

In today's modern world computing is becoming more fundamental to everything we do, and with that comes a focus on creating highly available and distributed services. With high availability services, many computers work together to make a service achieve the greatest uptime possible, this includes being fault-tolerant to various outage scenarios. Consensus algorithms are a foundational part of building these systems.

Raft Consensus

ITC303 – Assessment Item 2

Joshua Cameron;Sean Matkovich

Contents

Project Vision.....	4
Introduction.....	4
Positioning.....	5
Problem Statement	5
Product Position Statement	5
Stakeholder Descriptions.....	6
Stakeholder Summary	6
User Environment.....	7
Example Scenario 1: The On-Prem Password Manager company	7
Example Scenario 2: The database company	7
Example Scenario 3: The Video Game company	7
Product Overview	8
Needs and Features (Functional requirements).....	8
Other Product Requirements (Non-functional requirements).....	9
Business Justification for Functional and Non-functional Requirements	10
Requirement Model	12
Use Case Model	12
Short Use Case Descriptions - In Table Form.....	13
Actor Key	13
Domain model	15
Discussion regarding domain model	16
Does the analysis of non-functional requirements demonstrate an ability to identify, quantify, prioritise, and communicate required system qualities? (LO2).....	16
Architecture Proposal.....	17
Purpose.....	17
Architectural Goals and Philosophy	17
Assumptions and Dependencies	18
Architecturally Significant Requirements.....	19
Decisions, Constraints, and Justifications.....	20
Architectural Mechanisms.....	22
Architectural Mechanism 1 - Distributed Consistent Log.....	22
Architectural Mechanism 2 - Fault Tolerance	22
Architectural Mechanism 3 - Network Communication.....	22
Architectural Mechanism 4 - Security	22
Architectural Mechanism 5 - API Based Integration	22
Architectural Mechanism 6 - High Quality Code	22

Layers or Architectural Framework	23
Architectural Views	25
Use case view	25
Logical view.....	25
Physical view.....	25
Technical Competency Demonstrator	27
Class diagram.....	28
Example TCD Program	29
Example TCD Program output	30
Risk List	32
Generic Project Risk List	32
Project Specific Risk List.....	34
Master Test Plan.....	36
Testing Strategy	36
Test Driven Development (TDD).....	36
Unit testing(UT)	36
Integration Testing (IT)	36
Code review(CR)	36
Prototype(PT)	36
Implementation testing (IMPT)	36
Design validation(DV)	36
Tests to be Conducted.....	37
Project Plan.....	40
Introduction.....	40
Project practices and measurements.....	40
Deployment	40
Project milestones and objectives.....	41
Inception Phase assessment.....	43
Goals of Inception Phase and How They're Being Achieved	43
Gather accurate system requirements.....	43
Analyse functional and non-functional requirements.....	43
Produce diagrams expressing system functionality	43
Propose system's architecture and highlighting architecturally significant requirements.....	43
Produce a minimal proof of concept for demonstrating technical competency	43
Determine and analyse project risks	43
Provide clear plan for ensuring quality of project through testing	43
Produce a timeline for various milestones throughout the project.....	44

Justification of Technical Capability to Achieve Those Goals	44
Issues encountered.....	44
Status of Any Important Risks	45
Current progress of project	45

Project Vision

Introduction

In today's modern world computing is becoming more fundamental to everything we do, and with that comes a focus on creating highly available and distributed services. With high availability services, many computers work together to make a service achieve the greatest uptime possible, this includes being fault-tolerant to various outage scenarios. Consensus algorithms are a foundational part of building these systems.

Consensus algorithm work by ensuring all nodes in a cluster are in agreeance on the current state of the service and continues uptime of the service during node failure or network link failures between the nodes.

The vision for this project is to create and publish a fully featured implementation of a Consensus Algorithm as an open source code library which would allow for application developers to implement consensus/fault-tolerance into their services as easily as possible.

The benefit to an implementing company could be twofold; firstly, solving the problem of increasing uptime of a critical service, and secondly by taking advantage of consensus distributed consistent log feature they can maintain service uptime whilst offloading server computation to the end user devices, directly saving hosting costs for the company.

Create a fully featured open source code library to allow the most amount of developers access to adding consensus features as easily as possible to their projects

Positioning

Problem Statement

Consumer driven requirements for always available services has pushed companies to eliminate any downtime from their offerings. These downtimes impact user experience as well as trust in their platform.

When consumers have poor experiences and lose trust in a service they may consider alternatives solutions for the needs. This impacts the growth of a company, as well as their ability to extract the financial benefits of scale from having a large user base.

Companies which minimise or eliminate any impact from downtime for their end users may better compete in the marketplace for their services.

Product Position Statement

Our proposed project targets developers who are responsible for implementing services which require high uptime. This open source library aims to fill the market position of a consensus algorithm which focuses on ease of integration and which is written in the language the developer is most likely to be using.

The benefits to a company would include reducing overall product development time, reducing the total cost of developing a highly available service, and by reducing the need/costs for consensus specialists to implement the desired feature. Reducing these costs will also lower the barrier for entry for companies which previously could not have reasonably implemented this.

Driving down development overhead costs can not only improve a company's competitiveness in their market and allow them to capture the cost of unnecessary expenditure, the reduction of development time can also reduce the financial risk that companies accept during their projects which implement this technical feature.

Stakeholder Descriptions

Stakeholder Summary

Name	Description	Responsibility
Project Team	Our team; functioning as developers, as well as project sponsors and management	<ul style="list-style-type: none">• Ensuring system is maintainable• Ensuring market demand• Ensuring features are working as desired• Ensuring system is reasonably secure• Monitoring of project development progressInvesting their time into development
Team Leader/CTO	The individual who is responsible for selecting and assessing technologies that may be beneficial for inclusion into their services	<ul style="list-style-type: none">• Assess the amount of time and cost of implementation of this library as their feature• Confirmation of the benefits of the library matching the business requirements• Comparison of alternative solutions for their requirements• Proposing library as solution for their requirements to project sponsor• Managing the project of implementation
Project Sponsor	The individual who has requested the inclusion of the feature into their service, and provide the financing for implementation	<ul style="list-style-type: none">• Assess the proposal by the Team Leader/CTO in the context of cost to benefit to the company and realization of benefits• Responsible to the business for the success of the project and reasonable use of expenditure• Responsible for sponsoring the resolution of the business case
Developer - Consensus Specialist	The individual in the Project Team whose responsibility is to learn	<ul style="list-style-type: none">• Responsible for the intimate knowledge of the library• Responsible for technical practical knowledge to understand consensus algorithm and implementation• Liaising as the specialist consult inside the Project Team for practical implementation of the library into the service
Developer	The individuals responsible for the integration of the library into their service	<ul style="list-style-type: none">• Understanding their existing service such that they can liaise with the Consensus Specialist to plan how to implement• Implementing the feature into their service
End user	The end users who is pressuring the company for highly available service	<ul style="list-style-type: none">• Not applicable

User Environment

As the reliance on always available services grows, there becomes more companies and individuals who rely on their services having high uptime. These companies have many, sometimes thousands of users relying on the availability of their services to perform important roles in their lives.

Some examples of the users would be:

- Companies who offload server session to customers, such as multiplayer games.
- Companies offer As A Service (aaS) products needing high uptime
- Companies who want to add high reliability to their services
- Database programmers

Example Scenario 1: The On-Prem Password Manager company

This company produces a Linux webserver that their business clients run on premises, this server is a Password Manager for all their employee's company logins details and supplier logins in. Their biggest customer, a 1000 seat enterprise, has complained of the risk of reliability in this password web server and would like to mitigate the risk of failure of the server as it's stop hundreds of people from conducting their daily duties. They suggested that they were interested in a more reliable solution which mirrors itself between sites to maintain its uptime, is transaction safe, and importantly never loses data or returns incorrect results.

Example Scenario 2: The database company

Consensus algorithms are complicated, and databases are a critical part of almost all web services. In order to bring consensus to the masses, a database company is looking to implement clustering at the transaction log level of their database, so that web servers can failover to other nodes in the cluster if one falls over. The job of developers greatly simplified as they only need to write heartbeat checks to their database, and upon failure select another node in the cluster.

Example Scenario 3: The Video Game company

There is a video game company who runs a popular multiplayer turn-based strategy mobile game, many thousands of players are playing at any one time. Analysing their company costs, hosting costs accounts for around 40% of yearly expenditure, which is second only to staff costs. Each game server hosts up to 10 players at a time, meaning the company needs scale up to run hundreds and potentially thousands of these servers during peak playing times. The idea came up about offloading running the server onto one of the players playing in the server, however it was shot down due to the ability for players to leave at any moment causing bad experiences for the other players in the servers. They've now had the bright idea to employ a consensus algorithm which synchronises server state among their players, so if someone leaves the game server starts running on another player. This will dramatically cut hosting costs, while not impacting uptime for the players.

Product Overview

Needs and Features (Functional requirements)

Priority scale is between 1 and 12, with 1 being the most important

Need	Priority	Features	Planned Release
Consensus between distributed systems	1	Replicated log, with consensus algorithm	Proto.
Fault tolerant distributed service	2	Consensus algorithm allows for a fault tolerant distributed system	Proto.
Improved reliability of existing service	3	System is fault tolerance, so it will improve reliability	Proto.
Complete proven reliability	4	Based on proven algorithm	Proto.
Minimal additional surface area for failure	5	Complete coverage unit testing	Version 1.0
Cross Platform	6	Targeting .NET standard framework	Proto.
Mitigate project abandonment	7	Licensing allow for profit	Proto.
Minimal overhead/impact to service performance	8	Equivalent to leading consensus algorithm, Paxos in performance	Proto.
Minimal resource usage	9	Consensus Log compaction	Proto.
Ability to pick ideal leader	10	Fitness considered in selection of next leader (new leader calls for election of ideal leader)	Version 1.0
Warm nodes	11	Tracks log but doesn't vote	Version 1.0
Upgrade path	12	Versioning built in, backwards compatibility minor releases and single major	Final

Other Product Requirements (Non-functional requirements)

Priority scale is between 1 and 14, with 1 being the most important

Requirements	Solution	Priority	Planned Release
Reliability	1. Full coverage unit testing	1	1. Final
Usability	1. Designed to be as simple as possible to integrate. 2. Released as Nuget package	2	1. Proto. 2. Final
Documentation	1. Full coverage documentation for algorithm and API	3	1. Final
Quality	1. Full coverage unit testing 2. Strict adherence to style guide	4	1. Final 2. Proto.
Performance	1. Matches Paxos in performance of consensus 2. Own thread with ASYNC/non-blocking operations 3. Performance analysis	5	1. Proto. 2. Proto. 3. Version 1.0
Compatibility	1. Written in .NET the second most popular language. 2. Minimal dependencies. 3. Written in .NET standard, cross platform 4. Agnostic networking option 5. Designed to be as simple as possible to port languages	6	1. Proto. 2. Proto. 3. Proto. 4. Version 1.0 5. Proto.
Availability	1. Can be run between servers locally or across Internet	7	1. Proto.
Security	1. Network level authentication	8	1. Proto.
Privacy	1. Security measures to join cluster	9	1. Proto.
Scalability	1. Dynamic cluster membership, horizontal scaling	10	1. Version 1.0
Testability	1. Open source code, unit tests provided	11	1. Final
Extendability	1. Open source code	12	1. Final
Auditability	1. Open source code 2. Logging	13	1. Final 2. Proto.
Troubleshooting	2. Verbose logging	14	2. Proto.

Business Justification for Functional and Non-functional Requirements

First and foremost, the most important feature of this project is increasing the reliability of a User Application Service by maintaining a replicated log; hence the top 5 functional requirements link directly to this. Not only does the desired system require to spread a replicated log amongst nodes, it also needs to be fault tolerant in the case of failure of a minority of nodes and manage the start/stop of a UAS in cases where only one instance should run. An unconventional part of reliability is usability, therefore there is a focus on the design of the usability of the library to ideally eliminate issues to reliability caused during implementation. To reinforce usability for the product, there will also be complete documentation available.

Companies looking to run this software may look to do so on any operating systems (e.g. Windows, Linux and mobile), so it's important that the code is written in a way that allows for this portability. To implement this, the project will be written in the .NET Standard Framework, which allows portability over all platforms .NET is able to run on.

Part of the benefit of releasing all of the code for this project as open source is that the business risk of project abandonment by ourselves could be picked up by the client, or even a potential open source community. This would enable security updates, bug fixes, and new features to continually be added to the project; and the licensing the project is released under allows for companies to profit from the code.

A fact of implementing a consensus under the hood of a service is that there will always be some overhead; the consensus algorithm being implemented has the same performance as the more popular/complicated and error prone Paxos algorithm. So, using Raft gives us all of the benefits, without the understandability/usability downsides of Paxos. This minimal performance overhead allows companies implementing the library to not suffer from unreasonable latency overhead while waiting for servers to reach consensus, which would've caused a direct impact to their user experience. With careful design and the right conditions, it's even be possible to for this library to be used in a real time 30-60 tick game.

When only a single instance of the User Application Server (UAS) is required, such as a video game server, it may be important to the user experience that most ideal node in terms of latency, or hardware performance runs the UAS. Having the ability for the nodes to pick the most ideal leader for consensus based on performance enable this improved quality of service.

When a node fails or leaves the cluster, this impacts the redundancy/reliability of the cluster as a whole. To bring back a safe state, an additional node is brought in to bring the cluster back up to the desired number of nodes. However, if the cluster service has been running for a long time the replicated log may be so large that it takes several minutes to get another node ready. To resolve this issue, a requirement for allowing Warm nodes to be updated semi-regularly with committed entries to save time later is important. This directly improves reliability and user experience upon additional node failures/departures.

As companies require always on services, an important part is that the consensus library also include backwards compatibility, so allow for rolling software updates through the nodes in a cluster. This allows for the updating of UAS without causing service downtime to the service, avoiding any customer impact.

Security and privacy are also both highly important since these nodes will connect across the network, in a modern-day service there is an expectation of a duty of care to users for the protection of their potentially sensitive information or data.

Part of the easy of code maintainability is facilitating the recreation of any possible bugs in the library; as such, to enable the troubleshooting of this the library will output at multiple debugging levels based on desired configuration.

Implementing scalability in the cluster by enabling the dynamic adding and removing of nodes allows the integration of this library with the horizontal scalability necessary for modern infrastructure design.

Justification of ordering of requirements

The library being quite atomic in nature (i.e. it either works or it doesn't when ready for integration) has impacted the ordering of the requirements priorities. They have primarily been focused on functionality first before the extra "nice to haves", so that the product can get out the door as fast as possible, while never compromising on the reliability. For example, Security is absolutely necessary in a production release of this library as it may be conducted over the Internet (and other publicly known vulnerabilities are exploited in similar software, discussed elsewhere in this document), however it's technically not required to have a running consensus algorithm.

Requirement Model

Use Case Model

After investigation of the requirements the following [Use Case diagram](#) has been developed to outlining the initial set of interactions required

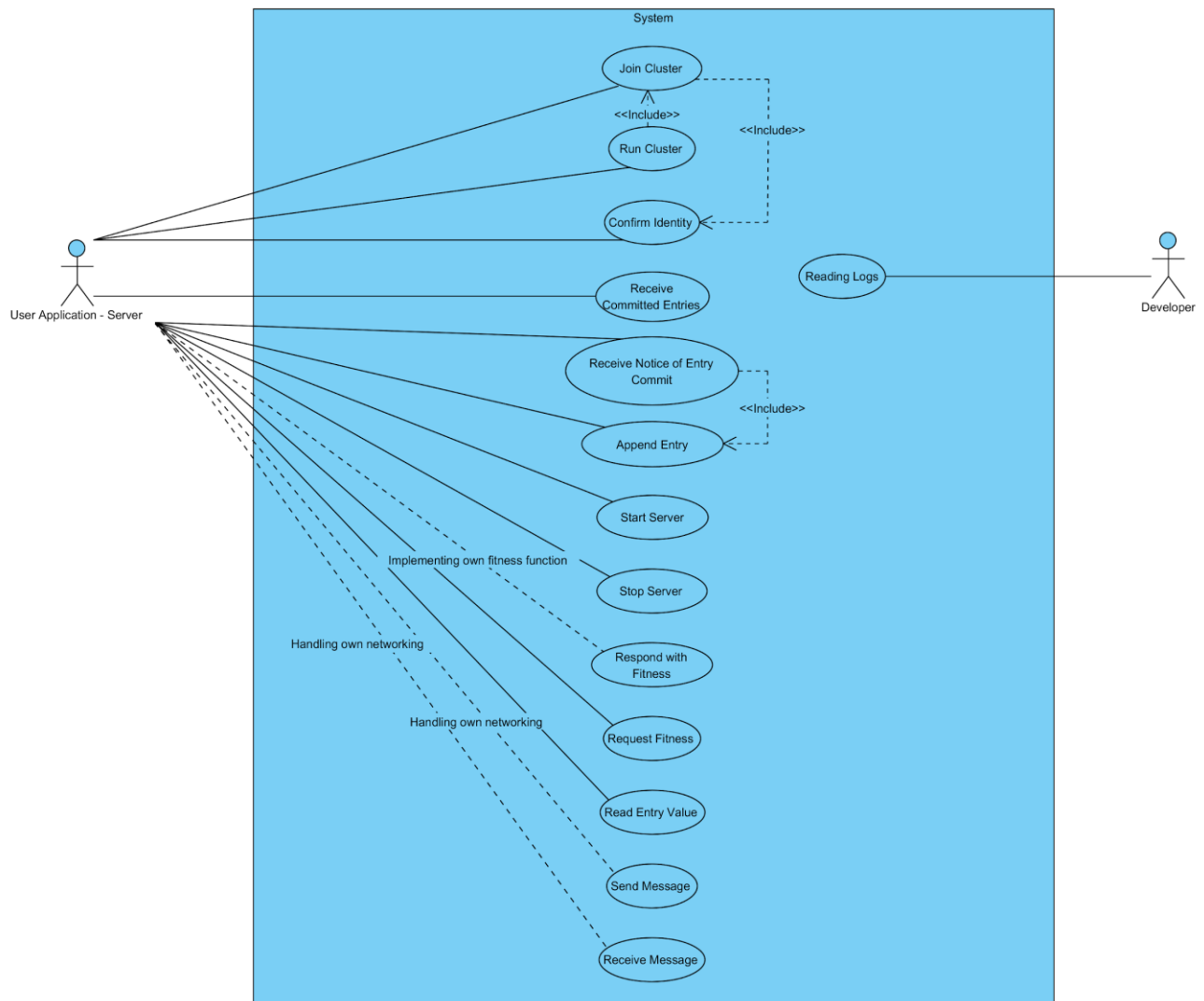


Figure 1

Short Use Case Descriptions - In Table Form

Actor Key

UAS = User Application Server

Dev = Developer

Use Case Name	Precondition	Actor	Need	Do Something	Basic Intent/Goal
Join Cluster	Has information about the cluster	UAS	Join the cluster	Join Cluster	Become a member of the cluster
Run Cluster	Join Cluster use case has run	UAS	Participate in cluster actions	Run Cluster	Start participating in the cluster
Receive Committed Entries	N/A	UAS	Bring itself up to date	Receive Committed Entries	Can maintain consensus
Append Entry	N/A	UAS	To add a message into consensus	Append Entry	Message gets committed into the consensus
Receive Notice of Entry Commit	N/A	UAS	To know if the message they requested to be committed, was committed	Receive Notice of Entry Commit	So that they may update the running UAS state
Receive Start Server	The current node should be running a UAS	UAS	To be running a UAS when required	Receive Start Server	They may go ahead and start up their UAS to provide services
Receive Stop Server	The current node should stop running their UAS	UAS	To stop running a UAS when required	Receive Stop Server	They stop acting as a UAS and providing service

Respond with Server Fitness	UAS implementing own fitness function	UAS	Wants to respond with its fitness for being the UAS	Respond with Server Fitness	The most fit node may be identified to run the UAS
Read Entry Value	N/A	UAS	Read log entries that have been committed	Read Entry Value	Can reference the committed data
Send Network Message	Offloading node communication to UAS	UAS	Send a message offloaded to it from the underlying consensus algorithm	Send Network Message	The underlying consensus algorithm can still communicate without relying on it's own networking stack
Receive Network Message	Offloading node communication to UAS	UAS	Forward a message offloaded onto it's networking stack to the underlying consensus algorithm	Receive Network Message	The underlying consensus algorithm can still communicate without relying on it's own networking stack
Confirm Identity	N/A	UAS	Confirm the identity a new node communicating for the first time	Confirm Identity	Establish security through trust, and ensure they cannot be man in the middled
Read Logs	N/A	Dev	Understand what the underlying consensus algorithm is doing, perhaps tracking a bug	Read Logs	Understand what the underlying consensus algorithm is doing

Domain model

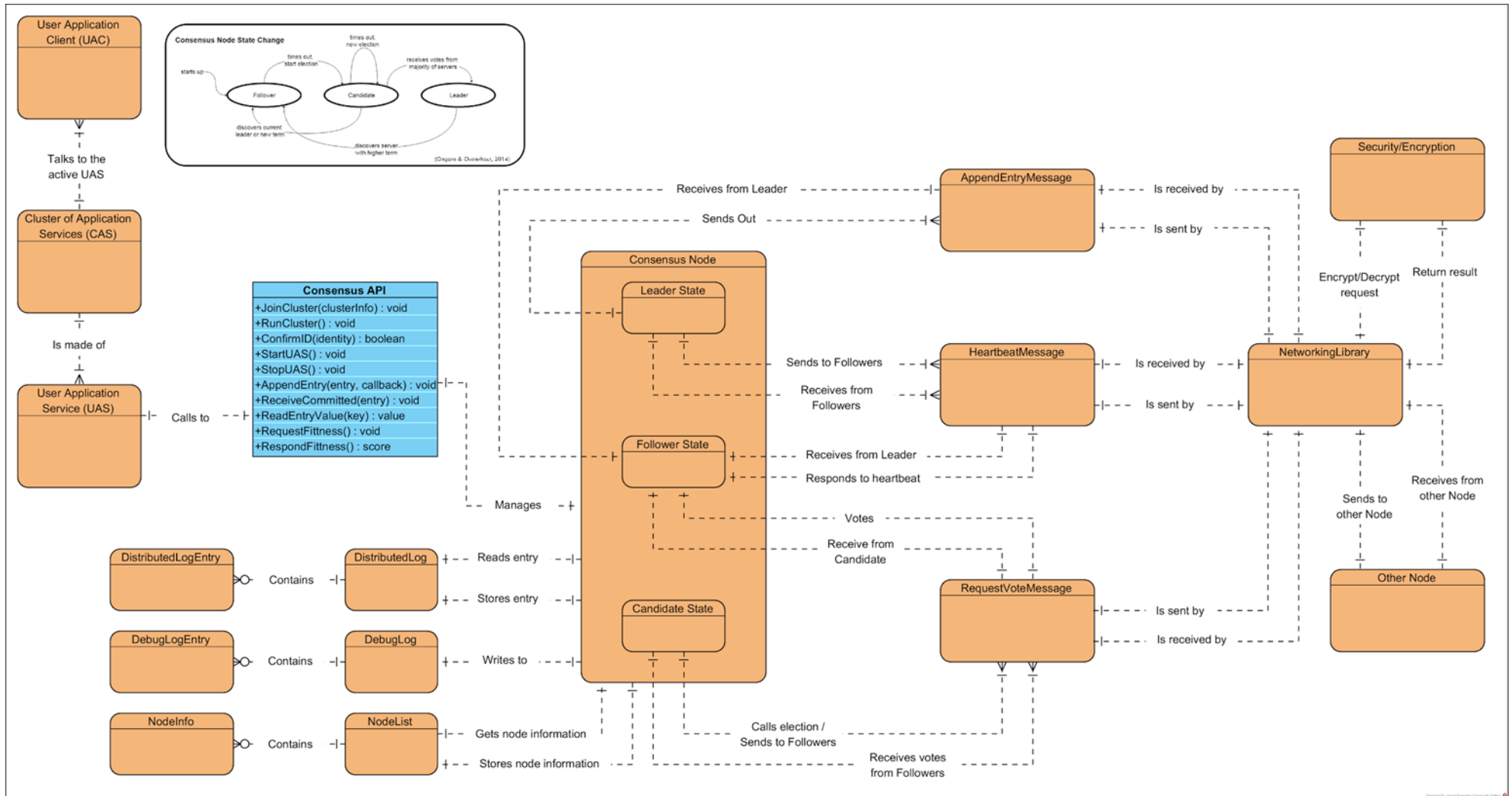


Figure 2 Domain Model

Discussion regarding domain model

We've found this is easiest to explain using an example of a distributed multiplayer game running a which is in a server/client setup. In this example the server is also running on one of the clients. In that example, the UAC would be the game client which the players would be interacting with, and UAS would be the game server all the game clients are talking back to. Our library would be integrated into the UAS.

Reading from the top left of the diagram it's shown how the UAC (i.e. game clients) talk to the active UAS (i.e. game server) in the Cluster Application Servers(CAS). Then it's shown how the CAS is made up of UASs. It's the library which links these UACs together into a cluster, and the UACs simply talk to whomever is the 'leader' UAS in the CAS. The UAC can simple just try each of the IPs in the CAS until they find the leader who responds, this cuts down on the complexity in the UAC software.

Next, the UAS uses the Consensus API to talk to its internal Consensus Node. This consensus node is the part that handles cluster operations, and part of that is instructing the UAS to let it know when it should and shouldn't be running the active game server. The nodes all communicate to each other through messages which are sent and encrypted by the Networking Library. Consensus between the nodes is through a shared distributed log (Distributed Log) which the active UAS commits entries into, and the follower nodes are updated with. Each node stores information about the other nodes which are in the cluster with it. And each node does verbose logging for troubleshooting reasons.

Does the analysis of non-functional requirements demonstrate an ability to identify, quantify, prioritise, and communicate required system qualities? (LO2)

This is the part of the document where the above would be analysed, however after discussing with Jim, he has confirmed that this is covered by our "Business Justification for Functional and Non-functional Requirements" section in the Project Vision above.

Architecture Proposal

Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

Architectural Goals and Philosophy

The most important goal of this project is increasing the reliability of a User Application Service; hence all the high priority non-functional requirements link directly to this and it's also the primary architectural design goal. By directly designing for reliability we incorporate it foundationally into our final product; part of this reliability focused design is a large amount of time initially invested into accurately and exhaustively planning out the project to meet all the functional and non-functional requirements.

An underlying requirement of reliability at the code level is rigorous full suite testing and a focus of strong adherence to code quality/style guidelines. This reduces errors, as well as improves maintainability.

Key to the implementation of reliability is usability, therefore there is also a focus on the design of the usability of the library. To reinforce usability for the product, there will also be complete documentation available to ease integration.

The final library needs to maintain functionality in a system with up to a minority of its nodes failed and will be put into potentially hostile environments in terms up frequently failing nodes.

Assumptions and Dependencies

Assumptions:

- Market need for consensus library, specifically in .NET
- Performance and latency overhead of consensus is manageable without negatively impacting customer experience
- Requirement for security and the impact of that overhead is acceptable
- The limitation of compatibility in cross-platform implementation for mobile is limited to Xamarin is acceptable
- Dynamic cluster membership, including increase and decreases cluster size/scalability is an important feature to the developers
- Designing this library as free and open source software will not detrimentally inhibit company adoption and that auditability is important
- That developers would utilise the debug logging features of the library to diagnose their issues
- That developers would report back any bugs or issues with library back so they may be resolved

Dependencies:

- Developer implementing the programming required to read/write User Application Service state information into the consensus log
- Developer is able to start a User Application Service off of solely the information contained within the consensus log
- Developer implementing or having load balancing logic in any User Application Client accessing the active User Application Service. This may simply include failover at the IP level between the Cluster of Application Servers.
- Developer will either manually specify node networking addresses or enable a lobby system in their own User Application Service to supply them as required to the library for the starting of the cluster

Architecturally Significant Requirements

- The system must be able to communicate over a network
- The system must be able to maintain consensus between distributed network nodes
- The system must be able to maintain a consistent replicated log
- The system must be able to survive failures up to a minority of its nodes
- The system must be as easy as possible to implement into existing projects
- The system must be as reliable as possible

Decisions, Constraints, and Justifications

Decision	Justification
C# language using .NET framework	C# is the second most popular language of 2017 , so using it will make our library available to a wide range of developers
C# over Java	The more mature library integration framework of C# (Nuget) over Java outweighs the 5 percentage points of difference in their popularity
.NET Standard	.NET Standard is a subset of functions in the .NET framework which is. Although this makes development more difficult due to the reduction of prebuilt libraries
Bitbucket	Developer familiarity and the ability to have repos private during development
Full suite unit testing	As reliability is the most important functional requirement, we want to ensure not only that the code is working as intended, but also that changes to the code impact functionality
Style guide	Part of releasing open source code while easing development and maintenance of it by disparate developers is ensuring strict adherence to coding standards
Code review	With reliability such an important non-functional requirement, systematic independent party examination of code is put in place to find bugs as early as possible, as well as maintain adherence to style guide.
Documentation	To enhance usability, thorough clear and consistent documentation
Implementing own network library	The existing functionality did not exist in an available library, or existing libraries were excessively bloated. With networking foundation to the node communication, controlling it was a practical decision.
Test driven development	As we've focused on exhaustive design for reliability reasons, TDD is a complementary fit due to its ability to ensure that all of the designed requirements are implemented
Visual Studio IDE	Industry standard .NET IDE, developer familiarity, ease of unit testing
Security	Always-on security was decided due to the nodes being designed to cluster across the public internet, as well as avoiding common issue of leakage or exploits such as the recent etcd and memcached .

Constraint	Justification
Consistent public API	Maintain ease of implementation for developers, regardless of underlying implementation changes
Minimalistic approach	When deciding on additional features or functionality, careful consideration must be taken to ensure a minimalist design is adhered to such as that the implementation of the software maintains being as simple as possible. This is due to reliability reasons, by aiming to avoid potential issues caused during implementation by over complicated or extensive options for developers.
Usability first approach	When deciding on approaches which the developers interacts with (such as the API), the most important constraint is usability and must be considered over unessential functionality. This usability first approach increases reliability, which has been justified multiple times above.
UDP	Although the guaranteed ordering and acknowledgment of packets in TCP would be ideal for a consensus algorithm, the additional round trip times required for this increase latency overhead unreasonably, and as the consensus algorithm takes cases of consistency and packet loss, UDP was chosen as the network protocol so we did not need to incur this overhead.
Universally standard data structure for distributed log	A key-value store (also known as “dictionary” in .NET) was chosen for it’s speed, reliability and non-complex implementation for developers.

Architectural Mechanisms

Architectural Mechanism 1 - Distributed Consistent Log

This is the component used by the User Application Service (UAS) to commit their running service's data into a distributed log amongst the consensus nodes. This is the foundational feature which allows other UASs to start up in the event of a running UAS failure.

Architectural Mechanism 2 - Fault Tolerance

This is the features which allows an increase in availability of a given service, it does this though enabling the failing over of a User Application Service to another available node in the cluster.

Architectural Mechanism 3 - Network Communication

This is the functionality which allows the distributed consensus nodes to communicate with each other. It will be based on the "fire-and-forget" / "connectionless" UDP protocol to reduce latency, while leaving the overhead of handling packet loss to the consensus algorithm.

Architectural Mechanism 4 - Security

This feature provides identification through public key cryptography, confidentiality through encryption, and integrity through HMAC, for all network communications.

Architectural Mechanism 5 - API Based Integration

The User Application Server (UAS) communicates with its consensus node through the use of a .NET class library. This single interface focused on usability is how the UAS communicates to the consensus algorithm.

Architectural Mechanism 6 - High Quality Code

There will be a dedicated and focused effort on ensuring the highest possible quality of code as part of this project. As this code is to be ideally used in ensuring UAS high availability, it's focus on quality must be paramount.

Layers or Architectural Framework

From the below diagram, it's shown where each Architectural Mechanism exists in the design. These architectural mechanisms in the diagram are numbered to reflect the assigned numbers previously in Architectural Mechanisms above.

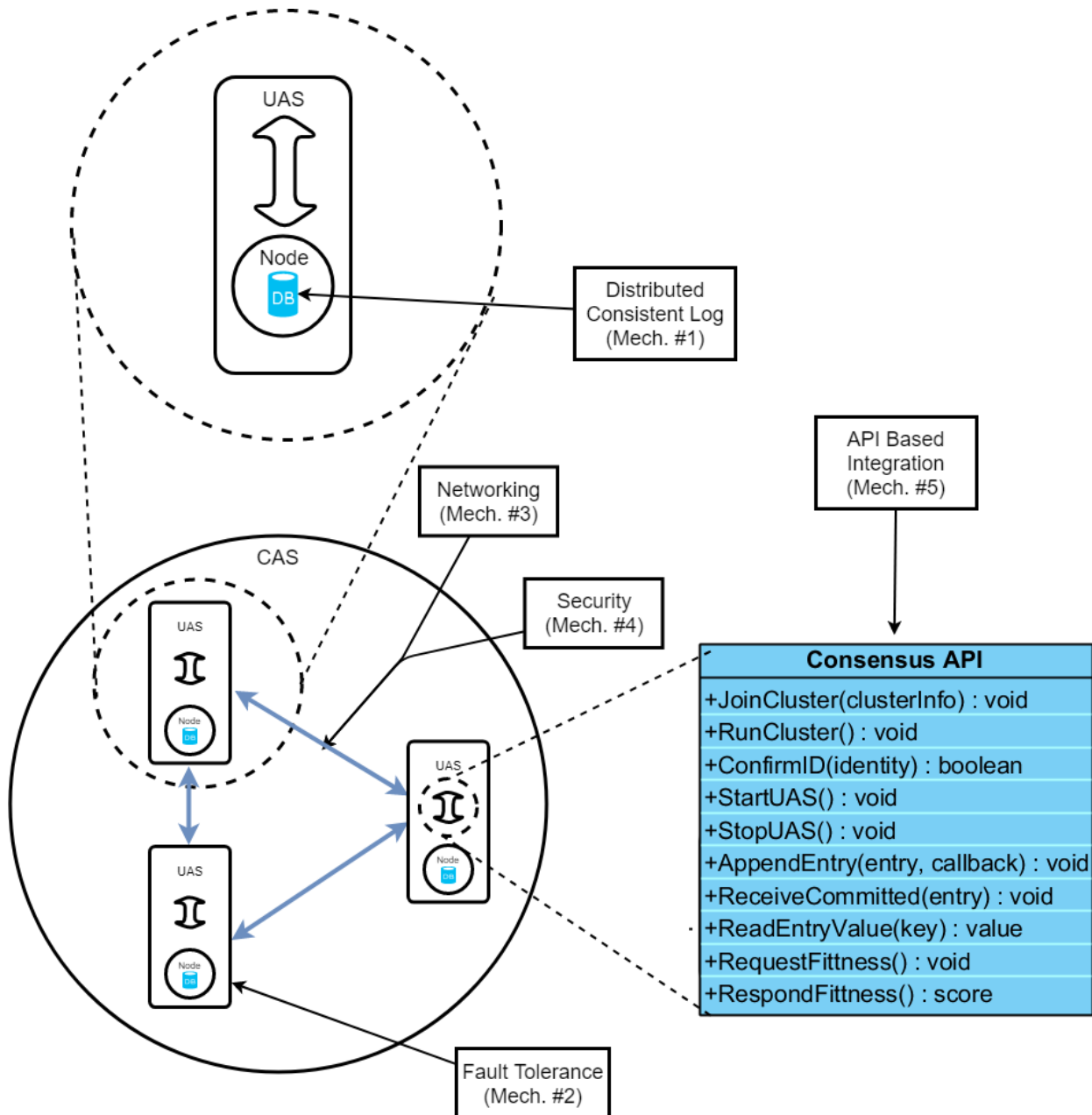


Figure 3

Where each NFRs is addressed in the architectural framework

NFR	Related Architectural Framework Mechanism
Reliability	Fault tolerance
Usability	API Integration
Documentation	API Integration, however handled externally to the system
Quality	High Quality Code
Performance	N/A, this is handled at the algorithm and code level
Compatibility	N/A, this is handled by choice of code language
Availability	Networking
Scalability	Distributed Consistent Log, handled in consensus algorithm
Security	Networking
Privacy	Networking
Testability	High Quality Code
Extendability	High Quality Code
Auditability	High Quality Code
Troubleshooting	High Quality Code

Architectural Views

Use case view

Please see Use Case Diagram

Logical view

Please see Domain Model

Physical view

The consensus library will be integrated into the User Application Service (UAS) and will be used to logically tie multiple UASs together into a Cluster of Application Services (CAS).

In the example case of a distributed network where UASs are offloaded from centralised company servers onto UACs there is only a logical difference between UAC and UAS, not physical.

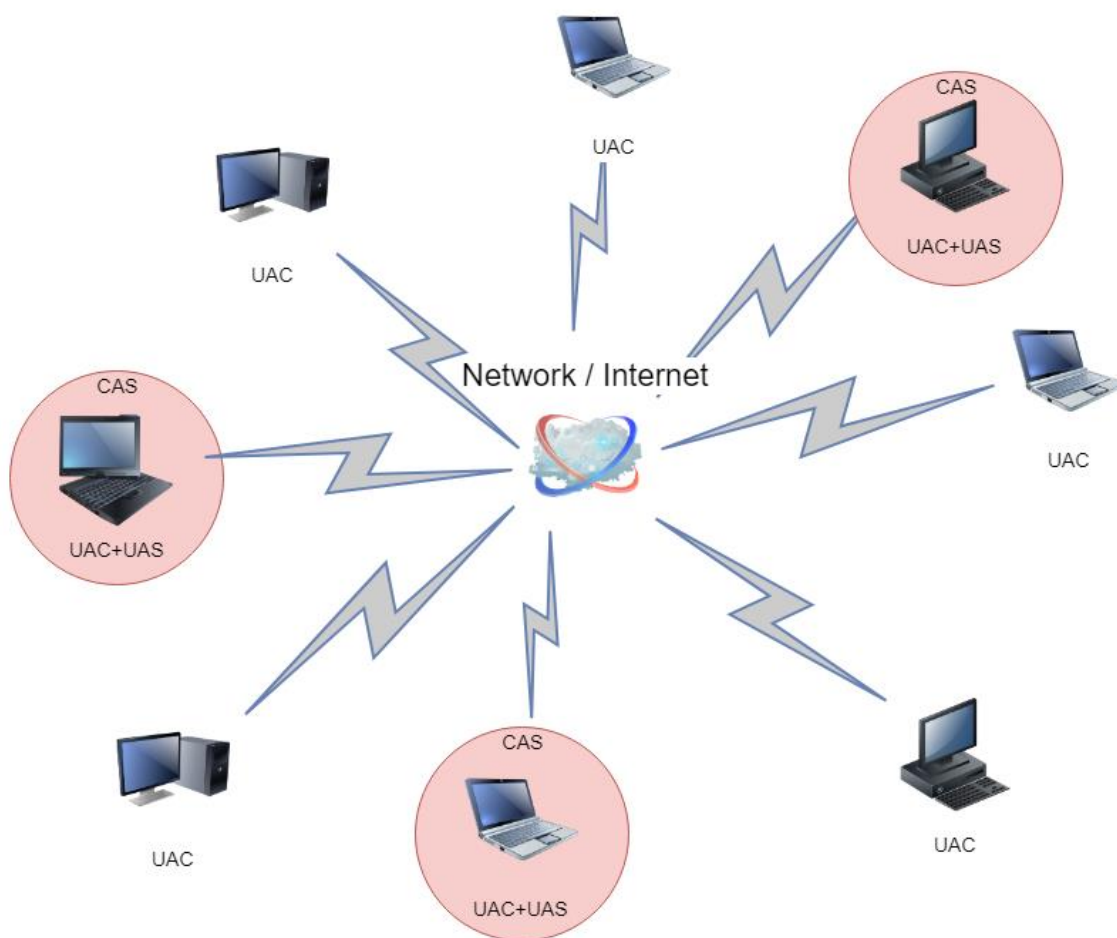


Figure 4

In the example case of dedicated linked servers which provide services for clients, there are multiple distinct servers which run the UASs forming a Cluster of Application Services (CAS). The User Application Clients communicate with the CAS through simple IP failover style.

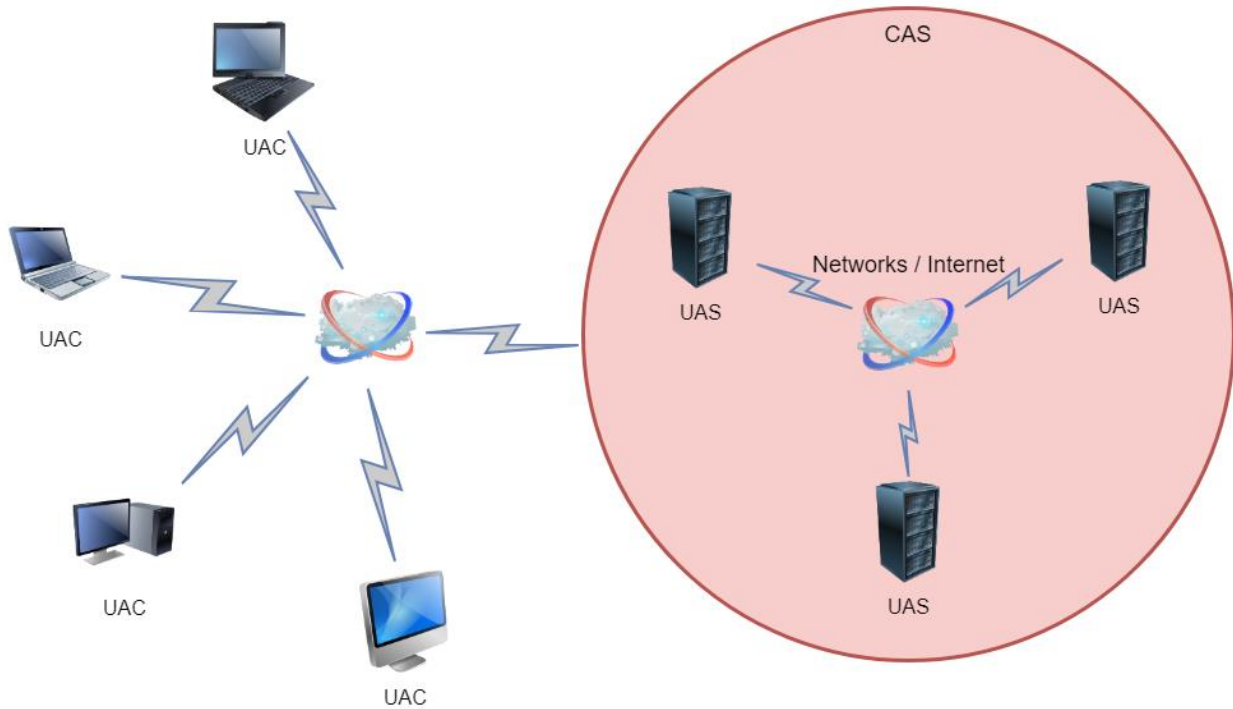


Figure 5

Technical Competency Demonstrator

[Link to the Technical Competency Demonstrator](#)

[Link to the networking code library \(RaftNetworking\)](#)

After analysing the project, the most fundamental technical process which we identified is the sending of messages between nodes. All functionality of distributed consensus is built upon the ability to send/receive messages from nodes in a timely fashion. After our initial findings of there not being a publicly available non-bloated networking library which suited our needs, we determined we needed to write our own. This is what we'll be doing for our Technical Competency Demonstration (TCD).

This decision was not taken lightly, as writing socket level programming in any language is fraught with error/difficulty and technical programming traps for new players. Fortunately, one of our team members has prior experience writing libraries like this, in C, C++, and the C# language as well.

When estimating performance bottlenecks in the raft consensus protocol running across distributed nodes, the largest factor which impacts consensus time of the network is the node's network latency from one another. Therefore, we chose UDP based networking over TCP, this was due to UDPs lack over acknowledgement message overhead for each message reducing overall message latency. The risk of running UDP however is that as the packets are connectionless, messages may be lost after transmission without the TCP resolving them in its host-to-host OSI layer (layer 4). This has been deemed not an issue for our project, as the raft consensus algorithm itself handles internally any packet/message loss issue.

Initially we thought that we'd be caught out with a maximum packet size issues, as with UDP networking over the internet you are only minimally guaranteed 576 byte messages due to historic networking device buffer sizes, and a likely modern packet size in the 1472 byte range. At such small packet sizes, we'd need to implement our own packet reassembly protocol into the messages to support our larger message requirements, however upon testing we found that Microsoft's socket wrapper `UdpClient` was already implementing this and allowed us to send messages with a total length of 65KB, much more than we'll reasonably require.

During one of our oversight meetings with Jim, he confirmed that our team's technical competency demonstration would have to have the following attributes:

- Serialization/deserialization of a message in JSON
- Sending the serialised message over the network, and deserializing it to write the output
- Single threaded

We decided to go above and beyond these specifications and implemented a multithreaded networking library consisting of 3 synchronized threads. We realised it wasn't worth writing code for a TCD to just throw it away after, and that we could take this time to write useful code now and then we'd be able to use it later in the project.

Class diagram

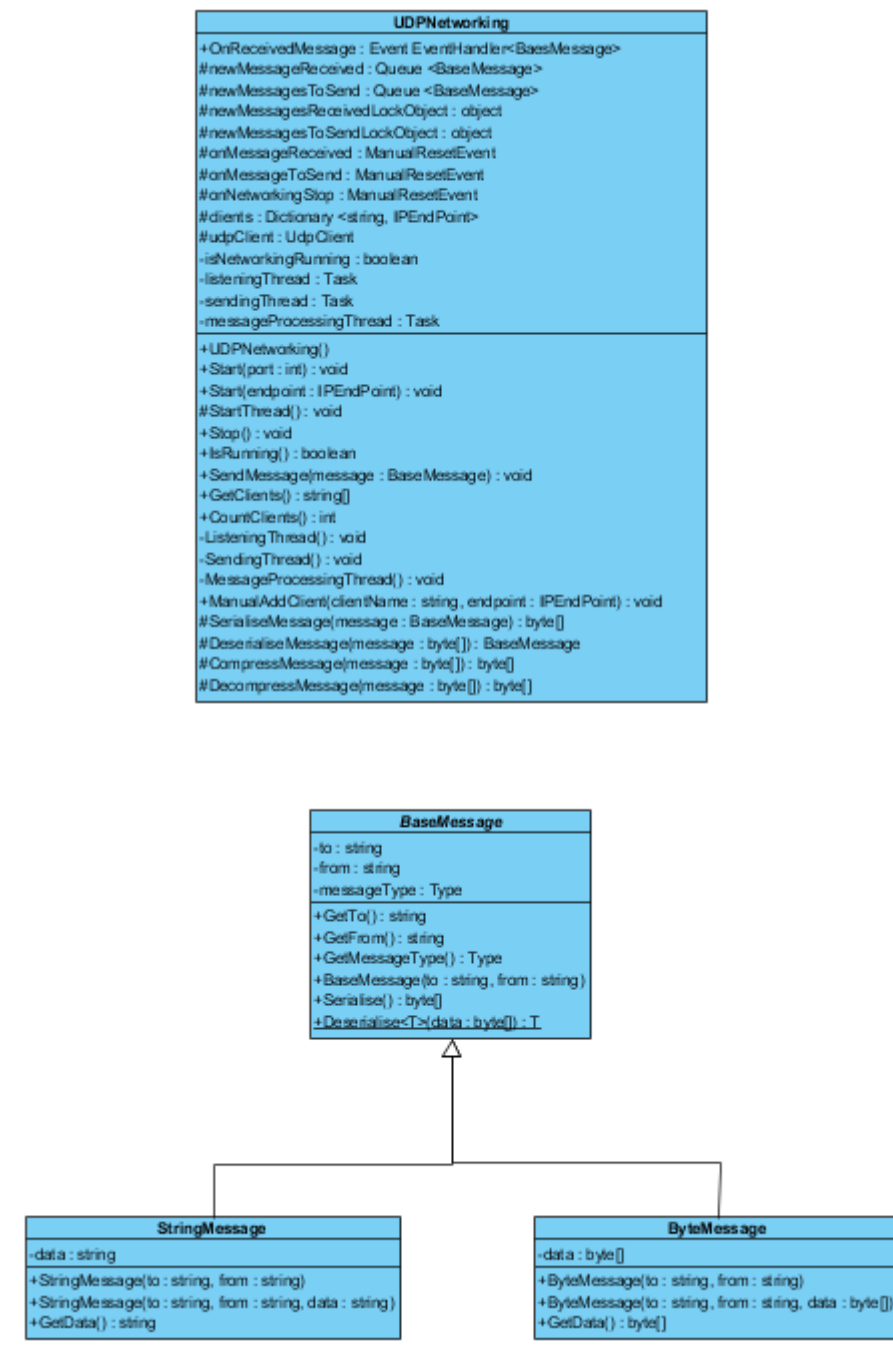


Figure 6 Class Model

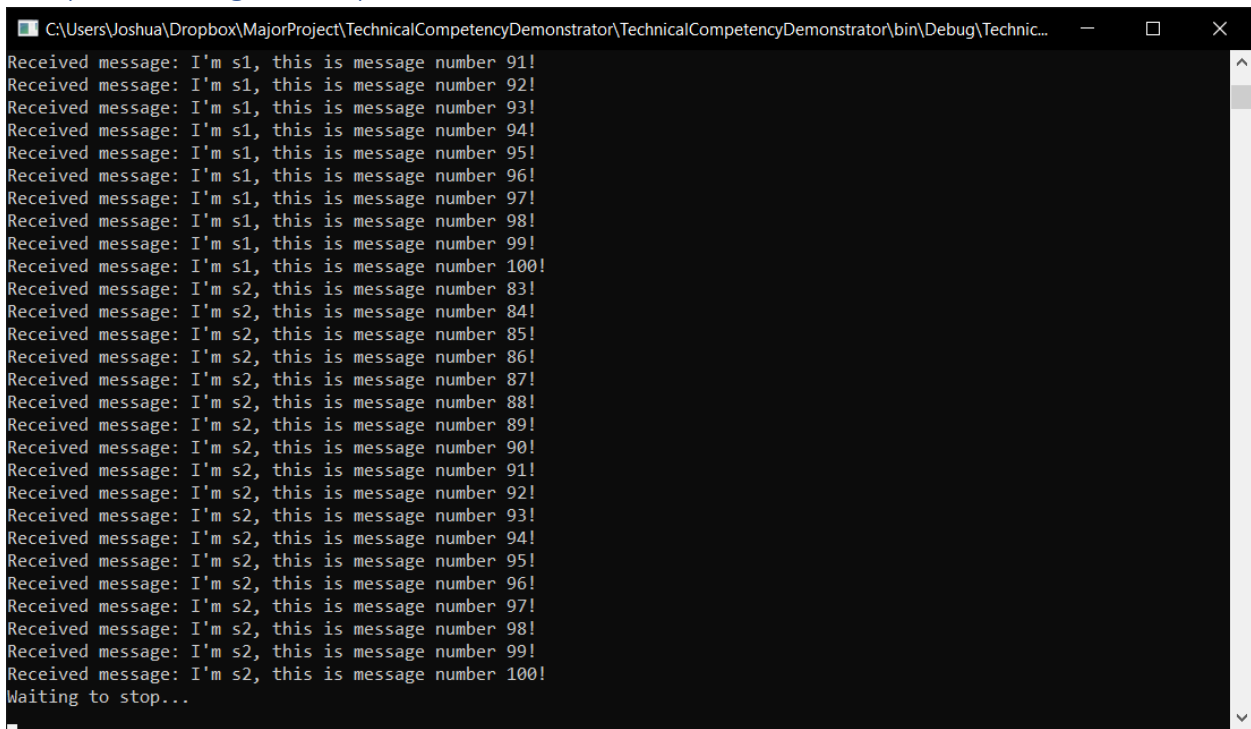
As you can see from the class diagram, the networking class itself is a single class which is interfaced with through its minimal public methods based on an interface, this class sends/receives messages derived from the **BaseMessage** class through the property of polymorphism. The library minimally only needs to send **ByteMessage** messages, however we've added **StringMessage** messages as well to simplify our example TCD program. In practise in the consensus library we'll be deriving our own messages based on our needs of message types.

Example TCD Program

```
8 namespace TechnicalCompetencyDemonstrator
9 {
10     class Program
11     {
12         static void Main(string[] args)
13         {
14             UDPNetworking networking1 = new UDPNetworking();
15             UDPNetworking networking2 = new UDPNetworking();
16             networking1.OnReceivedMessage += OnReceivedMessage;
17             networking2.OnReceivedMessage += OnReceivedMessage;
18
19             networking1.Start(4445);
20             networking2.Start(4446);
21
22             networking1.ManualAddClient("Localhost", new IPEndPoint(IPAddress.Parse("127.0.0.1"), 4446));
23             networking2.ManualAddClient("Localhost", new IPEndPoint(IPAddress.Parse("127.0.0.1"), 4445));
24
25             for (int i = 1; i <= 100; i++)
26             {
27                 StringMessage stringMessage1 = new StringMessage("Localhost", "Localhost", "I'm s1, this is message number " + i + "!");
28                 StringMessage stringMessage2 = new StringMessage("Localhost", "Localhost", "I'm s2, this is message number " + i + "!");
29                 networking1.SendMessage(stringMessage1);
30                 networking2.SendMessage(stringMessage2);
31             }
32
33             Thread.Sleep(1000);
34
35             Console.WriteLine("Waiting to stop...");
36             Console.ReadLine();
37
38             networking1.Stop();
39             networking2.Stop();
40         }
41
42         private static void OnReceivedMessage(object sender, BaseMessage e)
43         {
44             if (e is StringMessage)
45             {
46                 Console.WriteLine("Received message: " + ((StringMessage)e).Data);
47             }
48             else
49             {
50                 throw new Exception("Unknown message type");
51             }
52         }
53     }
54 }
55
```

Figure 7

Example TCD Program output



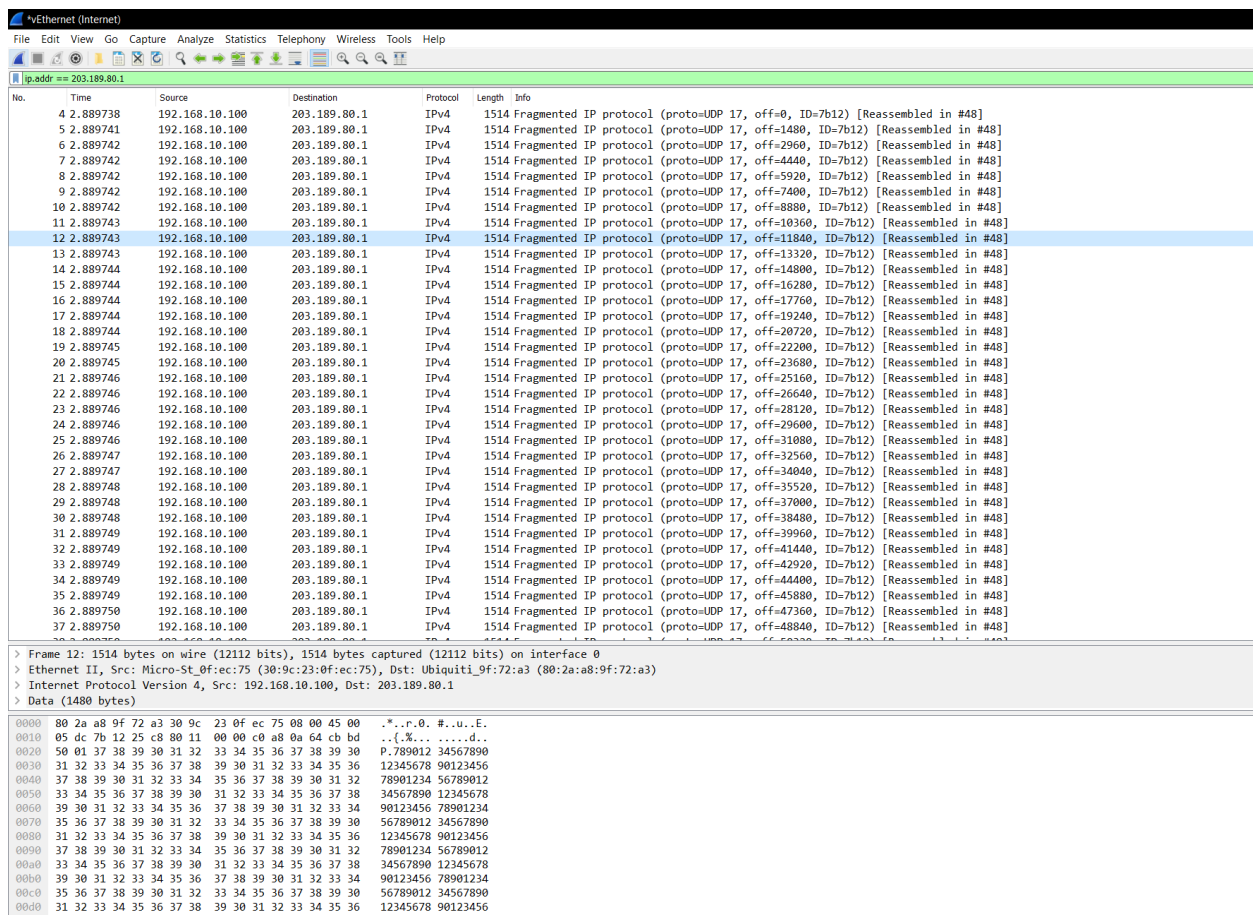
```
C:\Users\Joshua\Dropbox\MajorProject\TechnicalCompetencyDemonstrator\TechnicalCompetencyDemonstrator\bin\Debug\Technic...
Received message: I'm s1, this is message number 91!
Received message: I'm s1, this is message number 92!
Received message: I'm s1, this is message number 93!
Received message: I'm s1, this is message number 94!
Received message: I'm s1, this is message number 95!
Received message: I'm s1, this is message number 96!
Received message: I'm s1, this is message number 97!
Received message: I'm s1, this is message number 98!
Received message: I'm s1, this is message number 99!
Received message: I'm s1, this is message number 100!
Received message: I'm s2, this is message number 83!
Received message: I'm s2, this is message number 84!
Received message: I'm s2, this is message number 85!
Received message: I'm s2, this is message number 86!
Received message: I'm s2, this is message number 87!
Received message: I'm s2, this is message number 88!
Received message: I'm s2, this is message number 89!
Received message: I'm s2, this is message number 90!
Received message: I'm s2, this is message number 91!
Received message: I'm s2, this is message number 92!
Received message: I'm s2, this is message number 93!
Received message: I'm s2, this is message number 94!
Received message: I'm s2, this is message number 95!
Received message: I'm s2, this is message number 96!
Received message: I'm s2, this is message number 97!
Received message: I'm s2, this is message number 98!
Received message: I'm s2, this is message number 99!
Received message: I'm s2, this is message number 100!
Waiting to stop...
```

Figure 8

For usability/demonstration reasons, the one example program is used for displaying the two networking classes communicate. The TCD example program references the dynamic linked library (DLL) file compiled in the RaftNetworking project. It works by completing the following steps:

- Instantiation the `UDPNetworking` classes so they may be used for communication
- Registering an `OnReceivedMessage` functions for handling received message processing
- Starting the `UDPNetworking` classes listening on the designated ports, this starts all their multiple threads
- Manually registering the IP details of each client with one another so they're able to communicate
- Then there is a loop of sending 100 messages both ways between the two classes. When the `OnReceivedMessage` function is called, you can see from its console output the out-of-order properties of UDP messages and how they're also occurring asynchronously.
- Then we simply sleep to allow the previous asynchronous actions to perform before writing out our "Waiting to stop..." message, this is so the message displays at the bottom with its `readline` function
- After the user presses enter on their keyboard, the program then stops the networking classes, which stop their threads, and that completes the program

Network level Wireshark packet reassembly confirmation



No.	Time	Source	Destination	Protocol	Length	Info
4	2.889738	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=7b12) [Reassembled in #48]
5	2.889741	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=7b12) [Reassembled in #48]
6	2.889742	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=7b12) [Reassembled in #48]
7	2.889742	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=7b12) [Reassembled in #48]
8	2.889742	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=5920, ID=7b12) [Reassembled in #48]
9	2.889742	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=7400, ID=7b12) [Reassembled in #48]
10	2.889742	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=8880, ID=7b12) [Reassembled in #48]
11	2.889743	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=10360, ID=7b12) [Reassembled in #48]
12	2.889743	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=11840, ID=7b12) [Reassembled in #48]
13	2.889743	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=13320, ID=7b12) [Reassembled in #48]
14	2.889744	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=14800, ID=7b12) [Reassembled in #48]
15	2.889744	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=16280, ID=7b12) [Reassembled in #48]
16	2.889744	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=17760, ID=7b12) [Reassembled in #48]
17	2.889744	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=19240, ID=7b12) [Reassembled in #48]
18	2.889744	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=20720, ID=7b12) [Reassembled in #48]
19	2.889745	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=22200, ID=7b12) [Reassembled in #48]
20	2.889745	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=23680, ID=7b12) [Reassembled in #48]
21	2.889746	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=25160, ID=7b12) [Reassembled in #48]
22	2.889746	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=26640, ID=7b12) [Reassembled in #48]
23	2.889746	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=28120, ID=7b12) [Reassembled in #48]
24	2.889746	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=29600, ID=7b12) [Reassembled in #48]
25	2.889746	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=31080, ID=7b12) [Reassembled in #48]
26	2.889747	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=32560, ID=7b12) [Reassembled in #48]
27	2.889747	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=34040, ID=7b12) [Reassembled in #48]
28	2.889748	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=35520, ID=7b12) [Reassembled in #48]
29	2.889748	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=37000, ID=7b12) [Reassembled in #48]
30	2.889748	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=38480, ID=7b12) [Reassembled in #48]
31	2.889749	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=39960, ID=7b12) [Reassembled in #48]
32	2.889749	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=41440, ID=7b12) [Reassembled in #48]
33	2.889749	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=42920, ID=7b12) [Reassembled in #48]
34	2.889749	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=44400, ID=7b12) [Reassembled in #48]
35	2.889749	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=45880, ID=7b12) [Reassembled in #48]
36	2.889750	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=47360, ID=7b12) [Reassembled in #48]
37	2.889750	192.168.10.100	203.189.80.1	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=48840, ID=7b12) [Reassembled in #48]

> Frame 12: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
 > Ethernet II, Src: Micro-St_0f:ec:75 (30:9c:23:0f:ec:75), Dst: Ubiquiti_9f:72:a3 (80:2a:a8:9f:72:a3)
 > Internet Protocol Version 4, Src: 192.168.10.100, Dst: 203.189.80.1
 > Data (1480 bytes)

```

0000  80 2a a8 9f 72 a3 30 9c 23 0f ec 75 00 00 45 00  .*.r.0. #..u..E.
0010  05 dc 7b 12 25 c8 80 11 00 00 c0 a8 0a 64 cb bd  ..{....d...
0020  50 01 37 38 39 30 31 32 33 34 35 36 37 38 39 30  P.789012 34567890
0030  31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36  12345678 90123456
0040  37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32  78901234 56789012
0050  33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38  34567890 12345678
0060  39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34  90123456 78901234
0070  35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30  56789012 34567890
0080  31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36  12345678 90123456
0090  37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 32  78901234 56789012
00a0  33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38  34567890 12345678
00b0  39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34  90123456 78901234
00c0  35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30  56789012 34567890
00d0  31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36  12345678 90123456
  
```

Figure 9

The last image is of actual networking wire output captured through Wireshark, where it can be seen Microsoft’s socket wrapper does indeed break up a message of 65KB into 1514 byte packets with 1480 byte payloads. The target IP address had to be a non-localhost host address, as localhost traffic does not pass through the networking stack where Wireshark can read it.

Although this library is fully functional for our needs as a TCD, before implementing it into our own project we’ve highlighted additional features we’d like to add:

- Smoothly handling all exceptions with detailed error messages
- Adding full unit testing to ensure code quality
- Perform code review and ensure the code meets style guidelines
- Add a logging ability, likely with the NLog library
- Implement a heartbeat mechanism
- Ensure all functions are thread safe
- Derive a class which implements our desired security protocols
- Clean-up various code smells, closing of listening thread and `isNetworkingRunning` flag

Risk List

Generic Project Risk List

ID	Current Status	Risk Impact	Prob. of occurrence	Risk Map	Risk Description	Project Impact	Risk Area	Symptoms	Triggers	Risk Response Strategy	Response Strategy	Contingency Plan
1	Closed	Medium	Low	Green	Scope is poorly defined	Time required to complete may overflow available time	Schedule Project resources	Poor requirements	Requirements found not make viable project	Avoidance then contingency	Through initial requirement assessments	Redesign appropriate parts of project
2	Open	Medium	Low	Green	Scope creep inflates scope	Time required to complete may overflow available time	Schedule Project resources	Poor requirements	Member proposes scope increase	Avoidance then contingency	Through initial requirement assessments	Scope changes must be agreed on by all members
3	Open	Medium	Medium	Yellow	Estimates for milestones are inaccurate	Time required to complete may overflow available time	Schedule Project resources	Estimated to fail to meet project deadline	Failure to meet project milestone	Avoidance then contingency	Iterative review on expectations to increase accuracy	Overtime or reassessing target milestones
5	Closed	Medium	Medium	Yellow	Finish project too early	Poor allocation of estimations, Project lacks "Substantial challenge" requirement	Schedule Project resources	Not enough work to do in project	Plan shows project will finish early	Contingency	N/A	Add additional features
6	Closed	High	Low	Yellow	Team member conflict	Failure to complete project milestones or non-productive team member misses milestones	Schedule Project resources	Poor team member interpersonal skills	Team member officially complains	Avoidance then contingency	Ensure initial healthy team dynamic	Mediation, Team Charter resolution mechanisms
7	Closed	Medium	Low	Green	Poor team dynamics	Disagreements lead to loss of productive work on project	Schedule Project resources	Poor team member interpersonal skills	Team member officially complains	Avoidance then contingency	Ensure initial healthy team dynamic	Mediation
8	Open	High	Low	Yellow	Member is unavailable	Project work may overload remaining member, milestones required to be redesigned	Schedule Project resources	Team member expects to be unavailable	Team member is unavailable	Contingency	N/A	Member catch up or reassess milestones

9	Open	High	Low	Yellow	Member is lost	All project work will burden remaining member	Schedule Project resources	Team member expects to be lost	Team member is lost	Contingency	N/A	Reassess requirements and milestones
10	Closed	Medium	Low	Green	Architecture is not fit for purpose	Extra time required to re-architect the project	Feasibility Schedule Project resources	Extra design required during implementation phase	Recurring ad hoc resign during implementation	Avoidance then contingency	Through initial design	Redesign project
11	Closed	Low	Low	Green	Technology components aren't scalable	Final product may have limited use cases by users	Feasibility Schedule Project resources	Overhead of algorithm expected to limit number of nodes	Overhead of algorithm limiting number of nodes	Avoidance then acceptance	Through initial design	Reassess requirements
12	Open	High	Low	Yellow	Technology components have security vulnerabilities	Time taken to fix issues may extend project milestones	Feasibility Security Schedule Project resources	Security vulnerability suspected	Security vulnerability confirmed	Avoidance then contingency	Through initial design	Resolve security issues
13	Closed	High	Low	Yellow	Base technology not mature	Extra time required to re-architect the project	Feasibility Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance then contingency	Through initial design basing on known suspected tech	Redesign will alternative technology
14	Open	High	High	Red	Code quality issues	Project not fit for release at final milestone, extra time required to resolve issues	Schedule Reliability Project resources	Code quality semi regularly fails checks	Code quality regularly failing checks	Avoidance then contingency	Through initial thorough design and test-driven development	Perform additional quality checks
15	Open	Low	Medium	Green	User acceptance failure	No uptake by community of final product	N/A	Initial user interest is low	User interest is low	Acceptance	N/A	N/A
16	Open	Low	Low	Green	Users have inaccurate expectations	No uptake by community of final product	N/A	Initial user interest is low	User interest is low	Acceptance	N/A	N/A

Project Specific Risk List

ID	Current Status	Risk Impact	Prob. of occurrence	Risk Map	Risk Description	Project Impact	Risk Area	Symptoms	Triggers	Risk Response Strategy	Response Strategy	Contingency Plan
1	Closed	High	Low	Yellow	Bug found in consensus algorithm	Additional time required to resolve	Schedule Project resources	Bug suspected in algorithm, algorithm is unproven	Bug found in consensus algorithm	Avoidance	Basing on proven consensus algorithm	Redesign, implementing different consensus algorithm
2	Closed	High	Low	Yellow	Project isn't feasible in general	Fatal impact to project	Project	Not finding any successful implementation	Confirming project is theoretical or no successful implementation exist	Avoidance	Confirmed other implementation of library successful and project is practically possible	Shutdown project
3	Open	High	High	Red	Poor software quality	Additional time required to resolve	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance then contingency	Focus on code quality (unit testing, TDD, code review, pair programming)	Perform additional quality checks
4	Closed	High	High	Red	Networking library issues	Additional time required to resolve	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance then contingency	Through initial thorough design and TCD and focus on code quality	Redesign networking library
5	Open	High	High	Red	Security too complex	Additional time required to resolve	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance then contingency	Early completion in project, code designed to support later addition if required	Resign network security
6	Open	High	Medium	Red	Prototype failure	Additional time required to resolve	Schedule Project resources	Estimated to fail to meet project deadline	Failure to meet project milestone	Avoidance then contingency	Initial design, accurate time estimation of milestones, TCD	Overtime or reassessing target milestones

7	Closed	High	Low	Yellow	Tech stack not compatible	Additional time required to resolve	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance	TCD	Redesign required components
8	Closed	Low	Low	Green	Logging library of project too hard to implement	Additional time required to resolve	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance then contingency	Prototype completed to confirm difficult of usage	Additional time required to implement or deciding on alternative library
9	Closed	Medium	Low	Green	Project library is too difficult to implement	Additional time required to improve usability	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance	Focussing design of library interface on usability	Redesign project to focus on usability
10	Closed	Medium	Medium	Yellow	Software introduces unreasonable additional surface area for failure	Additional time required to resolve	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance	Focusing on code quality and minimal design	Perform additional quality checks
11	Closed	Medium	Low	Green	Consensus algorithm add unreasonable overhead for timely response	Additional time required to resolve	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Mitigation	Basing design on algorithm with provable consensus speed	Redesign required components
12	Open	High	High	Red	Multithreading introduces high level of difficult in troubleshooting	Additional time required to resolve	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance then contingency	Team members sufficiently skilled at debugging at this level	Additional time required to resolve issues
13	Closed	High	Medium	Red	Network level security issues related to usage of UDP	Additional time required to resolve	Schedule Project resources	Projection of issues during implementation	Issues during implementation	Avoidance then contingency	Designing to avoid UDP reflection issues	Additional time required to resolve issues

Master Test Plan

Testing Strategy

Test Driven Development (TDD)

Test driven development will be utilised in this project; this methodology focuses on testing the design specifications is accurately implemented in the code, rather than testing just the implementation of code itself. If all of these tests are written and pass, then it can reliability be confirmed that the code is accurate to the specification, no more no less.

Unit testing(UT)

These tests are directly related to the use of TDD and confirm that code is meeting the requirements of each unit's functionality. Unit testing is the set of atomic tests on the implementation of each object's public contract to ensure they meet requirements. If all of these tests are written and pass, then it can be reliably confirmed that each unit of code is correctly performing its functionality.

Integration Testing (IT)

These tests are directly related to the use of TDD and confirm that code is meeting the requirements of the design's use cases. These IT are simply groups of UTs which form together to make a use case. If all of these tests are written and pass, then it can be reliably confirmed that the code is performing all the use cases to specification.

Code review(CR)

During development, strict adherence to coding guidelines greatly improve maintainability with a flow on effect of code reliability. To perform code review, a developer writes or makes changes to the code base, and then a separate developer reviews and audits line-by-line those changes, ensuring quality and implementation ideas match design. This reduces, and ideally removes, obvious logic errors, code smells, improper implementation and other various code issues.

Prototype(PT)

The prototype will include a minimal functional feature set of the library which can be used to confirm successful implementation of the consensus algorithm. This prototype will also be confirmed by the TDD life cycle.

Implementation testing (IMPT)

Alpha/beta testing is not within a reasonable scope of our project due to the unreasonable level of burden it would place on an alpha/beta tester to integrate our library. To solve this the library will be integrated by ourselves into an open source program to confirm all integration functionality.

Design validation(DV)

Some functionality is not so black/white where it can be confirmed through software tests, for this we have the professional work of developers confirming functionality. For example, minimal resource usage cannot be unit tested, however a developer can give their professional results confirming the matter.

Tests to be Conducted

Feature/Functionality	Test method	Testing environment	Acceptance criteria	Role
Consensus between distributed systems	IT	VS	All functions properly respond and	Joshua
Fault tolerant distributed service	IT	VS	IT pass	Sean
Improved reliability of existing service	IMPT	VS/Workstation manual test	Reliability of existing service is improved, confirming still responds during node failure	Joshua
Complete proven reliability	IMPT	VS/Workstation manual test	Confirm function after all node scenario failures	Sean
Minimal additional surface area for failure	CR	Bitbucket Pull Request	All code is only entered into repository upon successful review from code review	Joshua
Cross Platform	DV	Developer assessment	Two developers confirm functionality	Sean
Mitigate project abandonment	DV	Developer assessment	Two developers confirm design	Joshua
Minimal overhead/impact to service performance	IMPT	VS/Workstation manual test	Two developers confirm functionality	Sean
Minimal resource usage	DV	Developer assessment	Two developers confirm results	Joshua
Ability to pick ideal leader	IT	VS	IT pass	Sean
Warm nodes	IT	VS	IT pass	Joshua
Upgrade path	IT	VS	IT pass	Sean
Consensus between distributed systems	IT	VS	IT pass	Joshua
Reliability	IMPT	VS/Workstation manual test	Two developers confirm results	Sean
Usability	IMPT	VS during implementation	Two developers confirm functionality	Joshua

Feature/Functionality	Test method	Testing environment	Acceptance criteria	Role
Documentation	DV	During implementation testing	Two developers confirm documentation existence	Sean
Quality	CR	Bitbucket Pull Request	Two developers confirm functionality	Joshua
Performance	DV	Developer assessment	Two developers confirm results	Sean
Compatibility	DV	Developer assessment	All code is only entered into repository upon successful review from code review	Joshua
Availability	IMPT	VS/Workstation manual test	Two developers confirm functionality	Sean
Security	IT	VS	IT pass	Joshua
Privacy	IT	VS	IT pass	Sean
Scalability	IT	VS	IT pass	Joshua
Testability	DV	During implementation testing	Two developers confirm functionality	Sean
Extendability	DV	Developer assessment	Two developers confirm design	Joshua
Auditability	DV	Developer assessment	Two developers confirm functionality	Sean
Troubleshooting	DV	Developer assessment	Two developers confirm functionality	Joshua
Can write to debug log	UT	VS	UT pass	Joshua
Can write to distributed log	UT	VS	UT pass	Sean
Can read from distributed log	UT	VS	UT pass	Joshua
Can do CRUD operations on Nodelist	UT	VS	UT pass	Sean

Feature/Functionality	Test method	Testing environment	Acceptance criteria	Role
Can propagate node info	UT	VS	UT pass	Joshua
Can send and receive messages from other nodes	UT	VS	UT pass	Sean
Can communicate using encrypted messages	UT	VS	UT pass	Joshua
Can confirm identity of a new node joining cluster	UT	VS	UT pass	Sean
Does library call UAS start/stop	UT	VS	UT pass	Joshua
Can respond with fitness	UT	VS	UT pass	Sean
Can request fitness from nodes	UT	VS	UT pass	Joshua
Can maintain service during node failure/loss	UT/IT	VS	UT pass, IT pass	Sean
Can recover from node failure/loss	UT/IT	VS	UT pass, IT pass	Joshua
Can hold successful election	UT	VS	UT pass	Sean
Bring node log up to date	UT	VS	UT pass	Joshua
Falls to follower when detecting newer leader	UT	VS	UT pass	Sean
Confirm library is one-click integration from Nuget	IMPT	VS during implementation	Two developers confirm result	Joshua
All functionality in prototype specification working	PT	Developer assessment	Two developers confirm functionality	Sean
Practical implementation full functionality working as required	IMPT	Developer assessment	Two developers confirm results	Joshua

Project Plan

Introduction

This part of the document outlines the high-level objectives for each iteration, and where each architectural element will be delivered. Further to these, it will identify all risk mitigation strategies and allow for the execution of contingency plans where required. This plan aims to be a complete and concrete expression of the concepts of the Unified Process.

There are no mitigation strategies scheduled during this project plan as we are going to utilise the benefit of our design-focused planning to enable avoidance of risks, where we then have a fall back of contingency in place.

Project practices and measurements

This project will be performed through the application of the iterative design methodology, this includes fortnightly outline of work to be completed each iteration and a review of the progress in the previous iteration. This includes lessons learnt and assessment of any issues and contingencies.

This project's code development will utilise multiple distinct programming techniques to achieve its non-functional requirement of exceptional code quality. These includes:

- TDD - throughout unit testing and integration testing to ensure functional outcomes are achieved
- Pair programming - A practice of two developers working together on the same screen (IDE, diagramming software, etc.), this technique has shown in practice to realise high levels of code quality
- Style guidelines - These are strict standards of code readability quality which each team member must adhere to
- Code review - This is the process of ensuring each member's code is reviewed by another before committing to the permanent code branch. This verifies quality through a two-party acceptance system.

Deployment

This project will have a smooth a simple deployment. It's project repositories on Bitbucket will be set to public, and the code will be packaged and made available online for use through the Visual Studio package manager, NuGet.

Project milestones and objectives

Subject	Phase	Iteration	Dates	Primary objectives (risks and use case scenarios)
ITC303 – Software Development Project 1	Inception Phase	I-1	12/03 – 25/03	Establish Vision Establish Initial Use Case Model Complete Preliminary Non-functional Requirement Analysis Identify/Document Candidate Architectures Establish Version Control
		I-2	26/03 – 8/04	Establish Risk List Complete Full Description for Critical Core Risky Difficult (CCRD) Use Case Implement Technical Competency Demonstrator Create Test Plan Establish Initial Project Plan Deliver Life Cycle Objectives Milestone (LCOM) Complete Inception Phase Project Assessment
	Construction Phase	E-1	9/04 – 22/04 (Session Break)	Contingency for the delivery of LCOM Mitigate Highest Priority Risk - Detailed diagramming of all desired features of the library, as well as the writing of integration tests as part of the TDD methodology Implement Highest Priority Architectural Element(s) to Support CCRD Use Case - High Quality Code Complete Development Testing for Highest Priority Architectural Element(s) - High Quality Code, TDD
		E-2	23/4 – 6/05	Mitigate 2 nd Highest Priority Risk(s) - Network Communication and Security Implement 2 nd Highest Priority Architectural Element(s) to Support CCRD Use Case - Network Communication and Security Complete Development and Integration Testing for 2 nd Highest Priority Architectural Element(s) - Network Communication and Security
		E-3	7/05 – 20/05	Mitigate 3 rd Highest Priority Risk(s) - Distributed Consistent Log and Fault Tolerance Implement 3 rd Highest Priority Architectural Element(s) to Support CCRD Use Case - Distributed Consistent Log and Fault Tolerance Complete Development and Integration Testing for 3 rd Highest Priority Architectural Element(s) - Distributed Consistent Log and Fault Tolerance Ensure the Functionality of the Prototype Confirm all Integration Tests Pass for CCRD Use Case by the Prototype
		E-4	21/05 – 3/06	Contingency for completion of E-3 Deliver Life Cycle Architecture Milestone (LCAM) Complete Elaboration Phase Project Assessment
	Mid-year Semester Break			

Mid-year Semester Break				
Subject	Phase	Iteration	Dates	Primary objectives (risks and use case scenarios)
ITC309 – Software Development Project 2	Feature Phase	C-1	9/07 – 22/07	Implement 2 nd Highest Priority Use Case(s) - Dynamic Cluster Membership (2 node, odd nodes, even nodes) Complete Development and Integration Testing for 2 nd Highest Priority Use Case(s) - Dynamic Cluster Membership (2 node, odd nodes, even nodes) Complete Internal User Acceptance Testing for 2 nd Highest Priority Use Case(s) - Dynamic Cluster Membership (2 node, odd nodes, even nodes)
		C-2	23/07 – 5/08	Implement 3 rd Highest Priority Use Case(s) - Choosing Ideal Leader and Warm Nodes Complete Development and Integration Testing for 3 rd Highest Priority Use Case(s) - Choosing Ideal Leader and Warm Nodes Complete Internal User Acceptance Testing for 3 rd Highest Priority Use Case(s) - Choosing Ideal Leader and Warm Nodes
		C-3	6/0 – 19/08	Implement 4 th Highest Priority Use Case(s) - Upgrade Path and Detailed Performance Analysis/Optimization Complete Development and Integration Testing for 4 th Highest Priority Use Case(s) - Upgrade Path and Detailed Performance Analysis/Optimization Complete Internal User Acceptance Testing for 4 th Highest Priority Use Case(s) - Upgrade Path and Detailed Performance Analysis/Optimization
		C-4	20/08 – 2/09 (Session Break)	Contingency for the completion of C-1 through C-3 iterations Deliver Initial Operational Capability Milestone (IOCM) Complete Construction Phase Project Assessment
	Transition Phase	T-1	3/09 – 16/09	Contingency for the completion of IOCM Deploy Application - Open Source the Code Library and Make Available through NuGet Complete 1 st Round External User Acceptance Testing - Through Practical Integration Into An Open Source Project Contingency - Resolve Any Identified Issues
		T-2	17/09 – 30/09	Complete 2 nd Round External User Acceptance Testing - Finalisation of Practical Integration Into An Open Source Project Contingency - Resolve Any Identified Issues Go Through All existing Project Management Documentation and Ensure Quality Being work on Product Release Milestone
		T-3	1/10 – 14/10	Contingency for the Completion of PRM and Any Other Requirements Deliver Product Release Milestone (PRM) Complete Final Project Assessment

Inception Phase assessment

Goals of Inception Phase and How They're Being Achieved

Gather accurate system requirements

We've successfully achieved this thorough analysis of the system requirements by gaining an understanding of the target audience's desired functionality. This was refined through various detailed discussions regarding references to other similar systems on 20th of March 2018 and determining an accurate use case model during 20-27th of March 2018. This allowed us to produce a Project Vision document and Requirements Model document.

Analyse functional and non-functional requirements

Following on from accurate system requirements earlier, we were able to determine an accurate list of functional and non-functional requirements and conduct detailed analysis on them, justifying them against the business case. This was conducted between the 19th and 24th of March. This allowed us to add our analysis into the Project Vision and Requirement Model documents.

Produce diagrams expressing system functionality

These high-level requirements expressed as both functional and non-functional were then cultivated into the Domain Model. Essentially this process has formed the crux of the elaboration phase of the project so far allowing clear insight into the architectural requirements.

Propose system's architecture and highlighting architecturally significant requirements

Through a combination of reviews of the functional and non-functional requirements and the newly produced Domain models we have been determine the architecturally significant requirements and propose a suitable system architecture. This was completed through 31st of March and 7th of April and allowed us to complete our Architectural Proposal document.

Produce a minimal proof of concept for demonstrating technical competency

Once the main components were identified from the system architecture stage, they needed to be shown to be technologically possible. This was done through the creation of a Technical Competency Demonstrator. This proof of concept was written to be above and beyond spec on 29 of March, and then was refined and documented on 7th of April. The reason we invested the extra time to go above and beyond spec was so that the code was reusable later in the project, and not a wasted effort.

Determine and analyse project risks

As project management is an established process there are multiple references to general project risks. We took various examples of these lists and analysed if, and to what extent, each of those risks applied to our project. We then looked at the risks included within our architecture and analysed those as well. We did this on the 7th of April to produce the Risk List document.

Provide clear plan for ensuring quality of project through testing

For this document, we took all the functional requirements, non-functional requirements, and use cases, then described how each of them would be tested to verify they're achieved during the project. By targeting these features, we were able to most accurately ensure the project will match design requirements. This was done on the 13th of April.

Produce a timeline for various milestones throughout the project

From our project requirements we're given a set time to complete the project in, and a section of stop work where no progress will be made. Working within these limitations, each of the use cases, risks, and testing steps were distributed upon the available iterations, focusing on the highest priorities first. The time for various documentation milestones (LCAM, IOCM, PRM) were isolated from development work due to lessons learnt about the amount of time it takes to achieve respectable quality standards, also there are contingencies added into the plan to ensure these are reliability completed without affecting the plan too much even if they run over time. This assessment was conducted on 7th of April and allowed us to create our Project Plan document.

Justification of Technical Capability to Achieve Those Goals

The technical capability of the members to achieve this project is seen as a substantially low risk as team members have experience using each of the skills required for this project, such as Test Driven Development, .NET, Git version control, and code review. Additional to these technical skills is Object Oriented System Analysis and Design (OOSAD), this incorporates all the documentation and design stages of the assessment, and both members also have formal qualification with these subjects.

As a practical proof of team member's understanding of the technologies used in this project a technical competency demonstrator (TCD) software program was produced. This TCD program showed the successful serialisation, transport and deserialization of JSON messages over a UDP network connection. This was later analysed in the "Technical Competency Demonstrator" document on 30th March 2018.

Issues encountered

Generally speaking we have not experienced technical issues thus far, however there is a real indication that we have overly optimistic view on how much time goes into tasks. This is highlighted by our initial estimates in the Iteration Plan 1 where our time estimations were out by around 1000%. Applying our learnings from this our second iteration plan's time estimations were out by 450%, an improvement nonetheless. Looking forward, this is one of our main focuses on project management which we'd like to become better accurate with, however we believe this is considered a skill and will mature over time.

Status of Any Important Risks

Risk	Open/Closed	Status
Code quality issues	Open	Neither symptom or trigger has occurred, migration currently not required
Poor software quality	Open	Neither symptom or trigger has occurred, migration currently not required
Networking library issues	Closed	Risk has successfully been resolved and demonstrated in Tech Competency Demonstrator
Security too complex	Open	Neither symptom or trigger has occurred, migration currently not required
Prototype failure	Open	Neither symptom or trigger has occurred, migration currently not required
Multithreading introduces high level of difficult in troubleshooting	Open	Neither symptom or trigger has occurred, migration currently not required
Network level security issues related to usage of UDP	Closed	Risk has successfully been resolved and demonstrated in Tech Competency Demonstrator

Current progress of project

The current state of progress is **satisfactory**; the reason we aren't considering the state as "**good**" is we are currently working within our contingency allowance iteration of LCOM. Moving forward we would prefer to finish our targeted outcomes on time and not to rely on these contingency time allowances when meeting our deliverables.

At this stage, the project has merit, the team has the technical skills required as demonstrated in the TCD. Provided we can improve our time estimation to adhere to the project plan, a minimal functional feature set of the library as a prototype can be delivered by end of the Construction Phase on the 3rd June 2018.