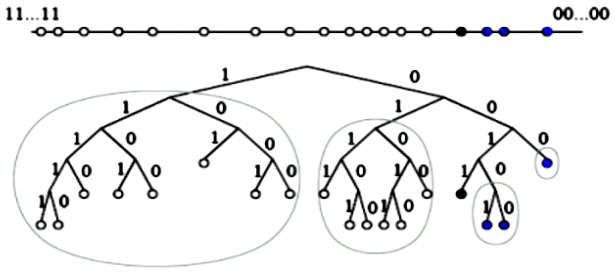

Fig 2: Kademlia tree, from [6]

The blue circles in Figure (Fig:2) represent the neighbors of node 0011. The location of an identifier (or key), which can represent a connected machine or a file in the network, relies on a unique search algorithm, unlike many other DHTs. Indeed, the space of identifiers is treated by Kademlia as a binary tree where each node of the network is a leaf corresponding to a unique identifier. If the tree is not complete, the search for an identifier may not converge to a unique identifier, but return all the nearest identifiers. For a given node, we divide the binary tree into a series of sub-trees that do not contain the intended node. The first sub-tree obtained corresponds to half of the starting tree not containing the node. The next one is the half of the remaining tree that does not contain the node, and so on. The Kademlia protocol ensures that all nodes in the network know at least one node in each of its subtrees. This guarantee allows any node in the network to contact another node by its identifier.

2.1.3. Arrival and Departure of a node

When a new peer *"A"* arrives on the overlay, it generates its own identifier *(id)*, then sends a message containing its key. When a peer receives a message, it updates the appropriate bucket. If the sending node is already in that bucket, it is moved down the list. Otherwise, if the bucket is not full, the new entry is inserted. If the bucket is full, node *"A"* sends a PING to the oldest neighbor in this bucket. If the latter responds to the PING, it is then moved to the end of the list and the new entry is ignored. Kademlia therefore keeps the oldest nodes in the network in its table. A peer is detected as unavailable when the other peers can no longer communicate with it due to a failure or breakdown. Thus, the other peers update their routing tables. Applications like Bittorent and mule work using Kademlia.

2.2. Principle of the Chord DHT

2.2.1. Protocol Operation

In a Chord overlay network [11], nodes are organized in a directed ring topology of 2^m peers, where m is the number of bits in the used hash function (typically 160 bits). Each peer is linked to a predecessor and several successors. A node maintains a routing table to nodes of identifiers $n + 2^{K-1}$ with k the number of bits used for identification. The table is thus of logarithmic size and the key search operations are performed in logarithmic time. The Chord protocol keeps track in its *K*-entry routing table and users can contact *K* nodes in the network directly to transmit their requests.

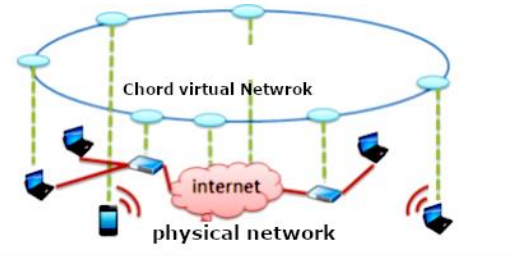

Fig 3: Architecture of a Chord network [19].

2.2.2. Routing in Chord

The Chord [11]: protocol is the simplest implementation of a DHT. There is no routing table. Each node communicates only with its successor in a stringlike fashion. For example, when a

query is made for a key, each node examines whether the key being searched for is less than or equal to its successor's identifier. If there is a match, then the node returns its successor's id. In addition, there is a more optimal routing. Figure (Fig:4) describes how in optimal routing one can very quickly find a user in this decentralized network. Suppose that *user "A"* is looking for user "B", because he has a file that he is looking for. User *"A"* needs the IP address of this user, but only knows the key *"Key(B)"* in the Chord network (this key alone does not give a route in the physical network of the Internet). The user *"A"* has stored in his memory the *log (N)* nodes corresponding to the log(N) additional links he has on the virtual ring. If B is among these nodes, then *"A"* finds directly the address of "B". Otherwise, *"A"* will send its request to the node whose key is nearest to *"Key(B)"* among the *log (N)* nodes that it has stored.

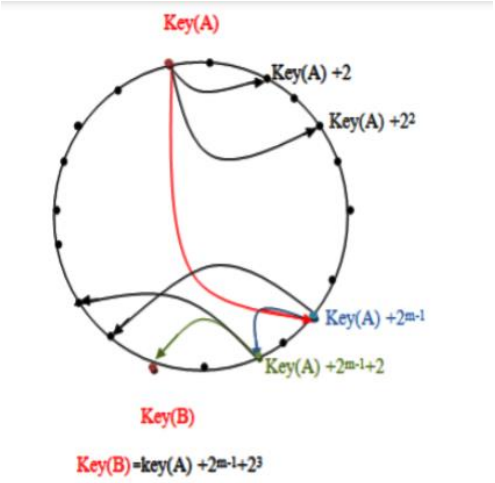

Fig 3: Optimal routing in Chord [19].

Chord supports well the scaling. Different works [14] have been conducted and show that the average length of path used to route a request evolves in *O(log(N))*, with *N* the size of the network.

2.2.3. Arrival and Departure of A Node

When a new peer X arrives on the overlay, it first takes its place in the ring according to its identifier (node id). Then it sends a message to the nearest successor, which sends all its lower keys to the peer X. Then the peers update their routing tables. A peer is detected as unavailable when the other peers cannot communicate with it because the failing peer has left the overlay or a failure has been introduced. Then the other peers update their routing tables. There are several applications that use Chord as their basic protocol. There are :

- CFS (Collaborative File System) [5]: a file system distributed at the scale of the Internet, and where each file is shared in blocks;
- ConChord [1]: a distributed infrastructure that uses CFS and issues SDSI (Simple Distributed Security Infrastructure) certificates.

2.3. Principle of the Pastry DHT

Pastry is a purely decentralized system [20] inspired by Chord's ring topology, but the identification space is circular and undirected, so searches can be done in either direction. The

identifier of a node has a size of 128 bits and is obtained randomly through a hash function based on the IP address or by using a public key.

The identifiers are generated and distributed uniformly over the identifier space. Routing is guaranteed by distinguishing the node in the table, presenting the longest common prefix with the resource identifier to make the next routing. Pastry is based on a prefix-based routing mechanism.

2.3.1. Routing in Pastry

Let's assume that a node "A" is looking for another node with to key "K" to transmit a message "m", it routes the message the node that has the identifier (*idNode*) nearsest to numerically to the key "K". In other words to a node where idNode shares with the key K, a common common prefix. This routing is based on the prefix matching. The figure(Fig:5) shows an example of message routing, where node 89B1CC routes a message to the node that key is E19BCA. In the first step, node 89B1CC sends the message to a node whose identifier has a digit in common with the key it is looking for. In this example the node is EF25BA. During the second step, the node E1CC5A retransmits "m" to a node whose identifier has three digits in common with the searched key, here the node E192C5. So on until *"m"* reaches the node whose key is E19BCA.

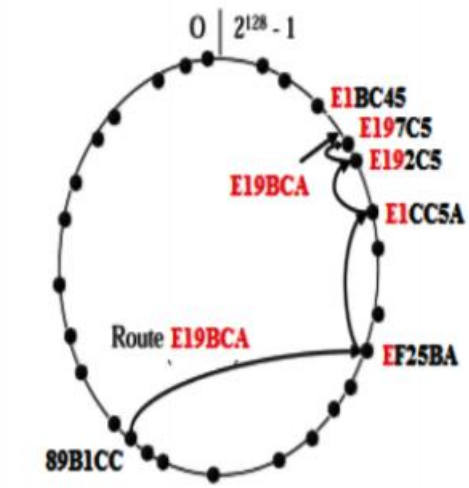

Fig 4:Routing a message from node to node in Pastry[20]

2.3.2. Arrival and departure of a node

When a new peer X arrives on the overlay, it generates its node id, then sends a message containing its key. The DHT Pastry forwards the message to the numerically nearest node id. Then each intermediate peer in this route sends a row of their routing table corresponding to the node id to X. X then establishes its routing table from this information. A peer is detected as unavailable when the other peers can no longer communicate with it, i.e. when the failing peer leaves the overlay or when a failure has been introduced. This causes the other peers to update their routing tables. There are also some applications based on Pastry:

- SCRIBE [23]: a multi-group broadcast application.
- PASTICHE [13]: a backup system which exploits the available capacities of the memories distributed in the network.

2.4. Principle of the CAN DHT

The CAN distributed hash table design described is based on a virtual cartesian coordinate space of torus space (Fig:6). This coordinate space is completely logical and has no relation to any physical coordinate system. At any point in time, the coordination space is dynamically shared among all the nodes in the system so that each node has its own distinct area in the space.

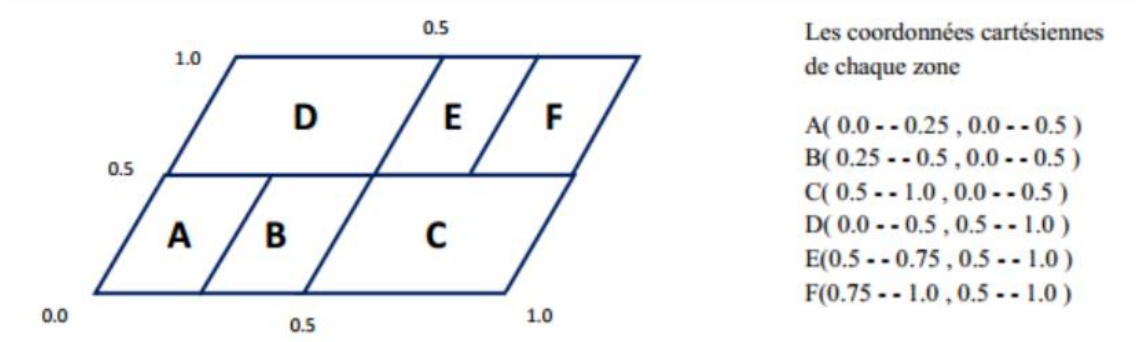

Fig 5: Space CAN on two dimensions [21]

The CAN DHT provides a multidimensional cartesian vector space where each portion of the space can be adopted by a node, and each node acquires information about its directly connected neighbors. Each identifier is converted into coordinates in CAN space. The routing in CAN is done from near to near, passing through all the neighbors until it reaches the target peer. To make a hop, the node receiving the request communicates it to the neighbor whose coordinates of that neighbor overlap on d-1 dimensions and are contiguous on the remaining dimension. At each hop, we can only change coordinates on one dimension. For example, the previous figure (Fig:7) shows a two-dimensional coordinate space of [0; 1] over [0;1] partitioned between 6 CAN nodes. This virtual coordinate space is used to store (key, value) pairs: to store a (key, Value) pair, the key is determined on a point P in the coordinate space using a uniform hash function. The corresponding (key, value) pair is then stored on the node that owns the field. To retrieve an entry that matches a key, any node can apply the same deterministic hash function to map the given key to point P and retrieve the corresponding value from point P. If point P is not owned by the requesting node or its immediate neighbors, the request must be routed through the CAN infrastructure until it reaches the node in the area. Efficient routing is therefore a critical aspect of the CAN protocol. Nodes in CAN automatically organize themselves into an overlay network that represents this virtual coordinate space. A node learns and maintains the addresses of nodes that contain coordinate areas adjacent to its own area. This set of nearest neighbors in the coordinate space serves as a coordinate routing table.

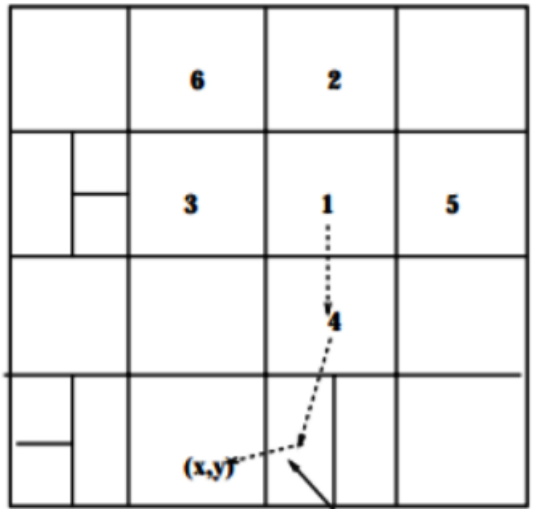

Fig 6: Routing CAN on two dimensional space [21]

In Figure (Fig:7), node 5 is a neighbor of node 1 because its coordinate area overlaps with 1 on the Y axis and follows along the X axis. On the other hand, node 6 is not a neighbor of 1 because their coordinate areas engage along the X and Y axes. Using its neighboring set of coordinates, a node transmits a message to its destination by a simple gluttonous routing gluttonous routing: a path that is constructed point by point to the neighbor with coordinates nearest to the destination coordinates. For a space divided into *n* equal areas, the average routing path length is $O(d/4)(n^{1/d})$. The hops and individual nodes maintaining the 2d neighbors. These scaling results mean that for a *d*-dimensional space, we can increase the number of nodes (and thus areas) without increasing the state of the nodes. However the average path length increases and can tend to $O(n^{1/d})$.Note that many different paths exist between two points in space, even if one or more of a node's neighbors should crash, a node can automatically route to another available path. But when a node loses all of its neighbors in a certain direction there are repair mechanisms that are triggered.

Note: the formula $(d/4)(n^{1/d})$, represents the average routing path length of CAN. The routing depends on two factors: the size of the network(*N*) and the dimension of the space(d). Thus by varying *d* from 2 to 4 and by taking *N=1000*, we have the evolution of the average number of hops on the figure (Fig :8). These graphs show that for a two (02) dimensional space the average number of hops goes from 5 per 100 nodes to about 16 per 1000 nodes. For a 3-dimensional space, the value of the average number of hops varies from 1.54 to 3.33 per 1000 nodes. This evolution tends the logarithmic order: log(N). For a 4 dimensional space, the value of the average number of hops decreases considerably. It goes from 0.8 to 1.4 per 1000 nodes. In conclusion: the more the space is multidimensional, the more the routing improves considerably.

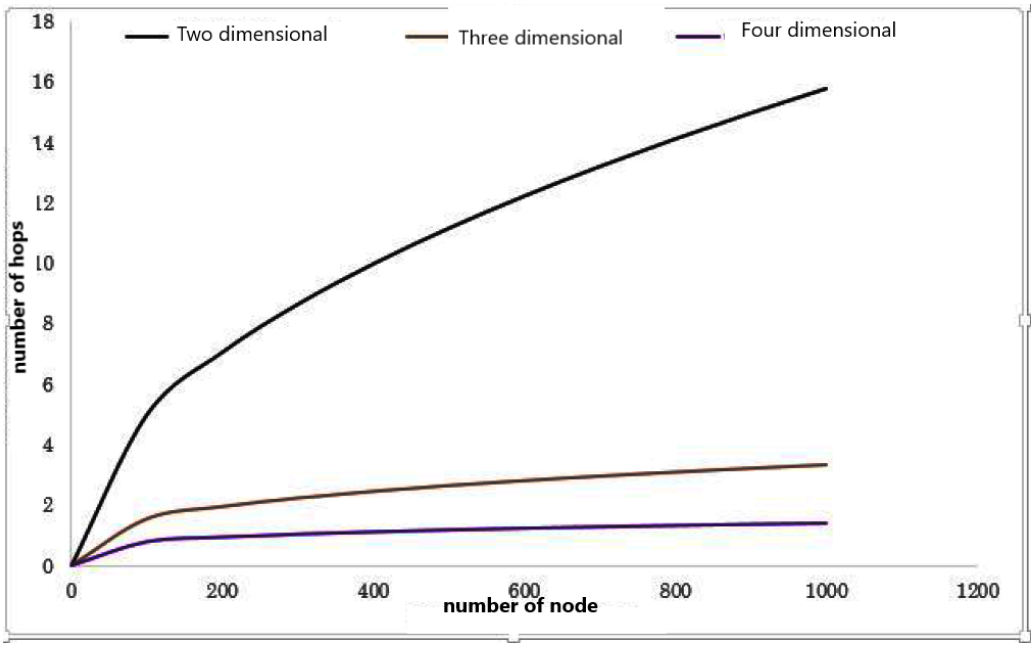

Fig 7: Performance CAN according to dimensions.

2.5. Optimization of DHTs with the De Bruijn graph

The routing of the various DHTs previously studied can be optimized by adopting the De Bruijn graph approach. De Bruijn's graph is a fixed degree directed graph with outgoing and incoming edges for each node. The degree is the total number of nodes. It has a diameter of $log_k N$ with N the number of nodes and k outgoing edges and k incoming edges. An arc is drawn between two vertices if the last *n-2* characters of the initial word correspond to the first *n-2* characters of the terminal word. The arc is labeled by the last character of the terminal word (Fig:9). The authors in [8] have proposed the construction of DHTs(CAN, Chord, Pastry) on a graph of fixed degree to obtain a small routing diameter. Indeed, with De bruijn's graph, the routing is done with half the time of the classical approach of CAN or Chord. Moreover, De Bruijn's graphs offer a diameter four times smaller than those of Chord or CAN. Thus, implementing CAN based on the De Bruijn graph would give it a better performance in terms of routing.

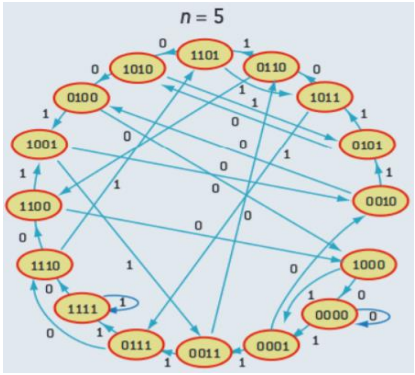

Fig 8: graph of De Bruijn [8]

2.6. Work on the Analysis of DHTs

A comparison of the performance of Chord and Kademlia DHTs in a context of high churn has been proposed by Elizabeth Perez and Miguel Lopez in [4]. This study shows that under similar conditions (experiment condition), Kademlia shows higher performance than Chord. Therefore, These authors concluded that DHTs must have mechanisms to deal with high churn throughout their existence. Otherwise they may not reach a state where peers are properly connected. In addition, the results suggest that DHTs should rely on less dynamic peers to improve their performance. An evaluation of structured peer-to-peer performance in mobile networks described in [7] evaluates the performance of five structured P2P protocols based on popular DHTs, namely: Chord, Pastry, Kademlia, Broose [7] and EpiChord [7]. This experimentation showed that Kademlia and EpiChord are good candidates for mobile networks that have an environment with a high churn rate. In addition, a study of Kademlia and Chord protocols in a peer-to-peer session initiation environment (P2P SIP) shows that Chord offers slightly better scalability in terms of messaging cost. The difference is mainly a consequence of Kademlia's higher frequency of keepalive . The second finding was that the average packet size was slightly smaller (about 14%) when using Kademlia. The results regarding routing performance revealed that Kademlia is much better compared to Chord by about 40% shorter search path length on average. The differences can be explained both by the general routing efficiency of the topologies used and by the means of updating the routing information. It must be noted that this work did not clearly establish a comparison of the CAN, Pastry, Chord and Kademlia DHTs. In this work ,we have conducted a comparative theoretical study of the DHTs Chord, Pastry, CAN, Kademlia. With a simulation, we will evaluate the performance in terms of latency, number of hops and number of transmitted messages. This study is done using the simple network layer of the peerfactsim simulator. The use of the single network layer can be considered as an ideal case because the node searches are done locally.

2.7. Existing Peer-to-Peer Simulator

There are many simulators for peer-to-peer networks. We give in this paragraph an non-exhaustive list of peer-to-peer simulators.

- P2Psim [16]: Simple to use but with limited statistics, scales to 3000 nodes maximum. It does not have a graphical display.
- PeerSim [16]: Scales up to 106 nodes in cyclic mode. The physical network is not modeled, there is little documentation on the discrete event simulation mode and there is no graphical visualization.
- QueryCycle [16] : specialized in file exchange systems with a network modeling in cyclic mode; scaling up to 20 nodes.
- OverSim [16]: scaling up to 105 nodes, physical network modeling, network visualization and statistics. Several DHTs implemented Chord, Kademlia. In case of error it blocks the simulation without displaying error messages.
- PeerfactSim [18] : the nodes of the physical network are modeled, possibility to visualize statistics. Several DHTs implemented like Chord, Kademlia, CAN, Pastry and also content distribution systems like gnutella. It scales up to more than 105 nodes, generates data files even in case of errors and consumes more memory compared to other simulators like PeerSim.

In the context of this study, we have chosen the peerfactSim simulator for various reasons that we will give in the following subsection.

2.8. The PeerfactSim simulator

The PeerfactSim simulator, based on Java, is a project launched by the Multimedia Communication Laboratory (KOM) for the simulation of large-scale peer-to-peer systems. It is often called PeerfactSim. PeerfactSim peer act is a discrete-event P2P simulator written in Java, developed by the Technical University of Darmstadt and published under the GNU General Public License. The stated goal of the project is to create an adaptive and lightweight simulation tool for efficient, accurate and realistic modeling of P2P protocols and applications.

2.9. Advantages

This simulator particularly meets our expectations in the sense that its architecture clearly separates the different levels of abstraction:

- at the level of the IP network layer, the simulator models the latency times for each message sent;
- at the overlay network layer, existing P2P protocols (CAN, Omicron, Chord, Kademlia, Pastry) are originally implemented in the simulator;
- at the user level, the user behavior part is totally detached from the corresponding physical node. Thus, we can implement a behavior model without having to modify the network parts. In addition, the spatial and temporal distribution of users is taken into account.

2.10. Drawbacks

The documentation is relatively weak and the only way to learn how the simulator works is to dive into its implementation and have some knowledge of the P2P protocols that are implemented. The basic architecture of PeerfactSim, can currently be divided in two parts: the functionality layer and the simulation layer (the PeerfactSim engine). Each layer provides well defined interfaces to express its individual functionality, operations and services provided. Specifically, the core components for each layer are identified and provide different services that can be used by other layers or components.The network layer of the simulator maintains a number of services that embody the functionality of the three lower layers of the ISO/OSI model.

3. THE CHARACTERISTICS OF THE SIMULATION LAYER

In general, all discrete event-based simulations have a common structure. We will briefly describe some of the components.

3.1. The Event Scheduler

This is the most executed component during a simulation. This method is executed before each event and can be called several times in an event to trigger other new events. The actions performed by this method are:

- the scheduling of event x at a time t;
- the cancellation of a previously scheduled event x;
- the obtaining of the simulation time;
- the determination of the final time of a simulation.

3.2. Simulation Time and Progression Mechanism

Each simulation has a global variable representing the simulation time. The scheduler is responsible for the progress of this time which will be updated automatically according to the time of the nearest event.

3.3. The Event Queue

The planning of events is done by keeping an ordered list of future events to simulate. Each simulation event contains the time at which it should occur and a reference to a simulation event handler who will be informed of the occurrence of this event.

3.4. Hosts in the simulator

It is important to understand that, during a simulation, each peer is represented by a host (Figure 10) in the current version. This host is implemented by the DefaultHost class. The host also includes an additional component called HostProperties that contains general information about the download rate,bandwidth consumed according to the requirements of the simulation. HostProperties can be extended with other properties such as CPU power, storage, RAM, etc.

3.5. Simulation With the Simple Network Model Implemented in Peerfactsim

The simple network model is intended to implement a stable and efficient network layer, without going into too much detail about the network characteristics. It is mainly applicable for debugging and testing overlays and provides a fast simulation. Other aspects include high memory efficiency and faster loading of network stacks at the beginning of the simulation. Currently, only transmissions with UDP (User Datagram Protocol) can be configured using this network layer. The use of this layer can be considered as the ideal case.

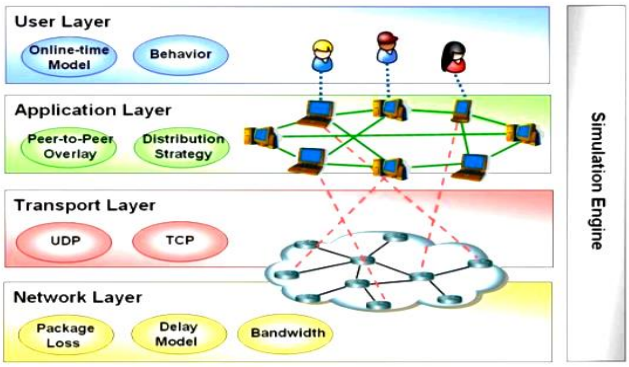

Fig 10:Architecture of peerfactSim [18]

3.6. Description of the Simple Network Model

3.6.1. The addressing

The addressing scheme applied to the simple network model is based on integers. Theaddress of a simulated peer in the network layer is assigned by the Simple Net Factory component of the peerfactSim simulator, starting with the identifier 0, and increased by 1 for each new network layer.

3.6.2. Behavior of the Peers

In the simple network model, when a peer sends a message to another peer, it first checks if the sender is online or offline. Messages from senders, who are offline, will be deleted immediately. If the sender is online, the subnet (SimpleSubnet) corresponding to the model calculates the latency for the transmission based on the time taken before the messages are received after a given time, which is determined by the latency model chosen. In order to ensure the orderly delivery of messages, different latency calculation strategies ensure that previously sent messages will always be delivered before later sent messages. When a packet is delivered, we check if the receiver is still online, otherwise the packet is dropped.

3.6.3. Size of messages

Messages are generated by the SimpleNetMessage component. The size of a SimpleNetMessage, of the simple network model, is only the size of the transported payload (the size of the data field). It does not add a header or footer to the packet.

3.6.4. Bandwidth management

Bandwidth management for the transmission of messages from one peer to another is not currently implemented in the simple network model. Peers have almost unlimited bandwidth. This means that neither the number of messages received or sent nor their size is a not problem. We are interested here in the simple latency model. In the simple latency model, the SimpleLatencyModel class assigns a random position to each peer based on a two dimensional torus surface, of size 40,000 * 40,000 km2.
The torus is a kind of cylinder curved on itself whose ends meet, forming a ring. The formula for calculating the latency for a transmitter S and a receiver R is based on Kovacevic's idea [17] as follows:

$$latency(S;R) = f:(df(n) + (dist(S;R))/v)$$

In this formula, *dist (S,R)* denotes the shortest Euclidean distance between the two peers *S* and *R* on the torus. *df(n)* is a time uniformly distributed between 0 and 31 ms, which is immutably related to each even n. The absolute term *V* represents the propagation speed of the signal, which is 100,000 km/s, and f is a random number between [0,1 :1].

In the rest of our work, we will use the network characteristics described previously, namely bandwidth capacity, message size, peer behavior and addressing to perform simulations. These simulations will be performed by switching from one DHT protocol to another using an xml file to retrieve data for analysis.

4. THE CHARACTERISTICS OF THE SIMULATION LAYER

4.1. The Configuration

Syntax and semantics of the configuration XML file In PeerfactSim, each scenario is defined by a configuration file. The configuration is represented by an XML file that can be divided into several formal parts and each part is delimited by a tag. Before we start highlighting the different parts, we first present the general XML-language schema, which is used to specify all components required for a simulation. Each element that defines a component in a scenario is a

sub-element of the configuration tag (root element). The name of the nested element is chosen according to the type of component and has an impact on the operation of the simulator. Therefore, the configuration mechanism looks for XML elements with listed names, which embody the requested functionality. The figure (Fig:11) is a capture of our xml file.

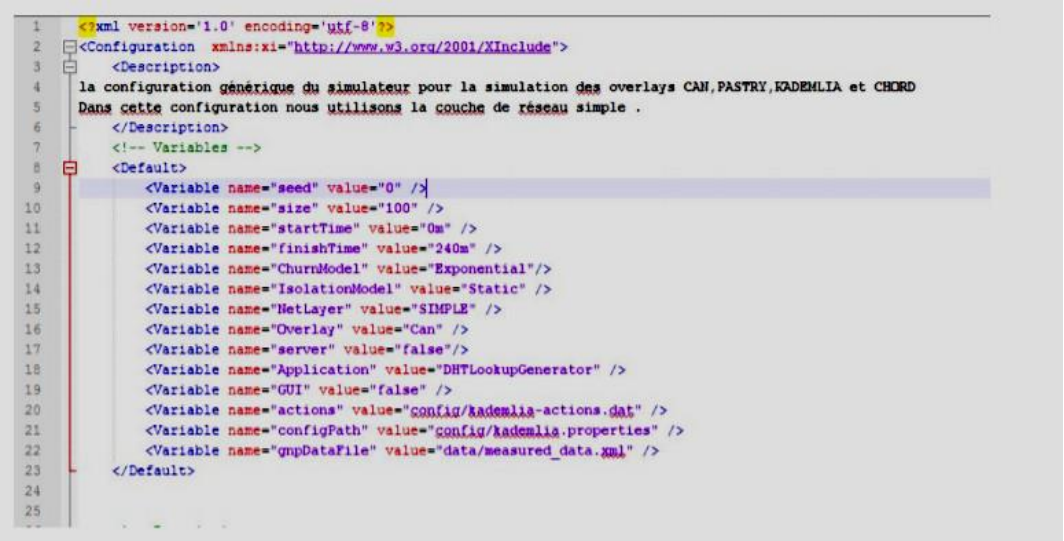

Fig 9: The variable values in the configuration file

In order to be able to switch from the execution of a protocol to another, we just changing the values specified in the ”Default” part. We have declared our components in this xml file by referring to other xml files. For example, we declared the component ”simulatorCore” in the ”Simulator.xml” file after the “Description” tag. The ”simulatorCore” component is the engine of the simulation. It is responsible for the management of all the components involved in the simulation. The seed variable (line 4 as well as line 6 of our global configuration xml file) is used to generate random numbers. If the same seed is used for several simulations, we end up finding the same random number sequences. The global simulation time is defined in the finish Time variable, here it is 240 minutes. After this period, the events are not processed anymore and the simulation is evaluated. After configuring the simulator engine and the approximate simulation times, it is necessary to define the components to be used in the simulation. An important component is the network layer. In our case, it is the simple network layer defined in the file ”SimpleNetLayer.xml”. To be able to deploy the CAN protocol on the nodes initialized by the simulator, we define the component ”CanNodeFactory” with the port 123 in the xml file CanNode.

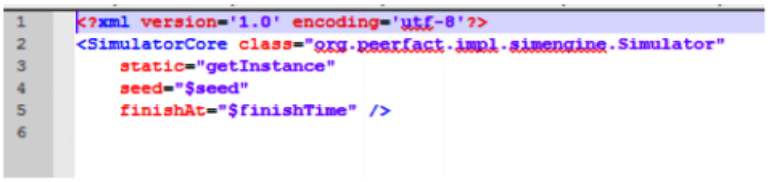

Fig 12: Specification of the simulatorCore component.

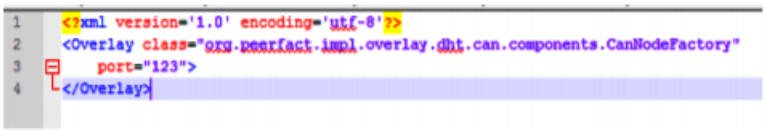

Fig 10: Specification of the CanNode component.

4.2. Process of Examining the Configuration in the XML File by the Simulator

This subsection focuses on the processing of the configuration file in the simulator and describes the components responsible for data extraction. Therefore, we first highlight the parsing of the configuration file and then explain the instantiation of the objects defined in the different parts of the configuration file. In order to obtain the information for the configuration of a simulation, the simulator needs the location as well as the name of the configuration file at startup. The path of the file is therefore specified in the IDE (e.g. the IDE in which the PeerfactSim simulator is integrated). After the provision of the necessary values when launching the simulator the class, taking into account the configuration, obtains the transmitted information. At present, the simulator only offers the class ”DefaultConfigurator” as an implementation, which is ready to run simulations and is sufficient for most scenarios. The simulation process of the peerfactSim simulator according to the following order:

the simulator is started and gets the configuration file;

- the component responsible of the configuration of the scenario is instantiated;
- the list of additional variables separated by spaces is processed and stored;
- the class representing and implementing the simulator, passes the control flow to the configurator to configure the required components.
- the configuration component parses the passed configuration file and creates an XML tree from the information in the file;
- on this tree, the configurator uses the first level to instantiate all the specified objects. Consequently, this step executes a loop, the number of iterations of which depends on the number of elements in the first level;
- Among these elements, the loop differentiates the definition of the variables, which are surrounded by the default element and between the components, representing the peers, the additional functionalities for the simulator and so on;
- in this step, the configurator creates the predefined variables and values out of the configuration file.

4.3. The Simulation Method

In this simulation phase, we will perform the operation several times taking the simulation time and the number of nodes as a factor. The idea is to find the average number of hops according to the network size and see if the time influences the routing performance. So by setting a certain number of nodes, we perform ten (10) simulations with a certain number of nodes set. For each simulation we calculate the average number of hops every 10 minutes for a duration of two (02) hours. To find the average number of hops, we perform the following operation let $x_1; x_2; x_3; \ldots x_{10}$, be the averages obtained for each simulation and Y the number of average hops.

$$y = (x1; x2; x3; \ldots; x10)/10 \quad (3)$$

5. RESULT

In this section we will analyze the results obtained from our simulation. This analysis also focuses on the average number of hops, the latency and the messages sent by each protocol. The performance parameters remain the time and the number of nodes. The use of the single network layer can be considered as an ideal case because the node searches are done locally.

5.1. Graph and Theoretical Comparison of DHT

Our analysis of the DHTs systems and other analysis works carried out in [12] and [2] allow to summarize in Figure (Fig:14), a complete analysis of their theoretical properties. For each of the topologies, we present the routing diameter and the neighborhood degree. The degree represents the size of the routing table. The diameter is the largest distance, in number of hops, between two peers. Finally, the flexibility of a DHT represents the degree of freedom of the nodes when choosing the nodes in the routing table and the next hop.

Table 1. Comparison of degrees and diameters of some P2P routing protocols [2].

DHT	**Degree**	**Diameter**
Chord	$O(log(N))$	$O(log(N))$
Kademlia	$O(k*log_2^b(N))$	$O(log_2^b(N))$
CAN	$O(d)$	$O(d)$
Pastry	$O(2^{b-1}*log_2^b(N))$	$O(log_2^b(N))$

b : is a configuration parameter typically equal to 2 (often 4);
N : the size of the network;
d : the dimension of the space;
k: (for Kademlia) the number of neighbors in each subtree of a node.

Figure (Fig:14) is a graphical representation of the performances of the CAN, Chord, Pastry and Kademlia DHTs. We have three curves instead of four. This is due to the fact that Kademlia and Pastry have theoretically the same performance in terms of routing. We see that in terms of routing CAN (3 dimensional) and Chord have a good performance compared to Pastry. Indeed, the performance of CAN depends more on the dimension of the space in which it carries out its routing. The more the space is with several dimension, the more the number of hops decreases. Thus, if in two dimensions, the routing is carried out from close to close according to two axes, in three or four dimensions, we have respectively three and four axes to reach a peer.

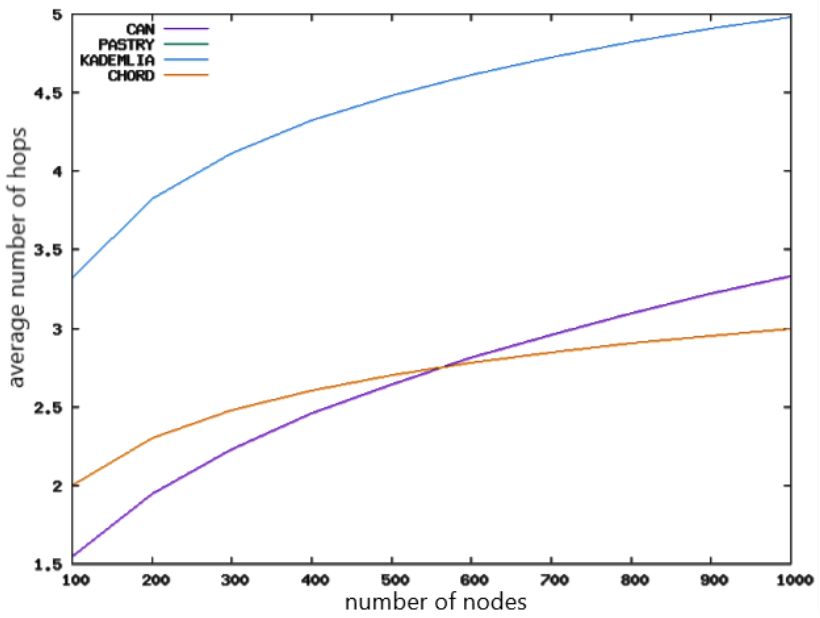

Fig 11: Theoretical comparison graph of DHTs.

5.2. Path Length Versus Number of Nodes

The figure (Fig :15) shows that the average path length increases algorithmically with the number of nodes, except for the CAN protocol. The blue curve, which represents the evolution of the path length of the CAN protocol, obtains the highest values (from 6.95 per 100 nodes to 21.5 per 1000 nodes), while the orange curve obtains the lowest values (from 2 per 100 nodes to 3.6 per 1000 nodes). Also, we have the brown curve that shows the evolution of the path length of Kademlia according of the size of the network (this value goes from 5.5 per 100 nodes to 6.4 per 1000 nodes) and the curve in Yellow which indicates that of Chord (the value goes from 4 per 100 to 5.8 per 1000 nodes). Finally to find the base of the logarithmic function of the routing order, we make a comparison with calculated values ($log_{10}(n)$, $log_2(n)$). The result is that Pastry's routing is closer to the logarithmic order with base equal to 10 ($log10(n)$) and Chord and Kademlia have a routing closer to $log_2{}^b\ (n)$. b: number of bits in the alphabet used (e.g. in hexadecimal b=4), n: number of nodes. For the CAN protocol, its routing is closer to $(d/4)\ (n^{1/d})$.

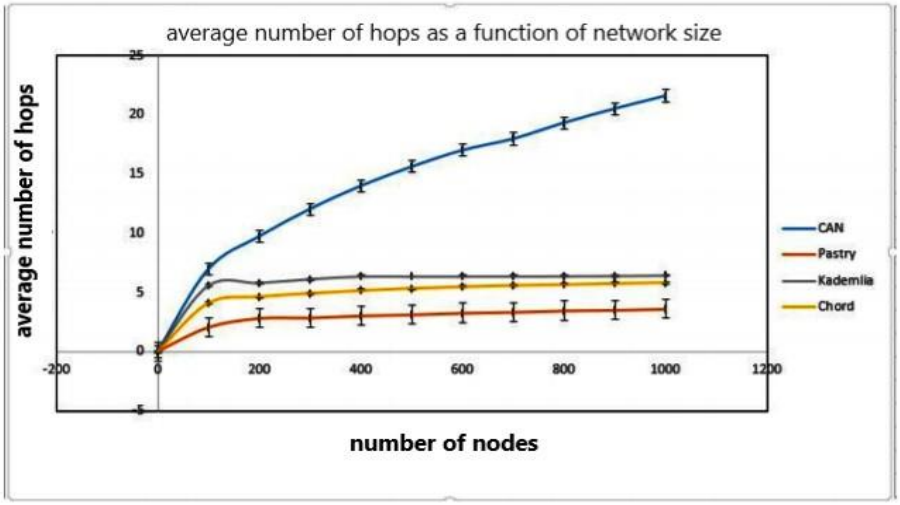

Fig 12: Evaluation of the average number of jumps according to the size of the network.

5.3. Path Length Versus Number of Nodes

variation of path length versus time Finally, to find the impact of time on the performance of the protocols in terms of path length, we performed the simulation experiment with 1000 nodes over a period of 2 hours. Figure (Fig:16) clearly shows that there is no significant influence of time on the path length, for a certain number of active nodes, whatever the protocol used. This proves the efficiency of self-organization of DHTs protocols. From the observations made on Figure (Fig:16) and Figure (Fig:12), we conclude that the best performing protocol in terms of path length, in the peerfactSim simulator is Pastry. And the Chord, Pastry, Kademlia protocols have a better scalability compared to the CAN protocol.

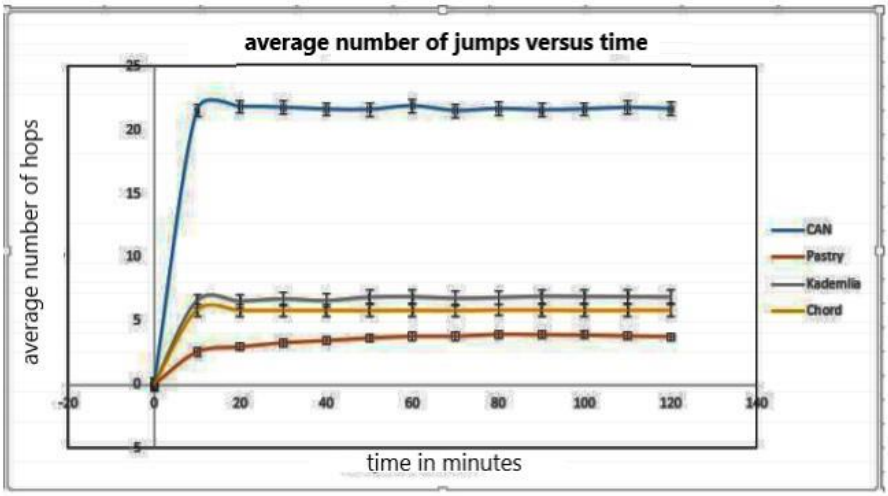

Fig 13: Evaluation of the average number of hops as a function of time.

5.4. Path Length Versus Number of Nodes

The absolute dispersion parameters indicate how much the values of a distribution deviate in general from the reference value. An absolute scatter parameter is always expressed in the unit of

measurement of the variable under consideration. The four most common absolute dispersion parameters are the range, the interquartile range, the mean absolute deviation and the standard deviation. In our case, we use the range because the averages calculated as a function of the size of the network vary slightly: 6,94, 6,89, 6,91 The range is the difference between its highest and lowest value. if we have values called X, we will have:

$$X = X_{max} - X_{min} \quad (4)$$

Thus we have: 0.5 for CAN, 0.1 for Chord, 0.8 for Pastry and 0.09 for Kademlia. These values are represented by segments on each curve in Figure (Fig:16).

5.5. Path Length Versus Number of Nodes

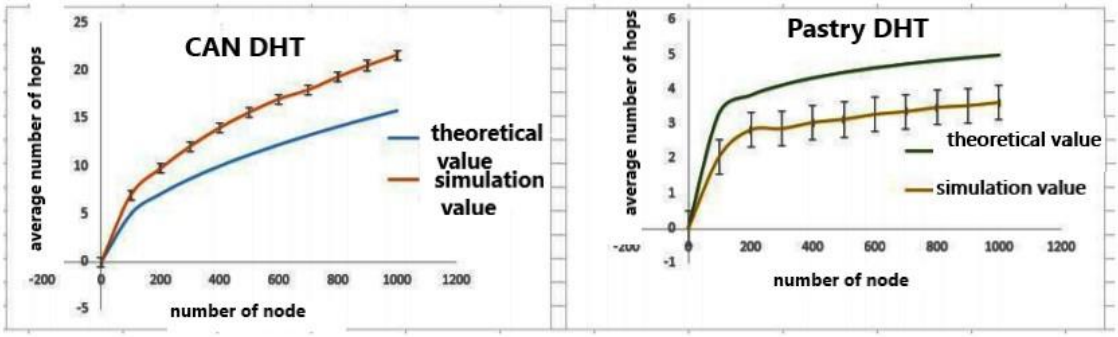

Fig 14: CAN and Pastry

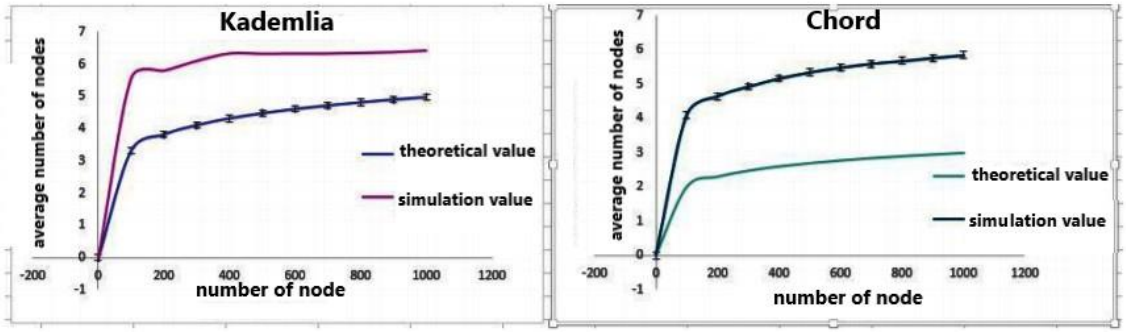

Fig 15 : Kademlia and Chord

On the figures (Fig:17) and figure(Fig:18), we have the curves drawn from the theoretical results and the results obtained after simulation. The curves with the bars are obtained from the data of the simulation. Note all the curves have the same shape and for each protocol, there is a difference between the theoretical values and the values of the simulation. CAN, Pastry and Kademlia have a logarithmic function according to the network size but with a difference in values.

5.6. Latency

Figure (Fig :19) shows the latency of operations performed during the simulation. Operations in PeerfactSim boil down to sending messages. latency is defined as the time between sending and receiving a message.

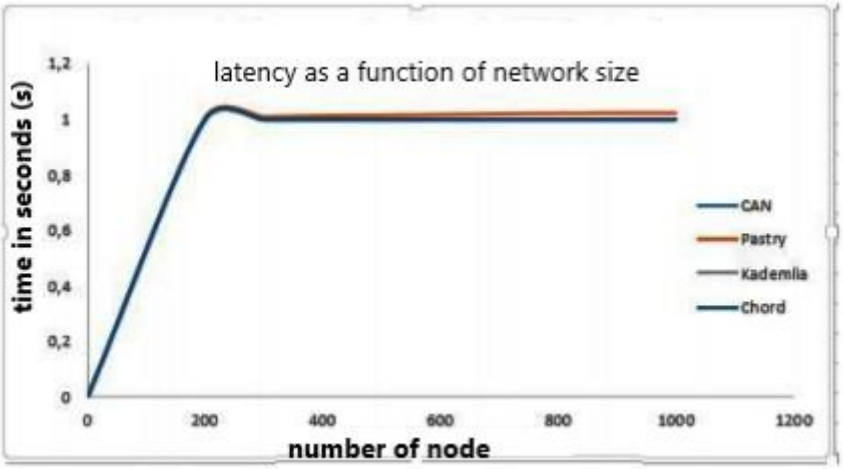

Fig 16: Latency time according to the size of the network.

The latency is approximately one second for these four protocols. Note that the physical network layer is local. So the search for keys in the identifier space and the requests sent remain local. This can optimize the latency time. We also notice that the average latency evolves weakly with the number of nodes, which confirms that these DHTs are scalable.

5.7. Average Number of Failed Messages and Number of Transmitted Messages

In this section we analyzed the initialized messages and the correctly transmitted messages to determine the average number of failed messages as a function of time.

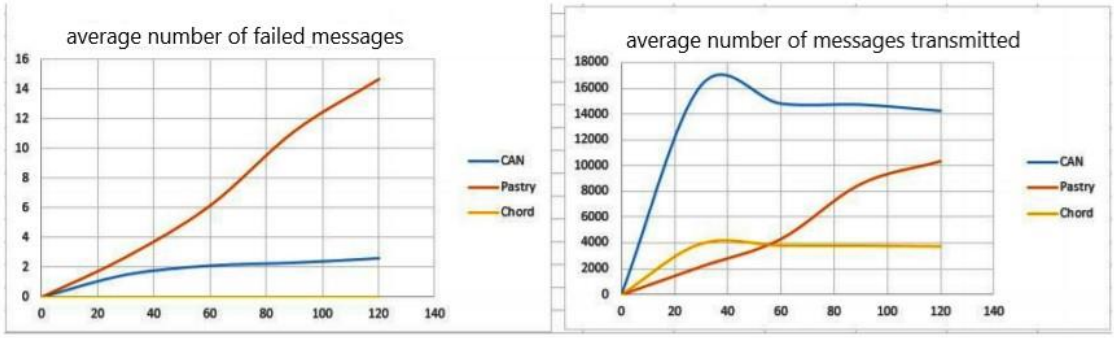

Fig 17:Number of messages transmitted versus time.

On the figure (Fig:20) we notice that the CAN protocol sends more messages. But this number decreases slightly with time and also the number of failed messages is low. The number of massages sent by Pastry increases with time and the number of failed massages increases proportionally with the number of messages sent. Chord sends less messages and without failure. Obviously the CAN protocol has a long routing time compared to the others but ensures the sending of a very large number of messages.

All previous results showed that Pastry is efficient in terms of routing and that these protocols have approximately the same latency. In terms of message delivery, DHT Chord is more efficient because it has less transmission failure. Kademlia has a middle position between Pastry and

Chord in terms of performance. CAN, on the other hand, is first in terms of the number of massages sent.

6. CONCLUSION

Peer-to-peer networks, structured or unstructured, are globally recognized as a viable solution for the implementation of various distributed applications. Existing applications and research projects indicate that structured peer-to-peer systems are very valuable technologies in a data sharing environment. Distributed hash tables provide reliable scaling while maintaining network performance compared to unstructured peer-to-peer technologies. Several comparative studies of DHTs have been done in order to classify them according to their performance but most of them do not take into account the CAN protocol. Therefore, in this work, we have highlighted CAN and its performance by comparing it to Chord, Kademlia and Pastry. The simulation not only gives us a view on their performance in terms of routing, latency and messages sent but also an overview on the theoretical and real performances. The simulation scenarios were built with the PeerfactSim simulator after a comparison between existing peer-to-peer network simulators. It was found that the Peerfactsim simulator implements all four protocols in our study and is open source. Today, with the advent of the cloud [22] and the techniques of parallel computing on the large amounts of data that are stored there, structured peer-to-peer protocols seem to be a solution to overcome the problems often encountered due to its strongly coupled architecture. In perspective, structured peer-to-peer protocols like CAN could be used to provide a fully distributed environment for storing and computing these large amounts of data. This will avoid bottlenecks and increase the availability of large-scale cloud applications.

REFERENCES

[1] Sameer ajmani, dwaine e. clarke, chuang-hue moh, and steven richman. conchord : Cooperative sdsi certi cate storage and name resolution. , pages 141154. Springer, 2002.,journal = In InternationalWorkshop on Peer-to-Peer Systems, year= 2002,.

[2] Krishna gummadi, ramakrishna gummadi, steven gribble, sylvia ratnasamy, scott shenker, and ion stoica. the impact of dht routing geometry on resilience and proximity. in proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications, pages 381394. ACM, 2003. conference on Applications, technologies, architectures, and protocols for computer communications, 2003.

[3] Jares-runser katia and cedric lauradoux. authentication planning for xor network coding in wireless sensor networks. in international symposium on network coding (net- cod2011), 2011. In International Symposium on Network Coding, 2011.

[4] Elizabeth P´erez-Cort´es Miguel Lopez-Guerrero Adan G. Medrano-Ch´avez. Peer-to-peer networking and applications. Peerto-Peer Networking and Applications, 2015.

[5] Kasyful Amron. Collaborative file sharing system using jxta p2p networking infrastructure – an application development. Journal of Enviromental Engineering and Sustainable Technology, 4:31–40, 07 2017.

[6] Ingmar Baumgart and Sebastian Mies. S/kademlia. A practica ble approach towards secure key-based routing. in parallel and distributed systems, 2007 international conference on, pages 18. IEEE, 2007., journal = International Conference , year = 2007,.

[7] F. Chowdhury and M. Kolberg. Performance evaluation of structured peer-to-peer overlays for use on mobile networks. In 2013 Sixth International Conference on Developments in eSystems Engineering, pages 57–62, 2013.

[8] IEEE Juan Casas Dmitri Loguinov, Member and IEEE Xiaom ing Wang, Student Member. Graph-theoretic analysis of struc tured peer-to-peer systems: Routing distances and fault resilience. IEEE/ACM TRANSACTIONS ON NETWORKING,, 2005.

[9] Marguerite Fayc¸al. Routage efficace pour les reseaux pair- a-pair utilisant des tables de hachage distribuees . PhD thesis, Telecom ParisTech, 2010.

[10] Bernard Cousin Houssein Wehbe, Gerard Babonneau. Optimisation de la retransmission des paquets perdus en streaming p2p. JDIR, Mar 2010, Sophia Antipolis, France., 2010.
[11] David Karger-M. Frans Kaashoek Ion Stoica, Robert Morris and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. acm sigcomm computer communication review, 31(4) :149160, 2001., journal = ACM SIGCOMM Computer Communication, year = 2001,.
[12] Steven Gribble-Sylvia Ratnasamy Scott Shenker Krishna Gum madi, Ramakrishna Gummadi. and ion stoica. the impact of dht routing geometry on resilience and proximity. in proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications, pages 381. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, 2003.
[13] Christopher D. Murray Landon P. Cox and Brian D. Noble. De partment of electrical engineering and computer science university of michigan, ann arbor, mi 48109-2122. 2007.
[14] Emmanuelle Lebhar. Emmanuelle lebhar. algorithmes de routage et modeles aleatoires pour les graphes petits mondes. phd thesis, ecole normale superieure de lyon-ens lyon. ´ Ecole normale superieure de lyon ´ , 2005.
[15] Rahmani A.M. Conti-M. Maleki, N. Mapreduce: an infrastructure review and research insights. j supercomput 75, 6934–7002. https://doi.org/10.1007/s11227-019-02907-5, 4, 07 2019.
[16] Shujaat Khan Mansoor Ebrahim and Syed Sheraz Ul Hasan Mohani. Peer-to-peer network simulators: an analytical review. https://www.researchhub.com/, 2014.
[17] Sebastian Kaune Patrick Mukherjee Nicolas Liebau Aleksan dra Kovacevic and Ralf Steinmetz. Benchmarking platform for peer-to-peer systems. it - information technology (methods and applications of informatics and information technology), 49(5) :312319., Sep 2007., journal = Informatics and Information Technology, year = 2007,.
[18] peerfact. Peerfactsim.kom : A simulator for large-scale peer-to peer networks. http ://www.peerfactsim.com. 2007.
[19] David R. Karger‡ M. Frans Kaashoek‡ Frank Dabek‡ Hari Bal akrishnan‡ Robert Morris‡, David Liben-Nowell‡. A scal able peer-to-peer lookup protocol for internet applications. https://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf.
[20] Antony Rowstron and Peter Druschel. Pastry : Scalable, decen tralized object location, and routing for large-scale peer-to-peer systems, pages 329350. Springer, 2001., journal = In IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, year = 2001,.
[21] Mark Handley Richard Karp Sylvia Ratnasamy, Paul Francis and Scott Shenker. A scalable content-addressable network, volume 31. acm, 2001. 2001.
[22] C. Wu T. Dillon and E. Chang. Peer-to-peer networking and appli cations. 2010 24th IEEE International Conference on Advanced Information Networking and Applications, Perth, WA, Australia,2010.
[23] Yan Wang. An advanced p2p multicast algorithm based on scribe. Advanced Materials Research, 2013

Authors

Laciné Kabre Currently I am a PhD student. I have a master's degree from Joseph Ki-Zerbo University in information systems and network. My research interests include Big data, Software Defined Networking, Distributed Hash Tables.

Telesphore Tiendrebeogo PhD and overlay network and assistant professor at Nazi Boni University. I have a master's degree in multimedia and real time system. My current research is on big data and image watermarking.