***Delta Team***

**Alternative Internet Name Service**

Members: Karthika Krishnamurthy (546810), Nguyen Van Tan (547301), Rohan Krishnakumar (546807), Bijay Basnet (545633)

# 1. Overview and overall architecture

**Overview:**

Alternate Internet Name Service (AINS) is a distributed service that maps names into some other information. The result information could be phone numbers, other contact details, IP addresses or something else [1]. The following section explains Alternate Internet Name Service (AINS) implementation done in this project. AINS is implemented similar to Domain Name System (DNS). DNS is a naming system for resources that are connected to the Internet. It translates domain name of each resource into IP address. Whenever a client types a domain name in a browser, a query is initiated from client to the DNS server which in turn resolves the domain name to IP address. Domain names are used for user friendly purpose. It is necessary to convert domain names to IP address to route the packet in a network. Routers understand only the IP address [2]. In this project, AINS is implemented very similar to DNS. Unlike DNS, it does not maintain a hierarchical naming system instead a single server has all domain names to respond to DNS queries. If the requested domain name is not present, then server responds client with a message "No entries found".

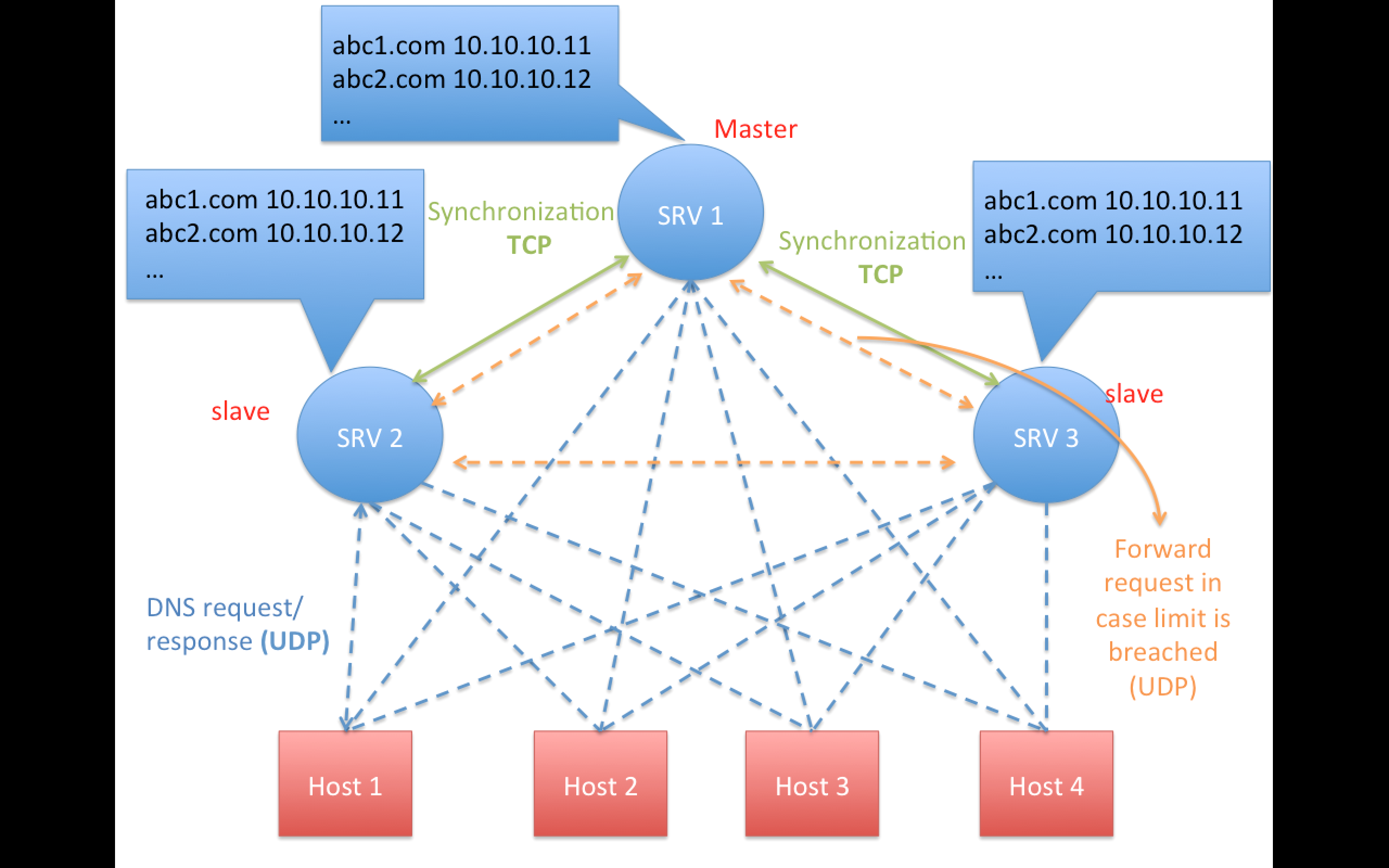


Figure 1 - Overall architecture

**Overall Architecture:**

**AINS database:**

AINS database has the mapping of domain name to IP address in a text file. It has domain name, IP address and action in a single line. AINS database can be manually updated anytime.

**Server:**

As part of this project, three AINS servers will used for implementation. Each server will maintain identical AINS database. Each server is capable of responding to client request independently. To each AINS query, servers responds with AINS reply which contains IP address to the requested domain name.

**Client:**

Each client sends query which contains a domain address to be resolved by the DNS server. Any client can query any DNS server. Each client maintains the list of DNS servers. List of DNS servers will be statically updated in each and every client. The result of DNS is used to forward packets. Forwarding packets is out of scope. Timeout happens in the client if it does not get reply from server.

**Server synchronization:**

This AINS implementation has three DNS servers. Out of the three DNS servers, one server is configured as master server and rest of the servers are configured as slave servers. Database can be manually updated only in the master server. As soon as the master server is updated, it updates rest of all the slave servers. The same design can be extended to any number of servers. Time delay in updating the slave servers is kept minimum. Updating the database has higher priority than responding to queries.

**Load balancing:**

Capacity of server is defined as the maximum number of queries that server can respond to. If server is loaded with more queries than its capacity, then server will start dropping the queries. To avoid this situation, it is necessary to balance the load between the servers. It is done in two steps. Step1: Clients pick servers randomly for each and every query. Step2: If a server is over-loaded, then it must re-direct the queries to other servers.

**Communication protocol:**

Client query and server reply will be implemented using UDP protocol and database synchronization will be implemented using TCP protocol. This project supports both IPv4 and IPv6 addressing.

# 2. Requirements description

**Functions to be implemented:**

**Client:**

- Client includes following functionality.

- sending DNS query to server.

- processing received DNS response.

- implementing timeout mechanism.

- preliminary load balancing.

**Server:**

- Server includes following functionality.

- processing DNS query.

- searching database with given query.

- sending response to client.

- allowing manual update of database.

- sending update to servers.

- processing database update request from master server.

- allowing to update database through synchronization.

- load balancing.

**How functionality is distributed between different components:**

* Client-Server: AINS query and response takes place between the server and the client. It is implemented using UDP protocol.
* Server-server: Database synchronization takes place between master and slave servers. It is implemented using TCP protocol.
* Load balancing: Client and server both implements load balancing logic.

**How software is used:**

* Client initiates a query to the AINS server with a domain name.
* Server after receiving request from client, searches its database and sends response back to client. If the requested server is over loaded, then query is re-directed to another server. If all servers are over-loaded, then query is dropped.
* Client has a timer. Query packet is considered lost if response is not received before the timer expiry.
* Since there are more than one servers, database in all servers must be identical. The master server sends update to the all slave servers after a manual update. Database in the slave servers cannot be manually updated.

**Programming language:** C

**Operating platform:** Linux

**Library dependencies:** No, except default C and Linux libraries.

**nwprog\* hosts:** Program any run in any Linux hosts.

# 3. Instructions

**How to build**

“Makefile” is present inside the src folder. Separate compilation commands are need for master server, slave server and client.

**Compilation command for client:**

make mode=client

**Compilation command for master server:**

make mode=server\_master

**Compilation command for slave server:**

make mode=server\_slave

**Command to clean all object (.o) files**

make clean

Binary files namely client, server\_master and server\_slave will be generated after successful compilation of client, master server and master slave inside **bin** folder. bin folder is present inside src folder.

**How to use:**

AINS service can be tested in the following way.

**Step 1**:Execute all compilation commands. Binaries are generated in bin folder.

**Step 2:**DB files must be present in the src folder.

**Step 3**:First, server\_master must be executed.

**Step 4**:Next, server\_slave binary must be executed.

**Step 5:**Text file with list of available server must be present in the src folder.

**Step 6:**client binary can be executed with any “domain name” as command line argument.

**Step 7:**client must receive reply from server with IP address corresponding to the given domain name.

**Note:** The above mentioned steps are subjective to change after phase-2. Exact steps will be updated in phase-3 document.

**Must work in Aalto login servers, or in course test servers:**

Code is being tested in both Aalto login server and course test servers

**Is any configuration needed:**

- No additional configuration or libraries are needed.

- But other than source code, software testing needs,

- DB file which has domain name to IP address mapping.

- File with available server IP addresses.

# 4. Communication protocol

**Describe the protocol messages and message interaction. Diagrams are likely useful**

AINS service uses two communication protocols

1) UDP: AINS uses UDP protocol to query server and servers send a UDP reply to those queries.

2) TCP: AINS uses TCP protocol to synchronize data bases between AINS servers. All slave servers will receive update from master server.

**UDP: AINS query and response**

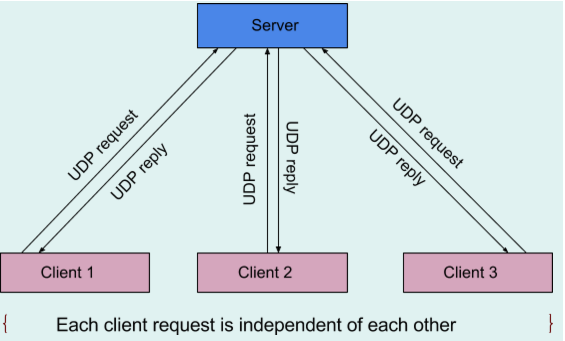
Each client will resolve domain name to IP address before accessing the Internet. To resolve domain name, client sends a query to AINS server. Thus all AINS servers must be up and running to respond to any client request.

- Server creates a thread during initialization which receives the UDP queries. Server uses a fixed port say 6000 to listen to the UDP queries. Server runs a loop such that it can receive and process UDP request all time.

- Client sends domain name [eg. home.flash.com] in the message to server's fixed port say 6000. Message to server consists of only domain address in the message.

- Client chooses server from list of available servers using random logic.

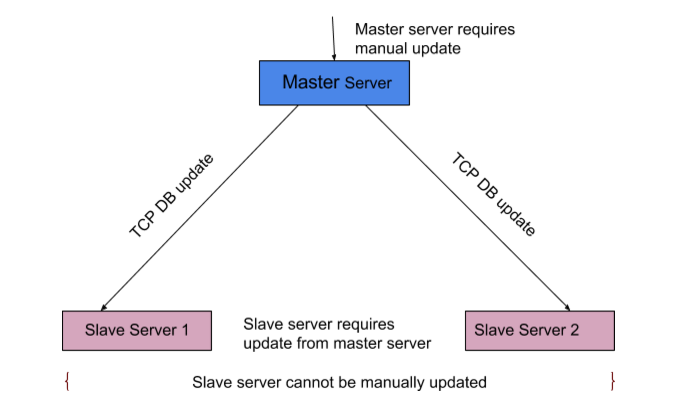
- Server as soon as receives a UDP query, searches AINS database and finds the IP address corresponding to domain address and sends response to client.



- Since server and client uses UDP, if a UDP request from client or UDP reply from server is lost or if server is down, it is not possible to identify the packet loss. To avoid this problem, two solutions were implemented. Client starts running a timer as soon as it sends a query to server. If reply is not received before timer expiry, then UDP request is considered lost and UDP query is sent once again to different server. DNS reply received after timeout is considered lost.

**TCP: AINS server database synchronization**

Servers need database (DB) files to respond to all AINS queries. Practically it is not possible to manually configure/ update DB present in all AINS servers. To overcome this problem, out of available AINS servers, one server is configured as master server and rest of the servers are configured as slave servers. In phase-2, one of the servers is manually configured as master server and the rest are manually configured as slave servers. In phase-3, a server\_list will be present in all servers. Servers with highest IP address is chosen as master and rest of the servers will act as slave servers. It is enough to update master server alone and master sends update to all slave servers.



- As master server boots up, it creates a new thread and accepts incoming TCP connection from other servers. Server does not accept connections from servers which are not part of the server list.

- Server listens to file descriptor of the DB. Once DB update is complete, master server sends all modified data to slave servers that are already connected to master server.

- Master server sends array of structures and each structure corresponds to one modified entry in DB. Before sending array of structures, master server will send to total number of entries to be sent to slave server.

- Each structure consists of already existing domain name, IP address , IP version number, new domain name, new IP address, new domain name and action. Action indicates whether entry has to be deleted, or updated, or created in DB.

- Action 1 modifies already existing data in DB. Action 2 indicates a new data has to be added to DB. Action 3 indicates existing data must be deleted from DB.

- DB update thread has higher priority than any other threads.

**What is done in terms of reliability? How to avoid congestion?**

- Synchronization messages are important as they are basis for AINS functionality. Thus it is implemented using TCP for reliability.

- AINS queries requires very less processing time. AINS queries will never cause storm or congestion as only one query packet is sent by each client application.

- Usually DNS query packets have higher L3 priority than normal user traffic. This project involves only application programming. Prioritizing the traffic is out of scope.

# 6. Testing

**Describe performance indicators/metrics you use to analyze your code**

- To measure minimum, average and maximum delay (RTT) for query and response.

- Rate of success of queries.

**Test:**

**Scenario 1 - From wireless network (e.g. Aalto Open) to a fixed server**

**Preconditions:**

- 4 hosts running on wireless network (Aalto Open)

- 3 servers running on course test servers: nwprog1.netlab.hut.fi, nwprog2.netlab.hut.fi and nwprog3.netlab.hut.fi

- IPv4 addresses

* **Test case 1\_Name service query - Found entries\_1 client**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Run ./client on 1 host | Program starts successfully |  |
| Input query: home.flash.com | Receives IP address … successfully | Time to query: |
| Input query: lab.flash.com | Receives IP address … successfully | Time to query: |
| Input query: office.flash.com | Receives IP address … successfully | Time to query: |
| Input query: uni.flash.com | Receives IP address … successfully | Time to query: |

* **Test case 2\_Name service query - No entries found\_1 client**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Run ./client on 1 host | Program starts successfully |  |
| Input query: h | Receives "No entries found" | Time to query: |
| Input query: home.flash.com.abc | Receives "No entries found" | Time to query: |
| Input query: office.com | Receives "No entries found" | Time to query: |
| Input query: qwerty | Receives "No entries found" | Time to query: |

* **Test case 3\_Name service query - Found entries\_4 clients**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Run ./client on 4 hosts | Program starts successfully |  |
| Input query: home.flash.com | Receives IP address … successfully | Time to query: |
| Input query: lab.flash.com | Receives IP address … successfully | Time to query: |
| Input query: office.flash.com | Receives IP address … successfully | Time to query: |
| Input query: uni.flash.com | Receives IP address … successfully | Time to query: |
| Verify load balancing in client side by reading the output in client (connected server details) | Client selects different servers randomly |  |

* **Test case 4\_Name service query - No entries found\_4 clients**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Run ./client on 4 hosts | Program starts successfully |  |
| Input query: h | Receives "No entries found" | Time to query: |
| Input query: home.flash.com.abc | Receives "No entries found" | Time to query: |
| Input query: office.com | Receives "No entries found" | Time to query: |
| Input query: qwerty | Receives "No entries found" | Time to query: |
| Verify load balancing in client side by reading the output in client (connected server details) | Client selects different servers randomly |  |

* **Test case 5\_Synchronization database - Add new records**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Add records in DB file in master server at location … | DB file is updated successfully |  |
| Verify DB file of 2 slave servers | DB files in slave servers should be updated accordingly to the changes from DB of master server |  |
| Send one query with new domain name from one client | Receives response from server successfully with new IP | Time to query: |

* **Test case 6\_Synchronization database - Delete record**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Delete a record in DB file in master server at location … | DB file is updated successfully |  |
| Verify DB file of 2 slave servers | DB files in slave servers should be updated accordingly to the changes from DB of master server |  |
| Send one query with old domain name | Receives "No entries found" from the server | Time to query: |

* **Test case 7\_Synchronization database - Modify domain name**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Modify domain name in DB file in master server at location … | DB file is updated successfully |  |
| Verify DB file of 2 slave servers | DB files in slave servers should be updated accordingly to the changes from DB of master server |  |
| Send one query with updated domain name | Receives IP address successfully which mapped to the updated domain name from server | Time to query: |

* **Test case 8\_Synchronization database - Modify IP address**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Modify IP address in DB file in master server at location. | DB file is updated successfully |  |
| Verify DB file of 2 slave servers | DB files in slave servers should be updated accordingly to the changes from DB of master server |  |
| Send one query with the domain name mapped with the new IP address | Receives new IP address successfully from server | Time to query: |

* **Test case 9\_Synchronization database - Modify IP address & Domain name**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Modify IP address and domain name in DB file in master server at location. | DB file is updated successfully |  |
| Verify DB file of 2 slave servers | DB files in slave servers should be updated accordingly to the changes from DB of master server |  |
| Send one query with the new domain name mapped with the new IP address | Receives new IP address successfully from server | Time to query: |

* **Test case 10\_** **Synchronization database - Processing queries dependency**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Precondition: makes synchronization longer than the timer on client |  |  |
| Send one query from client | Connection timeout, second query must get proper reply |  |

* **Test case 11\_** **Load balancing**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Precondition: All servers must be down |  |  |
| Send query to all servers which are not up and running | Client should stop forwarding queries after trying all all servers once |  |

**Scenario 2 - IPv6 compatibility**

**Preconditions:**

- Enable IPv6 in mininet

edit the following line in /etc/default/grub:

GRUB\_CMDLINE\_LINUX\_DEFAULT="ipv6.disable=1 text"

to GRUB\_CMDLINE\_LINUX\_DEFAULT="ipv6.disable=0 text"

then run sudo update-grub and reboot mininet

- Using **ipv6.py** script (being developed) to create topologies in mininet:

+ IPv4 at hosts and IPv6 at servers (ipv4ipv6)

+ IPv6 at hosts and IPv4 at servers (ipv6ipv4)

+ IPv6 at hosts and IPv6 at servers (ipv6ipv6)

* **Test case 1\_IPv4 at hosts and IPv6 at servers**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Use **ipv6.py** script and create topoly **ipv4ipv6** | Topology is created successfully |  |
| Run xterm command to open console of hosts and servers | Console of hosts and servers open successfully |  |
| Verify IP address of hosts and servers | Hosts have IPv4 addresses and servers have IPv6 addresses |  |
| Run ./client on hosts | Programm starts successfully |  |
| Input query "home.flash.com" on hosts | Receives IP address …. from server successfully | Time to complete 1 query |

* **Test case 2\_IPv6 at hosts and IPv4 at servers**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Use **ipv6.py** script and create topoly **ipv6ipv4** | Topology is created successfully |  |
| Run xterm command to open console of hosts and servers | Console of hosts and servers open successfully |  |
| Verify IP address of hosts and servers | Hosts have IPv6 addresses and servers have IPv4 addresses |  |
| Run ./client on hosts | Programm starts successfully |  |
| Input query "home.flash.com" on hosts | Receives IP address …. from server successfully | Time to complete 1 query |

* **Test case 3\_IPv6 at hosts and servers**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Use **ipv6.py** script and create topoly **ipv6ipv6** | Topology is created successfully |  |
| Run xterm command to open console of hosts and servers | Console of hosts and servers open successfully |  |
| Verify IP address of hosts and servers | Both hosts and servers have IPv6 addresses |  |
| Run ./client on hosts | Programm starts successfully |  |
| Input query "home.flash.com" on hosts | Receives IP address …. from server successfully | Time to complete 1 query |

* **Test case 4\_** **IPv6\_Synchronization database\_Add new records**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Use **ipv6.py** script and create topology **ipv6ipv6** | Topology is created successfully |  |
| Modify domain name in DB file in master server at location … | DB file is updated successfully |  |
| Verify DB file of 2 slave servers | DB files in slave servers should be updated accordingly to the changes from DB of master server |  |
| Send one query with updated domain name | Receives IP address successully which mapped to the updated domain name from server |  |

**Scenario 3 - High-latency mininet scenario ("latency")**

**Preconditions:**

- Use **topologies.py** script to create topology with name “**latency**” (bw=10, delay='400ms') in mininet: sudo mn -x --custom aalto/topologies.py --link=tc --topo latency

* **Test case 1\_Client\_Query Timer\_Greater than delay**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Pre-condition: timer = 500 ms (modify in client code) |  |  |
| Run ./client from 1 host | Program runs successfully |  |
| Send 1 query | Receives IP address successfully from the server |  |
| Read the time for 1 query from the output in client | Response after 400 ms and before 500 ms | Time to query: |

* **Test case 2\_Client\_Query Timer\_Less than delay**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Pre-condition: timer = 300 ms (modify in client code) |  |  |
| Run ./client from 1 host | Program runs successfully |  |
| Send 1 query | No response from server |  |
| Read the output from client | Connection timeout |  |

**Scenario 4 - Slow mininet scenario ("slow")**

**Preconditions:**

- Use **topologies.py** script to create topology with name “**slow**” (max\_queue\_size=5, bw=0.1, delay='200ms') in mininet:

sudo mn -x --custom aalto/topologies.py --link=tc --topo slow

* **Test case 1\_Client\_Query Timer\_Greater than delay**

Test multiple times

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Precondition: timer = 10 s (modify client code) |  |  |
| Run ./client from 1 host | Program starts successfully |  |
| Send 1 query | Receives IP address successfully from the server |  |
| Read the time for 1 query from the output in client | Response before 10 s | Time to complete 1 query:  Success rate: |

* **Test case 2\_Client\_Query Timer\_Less than delay**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Precondition: timer = 190 ms (modify client code) |  |  |
| Run ./client from 1 host | Program runs successfully |  |
| Send 1 query | No response from server |  |
| Read the output from client | Connection timeout |  |

**Scenario 5 - Over-buffered mininet scenario ("buffers")**

**Preconditions:**

- Use **topologies.py** script to create topology with name “**buffers**” (max\_queue\_size=200, bw=0.1, delay='200ms') in mininet:

sudo mn -x --custom aalto/topologies.py --link=tc --topo buffers

* **Test case 1\_Client\_Query Timer\_Greater than delay**

Test multiple times

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Precondition: timer = 10 s (modify client code) |  |  |
| Run ./client from 1 host | Program starts successfully |  |
| Send 1 query | Receives IP address successfully from the server |  |
| Read the time for 1 query from the output in client | Response before 10 s | Time to complete 1 query:  Success rate: |

* **Test case 2\_Client\_Query Timer\_Less than delay**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Precondition: timer = 190 ms (modify client code) |  |  |
| Run ./client from 1 host | Program runs successfully |  |
| Send 1 query | No response from server |  |
| Read the output from client | Connection timeout |  |

**Scenario 6 - Lossy mininet scenario ("lossy")**

**Preconditions:**

- Use **topologies.py** script to create topology with name “**lossy**” (loss=10, bw=10, delay='20ms') in mininet:

sudo mn -x --custom aalto/topologies.py --link=tc --topo lossy

* **Test case 1\_Client\_Query Timer\_Greater than delay**

Test multiple times

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Precondition: timer = 10 s (modify client code) |  |  |
| Run ./client from 1 host | Program starts successfully |  |
| Send 1 query | Receives IP address successfully from the server |  |
| Read the time for 1 query from the output in client | Response before 10 s | Time to complete 1 query:  Success rate: |

* **Test case 2\_Client\_Query Timer\_Less than delay**

|  |  |  |
| --- | --- | --- |
| Steps | Expected result | Actual result |
| Precondition: timer = 190 ms (modify client code) |  |  |
| Run ./client from 1 host | Program runs successfully |  |
| Send 1 query | No response from server |  |
| Read the output from client | Connection timeout |  |

**Scenario 7 – SIGNALS handling**

TBD

Many test cases are not executed yet. More test cases will be added in phase-3

# 8. Distribution of work

Total project is internally divided different stages:

1) Design stage – Phase 1

2) Implementation stage – Phase 1 & Phase 2

2) Testing/Bug fixing stage – Phase 2 & Phase 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Task** | **Task Owner** | **Duration** | **Phase** |
| 1 | Design phase (Everyone studied the project before design discussion) | All Team members | 2 hours | Phase 1 |
| 2 | Documentation for  phase 1 | Tan & Karthika | 8 hours | Phase 1 |
| 3 | Client: UDP to send DNS request and receive response, randomly select DNS server | Bijay | 10 hours | Phase 1  &  Phase 2 |
| 4 | Server: UDP to receive DNS request, read data and send response | Tan | 10 hours | Phase 1  &  Phase 2 |
| 5 | Timeout in client | Tan & Bijay | Not yet estimated | Phase 2 |
| 6 | TCP synchronization for master server | Rohan | 10 hours | Phase 1  &  Phase 2 |
| 7 | TCP synchronization for slave server | Karthika | 10 hours | Phase 1  &  Phase 2 |
| 8 | Makefile | Bijay and Rohan | 4 hours | Phase 2 |
| 9 | Load balancing on server side | Rohan | 15 hours | Phase 3 |
| 10 | Load balancing on client side | Tan & Bijay | 4 hours | Phase 2 |
| 11 | Server side logging & Mutex lock | Karthika | 6 hours | Phase 2 |
| 12 | Documentation for  phase 2 | Karthika | 8 hours | Phase 2 |
| 13 | High level test plan | Tan & Bijay | 15 hours | Phase 2 |
| 14 | Integration of code | Rohan | 10 hours | Phase 2 |
| 15 | Testing | Will be split between team members | Yet to be divided into sub tasks | Phase 3 |
| 16 | Bug fixing | Will be split between team members | Not yet estimated | Phase 3 |

This list is not yet completed. More tasks may get added in the future. Tasks under testing is not yet split or estimated. It will be done in phase -2 .

**Meeting schedule:**

**Meeting – 1:**

**Date:** 10/02/2016

**Location:** I346, Otakaari 5.

**Time:** 12 – 2 PM

**Agenda:** Project kick-off

**Participants:** Karthika, Tan, Rohan, Bijay

**Minutes of the meeting:**

1) Project manager “Karthika”, Communication channel (telegram / facebook messenger), Programming language “C”

2) Understood the requirements of the project.

**Meeting – 2:**

**Date:** 19/02/2016

**Location:** I346, Otakaari 5.

**Time:** 2 – 4 PM

**Agenda:** Design clarifications from advisor.

**Participants:** Karthika, Tan, Bijay, advisor (Arseny)

**Minutes of the meeting:**

1) Discussed possible designs for the project. 2) Clarified queries from advisor.

**Meeting – 3:**

**Date:** 24/02/2016

**Location:** Table in front of I346, Otakaari 5.

**Time:** 12 – 2 PM

**Agenda:** Freeze the design of the project.

**Participants:** Karthika, Tan, Rohan, Bijay

**Minutes of the meeting:**

1) Design for DNS implementation project was finalized. 2) Project implementation was divided into various phases. 3) Tasks of the first phase must be completed before/on march 4th 2016. 4) Next meeting is scheduled for next week (before March 4th class). 5) Before next meeting, each project member have to list all sub-tasks from assigned high level task.

**Meeting – 4:**

**Date:** 02/03/2016

**Location:** Table in front of I346, Otakaari 5.

**Time:** 1 – 2 PM

**Agenda:** To check the status of the project and deliverables of phase-1.

**Participants:** Karthika, Tan, Rohan, Bijay

**Minutes of the meeting:**

1) Status of the phase – 1 deliverable is checked. 2) Status of coding on various sub-tasks were checked. 3) Deadlines for the sub-tasks were fixed.

**Meeting – 5:**

**Date:**

10/03/2016

**Location:** Table in front of I346, Otakaari 5.

**Time:** 12 – 3 PM

**Agenda:** To check the status of the project and deliverables after phase-1. Checked out few new tasks.

**Participants:** Karthika, Tan, Rohan. Bijay

**Minutes of the meeting:**  
1) Integrator - Rohan HLTP (High Level Test Plan) - Tan, Bijay, Document - Karthika.  
2) Server synchronization must be implemented in phase-2  
3) Threads implementation must be done in phase-2  
4) Daemon process - Rohan & Tan.  
5) Lock implementation - Karthika  
6) Enhancements to be done in phase-3

**Meeting – 6:**

**Date:**

14/03/2016

**Location:** Table in front of I346, Otakaari 5.

**Time:** 11 AM – 12 PM

**Agenda:** To check the status of the project and deliverables of phase-2.

**Participants:** Karthika, Tan, Rohan

**Minutes of the meeting:** 1) Discussed issues in integration. 2) Status of tasks.

# 9. References

[1] [https://src.aalto.fi/ip/instructions/index.html#03\_topics](https://src.aalto.fi/ip/instructions/index.html" \l "03_topics)

[2] <http://compnetworking.about.com/cs/domainnamesystem/g/bldef_dns.htm>/

[3] <http://www.programminglogic.com/sockets-programming-in-c-using-udp-datagrams/>

[4] <http://www.binarytides.com/dns-query-code-in-c-with-linux-sockets>/

[5] <https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf>

[6] <https://www.git-tower.com/blog/git-cheat-sheet/>

[7] <https://tools.ietf.org/html/rfc1034>

[8] <http://tools.ietf.org/html/rfc3986>

[9] <http://stackoverflow.com/questions/4658409/whats-the-most-efficient-file-logging-in-a-server-written-in-c>