Master's Thesis

Autonomous Storage Distribution in a Surveillance System

Jeen Gullstrand
Malin Hägg

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, September 2014.

Master's Thesis

Autonomous Storage Distribution in a Surveillance System

By

Jeen Gullstrand and Malin Hägg

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Abstract

This report describes a solution to manage distributed resources (audio, video, metadata etc.) in a camera surveillance system. Based on the problem definition, a set of requirements was defined. These will lead the search for a network solution to handle the storage and communication autonomously by the participating nodes. Different network solutions were investigated and compared to the requirements, and one was selected to further study. To meet the requirements, the protocol was improved by various overlays. Finally, a test to confirm the fault tolerance of the system was done in a virtual environment.

Acknowledgments

We would like to thank our supervisor associate professor Mats Cedervall from Faculty of Engineering (LTH), Lund University, for guiding us on how to plan and finish a larger project perfectly, and helping us when we had difficulties. We would also like to thank our supervisor Mikael Ranbro from Axis Communications AB for his support and help. He introduced us to the company and its products, and gave us background information to the area.

We are grateful to Fredrik Hertzberg who assigned such a great and challenging project to us, and Ulf Ahlfors who gave us many valuable tips that put us in the right direction from the beginning. Last but not least, we want to thank all the colleagues from Core Technologies Analytics & Systems team at Axis.

Jeen Gullstrand & Malin Hägg

Table of Contents

CHAPTER 1

1 Introduction

Axis Communications AB is an IT company that specializes in developing and producing network cameras and network video surveillance solutions. The usages and demands of the surveillance cameras are growing rapidly year by year. There were already 5.9 million closed-circuit television cameras in England in 2011. IP cameras are widely used in schools, shopping malls, and museums [1]. In the studied network solution, the cameras are storing resources in one or several central servers. Any failures in the servers may prevent the user to retrieve the resources. Hence, a smarter solution is needed.

The design of our thesis is based on a decentralized network, where resources are divided and distributed over the network. We need to find a solution to store and efficiently retrieve resources in a secure environment.

1.1 Outline

The thesis will begin with an introduction to the area, and a problem description, followed by a set of requirements. Then the approach will be described, together with all necessary theory. We will then continue to describe our system choices and how we designed the solution by following the requirements. The testing of the system and the result of this will conclude our project, together with a discussion of the result.

CHAPTER 2

2 Description

2.1 Problem Definition

The surveillance system studied consists of a network of nodes (IP cameras, servers, and clients). Every node has a unique id, the NodeID, to communicate over the network. Resources in the network are audio, video, and metadata files. These could be stored in different locations, as described in *Figure 1*.

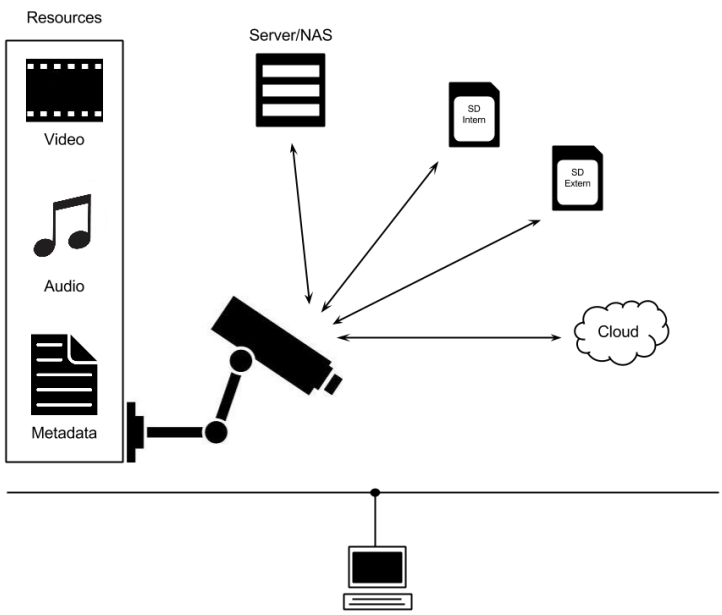


Figure 1. The resources could be distributed in many locations.

In the existing system, the resources are stored either locally on a camera and/or centrally on a server or NAS. If a node leaves the system or if the server connection is lost, some information will be unreachable. A possible scenario could be when a camera is located on a moving vehicle, for example on a bus, and the network signal is lost. A camera that usually stores resources on a server can now not reach the network, resulting in resources instead to be stored locally on the camera. The video will now be stored in different locations, causing video fragmentation. The fragments of the video need to be found and assembled in an efficient way.

The goal is to find a way to keep track of the distributed resources in the network, without the need of a central server. The proposed solution should include efficient search methods for resources in the network, and efficient retrieval of these when needed. All resources in the network should be available, and should be searchable in an efficient way. Since data could be fragmented, it should be possible to assemble the data throughout the network. The event of communication failure and/or data loss should be considered. The choice of method should meet the security aspects, as well as the synchronization requirements. Other aspects to consider are lifetime management and resource conflict resolution.

2.2 Use Cases

1. A user should be able to stream video from any chosen camera.
2. A user should be able to search for data (audio, video, metadata) in the network efficiently ($\sim O(log(N))$ or better).
3. A user should be able to retrieve data (audio, video, metadata) in the network efficiently ($\sim O(log(N))$ or better).
4. A user should be able to search for data without any knowledge about the underlying structure.
5. A user should be able to find data by knowing either the camera's name or location, and the desired time frame.
6. A camera should be able to record video and save information to another node. A user searching for the data can then connect to either of the nodes, to acquire the data.
7. A user should be able to find data even if a node crashes.
8. Two users should not be able to change the same file simultaneously.

2.3 System requirements

The following requirements of the system are specified based on the use-cases and on the background and goal of the thesis.

1. Storage management: Should be possible to handle all resources (audio, video and metadata) in the system.
2. Availability: Resources should be available.
3. Autonomous: The nodes should handle the communication and network organization themselves, without the need of a central communication point.
4. Efficient search of distributed resources.
5. Efficient retrieval of distributed resources.
6. Fragmentation: Fragmented resources should be possible to assemble.
7. Scalability: The solution should work for both small and large networks.
8. Load balancing: Preventing bottlenecks in the system
9. Reliability & Fault resiliency: The system should be able to handle node crashes and communications failures.
10. Lifetime management: Files should be removed after a beforehand specified time.
11. Resource conflict resolution: Handle simultaneous user access to files.
12. Security: Prevention of attacks and security flaws in the system.

CHAPTER 3

3 Approach

The project started with an investigation of an existing system, and research in the area of data storage and network solutions. The studied surveillance system consists of IP cameras, servers, and clients connected in a network. These could communicate over LAN, WAN or over the Internet, and videos can be streamed directly from a browser. To manage the recordings, a Video Management Software (VMS) is used. Depending on the demands of the user and on the size of the network, different VMS:s could be installed. Axis has three VMS solutions; AXIS Camera Companion (ACC), AXIS Camera Station (ACS), and Hosted video (cloud services). ACC is designed for small networks of 10 cameras or less, and stores resources either locally on the cameras SD cards or centrally on a NAS. ACS stores on an on-site server, and is made for networks with 50 cameras or less, and the hosted video solution is based on cloud storage. [2]

The local storage solution on a camera's SD card, also known as the edge storage solution, works as a backup system if the normal storage solution is not available. It can record temporarily, and when the storage on server is available again it can switch back. The videos can later be retrieved and merged together. This is the normal procedure when cameras are remotely located, or intermittently connected to a network. [3]

Since resources could be spread out in several locations depending on the network conditions and the storage solution chosen, we will assume henceforth that the data is arbitrarily distributed on the network. This is not always the case, but our solution should also be able to handle the worst case scenario. In our approach, we chose to divide the project into three sections; storage solution, resource management and security. However, this is not a strict division and the different sections will overlap over time. We begin by investigating existing network structures, and compare these to the requirements on the intended system.

3.1 Cloud

The first possible solution we investigated is cloud computing. Many companies offer different cloud services for various needs (see *Figure 2*). In the Infrastructure as a Service (IaaS) model, service providers, e.g. Amazon Elastic Compute Cloud, offer physical or virtual machines for server, storage, and networks solutions. In the Platform as a Service (PaaS) model, users can develop and run their own software on the existing platform. The operation system, database, and server are handled by a service provider, e.g., Google App Engine. The Software as a Service (SaaS) model delivers services that include management of the computer infrastructure and platform. Users can simply install applications on their own computers and get access to the service.

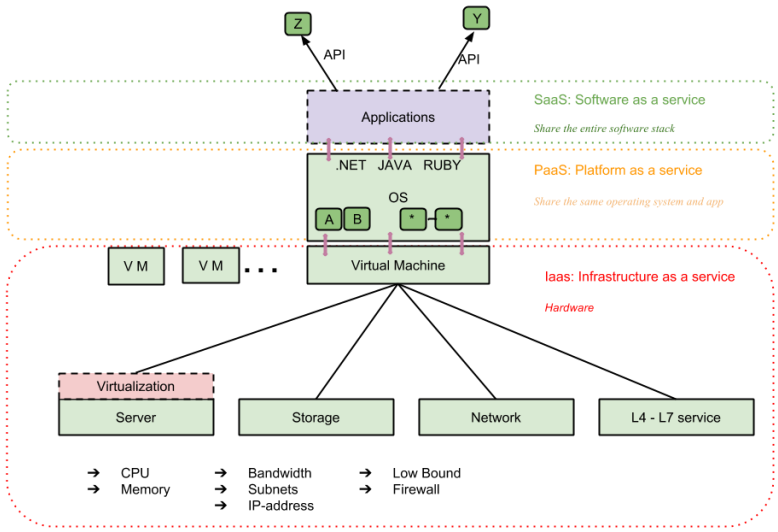


Figure 2. Cloud services on different layers

Cloud computing can handle all resources in the system. The customer can get access to the service anytime and everywhere as long as there is an internet connection. On the other hand, this is also a limitation. If the internet connection is down, we will lose connection to the previous saved resources, and live streaming will be unavailable. Hence, this will have the same limitation as in the server solution. Cloud computing is very scalable, and supports on-demand services. Resources are available when customers need it, and customers always pay for the capacity that they are actually using. In other words, customers get “unlimited” storage. Compared to the

system requirement, cloud is still a very suitable solution. Another advantage is that cloud computing replicates the data to reduce the possibility of data loss. However, there are security considerations, for example if we can trust a third party. There could be sensitive information that is stored in the system; therefore privacy is a very important issue. There is a central communication point on the service provider, but not at the customer's location. If the service provider's central communication point is under attack, the software and all the data can be lost. In our case, cloud computing can be used as a backup solution. [4]

3.2 Peer-to-peer (P2P) networks

In a peer-to-peer (P2P) system, the participating nodes are responsible for all network operations. Instead of a central communication point, the nodes acts as both server and client, communicating directly to each other. This increases the performance on the network. All nodes have knowledge of a subset of other nodes, connected to the network over logical links. This creates a P2P network topology, built on top of the physical layer. Two main typology types exist; unstructured and structured. Structured networks have, in contrast to unstructured networks, an underlying topology to organize the nodes. [5]

3.3 Ad-Hoc

The ad-hoc network is based on P2P communication using WiFi technology without an access point. In another word, all devices can communicate directly with each other. It does not need any infrastructure therefore the network can be deployed simply. One device can quickly join the ad-hoc network which is located in its range with the condition that the device fulfills the security requirements. Nodes in the network work as routers and terminals. Sometimes it is impossible to connect to the destination directly; therefore traversing multiple links to reach the destination is needed, see *Figure 3*. If device A wants to communicate with device B and the channel between A and B is not available, then device A should go through device C to B, that is A→C→B. There are many routing protocols for the ad-hoc network and all of them rely on the forwarding paradigm, e.g. if a node receives a packet that is not belonging to it, it will forward the packet according to the routing protocol. [6]

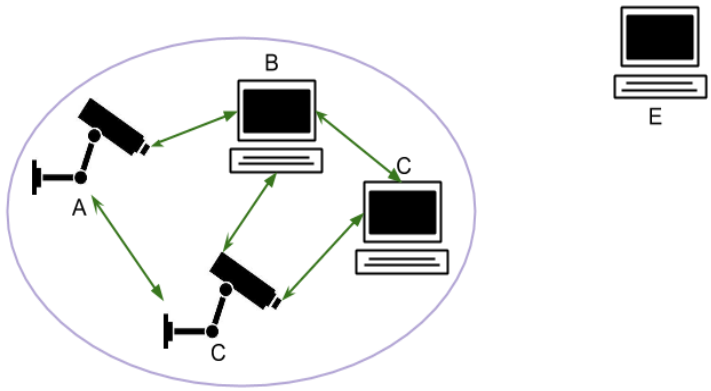


Figure 3. A simple Ad-Hoc network.

One variant of ad-hoc network is Mobile ad-hoc network (MANET). MANET is a self-forming and self-healing network. All nodes in the network are allowed to move around freely inside the network range. Thus, MANET needs an efficient routing algorithm, since the topology changes every moment. [7]

Nodes in an ad-hoc network are autonomous and this solution works for both small and large networks. However, the bit rate is limited. In a surveillance system which handles large amount of data, this is a major drawback. Also, due to that devices can join and leave an ad-hoc network relatively easy, the network is vulnerable if an attacker is inside the network range. Possible attacks are for example Denial of Service, signaling attacks, and flooding attacks.

3.4 WirelessHART

WirelessHART is a sensor network that is a wireless version of the "Highway Addressable Remote Transducer" (HART) protocol. It is a mesh architecture, where the nodes are self-organizing and act as routers for other nodes. It is easy to add and remove nodes, and messages can be forwarded as long as the devices are within range of other nodes. Often several different routing paths exist, and if one path fails another one can be chosen instead. [8]

This fits well to our requirements, since it is an autonomous architecture with self-healing properties. But it also has some drawbacks. WirelessHART operates on the 2.4 GHz radio band, the same as WLAN and Bluetooth. This could sometimes cause interference problems, resulting in increased packet loss if not configured properly. It also uses Direct Sequence Spread Spectrum (DSSS) and Offset-Quadrature Phase Shift Keying (OQPSK) modulation techniques, allowing a bit rate of 250 kbit/s. This is a very low bit rate, considering video streaming. WirelessHART is also vulnerable to denial-of-service (DoS) attacks. [9]

Hence, WirelessHART does not suit the requirements defined of our system. We therefore chose to look for other solutions.

3.5 DHT

A Distributed Hash Table (DHT) is a structured overlay network which provides efficient lookup services for participating nodes. In contrary to an unstructured network, the DHT structures the nodes in an organized topology where each member receives a unique NodeID. A structured solution has good scalability. No central server is needed, since all the nodes handle the network organization autonomously (self-organizing). Every node in the network is responsible for certain keys, which maps to values. When a file is requested, the lookup service maps the file to a node by a key. A key is a hash value of information about the file; the name, location, description and/or other data to identify the file. If the key is not present within the node itself, the request is forwarded to a node closer to the destination. If a node on the path fails, the lookup should still be possible using a different route or another node where the data is replicated.

DHT:s provide efficient searching, and has high scalability. It also provides a natural load balance when using a fair hash function. [10]

A DHT design can be divided into two general parts; routing-level and system-level design. On the routing-level, the routing behavior of the system is defined. Neighbors and routes are selected to form a routing table, based on latency (proximity) or other criteria such as hop count or geographical location. The flexibility in this selection is based on the underlying structure and the routing algorithm. All other higher design decisions are defined on system-level. [11]

Various routing algorithms can be chosen when creating a DHT-system. Each with a routing behaviour that often could be interpreted geometrically. Rings, tree-like structures and hypercubes are all examples of geometries used in DHT-systems. Hybrids of these structures are also common. In a ring, the nodes are distributed in a one-dimensional circular network. Chord is an example applying this structure. In a tree-based structure, each node represents a leaf in a binary tree. The distance between two nodes is the same as the number of bits where the identifiers differ, and each hop will correct one bit at a time. In a hypercube, every node "owns" a certain zone in the network. The distance between each node is similar to the tree, but each bit in the identifier will be corrected in any order. Depending on the choice of underlying geometry, the flexibility and routing performance will differ. [11]

In the following section, different DHT protocols will be discussed in more detail.

3.5.1 Chord

Chord is a protocol for a DHT network. It was developed by MIT and presented in 2001. Chord uses a ring structure which makes it easy to set up, and the search algorithm is efficient. Like other DHT protocols, Chord has no central node, and the system work load is distributed among the nodes. The latency in a chord network is scalable; lookup cost is always log of the number of nodes. In an N-node system, the lookup cost is in average $O(log(N))$. Chord uses consistent hashing, with the base hash function SHA-1. Consistent hashing ensures that each node only handles a limited part of the network, and provides load balance to the network using a fair

hash function. Keys and values are stored in their associated nodes, and are distributed over the network. A key's identifier is the hash value of the key itself. Chord has flexible naming, so the name of the key can include much information like location name, date, company name etc. A nodes identifier is the hash value of the IP-address. All nodes and keys are placed in an identifier circle. SHA-1 gives a 160-bit identifier space which allows the network to have 2^{160} nodes. Every node only has knowledge about a limited part of the network, so the request will be forwarded until it reaches the certain node. The routing table is called finger table and the size of the finger table is defined by the user. Each node has one or several predecessors and successors. The predecessor is the previous node and the successor is the immediately succeeding node in the identifier circle. In a 3-node system, node_1, node_2 and node_3 are placed clockwise. Node_1 is the predecessor of node_2, the node_3 is the successor of node_2, node_3 is the predecessor of node_1 etc. Successors in Chord are also sequential neighbors, because of the underlying unidirectional structure. Sequential neighbors secure that the message is sent to a node that is nearer to the destination. This increases the static resilience of the system, but it can lead to higher latency due to longer path ways. Thus, it is up to the user to decide what is more important. Is it more important to have more sequential neighbors to reach a higher resilience to path failures, or to have less sequential neighbors to get lower latency? [12]

3.5.2 Kademlia

Kademlia is a DHT protocol based on XOR (bitwise exclusive or) geometry. It uses the XOR metric to calculate the distance between keys, making it a symmetric communication between the nodes.

$$d(A, B) = A \oplus B = B \oplus A \quad (1)$$

The distance from node A to node B will then be the same as from B to A. Kademlia can then choose the best path by selecting the preferable route based on e.g. latency. Every node in the network keeps a list (called k-bucket) with information about nodes of distance between 2^i and 2^{i+1} from itself. It is sorted by recency, with the most recent contacted nodes at the end. The k-bucket will be updated every time a new message is received. One property of the k-bucket is that the nodes on the top have been alive the longest, and will have higher probability to remain alive. The list is also

limited in size and only inserts new nodes when old ones leave the system, preventing malicious attackers from flooding the system with new nodes (making it resistant to certain DoS-attacks).

Kademlia uses four commands; FIND_NODE, FIND_VALUE, PING and STORE. FIND_NODE is used to find the location of the node responsible for a certain NodeID. FIND_VALUE behaves like FIND_NODE, but returns the value if this is stored on the location. PING is used to verify that a node is still alive, and STORE saves a *<key, value>* pair on a node. Each command includes a random value to connect a response to a request.

To find a node, Kademlia sends parallel and asynchronous FIND_NODE requests to selected nodes in the k-bucket. It continues to send requests to known nodes until a specific number of nodes closest to the destination have responded. Kademlia can then choose a route among these to forward a request to. To keep the k-bucket updated, node searches are sent periodically to idle nodes. [13]

3.5.3 CAN

A Content-Addressable Network (CAN) is a DHT protocol with d-dimensional coordinate space, similar to a hypercube, where each node is responsible for "zone" in the network. A zone is a part of the network containing a certain range of keys which are assigned to the node. Every node holds information about the neighboring zones, and messages are routed zone by zone until the destination node is encountered (see *Figure 4*).

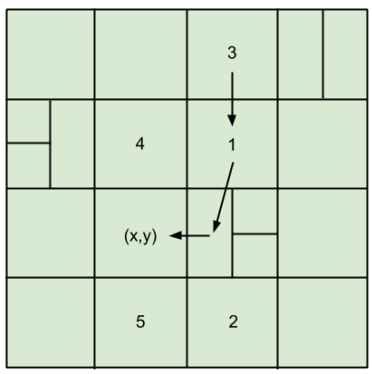


Figure 4. A simple CAN network with d = 2.

The distance between two nodes is the same as the number of bits by which their NodeID differ. If a node with NodeID 1205 sends a request to NodeID 1306, the distance between these would be 2. Each hop on the routing path corrects one bit in the identifier, until it matches the destination. This could be done in any order, making it possible to find another path even if some nodes on the way fail. A CAN thus has high flexibility in route selection. The average routing path length of a CAN is $O((\frac{d}{4}) \cdot N^{\frac{1}{d}})$, where d is the dimension and N is the number of zones. Every node maintains $2d$ neighbors, which means that the number of nodes can grow without increasing per node state.

As seen in *Figure 4*, some zones are divided into smaller sections. When a new node joins, an existing zone will be divided and assigned to this new node. The neighbors of this zone will then be notified of the network change. If a node instead leaves, its zone will either be merged to create a valid single zone or handed over to the smallest neighbor. To keep information on the current network state up to date, every node periodically sends update messages to their neighbors. [14][15]

3.5.4 Pastry

Pastry is one of the DHT protocols that are based on a prefix-based tree/ring topology. Its network structure is a combination of tree and ring, also called a hybrid. Pastry uses SHA-1 as its hash algorithm. Each node has a NodeID which consists of 128 bits. Thus, the network can contain the maximum of 2^{128} nodes. Similar to other DHT protocols, <*key, value*> pairs are stored in the nodes. Each node maintains a tree structured routing table, called a neighbor map, which leads a request to a destination digit by digit. Nodes forward the request to the node that has 1 or b bits more common prefix than the current node. A routing example is shown in *Figure 5*, where node_1234 sends a request to node_2443.

Figure 5. A routing example in Pastry.

The routing table size has $O(log\ _{2b}N)$ rows and $2^b - 1$ columns, where N is the number of nodes that is contained in the system. Leaf set stores NodeID and IP-address that is b nodes above and b nodes below. To reach the destination node, one need to route $O(log(N))$ hops in average.

Pastry has the following Application Programming Interfaces (API):

- *NodeID = pastryInit(Credentials, Application):* Used to join or start a network.
- *Route(msg,key):* Routes the message to the given key.
- *Deliver(msg, key):* Called when a message has arrived to the node responsible for the key.
- *Forward(msg, key, nextId)*: Called before the message is forwarded to the next node.
- *NewLeafs(leafSet):* Changes the node's leaf set. [16]

3.5.5 Comparison

After the investigation of existing network structures, some comparison and conclusions needed to be made. Which advantages and disadvantages are there in using a certain structure, and how does it fit with the defined requirements of our intended system? Is there an existing structure we can use, or take inspiration from, when creating our own?

First of all, we looked at a cloud based solution. This is a very convenient solution with almost unlimited storage capacity, where it is possible to choose the service level needed. However, the user will pay for each amount of storage used, and will have to trust in a third party for keeping

the resources secure. It could still be a suitable solution, but because of our security requirements we chose to look for other possibilities.

Next we investigated different peer-to-peer networks. These exist in two categories, unstructured (ad-hoc, WirelessHART, MANET) and structured (DHT). Every node act as both client and server for the other nodes, and all communication are handled by the nodes autonomously. It is a flexible and decentralized solution that fits well with our requirements. The structured solution also offers scalability, higher bit rates, load balance, and efficient lookup; which is exactly what we are looking for. The key benefits of using a DHT-system in our storage solution are:

- Autonomous and decentralized system
- Possible to manage audio, video and metadata
- Efficient search
- Efficient retrieval
- Fault tolerant and reliable
- Natural load-balance (keys are evenly distributed assuming a fair hash function)
- High scalability

Because of this, our next step was to take a deeper look at the DHT solutions. The different protocols, using various algorithms and geometries, were compared to find the most preferable one. *Table 1* shows the result from the comparison, where N is the number of nodes/zones, d is the number of dimensions, and c is a small constant.

TABLE 1: COMPARISON OF THE DIFFERENT DHT PROTOCOLS. [10][17]

	Chord	**Kademlia**	**CAN**	**Pastry**
Architecture	Unidirectional circular network	Network with XOR distance calculation	One or several d-dimensional coordinate space(s)	Plaxton-style global mesh network
Geometries	Ring	XOR	Hypercube	Hybrid = Tree + Ring
Flexibility	FNS+FRS	FNS+FRS (but with different path lengths)	FRS	FNS+ FRS (subtle)
Natural support for sequential neighbours	Yes	No	No	Default routing: No Fallback routing: Yes
Routing performance	$O(log(N))$	$O(log_B(N)) + c$	$O((\frac{d}{4}) \cdot N^{(\frac{1}{d})})$	$O(log_B(N))$
Routing state	$log(N)$	$Blog_B(N) + B$	$2d$	$Blog_B(N) + Blog_B(N)$
Lookup protocol	Retrieves the value of key directly or a node closer to the destination. Matches key with NodeID.	Retrieves the value of key directly or a node closer to the destination. Matches key with NodeID.	A {key, value} pair matching a point in the coordinate space using uniform hashing.	Matches key and prefix in NodeID with a node.
Fault tolerance	Failure in nodes will not cause network wide failure. Data replication consecutive among nodes provides redundancy. Application retries on failures.	Failure in nodes will not cause network wide failure. Data replication on multiple nodes provides redundancy. Application retries on failures.	Failure in nodes will not cause network wide failure. Data replication and responsibility among nodes provides redundancy. Application retries on failures.	Failure in nodes will not cause network wide failure. Data replication on multiple nodes provides redundancy. Keeps track of different routes to each peer.
Static resilience	Best	Good	Better	Good

The table shows the comparison of the different geometries and algorithms, but also the flexibility, performance and fault tolerance of the protocols. Flexibility is important, since it impacts the static resilience (i.e. the ability to route around failures) and latency of the system. A system could have flexibility in route selection (FRS) and in neighbor selection (FNS), depending on the average number of next-hop choices and the number of neighbor choices. A high FRS gives a better static resiliency, but a high FNS is more important for the latency. According to the table, the ring geometry has the best flexibility in both aspects. [17]

Chord and the leaf set in Pastry naturally support sequential neighbors. This guarantees that a hop on a routing path makes progress towards all destinations. The ability to have sequential neighbors plays a crucial role in recovery algorithms, and it improves proximity and resilience. A higher number of sequential neighbors increases the ability to handle path failures, but at the cost of increased path stretch. [17]

When looking at the routing performance, every DHT protocol studied is capable of an average path length of $O(log(N))$. This is under the precondition that $d = log(N)$ in CAN. [18]

All protocols are resilient to failures, and a node crash in the network will not cause network wide failure. Data replication is supported, adding redundancy to the system. When comparing the static resilience of the different protocols in [17], the ring network Chord showed the highest performance.

CHAPTER 4

4 Structure and system design

Based on the requirements, Table 1 and the comparison of the different protocols, the DHT Chord protocol was chosen for further investigation since it suits the requirements well. It is a simple structure with very high flexibility that is easy to build on top of. It has natural support for sequential neighbors, good routing performance, is scalable and has the best static resilience of the studied protocols. [17]

4.1 Resource management

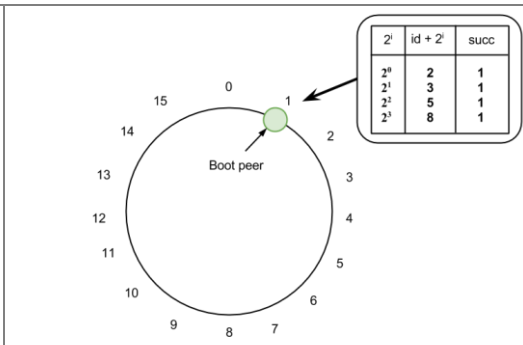
According to the requirements, the system should be able to handle all resources; audio, video, and metadata. This should be done in an autonomous and decentralized manner. Chord provides this, with <*key*,*value*> pairs distributed among the self-organizing nodes. When used with a fair hash function which evenly distributes the keys, this will also provide a natural load balance. We choose to use SHA-256 hashing instead of SHA-1 that was originally used, since it provides a fair distribution of the resources and better collision resistance [26]. A key is allocated by hashing the value of the key itself, and is placed in the identifier circle. Since Chord has flexible naming, the key could be set to anything. In our solution, a reasonable key would be the camera name or location, and the time of which the video was recorded.

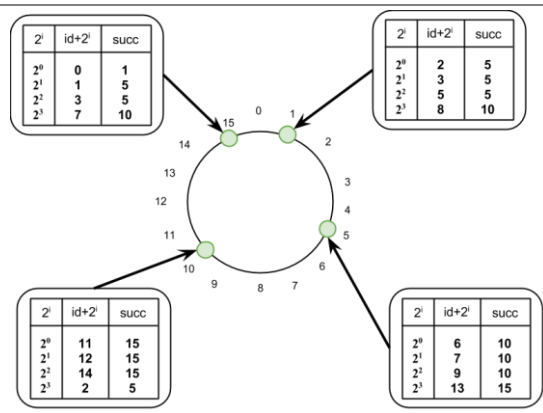
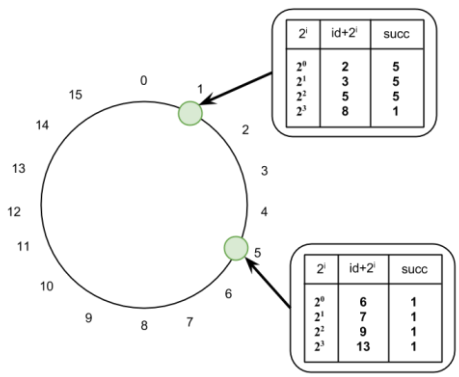
4.2 Consistent hashing

Chord is based on consistent hashing, which makes it highly scalable. The consistent hashing utilizes a hash function that assigns keys and nodes. By using a good hash function, the keys will be evenly spread among the nodes. This also gives the network a natural load balancing. When N:th nodes join or leave the network, only $O(1/N)$ part of the keys need to be replaced. Each node only has knowledge of a limited part of the network, making node changes more manageable. Due to only the successors and predecessors of the altered node state is affected, only these will have to be notified of the change. This makes it possible for the protocol to be used in both small and large networks, with thousands of nodes. The ability to handle node changes also benefits the fault tolerance of the system, since a failure in a node will not cause network wide failure.

4.3 Booting

To start up a Chord system the following procedure is done:

The boot peer gets a NodeID and the network is started.	

1. The joining peer queries a JOIN request to the boot peer 2. The boot peer sends a NodeID to the joining peer, in this case, the joining peer's NodeID is 5 3. The joining peer joins the identifier circle 4. The joining peer sends a COPY ROUTING TABLE request to the boot peer 5. The boot peer sends its finger table to the joining peer and updates its finger table 6. The joining peer creates a finger table	
Since nodes have the same functionalities, node_10 and node_15 could query their requests to an arbitrary existing node. After that, nodes will finish joining process by following previous steps.	

[12]

4.4 Searching

Searching in a Chord network is efficient, with the lookup cost of $O(log(N))$. A request returns either the key value immediately, or a NodeID nearer to the destination. The client can pick up a node randomly and do the lookup process. When a node gets a request, it will first check its finger table (see *Figure 6*). If the routing information of the destination node could not be found in the finger table, the node will find the closest node to the destination and query the request to that node. If the closest peer is the same as the last contacted peer, the lookup has failed. [12]

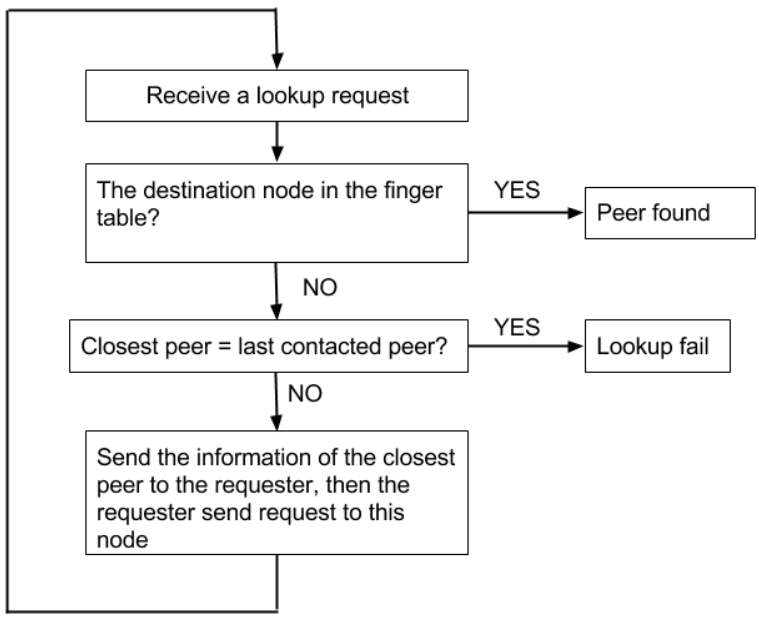


Figure 6. The Chord lookup process

For example, a client asks node_1 where the file belonging to key_14 is (see *Figure 7*). Node_1 checks its finger table and wants to send a request to the node closest to the key, in this case node_8. But since node_8 does not exist, it will send the request to the node responsible for node_8 which is node_10. Node_10 finds that node_15 in its finger table is responsible for key_14, and will then answer node_1 with node_15's routing information. File is found.

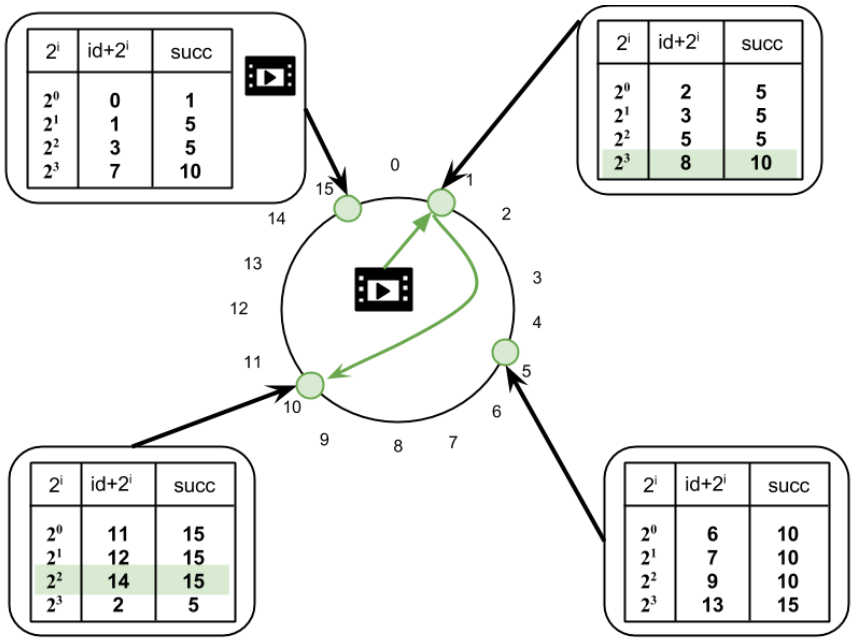


Figure 7. Searching for a file.

4.5 Retrieval

When a node is found, the resources will be sent to the requester. To make this more efficient, it is possible to retrieve data from several sources at the same time. Parallel retrieval of data is used for example in Pseudo-DHT [19]. Pseudo-DHT also has support for proactive switching, which prepares the next node before a switch occurs. This is useful when resources are spread out in different locations, and makes it possible for continuous switch between retrievals. Another property in Pseudo-DHT is the use of buffers. Every node stores the most recently captured video in the buffer, to be streamed live to connected nodes. This prevents jitter and network variations [19].

4.6 Routing

The route decision could be based on e.g. number of hops, geographic location, or proximity. Proximity routing makes a decision based on both the path length, and on the latency [20]. In our case, this will be the best choice considering the latency. When used in routing decisions and initiating the finger table, it is known as Proximity Neighbor Selection. Chord does not consider proximity at all, but this can be improved. An example of a Chord system based on proximity is PChord [23], which uses a proximity list together with the original finger table to find the best path.

4.7 Caching

Chord is based on unidirectional routing, with the property that all routing paths converge towards the destination regardless of the originating sender. This is an advantage considering caching frequent queries. If the converging node caches the resource, it will be more available and the other nodes will have better access to it. It also reduces routing "hot spots" in the network. And because of the consistent hashing property in Chord, the number of caches in the network will always be limited. [10][21]

4.8 Static resilience

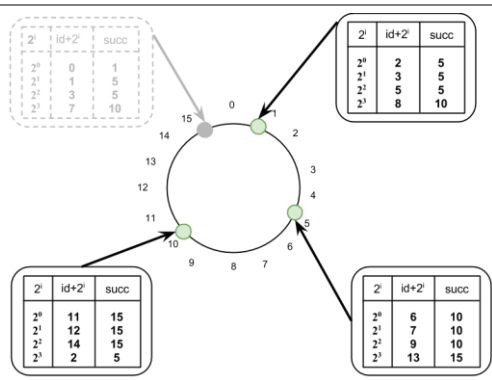
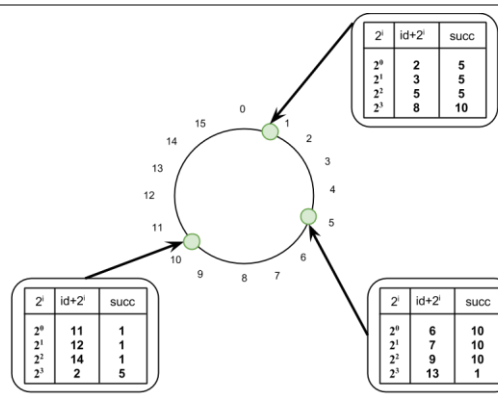
The sequential neighbors in Chord ensure that every message always routes closer towards the destination. This increases the static resilience, but can also lead to longer path ways. Static resilience is the ability to route around failures in the network, and is a very important property in our system. Latency is also an essential variable to consider, since the network will be used for live video streaming. Hence, a tradeoff has to be made between a more fault tolerant system and a shorter response time. Another contributing factor to the static resilience and latency, as earlier mentioned, is the flexibility of the system. Chord has high flexibility in both route selection and neighbor selection, which is a great advantage.

4.9 Redundancy

Chord also provides redundancy through data replication. The data is replicated consecutively among nodes, and the number of replications is defined by the user. If one of the nodes leaves, there will still be other nodes to route to. The leaving node's resources will then instead be handled by the successor, which will replicate the resources to keep the number of replications constant. Because of the redundancy, the system will be able to repair itself even under a massive crash, as long as only one node with the specific resource remains in the ring. Update messages of the node states will be sent periodically between neighbors, to keep the finger tables up-to-date.

4.10 Leaving node

The following procedure happens when a node leaves the network:

Node_15 sends a LEAVE message to all nodes (in this case, node_1, node_5, and node_10) in its finger table and then leaves the network. [12]	
Node_1, node_5, and node_10 update their finger tables. The resources of node_15 will now be handled by node_1, which will then replicate the resources to keep the number of replications constant. [12]	

4.11 Fragmentation

If a network failure occurs, the sender will retry sending the message. But if the network connection is down, we will need some automatic failover solution. Like with the example with the bus (in section 2.1, Problem Definition), there will not be so many choices. The camera node will save resources locally on the SD card. Then, when the connection is reestablished, the videos can continue to stream to the network. This will cause video fragmentation, which is not desirable. If the video is divided into too many parts, the video management will be harder. Each fragment will have a separate segment identifier to keep track of the different parts, resulting in new keys and a higher spread. The fault tolerance will decrease, since the probability of losing data in a node crash will increase. Hence, we aim on having as few segments as possible, and only fragment the video when it is absolutely necessary. To reduce the number of fragments further, it should also be possible to assemble the video parts together. We have chosen to give each segment a unique consecutive id, and have a metadata file to keep track of the number of segments. The metadata file should also give information about time of creation, time-to-live (TTL), file size, etc. When assembling a video, the metadata file is first consulted. If the video is fragmented, it informs the stitching application and the assemblage can begin. The full implementation of this application is considered to be future work.

4.12 Lifetime management

The time of creation and TTL in the metadata-file is used when considering lifetime management. A resource or video should in some cases have limited lifetime, determined by law. The file should then be erased. We have chosen to let each node handle the lifetime management of its assigned resources.

4.13 Resource conflict resolution

When two or more users want to access and modify a resource at the same time, a conflict will occur and several versions of the file may end up being stored. The resource conflict resolution will determine which version to keep, based on some application specific requirements. In Bayou [22], each write or modification to a file needs an accept-stamp by the server. This could be applied to our solution by requiring an accept-stamp from the node. This accept-stamp should also be accepted by all replicated nodes before the write, to provide consistency of the file. Otherwise, the modification will be discarded.

4.14 Security

4.14.1 Routing attacks

In a structured peer to peer network, each node plays a very important role. The knowledge of other nodes is limited because the routing table only contains a part of the whole network. If a node receives a request, and is not the final destination, it will always forward the request to a node that is closer to the destination. A "bad" routing performance will lead to communication failure; therefore it is important to ensure that the route is correct. Sometimes, it is difficult to detect malicious nodes in the system because they behave like all the other nodes. The way to defend against a routing attack is to detect an ongoing attack. [24]

4.14.2 Incorrect Lookup Routing

Incorrect lookup routing means that a malicious node queries a request to an incorrect node, or responds with a wrong result. In a 16 node network, node_0 wants to initiate a request to node_15, see *Figure 8* and *Table 2*. The correct route should be node_0 → node_8 → node_12 → node_14 and then node_15. Suppose node_12 is the malicious node, then this node could forward the request to node_7 instead, which leads to a failed lookup.

TABLE 2: FINGER TABLES FOR THE NODES, WHERE NODE_12 IS THE MALICIOUS NODE.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2^0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
2^1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1
2^2	4	5	6	7	8	9	10	11	12	13	14	15	0	1	2	3
2^3	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7

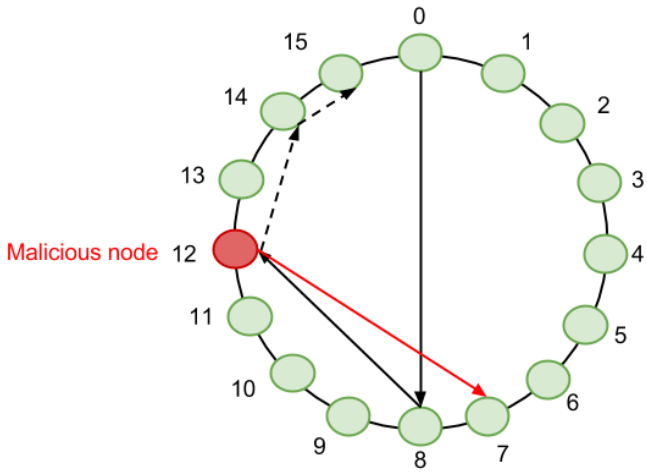


Figure 8. Incorrect Lookup Routing

To identify the malicious node, the query requester can monitor the lookup process. The query requester should control all the steps when a request is approaching the destination. Like we mentioned before, the chord is a unidirectional ring structured network. The distance between node_6 and node_7 is not equal to the distance between node_7 and node_6. All finger tables need to be visible during the backtrack control. In this case, node_12 queries the request to node_7. The query requester will then check node_12's finger table and find out that node_7 is not forwarding closer to the destination and can thus be identified as a malicious node. After the malicious node is discovered, the query requester can backtrack the routing to the latest "good" node and ask for an alternative way. Node_0 can send a request to node_8 instead, and take node_10 as a second choice. [25]

4.14.3 Incorrect Routing Updates

Since each node's finger table has limited space, nodes must update their finger tables when something happens in the network, e.g. a join, or leave, etc. A malicious node could provide incorrect updates to other nodes. This attack can mislead the request to a node that does not exist, or make the routing path longer leading to a higher latency. The solution to this attack is to let nodes try to reach the remote node; if the remote node is reachable then perform the update. [25]

4.14.4 Incorrect Routing Network Partition

One set of malicious nodes builds a parallel chord network that just behaves like a legitimate network, where one or several of the malicious nodes also participate in the real network. In the previous section, we described how a node joins a Chord network. The node must query a join request and bootstrap via an existing node. It is possible that an existing node is a malicious node, thus, the new node is hijacked by the malicious node. The new node is now partied from the real network, and joins the malicious network. In that way, the malicious network can now get the resources from the new node and track the metadata.

To prevent this attack, first of all, the network needs a trusted node to handle the join request. If one node joins the network initially, then it bootstraps only via the trusted node. Nodes within the network create their own lists of trusted nodes that they know from the past. When the node joins the network again, it contacts one of the nodes in the list. [25]

4.14.5 Storage and Retrieval Attacks

Storage and retrieval attacks occur in a network that contains malicious nodes. Unlike a routing attack, malicious nodes could reject all the data that should be stored there, and it could also receive the data first and then delete it. Moreover, even if the malicious node stores the data it receives and is responsible for, it may also deny all the requests from a client or serve a client with modified data. These attacks can be prevented by replicating data, and avoiding single point responsibility. Therefore, in our case, the data is replicated in 3 copies and stored in 3 successors. If one

node is down, a client could still request replications from successors. In other words, there is no single point of failure. To make sure that the data is correct, a client should request copies from two other nodes and compare the result. [25]

4.14.6 Denial-of-service attacks

The key of the Denial-of-Service attack (DoS attack) is to make the victim node unreachable. The malicious node sends a certain amount of requests to the victim node, causing the victim to be overloaded by requests and preventing it to get the real requests. In the Chord network, resources are divided and distributed over the network. Since resources are replicated, and because of the fault tolerance in Chord, the network can still work even if there is just a small amount of nodes left after a crash with possibly longer latency. There is a method to minimize the effects of a DoS attack, where each node accepts only a certain amount of requests from each client. The request load will then be balanced in the system. [25]

4.14.7 Rapid joins and leaves attack

When a node joins or leaves the network, the existing interrelated nodes update their finger tables. When a finger table is updated, a new node should take over all the responsibilities belonging to its identifier, e.g. stored resources replicated from its predecessors, take over the data that it is responsible for, etc. A node can join the network and then leave immediately. When the attack takes place, the malicious node would repeat this action to overload the network. To prevent this kind of attack, a node that connected to the network recently should not query any requests from other nodes. To do this, the system sets a fixed timer for the new node, so that it can not work properly as other existing nodes until the timer expires. [24]

4.14.8 The Sybil attack

Since P2P is a decentralized network, when two nodes connect to each other without any physical connection, it is difficult to authenticate each other's real identity. In a DHT network, nodes can look up each other’s

NodeID. The malicious nodes could use this to forge multiple identities [27]. Chord networks are vulnerable, a malicious node could combine the Sybil attack with other attacks that we mentioned before and create huge amounts of identities, do rapid join/leave attacks or send the incorrect routing update to fill other nodes' finger tables etc. Sybils can then not give any response to all requests or queries from other Sybils, and will fail the routing procedure. The network can not perform the correct routing when the Sybils exceed 25% [28]. To prevent this attack, the network should limit the joining of new nodes. When a node gets a request, it should reply with all its finger tables instead of just the best routing solution. [28]

CHAPTER 5

5 Testing

The testing of the system was done by the use of the open source project OpenChord [29]. A better simulation would be to do this in a real IP camera environment, but this would not be feasible considering the number of cameras that would be needed and the time to set up such a system. We wanted to simulate how well a large Chord network can handle massive crashes, and how much data loss this would cause. By inserting fragmented texts into the network, we could test the searching, retrieval, fault tolerance and stability of the system. We simulated this with 500 virtual nodes, using two different texts: Shakespeare's Sonnet 18 and TCP Joke. We refer to these as Poem and Joke.

SHAKESPEARE'S SONNET 18

Shall I compare thee to a summer's day?
Thou art more lovely and more temperate:
Rough winds do shake the darling buds of May,
And summer's lease hath all too short a date:
Sometime too hot the eye of heaven shines,
And often is his gold complexion dimm'd;
And every fair from fair sometime declines,
By chance, or nature's changing course, untrimm'd;
But thy eternal summer shall not fade
Nor lose possession of that fair thou ow'st;
Nor shall Death brag thou wander'st in his shade,
When in eternal lines to time thou grow'st;
So long as men can breathe or eyes can see,
So long lives this, and this gives life to thee.

TCP JOKE

"Hi, I'd like to hear a TCP joke."
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out. Hello, would you like to hear a TCP joke?"

Both of these texts need to be fully retrieved after the crash for the test to be considered a success. In the first test, for each node, we used 3 replications distributed on 3 successors. The texts are divided on respectively 2 keys, which is to be considered as a fragmentation. All crashes are done simultaneously and randomly, hence every test will be unique and not be affected by one another. We will repeat every test 50 times, to get reliable results. The result from the test will be shown in the following section.

CHAPTER 6

6 Results

Here is the result from the tests.

✓ = All resources were retrieved after the crash.
P = All resources exist, but some resource(s) could not be retrieved.
× = Failure, some or all resources are lost.

Test 1.1: Crashing 10 % of the network, e.g. 50 nodes simultaneously.
Number of tests: 50

✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓
✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓

Comment: Works well, system recovers.

Test 1.2: Crashing 20 % of the network, e.g. 100 nodes simultaneously.
Number of tests: 50

✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓
✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓P

Comment: Works well, system recovers.

Test 1.3: Crashing 30 % of the network, e.g. 150 nodes simultaneously.
Number of tests: 50

✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓
✓✓✓✓P	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓P

Comment: Works well, system recovers

Test 1.4: Crashing 40 % of the network, e.g. 200 nodes simultaneously. Number of tests: 50

✓✓✖✖P	✓✓✓✓P	✓✓✓✓✓	✓✓✓✓✓	✓✓✓✓✓
✓✓✓✓P	✓✓✓PP	✓✓✓✓✖	✓✓✖✓✓	✓✓P✓P

Comment: Mostly works, but some ring partitioning has occurred.

Test 1.5: Crashing 50 % of the network, e.g. 250 nodes simultaneously. Number of tests: 50

✓✓✓✖P	✓✖✖✖✖	✖✓✓✓✓	✓✓✓✓✓	✖PP✓✓
✓✓P✖✖	P✓✓✖P	✓✓✓✓P	✓PPPP	PPP✓✓

Comment: Mostly works, but some ring partitioning has occurred.

Test 1.6: Crashing 60 % of the network, e.g. 300 nodes simultaneously. Number of tests: 50

✖✖✖✖✖	✖✖✖P✓	✓✓✖✓P	✓✖✖✖P	✓✓PPP
✖✖✓✓P	✖✖✓PP	✓✖PPP	✓✖P✖✖	PPPPP

Comment: System struggles, more partitioning and failures.

Test 1.7: Crashing 70 % of the network, e.g. 350 nodes simultaneously. Number of tests: 50

✖✖✖✖✖	✖✖✖✖✖	✖✖✖✖✖	✖PPPP	PPPP✓
✖✖P✖P	✖PP✓✖	PP✖✖P	✖✖P✖P	✖P✖✖P

Comment: Most failures, but with some succeeding retrieval.

The testing could be summarized into the following table and graph. A test will be considered a "Success" if all resources were retrieved after the crash. Sometimes resources exist in the system, but could not be retrieved. This is marked as "Partitioned". When some or all resources are lost, it is considered a "Failure".

TABLE 3: RESULTS FROM THE TEST.

	Success	Partitioned	Failure
Test 1.1	50	0	0
Test 1.2	49	1	0
Test 1.3	48	2	0
Test 1.4	39	9	2
Test 1.5	26	19	5
Test 1.6	12	18	20
Test 1.7	2	19	29

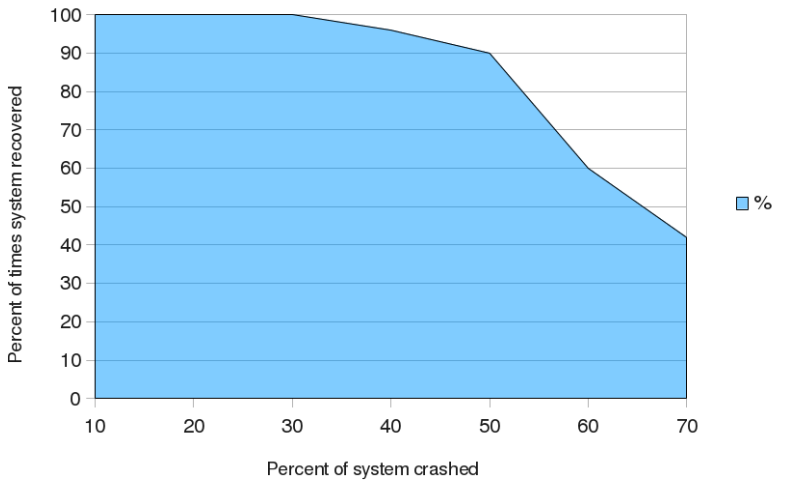


Figure 9. Percent of times system recovered after crashes.

CHAPTER 7

7 Conclusion

According to the test, Chord is in most cases capable of handling massive node crashes and still recovers to full capacity (see *Figure 9*). Not until the 60 percent system crash, Chord begins to show some struggle. This is a very good result, considering 300 randomly chosen nodes out of 500 crashed at the same time. And if compared to an ordinary server-client solution, with a single-point-of-failure, this provides a much better resiliency.

However, one thing we did not realize before the testing was the ring partitioning problem [30]. Sometimes, during massive node crashes, a part of the ring could separate to an own entity. The new entity nodes do not realize that a partitioning has occurred, and will continue to route to their only known neighbors. This is probably why we sometimes could not retrieve the resources from the calling node, even when it existed in the network.

Otherwise, the Chord network is a good solution considering the requirements.

1. It provides a storage management system capable of handling all resources in the network.
2. The decentralized autonomous structure lets the nodes handle all communication and network organization themselves.
3. Resources are very available to the network through high fault tolerance and redundancy.
4. Chord provides efficient search and retrieval of distributed resources, with an average lookup performance of $O(log(N))$.
5. Chord is highly scalable, and will work for both small networks to larger networks.
6. Chord has a natural load balance when using a fair hash function. We chose to use the better hash SHA-256 instead of SHA-1.

7. The system handles node crashes and communications failures well. The protocol has high fault tolerance and the best static resilience of the studied protocols. It also provides redundancy by data replication.

To fulfill the other requirements and to improve it even further, the following additions to the protocol have been proposed:

1. The system is capable of storing fragmented resources, but an additional overlay with gathering of video parts and a assemble application is needed to fulfill the requirement of fragmentation.
2. Proactive switching and parallel retrieval can be used for a more efficient retrieval.
3. Files are removed after a beforehand specified time, or TTL, specified in the metadata file. The nodes are responsible of this removal.
4. The Resource conflict resolution will determine which file to keep by an accept-stamp admitted by all replicated nodes.
5. Buffers will prevent jitter and network variations, improving latency, and make it possible for live streaming.
6. Proximity routing is used to decrease latency.
7. Caching at a converging node will prevent routing “hot spots”.
8. Security aspects have been considered and preventions to the attacks have been proposed.

Other DHT protocols were also considered during the research, some of which were also well suited to our requirements. But since Chord compared better to the others, this protocol was our first choice.

CHAPTER 8

8 Future Work

Many aspects have been taking into consideration during the project, and limitations were necessary to narrow it down to a reasonable scope. We have taken all requirements into consideration during research, system development and testing. But it was not possible, because of the timespan, to develop a complete functioning system of this size on actual cameras. Then our testing could also have included other aspects, such as latency, which were not considered in our virtual test environment. The described system is only a proposal, and can be seen as a reference for further development.

To take this project further, a complete system with test environment on actual cameras needs to be developed. This together with a solution to the ring partitioning problem we encountered during testing.

A video assemblage software is needed to decrease the number of fragments in the system. A lower amount of fragments will make the searching more efficient, and the system less complex.

The security preventions need to be embedded in the system. Also other security aspects need to be considered, such as authentication and permission to data etc. A user without permission should not be able to get access to sensitive information.

References

[1] One surveillance camera for every 11 people in Britain says CCTV survey [read: 2014-09-05] <http://www.telegraph.co.uk/technology/10172298/One-surveillance-camera-for-every-11-people-in-Britain-says-CCTV-survey.html>

[2] Axis, “Choosing the right video management software solution” [read: 2014-08-20] <http://www.axis.com/files/whitepaper/wp_choosing_right_vms_50485_en_1302_lo.pdf>

[3] Axis - “Edge Storage” [read: 2014-08-20] <http://www.axis.com/files/datasheet/ds_edge_storage_en_1109_lo.pdf>

[4] John Rhoton, Risto Haukioja, “ Cloud Computing Architected: Solution Design Handbook”, 2011

[5] Ammar Waysi AITuhafi, Sureswaran Ramadass and Yung-Wey Chong, “Concepts and Types of Peer-to-Peer Network Topology for Live Video Streaming” Sept. 2013

[6] Jan Suwart, “Wireless Ad Hoc Networks: Limitations, Applications and Challenges”, april, 2008

[7] Introduction to Mobile Ad hoc Networks (MANETs) <http://user.it.uu.se/~erikn/files/DK2-adhoc.pdf>

[8] WirelessHART - How it works <http://en.hartcomm.org/hcp/tech/wihart/wireless_how_it_works.html>

[9] Stig Petersen, Simon Carlsen, “Performance Evaluation of WirelessHART for Factory Automation” Sept. 2009

[10] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, Steven Lim, “A survey and comparison of peer-to-peer overlay network schemes”, 2005

[11] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, I. Stoicak,
“The Impact of DHT Routing Geometry on Resilience and Proximity” 2003

[12] Jan Seedorf, Christian Muus, M. Frans Kaashoek, Hari Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, 2001

[13] Petar Maymounkov and David Mazieres, “Kademlia: A Peer-to-peer Information System Based on the XOR Metric” 2002

[14] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, “A Scalable Content-Addressable Network”. august, 2001

[15] “Overlay and P2P Networks Structured Networks and DHTs”, [read: 2014-08-12]
<https://www.cs.helsinki.fi/webfm_send/1338>

[16] Antony Rowstron, Peter Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems”, 2001

[17] The Impact of DHT Routing Geometry on Resilience and Proximity <https://www.cs.cmu.edu/~dga/15-849/papers/gummadi-sigcomm2003.pdf>

[18] Laurence T. Yang,Minyi Guo, High-Performance Computing: Paradigm and Infrastructure

[19] Jeonghun Noh, Sachin Deshpande, “Pseudo-DHT: Distributed Search Algorithm For P2P Video Streaming”, 2008

[20] Weiyu Wu, Yang Chen, Xinyi Zhang, Xiaohui Shi, Lin Cong, Beixing Deng, Xing Li, “LDHT: Locality-aware Distributed Hash Tables “, jan, 2008

[21] Jan Seedorf, Christian Muus, “Availability for DHT-Based Overlay Networks with Unidirectional Routing”, 2008

[22]Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer and Alan J. Demers, “Flexible Update Propagation for Weakly Consistent Replication”, 1997

[23] Feng Hong, Minglu Li, Jiadi Yu, Yi Wang, “PChord: Improvement on Chord to Achieve Better Routing Efficiency by Exploiting Proximity”, june, 2005

[24] Emil Sit, Robert Morris, “Security Considerations for Peer-to-Peer Distributed Hash Tables”, 2002

[25] Quang Hieu Vu, Mihai Lupu, Beng Chin Ooi, “Security in Peer-to-Peer Network” in the “Peer to peer computing, Principles and Applications”, chapter 6, 2010

[26]Monica Haladyna Braunisch, “Chord and Symphony: An Examination of Distributed Hash Tables and Extension of PlanetSim“ (2006), [read: 2014-08-31]
<http://projects-deim.urv.cat/trac/planetsim/chrome/site/HaladynaBraunischMALMThesis20060430.pdf>

[27] John R. Douceur, “The Sybil Attack”, 2002

[28] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek, Ross Anderson, “Sybil-resistant DHT routing”, 2005

[29] The OpenChord Project <http://sourceforge.net/projects/open-chord/>

[30] Monica H. Braunisch, “Chord and Symphony: an Examination of Distributed Hash Tables and Extension of PlanetSim”, 2006

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2014-410

http://www.eit.lth.se