A Framework for Collaborative Applications using
a Client-Server Network With Supernodes

by

YiYun Zhao
B.Sc., East China Normal University, 2013

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SECIENCE

in the Department of Computer Science

A Framework for Collaborative Applications using
a Client-Server Network With Supernodes

by

YiYun Zhao
B.Sc., East China Normal University, 2013

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Sudhakar Ganti, Departmental Member
(Department of Computer Science)

ABSTRACT

Today's product managers must quickly determine viable avenues for innovation while carefully balancing the costs and benefits involved. Agile methodologies are highly incremental and often seen as lacking in rigour and due diligence. This thesis explores the relationship between processes and tools that are commonplace for product managers versus those that tend to be reserved for researchers. A case study reveals key opportunities for the practices in each domain to inform each other, and further identifies the need for gaps in the tooling to be addressed. The study uses *Think Together*, a collaborative mobile application for interactive presentations with rich media content. The application supports individual action layers for each user and session replay, creating several challenging bottlenecks that jeopardize the scalability of the original implementation. A proposal for an alternative network configuration for communication to address these bottlenecks is examined from both a product management viewpoint and from a more traditional research perspective. A simulator is used as a means to analyze and evaluate the proposed configuration, revealing essential trade-offs in terms of efficiency and productivity. Unlike testing on real devices, the simulator is much more in line with agile processes, enabling more power and flexibility without the limitations of physical resources. However, the extent to which simulated results are practical in the real world, in particular to product managers, is an open question. We demonstrate how a lifecycle involving both traditional approaches to research and incremental implementation strategies in agile environments complements each other, and further identify current obstacles involved.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

This is a long and tough journey, but I would like to show my appreciation to those who are supportive for me and accompanied with me during the period.

- I want to thank my parents for their unconditional support and love. I could not achieve what I have without them.
- I would like to express my gratitude to my supervisor Dr. Yvonne Coady, who mentored and guided me through the whole experience patiently as well as offered me opportunities to pursue what I wanted.
- I appreciate the help and encouragement from everyone in Two Tall Totems, especially Chris, David and Josephine.
- I need to thank Lei Dai for understanding and bearing with me during this time.
- I am also grateful for the weather in Vancouver, and my furry bear that is always there for me.

Chapter 1

Introduction

In the world where more and more people have access to the Internet, widespread communication is becoming significant in our daily lives. In particular, social networking and collaborative activities supported by software on mobile devices have become mainstream in recent years. *Think Together* [66] is a collaborative application developed specifically for touch-screen iOS devices. The goal of this application is to support interaction between participants during meetings with shared media, such as images. The uniqueness of the application is the way in which it plans to integrate shared media while recording interaction such as on-screen annotations and voice. Further, in *Think Together*, any participant can not only initiate a meeting but also replay and share the meeting with a third party after the meeting, recording and filtering voice and collaborative events at a per participant granularity.

In order to launch a product like *Think Together* on the market, the company needs more than a prototype with fundamental features. They need a stable product with acceptable performance, and a solid plan to continue to scale the application. The current implementation consists of a simple client-server network structure, where the server is located in one of the participating devices. Discovering and joining a meeting is accomplished using Bonjour [19], Apple's implementation of a zero-configuration networking service. Nevertheless, there are various deficiencies when the server supporting the whole network is on a resource-poor mobile device. From a user perspective, the key issue that requires further investigation is productivity, which can be validated through metrics such as stability and scalability of the underlying infrastructure.

As a first step towards scaling an application such as this, a product manager needs to determine the behavior of a resource-poor server when subject to specific

workloads involving a large amount of data transmission. In this case, the data could be streaming audio or video, on a large scale. The feasibility of accommodating more participants while preserving a minimum quality of service for all users can in part be determined by understanding the tolerance for increasing response times during collaborative work.

This thesis investigates this problem from two perspectives: (1) as a product manager using agile methodologies, and (2) as a researcher using more standard robust methodologies and tooling. We consider both an exploratory study for product validation and also a proposal for a modification designed to improve scalability in future versions of the product. Specifically, we explore the overlap and synergy between these two perspectives applied to the problem of improving infrastructure for communication.

As opposed to the simple client-server infrastructure currently used by the application, we consider more popular network alternatives such as peer-to-peer (P2P) networks [4] and supernode structures [40]. A network configuration is designed and evaluated as a simulator that is capable of reproducing core behaviors associated with the application. Though simulation provides a powerful and flexible methodologies for inspecting the configuration while we examine its characteristics from differing perspectives, the question of whether or not this type of approach is useful from a project management [55] perspective remains.

We argue that, from a product management perspective, the research effort involving the simulation provides critical insights to the product roadmap [50]. However, in its current form, this kind of support to make decisions during the planning of a product does come at a cost in the form of a substantial learning curve. Similar to a researcher, a product manager seeks metrics to evaluate a solution to a question. However, the scale of emphasis between the two approaches is distinct. While a researcher might seek to establish an absolute truth in the context of many scenarios, a product manager is typically trying to identify a more modest relative improvement in a short period of time. This means that the benefits of adopting more elaborate processes and tools borrowed from the research community may have diminishing returns. We outline the costs and benefits of these tradeoffs in the context of our exploratory study and network simulation of the *Think Together* application.

1.1 Case Study Background

As platforms for teamwork are gaining popularity in industry [25], collaboration is emerging as an important future trend. Communication, sharing, and cooperation are becoming key values associated with success and many products are trying to create an appropriate platform using mobile technology. Though the market demand is clear, many research efforts have identified critical challenges in architecture and implementation when relying solely on mobile devices. While product managers are struggling to identify feasible product plans, relatively simple tools used every day by researchers could inform them of the likelihood of improvements in infrastructure design.

Clearly, untapped potential for market differentiation exists, even just in terms of blending technologies such as cellular, wireless and Bluetooth [10] under the right circumstances, depending on the connectivity and proximity of participants. However, it is exceedingly difficult for product managers to reason about these kinds of tradeoffs without the right tools to explore the space quickly and efficiently.

1.2 Contributions

In partnership with the industry stakeholders responsible for the *Think Together* application, this work was conducted in three stages: (1) an exploratory study involving the application in a collaborative setting, (2) the development of a simulation tool to explore tradeoffs in alternative network infrastructures, and (3) an assessment of the kinds of tradeoffs the simulation tool reveals, and the applicability of this kind of assessment to a typical product manager's toolkit.

The exploratory study was conducted in a school classroom setting, where *Think Together* was shown to be potentially effective for collaboration between the participants. Both students and teachers anecdotally reported that it was beneficial to utilize collaborative tools for lecturing, allowing more explicit interaction compared to traditional modes of education. In this exploratory study, we also conclude the need to modify the infrastructure in order to scale with the number of participants involved.

The simulator was built explicitly for exploring the application at scale. Given the current simple client-server structure, it is not surprising that a heavy workload or a large number of participants could crash the server. In order to alleviate this

bottleneck, we establish alternative infrastructures that delegate work to other nodes in the network besides the server.

Based on the metrics established in the exploratory study, we investigate the current infrastructure relative to a newly proposed configuration. In conjunction with industry stakeholders, we further assess the value of this simulation tool in the hands of a typical product manager. We identify some of the costs and benefits of including everyday research tools in a product manager's toolkit. Costs are largely associated with learning curves and integration with other more common tools. Benefits include the flexibility and accuracy afforded by the approach. In particular, if the tool can be incorporated into an agile lifecycle, where parameters are informed by prototype implementations, the benefits could outweigh the costs of learning to use the tool.

Table 1.1: Comparison between typical research questions versus product management questions

	Exploratory Study	**Network Simulation**
Sample Research Questions	Was learning more effective? What metrics are critical for identifying improved mode of education? How does this relate to other publications about technology in the classroom?	What spectrum of configurations are feasible? What metrics are critical? How are they informed by real-world results? What is the relationship between a theoretical result, one shown in the simulator, and the real-world?
Sample Product Management Questions	Is there a market for the product? Does the product appeal to the target market? What are the competing products?	What are the bottlenecks in the current system? What is the most cost-effective way to mitigate the bottleneck for the next version of the product?

Table 1.1 shows a high-level comparison between what could be considered typical research questions and typical product management questions. In general, it would be reasonable to expect research questions to pursue in-depth characteristics and nuances between alternative infrastructures that would extend beyond proof-of-concept product management needs. In order to achieve high productivity in practice, a prod-

uct manager seeks a low-cost solution that can solve immediate problems [50]. There is definitely overlap between the two perspectives, possibly aligning methodologies though they tend to operate at different scales. For example, with the network simulation, research into the performance of a given infrastructure can involve a spectrum of metrics and a spectrum of configurations [41]. However, a product manager may instead be satisfied once the validation of the performance given by a single proposed network simulation reaches a reasonable standard. It may in fact be less important to consider whether a solution is optimal in an absolute sense, in particular in *agile* product management [62, 49]. It would be exceptional to even consider the time and cost involved in building an elaborate infrastructure to explore alternatives.

However, agile development teams may benefit from commonly used research tools that could address risks in product planning. Though our results are preliminary, this study opens the door for bridging the gap between product managers and researchers. We identify user interface issues with tooling as a possible obstacle between the two worlds but seek to understand the commonalities between the role of a researcher and the role of a product manager.

This work shows concrete ways in which researchers and product managers are asking similar questions that pursue facts and metrics. Generally speaking, research requires comprehensive study in-depth, while a product manager is more concerned about delivery of a working product that is on time and on budget. Though we can establish the ways in which there is overlap, in practice research and product management tend to use their own unique tool chains. Evaluation tools that researchers are familiar with can be foreign to a product manager, and incur the costs of a steep learning curve that could result in diminished returns. Therefore, we identify the need to carefully balance the costs and benefits in any approach to bridging the gap between these two roles.

1.3 Outline

The rest of this thesis is organized as follows. Chapter 2 presents background and related work relevant to the infrastructure for communication within collaborative applications, including several possible alternative network structures. Chapter 3 presents an exploratory study showing *Think Together* in a school environment, with a concrete demonstration and discussion. This study reveals that the practices of the product manager would most likely be much lighter weight than a typical researcher

at this stage of the work. Chapter 4 elaborates on a detailed alternative configuration design and implementation policies. Based on the proposed network configuration, evaluation and analysis with a simple simulator is presented in Chapter 5. Unlike the exploratory study, this study reveals the potential for significant overlap and the opportunity for synergy between the practices common to researchers and product managers. The potential costs and benefits of using a common research tool from a product manager perspective are provided. Finally, we end with conclusions and suggestions for future work in Chapter 6.

Chapter 2

Background and Related Work

This chapter provides context for our work. We first define what we mean by product management and some of the activities involved. We then establish fundamentals for three basic network configurations: client-server, peer to peer (P2P) and super-peer networks. We conclude with a short overview of network performance, and tools typically used by researchers for network simulation.

2.1 Product Development and Agile Methods

Agile is a popular iterative approach to manage projects and teams, now commonly applied in software development [22]. It enables a team to deliver a product incrementally throughout its life cycle instead of an all-in-one delivery in the end. This methodology also supports launching the product to market early and allows for the response to feedback from customers to be timely [16]. As an important principle in agile development, testing is involved throughout the process [60, 61]. Moreover, agile development engages everyone in the team and helps with collaboration [54].

There are several approaches to agile methodology available in practice, such as SCRUM [52], Extreme Programming (XP) [6], adaptive software development (ADP) [29], feature-driven development (FDD) [46], and Crystal [30]. Nevertheless, most of the successful agile teams adapt these classic methods with their teamwork style and create a particular agile development methodology suitable for them [42]. It is common to see the combination of SCRUM and XP in lists of good practices [38].

2.1.1 Planning

At the start of a project, developers will not have a good idea of how much they can do and how much time each feature will require and cost. It is difficult to provide accurate answers and plan carefully. Instead, in agile approaches, short-term goals are identified. An estimation is typically based on criteria that it is good enough to make a start and the idea that the group will learn from successes or failures through each development cycle [13].

2.1.2 Simple Design

One rule of agile is to make things simple [42]. As a result, a development team will put their best effort on finishing features by using the most straightforward methods available. For example, if a feature of storing log entries is requested, the team might only have a prototype to put log files on local disk, without involving databases or other tools that may scale better in the future. Unless something is clearly stated to require an extensible framework [18], simplicity of design and implementation will dominate decision processes.

2.2 Network Configuration Alternatives

Keeping in mind this criterion of simple design from agile methodologies, we now consider design alternatives for scaling communication within the application considered in this case study.

2.2.1 Client-Server Network

Client-Server is an architecture model which considers the two parts for data processing [9] as a requester and a provider, which is also known as client and server. A client sends requests to the server to initiate a connection. A server provides service that responses to the requests from the client. As shown in Figure 2.1, the clients are on the left side and the server is on the right side. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources

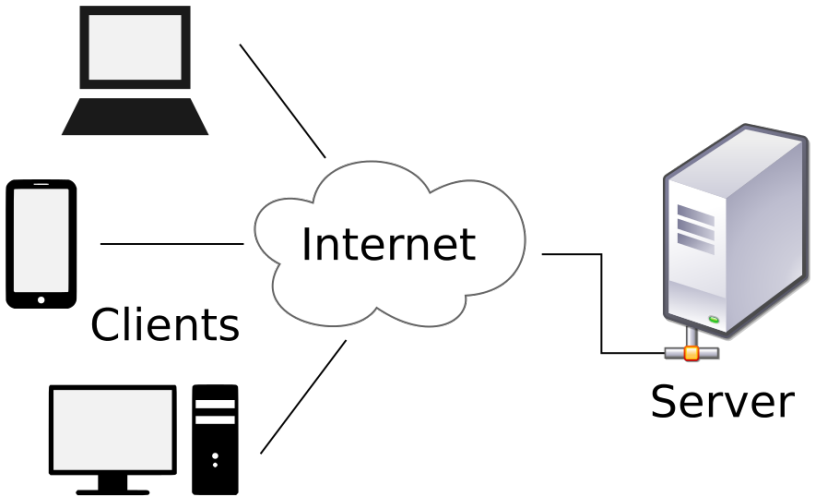


Figure 2.1: Client-Server Model

but requests a server's content or service function. Clients, therefore, initiate communication sessions with servers which await incoming requests [9]. Examples of a client-server application include web browser [34] and e-mail [68].

In a client-server model, one of the advantages is greater ease of maintenance [9]. For instance, all the clients will not be affected when the server is going through an update or replacement, which is also referred to as encapsulation [57]. Furthermore, it more secure to store all the data on the servers rather than clients. Servers can better control access and resources, to guarantee that only those clients with the appropriate permissions may access and change data [9]. Moreover, it is also easier to update the data in the centralized server than in a distributed network. Typically, the server has the capability to connect to different types of clients.

However, there are problems of network traffic in a client-server model. As more and more clients send a request to a server at the same time, the server is likely to be overloaded with a delayed response due to limited bandwidth [51]. Instead, in the case of growth in the number of clients in alternative configurations such as peer-to-peer (P2P) networks, the bandwidth is also growing as the number of peers increases. The sum of the bandwidth available in a P2P network is roughly the sum of the bandwidth of each node in the network [33, 56]. Additionally, a request from a client can not be processed if the server fails in a client-server network.

2.2.2 P2P Network

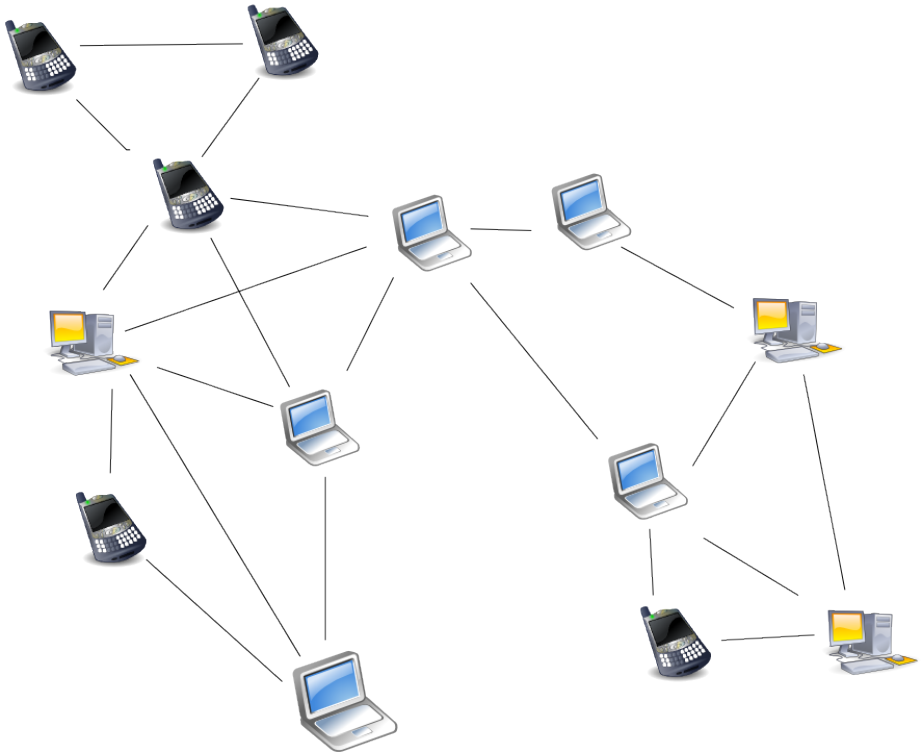


Figure 2.2: P2P Systems Classification

Peer-to-peer (P2P) computing or networking is a distributed application architecture in which peers are equally privileged in participating tasks or workloads [4, 35]. In the late 1960s, Time Berners-Lee proposed the World Wide Web (WWW) [8], which was very similar to a P2P network because users of the web are both consumers and contributors to create and link content to form an interlined web of links [7].

In a P2P network, unlike the client-server model, all resources and content are shared by all the peers, which is more reliable because the failure of one node will not bring down the whole system [65]. The servers are usually distributed in different nodes in the P2P network. Therefore, the requests will be served even though a couple of servers fail [9, 4]. Additionally, since every peer has the same privilege, there is no administrator in this network, everyone manages his/her own machine, and the configuration of the network is not the job of one person. Last but not least, the cost to build a P2P network is comparatively low relative to a traditional client-server network [58].

At the same time, because every peer might have its own unique content, it is potentially difficult to recover from a data loss. P2P networks are also notoriously hard to administrate because there is no centralized location for content. It is extremely difficult to control what is being shared, how it is being shared and where it is being shared from. For example, P2P applications are commonly used in illegally sharing rich media and books. Moreover, security is highly dependent on each peer because the content is shared without auditing in place [58].

2.2.3 Super-Peer Network

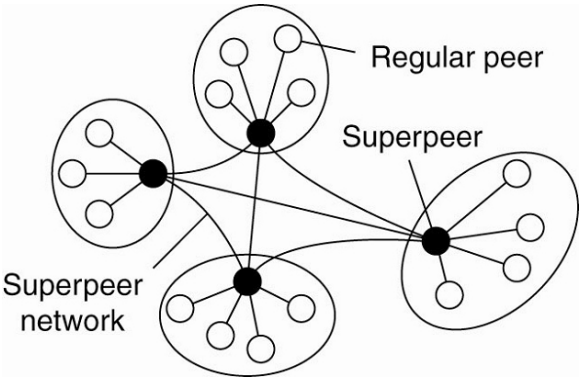


Figure 2.3: Super-Peer Network

A super-peer network is a network structure between P2P network and client-server networks [71]. As shown in Figure 2.3, in a super-peer network architecture, super-peers sit between server and leaf peers, provide services to leaf peers and index contents that leaf peers provide. Requests will come to super-peers first, then will be forwarded to the relevant leaf peer based on the index table.

Super-peer networks have better searching performance than P2P networks because they do not need to query each peer in the network [45]. It is easier to administrate because content has to be accessed through a limited amount of super-peers, monitoring super-peers is equivalent to monitoring the entire network [4]. On the other hand, the searching performance is still not as good as client-server model [39, 44]. The network structure increases the complexity of the network and super-peers need to be carefully chosen so that they rarely accidentally leave, meaning that all their leaf peers would suddenly become unavailable [2].

2.3 Network Performance and Simulation

There are various methods to evaluate the performance of a network [47, 53]. In many circumstances, network performance can be analyzed by modeling or using simulators [32, 64, 11, 12]. We often consider some set of classic metrics important [32], such as bandwidth [69], throughput [23], latency [59], jitter [27] and error rate [31].

Network simulation [36] is commonly used in network research area to simulate and predict behaviors of a network. It can help to valid the behavior of a network, study the scalability in a controlled environment and conduct comparison easily with other networks [11].

2.3.1 Available Simulators

Most network simulations model the network as a discrete sequence of events in time [36], which is also known as discrete event-based simulation [26, 43] (DES). The early research on DES of computer networks was established more than two decades ago [21, 37]. Since then, ns-2 [15] has become virtually the standard for network simulation as a direct successor of those early efforts [67]. However, ns-2 has its limitations on the scalability [28], which is critical to the research exploring large-scale networks. There are alternative network simulators in academia and in commercial use. For instance, SensorSim [48] is based on ns-2, JiST [5] based on Java, OPNet [14] modeler is a commercial tool and OMNeT++ [41]. Among these available simulators, OMNeT++ has better performance by evaluating the comparisons of the execution time incurred and memory usage [70] in a large scaled simulation. It tends to be a popular tool for research standards.

2.3.2 OMNeT++

OMNeT++ is a simulation library and framework based on C++, mainly used as a network simulator [1]. It is developed as a DES for modeling networks and systems [63]. Unlike OPNet that is an expensive commercial simulator, OMNeT++ is free to use for non-profit use in academia. OMNeT++ has better performance than the research-oriented ns-2 in large scale scenarios. It also supports extensive user-friendly GUI in multiple platforms [1].

Chapter 3

Exploratory Study

This chapter is an exploratory study on a demonstration of *Think Together* in a classroom setting. It will also introduce *Think Together* in detail and present its features and possible bottlenecks. The challenges in the next version of the application are uncovered in the study. The analysis was performed by observing videos taken during the hands-on study.

3.1 Overview

Think Together is a collaborative platform initiated by Two Tall Totems. It is developed on iOS devices as a mobile App and also represented as a creative Bring Your Own Device (BYOD) solution. While valuing teamwork more, people have realized that we need more collaborative tools in every industry. For that reason, *Think Together* serves this purpose and fulfills the gap. We are aware of the potential that it has the ability to deliver distinct solutions with different marketing strategies. Such tools in collaboration are significant since people can truly be brought together and produce outstanding ideas in a simpler and faster way. Our research is going to explore and identify the requirements for this collaborative platform.

3.1.1 Features and Highlights

Think Together only works on iOS devices, more specifically on iPads, at the current stage. Users of *Think Together* are able to create, join or replay a session, which could be a business meeting, a solo presentation or a collaborative lecture. It is natural to associate with Skype once we mention analogous features. Unlike Skype,

however, *Think Together* places more emphasis on content collaboration rather than communication.

Every time a session is created, whoever launches the session is assigned a presenter role by default. The presenter needs to host the session and distribute any essential media content to others, such as conference agendas, business documents or lecture slides. Users who join that session are set to the role of the attendee.

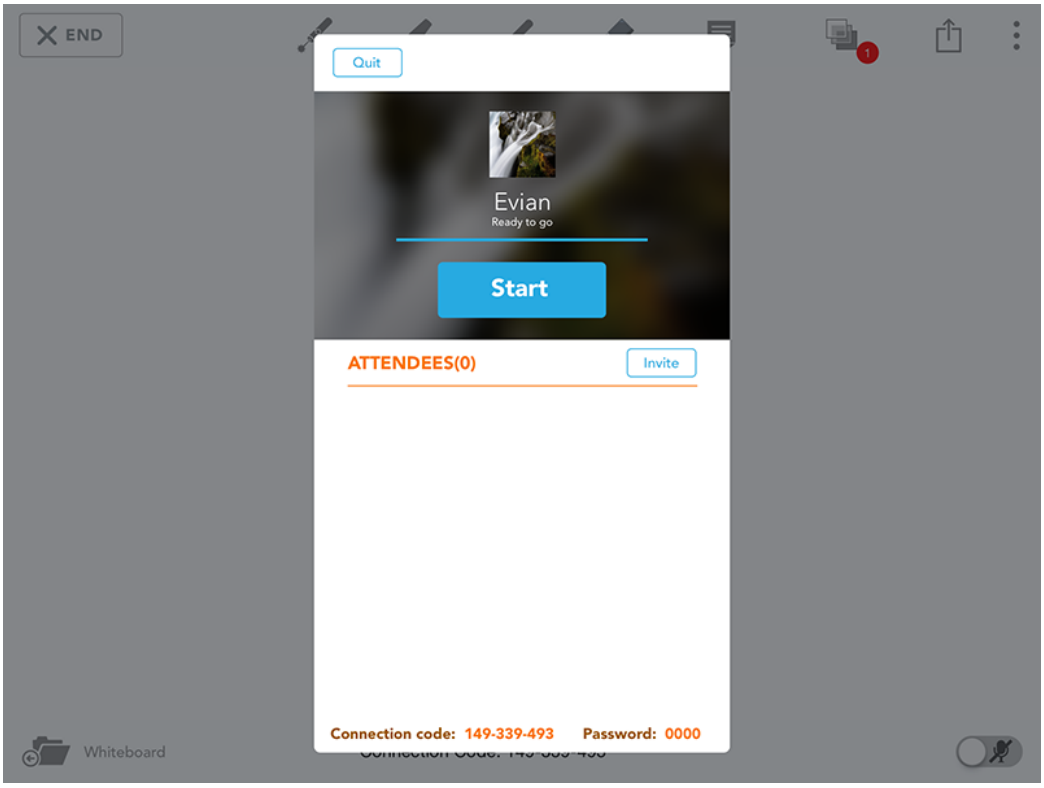


Figure 3.1: Starting Session

As shown in Figure 3.1, a list of attendees is displayed to the presenter before the session starts. The presenter can have an idea of who are attending this session and how many there are in the starting screen. Everyone can also view the process bar that represents the process of receiving media content from the presenter. The connection code and password for this session are displayed at the bottom in the view, where the password indicates whether this is a private session. Meanwhile, the same details are presented to attendees as well, except that only the presenter has the right to start the session.

To join a session, attendees can easily select any available local session shown in the screen as illustrated in Figure 3.2. These sessions are created in this local network

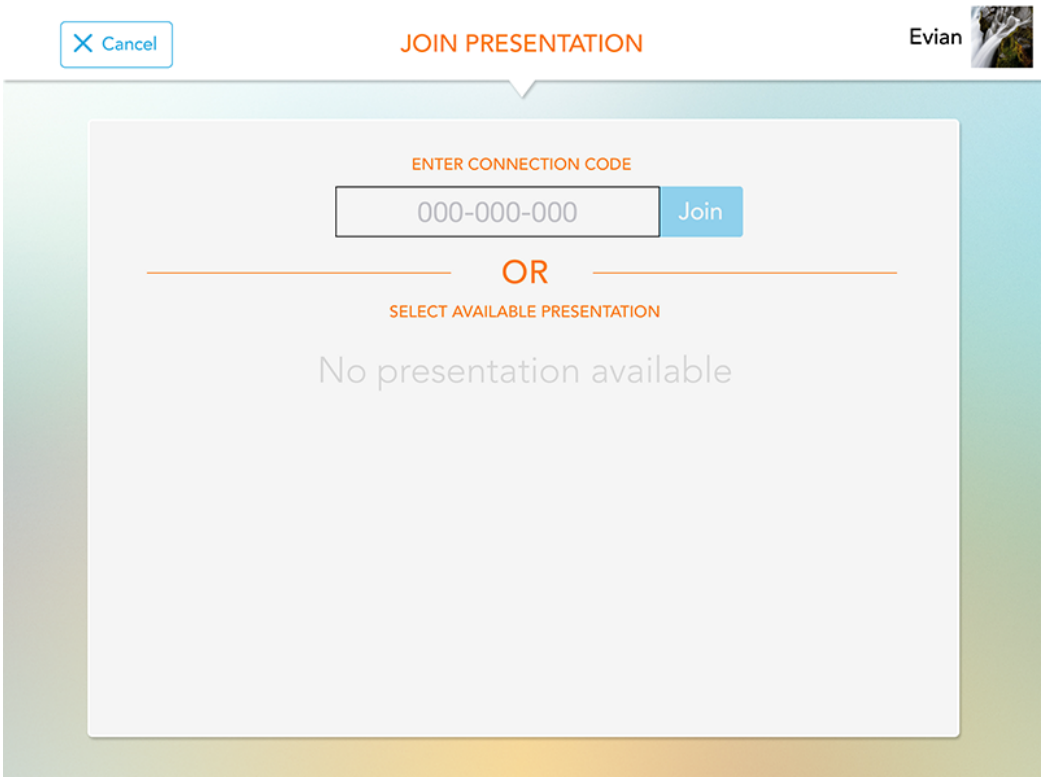


Figure 3.2: Joining session

and detected by the system. Otherwise, attendees can enter the connection code with a password if applicable to have access to that session. When joining a session that has not started, the attendee will go to the screen that is shown in Figure 3.1.

After the session is started by the presenter, everyone in the session will move to the session screen as Figure 3.3 indicates. During the session, all users can add annotations to the session, such as laser pointer, highlighter, pen, and notes. In addition, users can erase the annotations of highlighters and pens, and the notes are free to be edited or deleted. There are various colors and various sizes to choose from for the laser pointer, highlighter or pen. It is convenient for users to apply these tools to do markups during a session. Users can always erase markups they do not want. Furthermore, notes can be pinned to the session, and they can also be edited or deleted by the user. All markup annotations are associated with users, and users can filter markups they don't want to see.

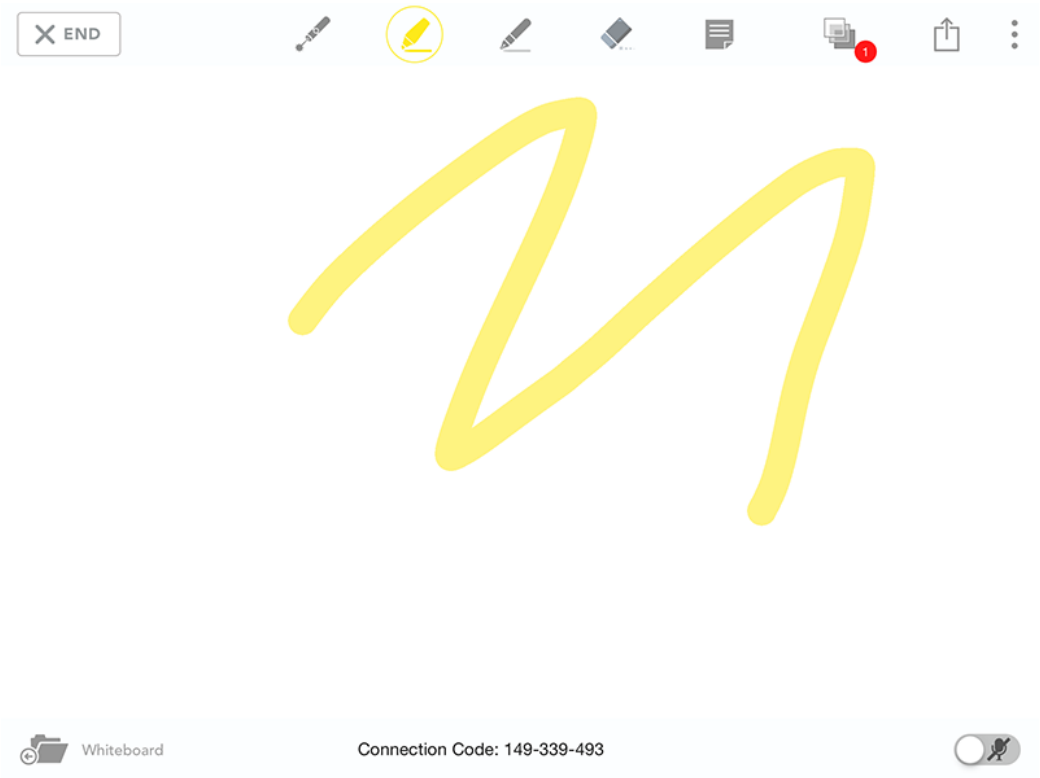


Figure 3.3: Session

3.1.2 Multilayer Collaboration

At the right end of the toolbar in the session screen, there is a button for the action layers to be displayed. In Figure 3.4, we can see that *Action Layers* displays the users who are in this session as well as their status. The users, including the presenter and attendees, have their own action layer, and each layer is always independent of each other. The status on each layer indicates the actions performed by the user.

Two types of user status, visibility, and live status, could be found on the action layer. Users can disable any layers except the presenter's as invisible during a session, which ensures the users to be protected from interruptions or distractions from the layers of other users. Moreover, the presenter can broadcast and present any attendee's layer to all attendees, which helps to make sure the attendees are sharing the same content during a session.

In addition, the live status implies whether an attendee is following a session or not. A user can be in offline during a session while they might need some time to add markups or take notes. After they are done, they can come back online and catch up with the session.

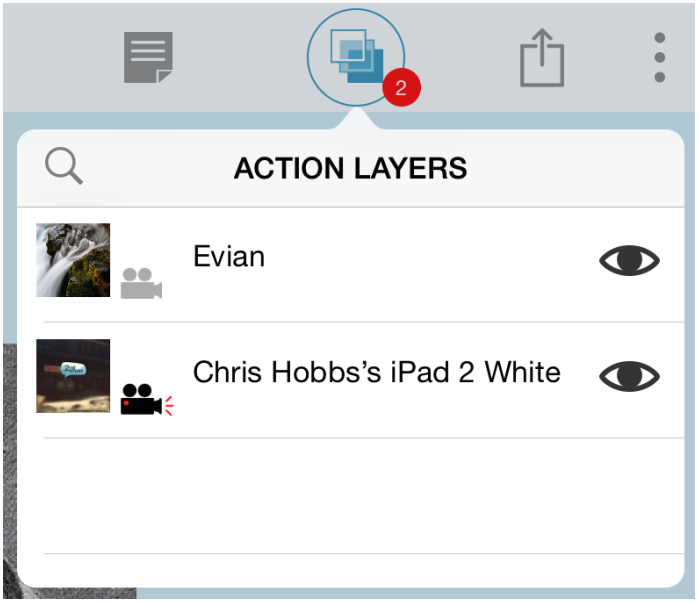


Figure 3.4: Action Layers

3.1.3 Session Replay

Session histories are displayed in a calendar as you can see in the Figure 3.5, where users can select a session on a certain date to replay and review. In the current business plan, there are two types of users. One is the basic user, also known as the free user, and another one is pro user. Basic users are only accessible to the last session they have while the pro users are allowed to access up to fifteen previous sessions. In session replay, a complete session would be replayed including all interactions like annotations or notes. Furthermore, users can even add more interactions during the session replay, which are very helpful to review the class lectures or any conference notes.

3.2 Demonstration

Two Tall Totems has conducted a demonstration of *Think Together* in an elementary school in Surrey, Canada. The demonstration has given some insights of how the teacher and students make use of *Think Together* in classrooms. Besides, feedbacks given from the participants are valuable as well, which helped us to understand the needs from customers point of view.

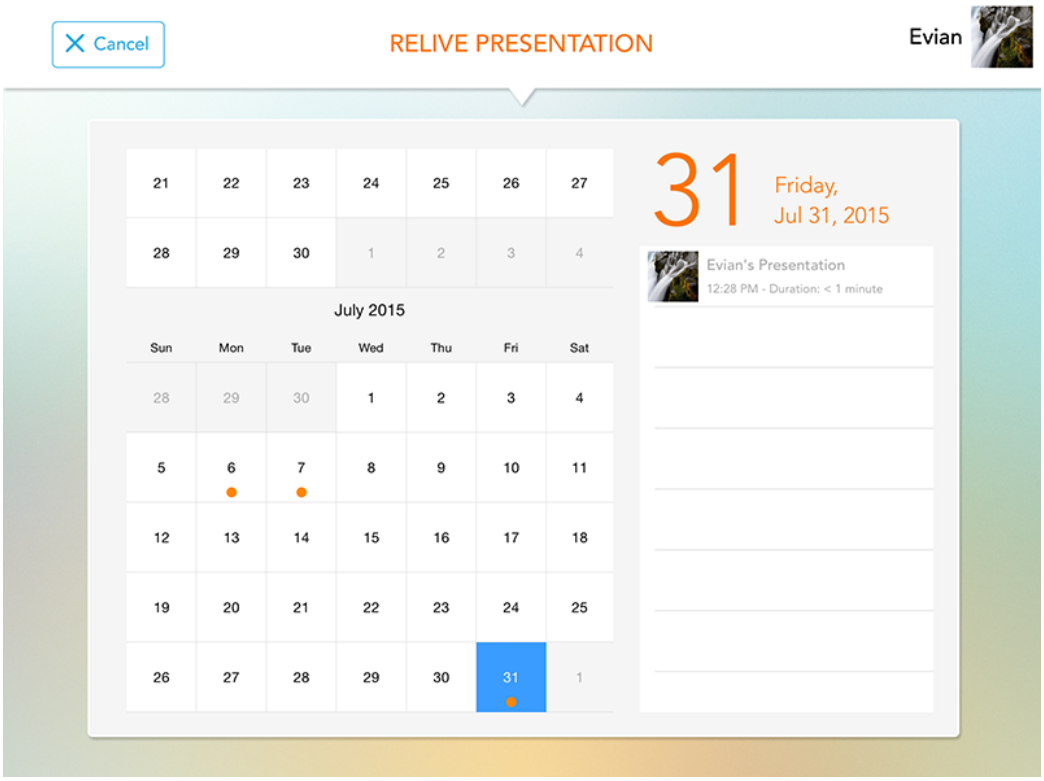


Figure 3.5: Session Replay

3.2.1 Setup

The demonstration was held in a class with 28 students and a teacher. An iPad installed *Think Together* was assigned to each of them. The teacher created a session for the lecture as a presenter, and students joined the session as attendees. After assuring of that all the students were involved, the teacher began the class with the session.

3.2.2 Findings

There are definitely valuable findings and feedbacks from the demonstration. To start with, students were found to be more excited and curious when using *Think Together*. They were able to understand and accept the App a lot faster than expected even without any instructions on how to use it. It was also exciting for them to discover the collaboration features with the action layer. They were exhilarating and joyful to either sharing their ideas and thoughts or noticing work from other students.

On the other hand, the teacher acknowledged that *Think Together* was definitely

beneficial in classrooms. She liked the multilayer feature because she could have control over all the layers and she could also see what the students were doing. Aside from that, she was also impressed by the App that allowed her to share students' notes or markups as well as students could have a look into work from others. The App engaged the students to connect with each other in the classroom, which books could not achieve. The collaboration between students made them more active and engaged since it is easier to share and work together. In addition, even for the students who were really shy, she pointed out that *Think Together* offered new communication options for them. More specifically, she could response to a student privately during a session, which was less embarrassing for these shy students.

(a) Showing a photo (b) Zooming in a photo

Figure 3.6: Synchronization

Moreover, we can see in Figure 3.6 that 24 iPads were having a session with a photo being shared. One iPad was acting as a presenter while the rest of iPads were attendees. From Figure 3.6a and Figure 3.6b, we can discover that the session was in a good synchronization.

3.3 Discussion

In Section 3.2, several findings of the demonstration are revealed to us. As shown in Table 3.1, we will discuss the findings and their consequences on the application.

Collaboration is a significant feature in *Think Together*. It is obvious that not only the teacher but also students are fond of using the App that enables them working together. Such essential feature of cooperation should include supports for audio or video to help users communicate better. However, it might burden the cloud server

Table 3.1: Findings of Demonstration

Findings	Consequences
Collaboration with audio/video	Heavy traffic flow on server
Interactive user experience	Everyone on synchronization in real time
	Require transmission efficiency and quality

once getting audio or video involved in the session. More specifically, the traffic flow in the network would increase quite a lot, especially with plenty of users and devices. The heavy load on the server is unavoidable which makes it the bottleneck of the system. It is necessary to enhance the performance in this aspect.

In addition, well-designed interactive user experience plays an important role in the collaboration. You will not enjoy working with other people when there is an apparent delay among the group. Accordingly, synchronization on an individual device should be close to real time as a part of user experience, which requires high transmission efficiency in the network. Therefore, we consider the transmission efficiency quality as an important performance metric.

Table 3.2: Typical research questions versus product management questions in exploratory study

	Exploratory Study	**Question Status**
Research Questions	Was learning more effective? What metrics are critical for identifying improved mode of education? How does this relate to other publications about technology in the classroom?	Not answered in depth. Formal user study would involve ethics, quantitative and qualitative evaluation.
Product Management Questions	Is there a market for the product? Does the product appeal to the target market? What are the competing products?	Answered sufficiently for market validation.

Besides, we demonstrate how the typical research questions and product management questions are answered in an exploratory study as shown in Table 3.2. In the exploratory study, the qualitative research questions are not answered while the

product management questions are all answered through the experiment. As a researcher, we use the qualitative research to discover and understand the insights of a problem [24]. It uncovers underlying reasons and reveals the thoughts and opinions in depth. As a product manager, however, it is not necessary to conduct a full qualitative research on the problem. Qualitative research is too time-consuming and costly for a product manager to afford though the results from the research are accurate and comprehensive. In this case, a qualitative research could have been conducted but the exploratory study is enough to answer the questions.

3.4 Conclusion

In this chapter, we presented features and highlights of *Think Together*. We also conducted an exploratory study on a demonstration with the application. Except the main feature of having a shared session, users could also have the experience to perform on their own action layers to express and share ideas or thoughts. Moreover, users could see other people's work to cooperate in a better way. Besides, a session is allowed to share to a third party who is not a part of the session, which provides more flexibility in collaboration.

Meanwhile, the usage of *Think Together* in the classroom is proven to be beneficial. Students were showing more active and focused behavior in the classroom with *Think Together* comparing to their behavior with books. They were also excited and delighted to share the work on their layers as well as discovering the work on others layers. Such a collaborative tool has a positive impact on students in this scenario. On the other hand, the teacher felt that the application was truly useful and helpful for her to have controls over the lecture session. She could interact with students and engage the students in the class more effectively. Therefore, the findings from the demonstration enlighten us from different perspectives. The potential needs for audio or video in this collaborative tool will lead to heavy traffic flow on the server. Moreover, high transmission efficiency is required in the system to provide a high-quality interactive user experience.

We have compared the typical research questions with the product management questions in the exploratory study. The product management questions are all answered sufficiently, but not the research questions. A qualitative study including securing ethics for user-testing is relatively time consuming from a product manager perspective working within an agile methodology.

Chapter 4

Network Configuration Design

This section includes an overview of the *Think Together* application architecture and various strategies of dynamic network constructions. The original architecture was conceived by the industry stakeholder, and the strategies have been developed during internship with the company as part of this thesis.

4.1 Application Architecture

Table 4.1 shows the overall application architecture in *Think Together* as five layers, organized as many collaborative applications that manage communication that essentially updates a shared view. We present the details of the architecture from the bottom layer to the top.

Table 4.1: Application Architecture of Think Together

Layer	Feature
APPLICATION	
MODEL (filter)	Messages
RECORDING (filter)	SQLite Time Machine
ROUTING	Handler Registration
TRANSPORT	Advertise/Browser Client/Server Encoding/Decoding

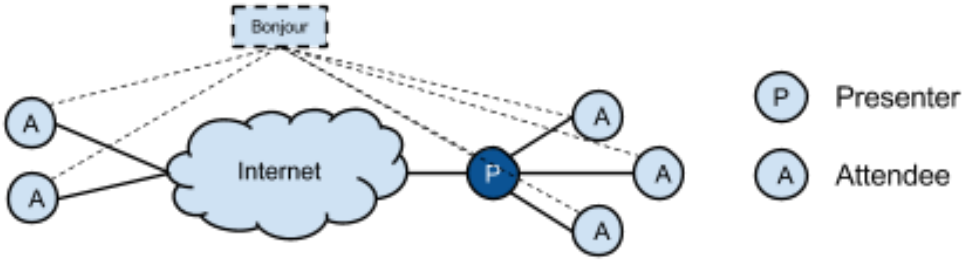


Figure 4.1: Preparation for session

TRANSPORT LAYER: The transport layer is the base layer of the architecture that directly connects to the network. It includes advertising and browsing service that broadcasts or discovers the session before a user can join. If the user is a presenter, the layer also acts as a server to process the messages received from clients. Additionally, this layer is responsible for encoding outgoing messages and decoding incoming messages.

ROUTING LAYER: The application on different devices communicates by sending messages to update each other's shared state. There are various types of messages in this application, such as note message or annotation messages. Each type of message requires a handler, and the routing layer needs to register the handler.

RECORDING LAYER: The recording layer is mainly used as data storage. The application has SQLite on this layer, which is the database for the data generated by users. Furthermore, management of session data and timelines are also implemented on this layer.

MODEL LAYER: To avoid massive messages routing all over the network, the model layer is added as a life saver. Messages are filtered by the model layer and be passed to the correct handler to process.

APPLICATION LAYER: The application layer is the top layer in the architecture, which contains functionalities of user interface and interactions.

4.1.1 Bonjour

Bonjour is a service provided by Apple that is used for session discovery in the application. The presenter is always advertising the session as a server, and the attendees are able to find the session through the Bonjour service on iOS. The advantage of adopting this service is that it can be plugged in as a component that is known to work well. The disadvantage is that this service is specific to this vendor, and moving the application to another platform would require changing this service.

The remainder of this chapter represents the portion of the planning for the future of the application that constitutes the case study of this thesis. That is, we consider the issue of scaling the application as a problem encountered from two perspectives: the project manager versus a researcher.

4.2 Network Configuration Design Overview

There are currently two types of users in the application, a presenter, and attendee(s). In order to better scale this into a more decentralized organization, we propose to have new roles for users in the configuration. In Table 4.2, we compare the old roles to the new ones in terms of user types.

Table 4.2: Comparison of Roles in Old Model and New Model

Type of users	Old Role	New Role
Presenter	Server	Relayer/Listener
Attendee	Listener	Relayer/Listener

There are three types of roles that integrate together in the new model. The first role is a *listener* which is an end user in a session. The second role is a *relayer* which represents both an end user and a distributor who is responsible for distributing messages to other listeners to increase efficiency. Both presenter and attendees can be either a listener or a relayer according to their own network situation in order to provide the best efficiency in the application. The third role is a *server* which originally was the presenter's iPad.

Due to the needs of a large scale system, the server can no longer be placed in an iPad that does not have enough power to process and distribute to many peers. An iPad is also not able to store a massive amount of data from sessions or interactions

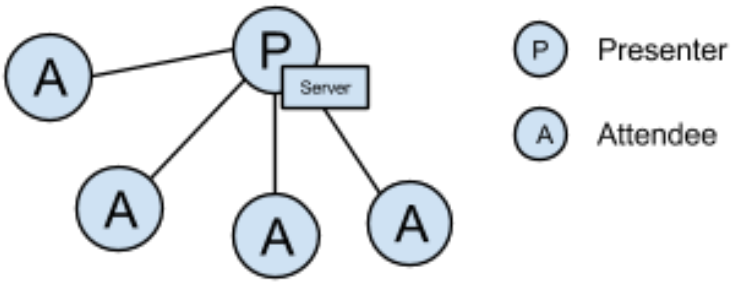


Figure 4.2: Original Configuration

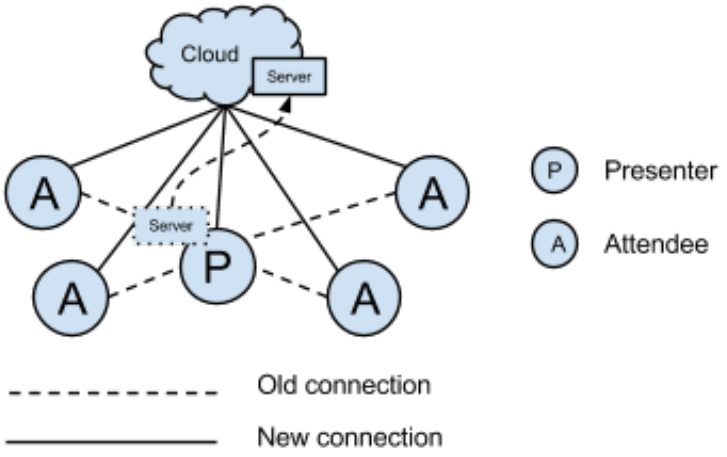


Figure 4.3: Move server to cloud

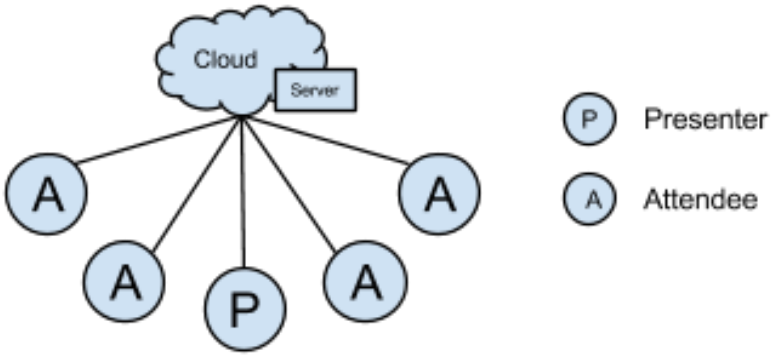


Figure 4.4: New Configuration

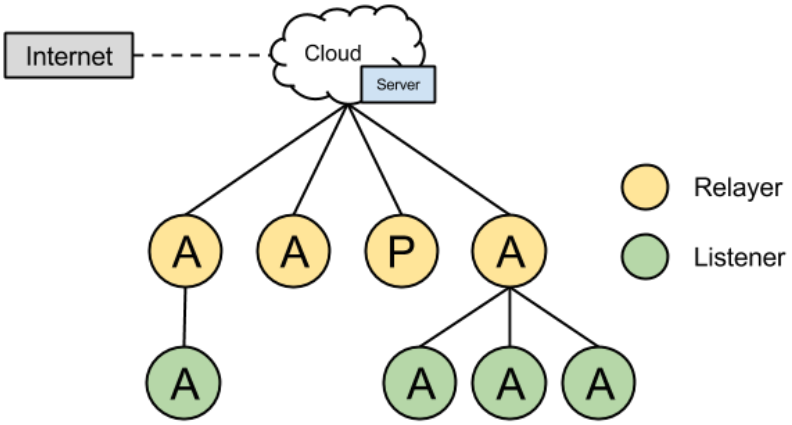


Figure 4.5: Model Design in Local Network

while all peers are connected to the presenter. The old model that is presented in Figure 4.2 is more suitable in a relatively small network but not in a scalable network. Therefore, we bring the server up to the cloud as shown in Figure 4.3 to reduce the traffic on presenter's device. In Figure 4.4, the server in the new model has more power to process and handle the data from all users during the session. Also, the strategy to move the server brings more potential features since it can serve any user through the Internet. The disadvantages are the increased latency that results from having to potentially connect to a server that is very far away.

As for the desire to create a scalable network, we introduce the concept of the *relayer*. The relayer is similar to the supernode in P2P network. The dynamic adaption between the nodes based on network condition can help to reduce the redundancy in the network.

In a local network, relayers connect to the cloud server and distribute the data to all the listeners in a hierarchical architecture as Figure 4.5 indicates. All the annotation messages, ping messages or media stream from other relayers are allocated through the relayers to the listeners from the cloud server. Correspondingly, listeners upload all of the action data to their parent relayer, which will be packed up with relayer's data and sent to cloud server together in a batched strategy.

The relayers can be either a presenter or an attendee based on the network situation. The roles are dynamic so that the dissemination tree structure could achieve the optimized efficiency. A relayer can become a listener if, for example, their network is slow at that moment. Similarly, a listener can become a relayer if more bandwidth

is available. However, a listener who is in an unsatisfied network condition has no ability to spare any bandwidth like a relayer. As we proposed, listeners can connect to a relayer in the local network according to the dynamic situation at the moment for the best efficiency. It can be a common scenario that the users in a session are in the same classroom, where they probably share one local network.

The server gathers data from the relayers and keep them as the record in the database on cloud. It is assumed that there will not be any obstacles, including costs, that would prevent users to access the data through the Internet. More specifically, this architecture enables a lot more potential areas we can enhance the application. For instance, it would be able to support the live web browser for watching the session without installing any application.

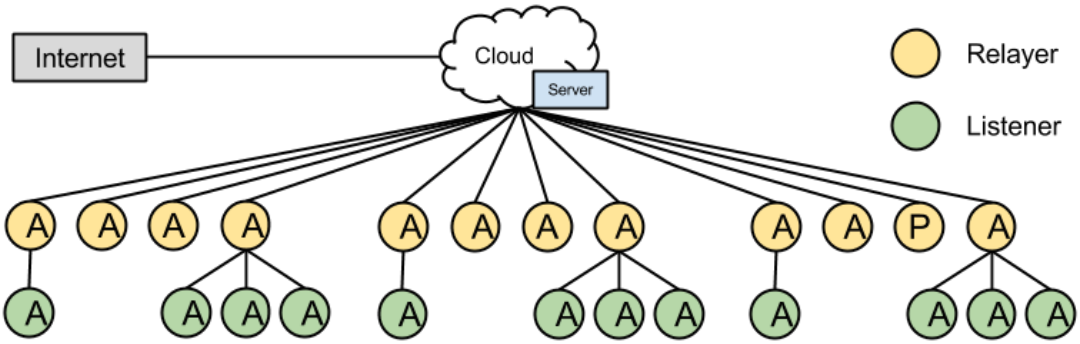


Figure 4.6: Big Picture of the Model Design

The big picture of the model is described as Figure 4.6, where the cloud server, which could itself be elastic and scaled, is connected to four parts, three local networks and the Internet. Each part of local network is explained earlier on page 26 referring to Figure 4.5. The synchronization of devices in a local network is more critical and would be designed to minimize the delay as that would have a major impact on user experience. The model is designed to serve the needs of local efficiency.

Take an educational scenario as an example scenario, where the application is being used in a lecture in a classroom. The presenter is an instructor who gives the lecture with *Think Together*, and the attendees are students who are in the classroom. When the presenter creates a session, the students in the classroom are in the same local network, as well as the ones who know the connection number and the password of the session could have access to the session. The lecture material is uploaded by the presenter to the cloud server and be stored as well as shared out from server to

all the listeners through their parent relayers. Instead of downloading and uploading data directly through router individually, some users need to take the lead as a relayer since they are in a better network condition which could help the system to lower the burden by grouping listeners. The server could always provide a backup service because it works in a stable status. Either listeners or relayers could retrieve the missing content from the cloud server in case of any data lost due to situations like network interrupt or disconnection.

4.3 Maintenance Policies

The section presents details about key concerns in the network configuration design and proposes related maintenance policies. The section includes policies of the peers on joining, leaving, recovering from unexpected quitting from the system, as well as peer adaption. For a project manager, it would be important to include these policies as part of the project plan. Though these are admittedly highly simplistic scenarios, and issues of determining behaviour when there are consensus issues and timing dependencies would also have to be considered, they are considered out of the scope of this initial first step which is in line with the simplest incremental change that a project manager might take.

4.3.1 Peer Join Policies

The procedure of joining a session is similar to that of a node joining a P2P network with supernodes. Our peer connects with the server to let the server know it wants to join, which is like the node in the P2P network registering with the authentication server. The cloud server is able to maintain all the peers and up-to-date status in the network. During the initial negotiation stage of a peer joining, the server would have to also have the knowledge of the network states of the joining peer. The network condition incorporates factors like round-trip latency, bandwidth as well as throughput. The server determines whether a peer is suitable to be a relayer by evaluating its network environment. Providing that the network setup is ideal to act as a supernode, the peer would be assigned by the server as a relayer, otherwise a listener.

If the peer is allocated as a relayer, it needs to take the responsibility to bridge the communication between the server and the listeners. In the worst case no node

meets the minimal criteria to be a relayer. If the load requires that one be selected, the policy would resort to the node that would do the least amount of harm.

Supposing that the peer is delegated as a listener, it will be given a list of potential parent relayers as well. The server, itself, would always be the last one in the list as a backup. Given the parent linking list, the peer starts to reach out to the nodes based on the order in the list. The peer would try to connect to the next node in the list when the current one is no longer available. It is very unlikely that failure occurs in every connection so as to prevent unexpected single point failure on relayer in the network. After establishing the interdependence with the parent relayer, the peer would transfer all data and information through the relayer.

We offer the detailed algorithm of a peer joining a session. The procedure Algorithm 1 is displayed as the following.

Algorithm 1 Peer joining a session

```
1: set parameter max waiting time;
2: set peer.position as listener;
3: create hello message including current network information;
4: peer sends the hello message to server;
5: while used time > max waiting time do
6:     if received response then
7:         set peer.position to response;
8:         break;
9:     end if
10: end while
11: if peer.position = listener then
12:     peer sends join request to server;
13:     while used time > max waiting time do
14:         if received response then
15:             set response to peer.parentList;
16:             break;
17:         end if
18:     end while
19: else
20:     set peer.parentList[0] as server;
21: end if
22: Send ACK message to peer.parentList[0] to initiate the connection;
```

4.3.2 Peer Leave Policies

There are two types of leaving strategy that a peer may have in our configuration, which are leaving the network gracefully and ungracefully. It is apparent that each of these methods result in different behaviors. Additionally, it can be shown to be impossible to determine if a peer is slow or actually no longer functioning.

On one hand, a peer is intended to leave the network gracefully in order to allow other components to give necessary response in a reasonable time. Accordingly, the peer sends the message informing the server that it is leaving first, and server can get prepared to react and make adjustment in the network.

If the leaving peer is a listener, it is required to notify the parent relayer node. The knowledge of the leaving listener node ensures relayer to remove this listener node and update the listener list. By removing the listener node, the relayer discontinues sending data and information to the listener node so as to avoid consuming the resource, such as bandwidth. If the leaving peer is a relayer, the leaving message would be dispatched to all the children listener nodes. Hence, the listeners can get ready for parent relayer leaving to ensure that the connection to server is not broken and always available. Usually, the listeners would go through their backup relayer list and connect to the next available relayer. After all children listeners are switched to other relayers, the leaving peer can depart from the network gracefully.

The explicit algorithm for a peer leaving the network gracefully is illustrated as Algorithm 2.

Algorithm 2 Peer leaving a session gracefully

1: *peer* sends leaving message to *server*;
2: **if** *peer.position* = *listener* **then**
3: *peer* sends leaving message to parent *relayer*;
4: *relayer* updates *childrenList* that removes *peer* from the list;
5: **else if** *peer.position* = *relayer* **then**
6: *peer* sends leaving message to children *listener* nodes;
7: *listener* nodes connect to *server* to re-join;
8: **end if**
9: disconnect and remove *peer* from the network;

On the other hand, it is practical to consider the case that a peer leaves the network ungracefully by accidentally quitting the application or losing network connection. In such circumstances, other components need to react with proper behavior in the network. Normally, any peer will monitor the status of children to inspect if the

nodes are active or not if it has any. On that account, there are three scenarios we need to take into consideration.

The first is that the listener discovers that the parent relayer is missing or gone. The listener fails to retain the connection to the network when missing the parent relayer because the parent relayer is the connection between the listener and all other nodes in the network. Nonetheless, every listener has a backup list of relayers and the listener could obtain next available relayer in the list to associate with. Secondly, the server might discover that the relayer is absent because of the regular check on relayers. As stated by the first case, the listeners should try to connect to other relayers when their own relayer is missing. The server will follow up with the children listener nodes to verify whether the listeners are relocated as designed. If not, the server will help the listener to re-join the network. While all the listeners are settled on, the server removes the disconnected relayer from the relayer list. The third situation is that the relayer recognizes that a listener leaves the network suddenly. This is a simpler scenario due to that listeners have a lot fewer responsibilities in the network compared with relayers. Owing to the fact that the listener only joins up to a relayer in the network, the relayer will share the information with the server once being aware of the lost listener. Subsequently, the relayer will remove this listener from the children listener list to keep itself updated and avoid any redundancy in the network.

We describe the a peer leaving a session ungracefully as algorithm 3 indicates.

Algorithm 3 Peer leaving a session ungracefully

1: **if** *listener* discover a *relayer peer* leaving **then**
2: *listener* nodes connect to *server* to re-join;
3: **else if** *server* discover a *relayer peer* leaving **then**
4: *server* asks children nodes of *peer* to re-join;
5: **else if** *relayer* discover a *listener peer* leaving **then**
6: *relayer* notifies *server* of leaving *peer*;
7: *relayer* removes *listener* from the connection list;
8: **end if**

4.3.3 Peer Recovery Policies

When a peer disengages with the application and expects to recover from where it left off, the peer is able to catch up with the session in the process and send annotation

data to the server as well as other nodes. In this situation, we consider the peer leaving the network gracefully as algorithm 2 outlines, and it will join another time as algorithm 1 reveals.

4.3.4 Peer Adaption Policies

The cloud server is taking charge of keeping the status of the network under surveillance. As a result, the server monitors each peer periodically, and the peers must report their network perspective [17]. By evaluating the condition of each peer, the server could dynamically adapt the structure in order to achieve the highest productivity and efficiency. Though this approach may have benefits in overall efficiency, this would have to be balanced by the costs in a reconfiguration scenario, and avoid the situation where reconfiguration is always happening.

From the perspective of a relayer, this role demands for high bandwidth consumption, throughput, etc. The server has to put a relayer to the lower position, listener, if the relayer can no longer hold up such a prerequisite. The scarcity of resources to support the requirements is likely to result in poor performance in the network as well as unsatisfactory user experience with the application. As a consequence, the peer would be appointed from relayer to listener when the network situation is not desirable. Again, policies to avoid continuous reassignment in the case where a node is constantly in that threshold would have to be put in place.

During the adaption from relayer to listener, it is obligatory to assign the children listener nodes to other relayers or assign them back to the server. It is almost identical to the procedure of a peer leaving gracefully in Algorithm 2. Following that, the peer should have received response from the server including the backup relayer list and store the list. With the given list, the peer could connect to the relayer, which is similar as Algorithm 1 indicates.

From the point of view of a listener, it will receive a message from the server to adjust to another position only if the server is convinced that the resource the peer has is sufficient to serve as a relayer between listeners and server. Similar to the former viewpoint of relayer, turning a listener to relayer also depends on the network condition.

If a peer is going to be changed from a listener to a relayer, the response from the server would be notified to the peer. When receiving the response from the server, the peer would set the first value in backup relayer list as the server. As a result, the

peer could communicate with the server directly as a relayer. Furthermore, the peer would inform the parent relayer node of the transforming as a listener, that is close to peer leaving gracefully in Algorithm 2 as well. On that account, the parent relayer of the peer could remove the peer and update the children list in time to prevent a waste of exchanging data.

In Algorithm 4, we demonstrate that how the peer is adapted and assure the efficiency in the network.

Algorithm 4 Peer Adaption

```
1: set interval time
2: for every interval time do
3:     server pings relayer and listener nodes to collect information on their network condition
4:     if peer.position = relayer and is in an undesirable network situation then
5:         send leaving message to children listener nodes;
6:         receive response from server;
7:         set response to peer.parentList;
8:         set relayer.position as listener;
9:     else if peer.position = listener and is in a desirable network situation then
10:        receive response from server;
11:        set peer.parentList[0] to server;
12:        inform parent relayer to remove peer from his children list;
13:        set relayer.position as relayer;
14:    end if
15: end for
```

Chapter 5

Evaluation and Analysis

In this chapter, we quantitatively evaluate the efficiency of the new network configuration that we proposed for *Think Together* with a simulator. Admittedly, a project manager may not invest the time to build such a tool, however, we are exploring the ways in which tools such as this can inform project planning and be beneficial without introducing unnecessary costs.

There are two parts of this experiment. First we set up the simulator with default parameters that may or may not have meaningful representation under real workloads with the application. The point is that these parameters can be made to be easily changed, so that a product manager can use real data in further iterations of application development. We then analyze the performance with various impact factors of our network configuration and compare the configuration to a classic client-server model. Our experiment methodology, simulation metrics as well as approaches are described for the performance evaluation in both parts. Furthermore, we present results from a sequence of simulation experiments using simple tools that product managers would be using for other elements of the management landscape. This integration in the tool chain may reduce the costs of adoption of the simulation framework in the long run.

5.1 Methodology

We have installed OMNet++ as the simulation platform on Mac OS X. It is more practical to experiment with a simulator than real devices due to the limitation of the hardware resources. Thus, a simulator is a good choice to solve these problems if

it can be made to be an accurate enough portrayal of the real world factors involved; also, it can help us evaluate the network configuration in different perspectives. The flexibility of this approach enables potential research in a large network or with a large number of active nodes at scale. We also set up a traditional client-server model [9] with the same basic network setup for comparison.

5.1.1 Network Setup

In OMNet++, a NED (Network Description) file is used to describe a topology in the NED language, which defines module or labels modules as a network with plain syntax [3]. The NED file mainly defines components like modules and channel connections.

In our experiment, we create instances of nodes, relayers and a server. We use simple module type *T3_Node* to represent the nodes, *T3_Relayer* to represent the relayers, *T3_Server* to represent the server. They are represented by their corresponding C++ class. We use channels to implement in a naive way to connect instances. We apply a predefined type *ned.DatarateChannel* to specify parameters and behavior associated with connections.

Setup for Network Configuration

As we proposed earlier, the network configuration mainly includes a server as a server node, multiple relayers as relayer nodes and multiple listeners as leaf nodes. As we can see from Figure 5.1, we define a network that simulates the structure we proposed. In this network, there are three simple modules as components: a server host, multiple relayer hosts, and multiple node hosts. This simple decomposition is something that a non-expert project manager could identify without much of a learning curve, though the language itself does require some learning.

The server host represents the cloud server in the new network configuration, and is responsible for receiving data from client nodes and distributing the data. In our case, the server will take the data from a relayer and forward it to all other relayers. The relayer host represents the relayer node. It is the middle node between the server and the listener. Firstly, a relayer can act as a client node, generate data and send out data. It sends out not only to a cloud server but also to the local listeners connected with it. Secondly, the relayer needs to handle the data received from the cloud server as well as from local listeners. It needs to forward the data from local listeners to the cloud server and distribute the data from the cloud server to local listeners.

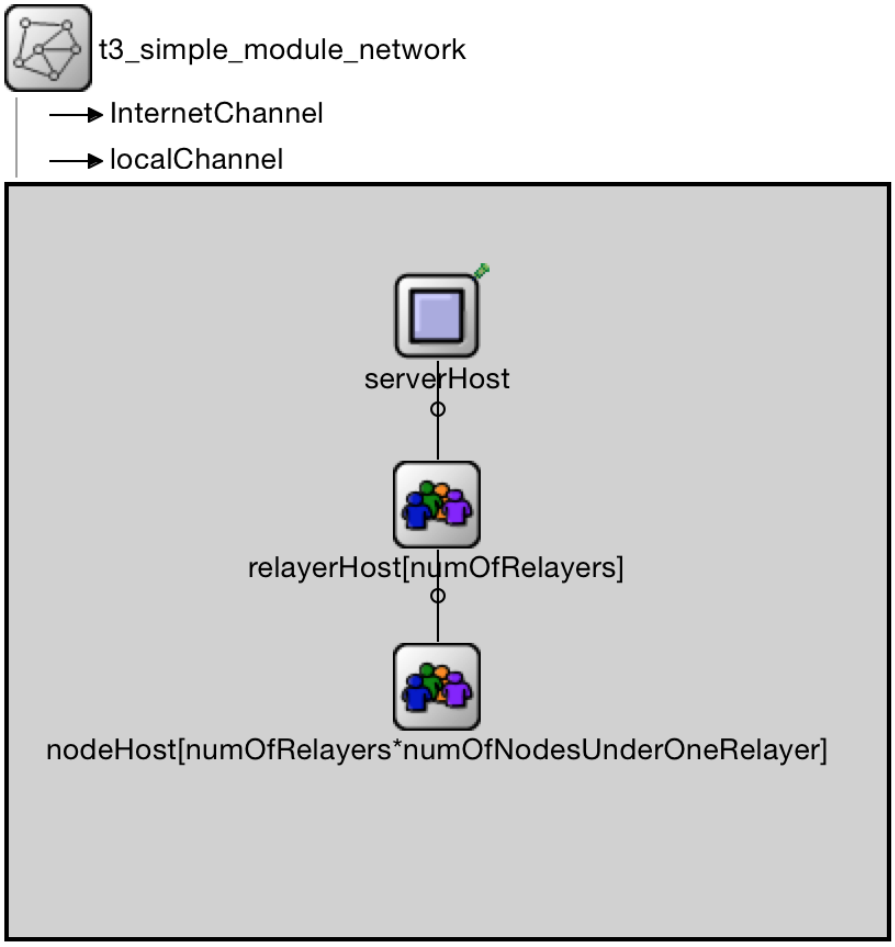


Figure 5.1: Graphic overview of the network structure

Table 5.1: Connection from OMNet++ for basic network setup

Connection
serverHost <—>InternetChannel <—>relayerHost[..]
relayerHost[..] <—>localChannel <—>nodeHost[..]

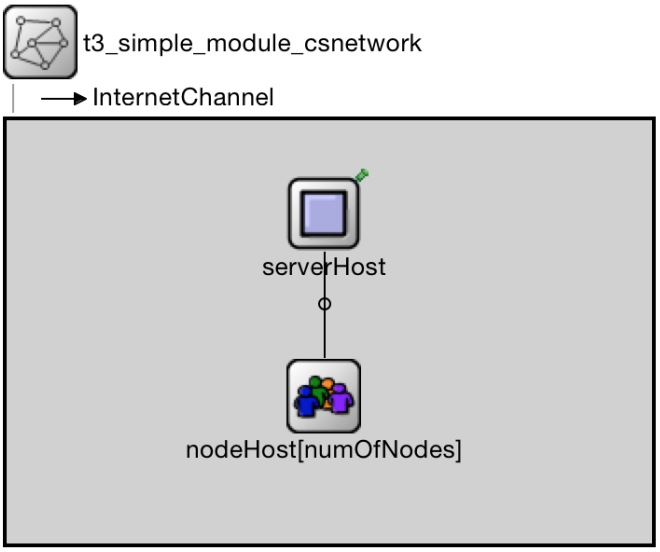


Figure 5.2: Graphic overview of the network structure

Table 5.2: Connection from OMNet++ for Client-Server network setup

Connection
serverHost <—>InternetChannel <—>nodeHost[..]

There are two types of connections required in this network. The connections are defined in Table 5.1. A connection is handled through a channel. The serverHost has to connect to each of the relayerHosts through the Internet because the serverHost acts as a cloud server. Thus, we named the channel the InternetChannel. Since we consider a relayer and its listeners to be all on the same local network, their connections should be under the standards of a local network. Here we named the channel as the localChannel.

Network Setup for Client-Server Model

In order to conduct a comparison in next stage, we propose another traditional client-server network structure to help evaluate and compare the simulation results of our network configuration.

We define a client-server network structure as in Figure 5.2. The basic network setup is the same as our configuration without relayerHost. Here, nodeHost is acting as a client and serverHost is acting as a cloud server.

Table 5.3: Configuration parameters for network setup

Parameter Name	Value
sim-time-limit	20s
numOfRelayers	variable
numOfNodesUnderOneRelayer	variable

In Table 5.2, we show that the server host connects to the relayer hosts through the channel that connects to the node hosts in the client-server model. They are probably not in a local network because the server is on the cloud so the InternetChannel is applied in this case. We believe that this setup is reasonable and would correspond to expectations of a product manager looking at the setup. We have also chosen the client-server model to be the same basic network setup, ensuring that we are controlling all environment variables in a similar way, and only comparing the differences between two structures in this initial phase.

5.1.2 Parameters Setup

An OMNet++ project requires a NED file to setup the network environment and an INI file to configure the setup. We need both in our evaluation.

Parameters for Network Configuration

In Table 5.3, we can see that we have four parameters affecting the experiment in this case. These correspond to key issues identified in the exploratory study presented in the previous chapter. Though they are not exhaustive, they serve as a basis upon which we can consider this configuration from a product management perspective in the context of agile methodologies. They are as follows:

sim-time-limit: This is the parameter that controls the running time of the simulation. The simulation will not stop running until it reaches the limit or some error occurs in the system. We set the sim-time-limit to 20 seconds for each test run to prevent the simulator from running forever. The same simulation running time guarantees the same standard for different setups so that we are able to compare and analyze the results.

network: This parameter is required if there are different network setups. The network is defined and specified in a NED file, where one can designate multiple

Table 5.4: Configuration value for variable numOfNodesUnderOneRelayer

Network Size	20	30	40	50	60	70	80	90	100	110	120
1 Relayer	19	29	39	49	59	69	79	89	99	109	119
5 Relayers	3	5	7	9	11	13	15	17	19	21	23
10 Relayers	1	2	3	4	5	6	7	8	9	10	11
15 Relayers	-	1	-	-	3	-	-	5	-	-	7
20 Relayers	0	-	1	-	2	-	3	-	4	-	5

networks. In our case, this parameter is used to specify the name of the network we want to use in the simulation.

numOfRelayers: This is a parameter that we defined in NED file to determine the number of relayers in the network. It can be either specified in an INI file or entered through a GUI during runtime.

numOfNodesUnderOneRelayer: This is a parameter that we defined in NED file to determine the number of nodes each relayer has as child nodes. With the multiplication of numOfRelayers and numOfNodesUnderOneRelayer, we can get the number of listener nodes in the network and the network size. Similar to the parameter numOfRelayers, it also can be either specified in INI file or entered during runtime. The convenience of the GUI from the product manager's perspective is that it may reduce the learning curve involved in using the simulator.

We need to revisit different sets of parameter variables in the experiment to achieve more accurate and practical results. In order to make comparisons between different setup parameters, we set the network size, which includes the number of relayers and number of listeners, as a certain value. The network size varies from 20 to 120, which can simulate a small group to a large group. In each run, the network size increases by 10. The relation between performance and network size can be observed with these setups. We assign different values to numOfRelayers so as to find how the different number of relayers affects the performance in the network. In addition, we can get the value for variable numOfNodesUnderOneRelayer displayed in detail in Table 5.4 based on the network size and the number of relayers planned.

Configuration for Client-Server Model

Table 5.5, shows three key parameters we are isolating in our client-server network.

sim-time-limit: This parameter is set the same as the configuration and can be

Table 5.5: Configuration parameters for network setup

Parameter Name	Value
sim-time-limit	20s
network	client_server_network
numOfNodes	variable

Table 5.6: Configuration value for variable numOfNodes

Network Size	20	30	40	50	60	70	80	90	100	110	120
numOfNodes	20	30	40	50	60	70	80	90	100	110	120

Table 5.7: Simulation Metrics

Category	Metric
Productivity	Server Bandwidth Consumption
Responsiveness	Packet Life Time

referred to in Table 5.3.

network: This parameter is set the same as the configuration and can be referred to in Table 5.3.

numOfNodes: This is a parameter that we defined in NED file to determine the number of nodes in the network.

In the client-server model, we set the network size the same in our network configuration. The network size starts from 20, increasing by 10 until 120. Because the network model only involves listener nodes, the number of nodes is the same as the network size as shown in Table 5.6.

5.1.3 Simulation Metrics

Though this is a proof-of-concept experiment, it serves to provide prospectives that align the goals of both project management and research. As Table 5.7 indicates, we have evaluated our configuration with the following metrics. These are the key metrics from different perspectives that help us with performance analysis.

From the view of productivity, we apply the server bandwidth consumption to measure the capacity of the configuration. The bandwidth consumed on server side is the bottleneck of the original network. It helps to improve the performance in the whole network if our configuration lightens the burden on the server. The simulator gives as an indication of what the tradeoffs might be.

From the view of responsiveness, we use packet lifetime in the network as the measure. The duration of a packet that exists in the network reveals how efficiently a packet will be transferred in the network. The comparison between different network structures explains how the configuration influences the performance from this point

of view.

5.1.4 Simulation Procedure

There are three parts of the procedure of the simulation. In the first part, we evaluate server bandwidth consumption for productivity performance. This metric is evaluated based on the number of packets forwarded by the cloud server that helps us to estimate the bandwidth consumed by the server. With our network and configuration setup earlier, all the cases with different network sizes, the number of relayers and number of nodes will be explored. The experiment of how the components in the system are distributed affect the productivity performance will be conducted with different sets of experiments. Moreover, the comparison with the client-server model will also provide a relative perspective on performance improvements.

In the second part, we evaluate packet life time for the efficiency of responsiveness. We collect the packet life time data from each component, which includes packet life-time overall, packet life time in local networks and packet life time in wide networks. Similar to previous parts, we also run through every combination of the setup that specified the number of relayers and number of listeners. Once the initial configuration has been built, the job of collecting data is somewhat trivial, and therefore allows for product managers to inexpensively generate informative data sets.

5.2 Results and Analysis

Server Bandwidth Consumption: This experiment is to appraise the bandwidth consumption in server host. We evaluate the number of packets the server sent in different network configurations, which we use to represent the bandwidth consumption.

As described in Figure 5.3, we can observe the data trend with different setup of network size and distribution. Not surprisingly, these results confirm that the larger the network size is, more packets are sent from the server. It is natural that the server needs to send out more packets when there are more client nodes communicating with a server in a larger network. The question is, at what point is the server overloaded? In order to answer that, testing in a practical sense will assist in setting the parameters for the simulation. Overall, that question is not answered in this simple case.

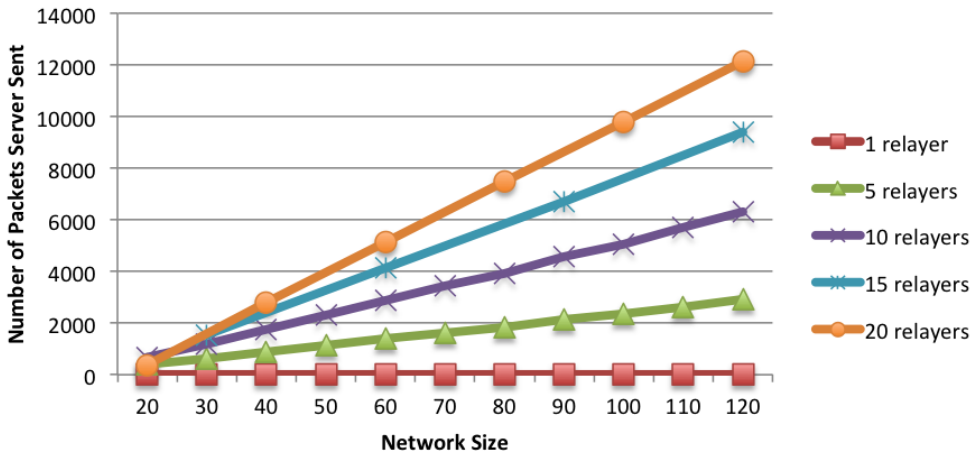


Figure 5.3: Server Bandwidth Consumption In Different Network Size

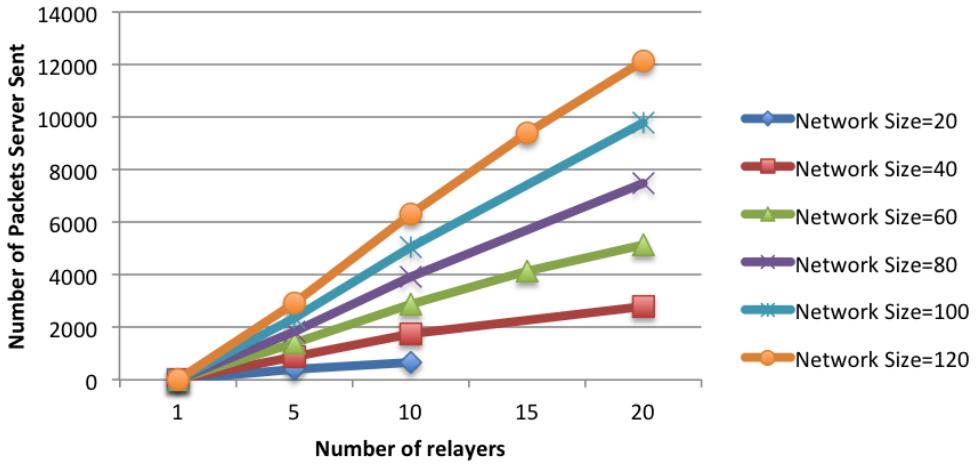


Figure 5.4: Server Bandwidth Consumption With Different Number Of Relayers

However, there there appears to be an exception if the network has only one relayer. As illustrated in Figure 5.3, the number of packets the server sent when there is one relayer is always zero no matter what network size is in this case. The reason is that all the other client nodes except the relayer node are listener nodes, who always communicate with the relayer instead of a cloud server. The communication between all the client nodes is in a local network while the only communication through the Internet is the link from relayer node to the cloud server. More specifically, in this set up, the server only receives from the relayer node and does not need to forward any packet. This is a huge advantage for the server as a special scenario in terms of upload bandwidth consumption for the server when all the clients exchange data through the relayer in a local network. Again, this highlights a trend for a project manager to consider, but realistic parameters would be required for rigorous research of behaviour with a more detailed setup.

Furthermore, we can discover that when the number of relayers gets larger, the server will send more packets out in accordance with the lines in Figure 5.3. This would indicate that there is an advantage to keeping the number of relayers small. Again this is a trend, and not a detailed analysis. However, with more realistic parameters this trend could be explored in more detail.

From another point of view, it is important to consider the relation between network size and the number of relayers as Figure 5.4 illustrates.

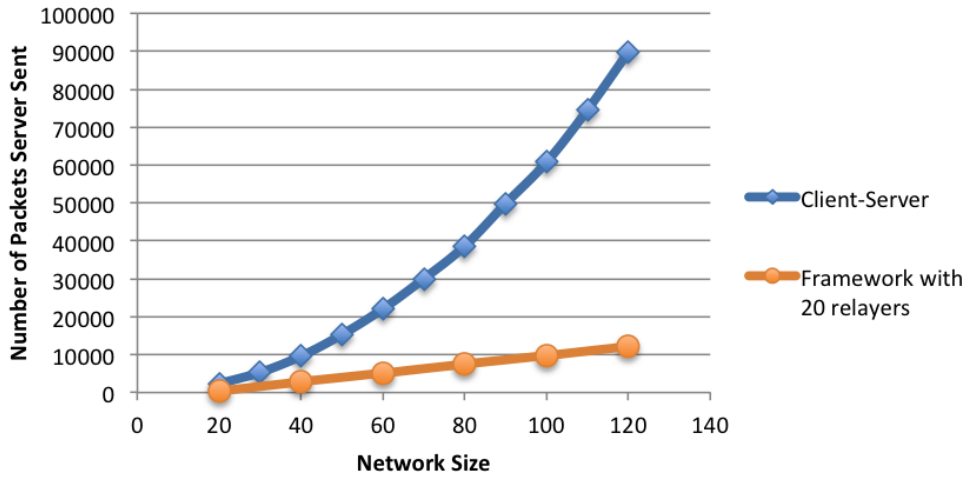


Figure 5.5: Server Bandwidth Consumption Comparison With Client-Server Network

Similar to Figure 5.3, this figure shows a clear increasing trend for the number of relayers. With the different size of the network, the number of relayers builds up the number of packets the server sends. It follows that the server utilizes more bandwidth when there are more relayers in the network. Bearing in mind network size may grow quickly as participants come online, we might have need of making use of the number of relayers.

Next, we compare the configuration with 20 relayers with the client-server model network.

The scenario with 20 relayers has the worst performance in the sense that the server sends out more packets than any other scenarios in compliance with Figure 5.3. This means that the performance of server bandwidth consumption is much less favorable as Figure 5.5 demonstrates. The trend of the bandwidth consumption is also associated with network size in client-server Network. The server sends out more packets with more client nodes. Though this is not surprising, the ability to more accurately quantify and express these trends could be of great benefit to a product manager communicating with stakeholders who are investing in the next iteration of the application.

Comparing with our configuration with 20 relayers, we can discern that the number of packets the server sent dropped dramatically. Our network configuration reserves a great amount of bandwidth which brings the potential for

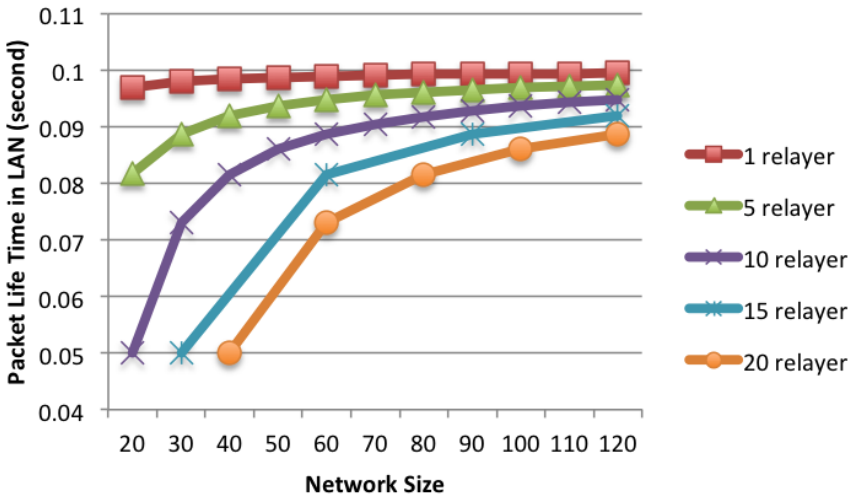


Figure 5.6: Packet Life Time In LAN

better network utilization in the system.

Packet Life Time: Packet life time represents the length of time that a packet exists in the network, from when the packet is sent to when the packet is received. The evaluation of packet life time in the network is for analyzing the responsive efficiency of our configuration. The packet life time behaves differently in LAN and WAN because of varying delay in each of the network. Therefore, we look at the packet life time in LAN, in WAN and overall.

First of all, we are going to start from looking at the packet life time in LAN given different network sizes.

In this case, we still have the setup for altered network size with five numbers of relayers. As shown in Figure 5.6, there is a clear trend that the packet life time rises when the network size gets bigger no matter how many relayers the network has. Equally, we can see that the packet life time is reaching close to 0.1s which is the network transfer limitation we defined in the experiment. This can be modified for further experiments. Accordingly, the small network size will have shorter packet life time and better performance in LAN traffic transmission.

The pattern of the influence from the number of relayers is straightforward to be noticed from Figure 5.6. The line with fewer relayers is always above the line with more relayers. Thereby, the packet life time is shorter if the listener

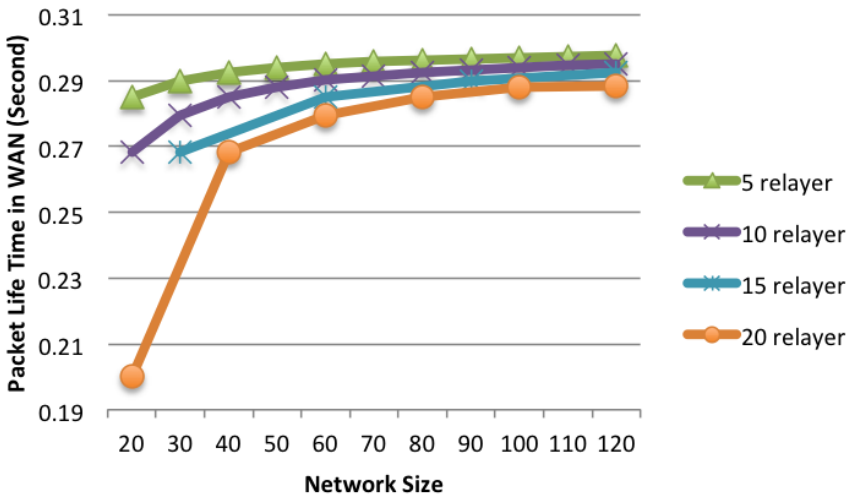


Figure 5.7: Packet Life Time In WAN

nodes are allocated by more relayers. This confirms the intuition that relayers help to improve the efficiency of the network.

As Figure 5.7 exhibits, the tendency of the graph is similar to the one in Figure 5.6 of packet life time in LAN with multiple relayers. In WAN, the growth of the network size extends the packet life time as well. Additionally, with increasing number of relayers, the packet life time climbs close to 0.30 seconds, which is the limitation in the network setup. Consequently, it is reasonable to draw the same conclusion as the experiment on packet life time in LAN. The network size escalates the packet life time which slows down the performance. More relayers give assistance to share the responsibility to transfer packets, but at a cost. The simulator allows both the researcher and the product manager to explore costs precisely. While the researcher may want to consider a wide spectrum, the practitioner may want to focus on metrics that specifically apply to user experience with the application in mind.

In the same experiment scenario, we contrast our network configuration with the simple client-server model. The packets in a client-server network always take less time to be received than in our configuration. As we can see in Figure 5.8, the packet life time with 20 relayers, which is the lowest result shown in Figure 5.7, is always greater than the packet life time in the client-server network except the case of 20 as network size. This is a special case that all 20 nodes

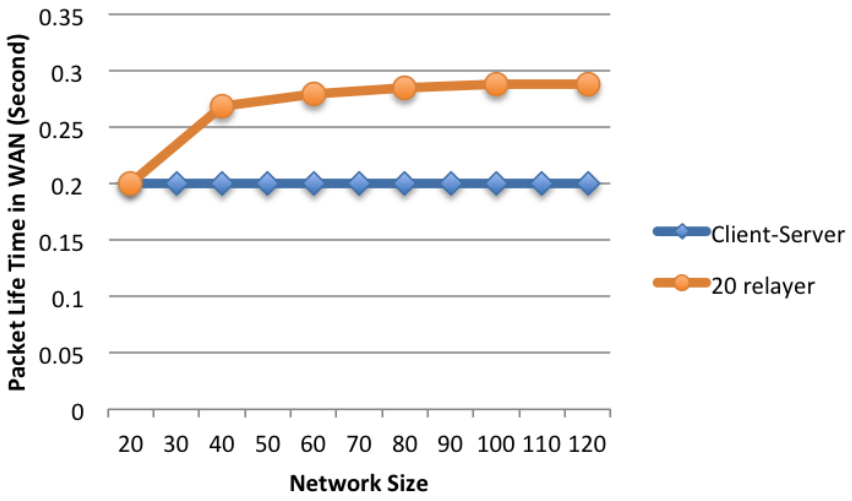


Figure 5.8: Comparison Of Packet Life Time In WAN

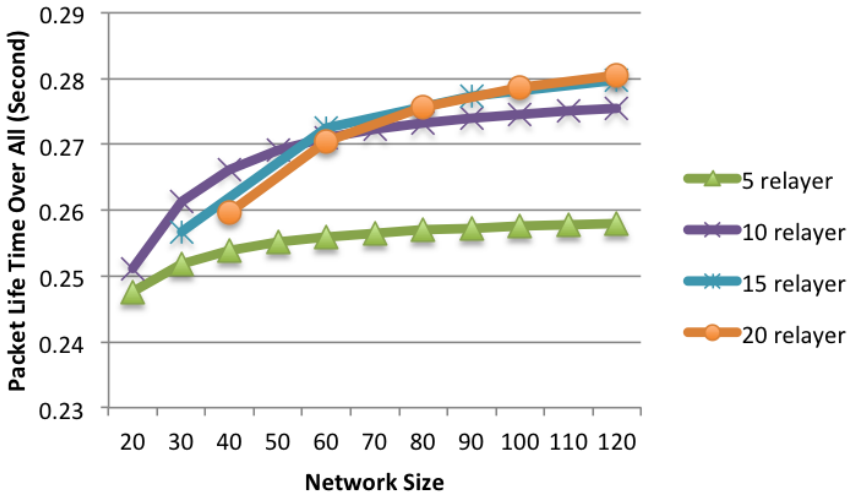


Figure 5.9: Average Packet Life Time

are relayers since the network size is 20, which is the same as a client-server network with 20 nodes. As a result, the packet life time is 0.2 in both cases we mentioned, which is the limit set in our particular setup.

In addition, the packet life time is always 0.2 and does not change in a client-server network. Due to the behavior in a client-server network, we can see that the client nodes only communicate with the server, not with any other clients, so that the network size has no impact on the packet life time in the client-server network. This naive approach demonstrates the need to incorporate the server load as an additional metric that has an impact in the client-server model. Taking any one metric in isolation can be deceiving, but still informative from a quantitative perspective.

Figure 5.9 shows the average packet life time over all, combining costs in LAN and in WAN. The movement of the average packet life time follows the trend from the analysis in LAN and in WAN. The packet life time slightly amplifies accompanied by the network size. The x-axis in Figure 5.9 is relatively small, that only from 0.23 to 0.29, where the lines can be considered as stable, though very much ideal, as real networks would be very noisy. The intersting point here is that it is someone difficult to see the influence of the number of relayers on the packet life time. The relayer is the distributor between the cloud server and the listener nodes. The number of relayers decides the proportion of the

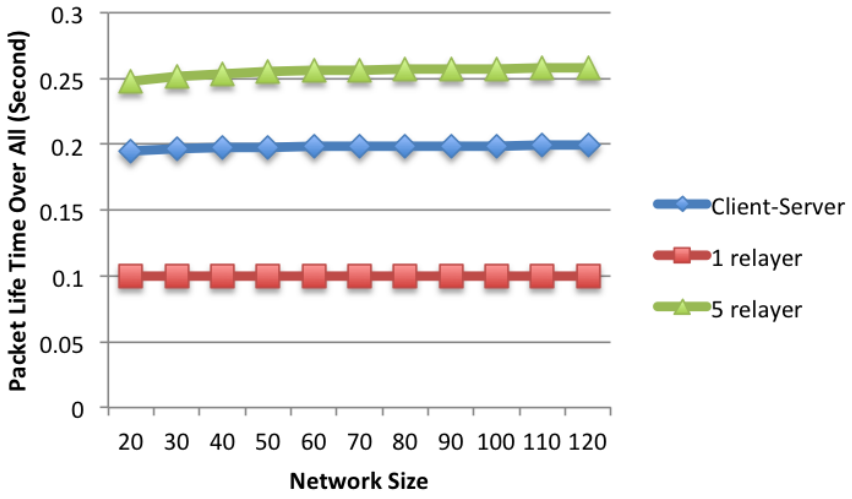


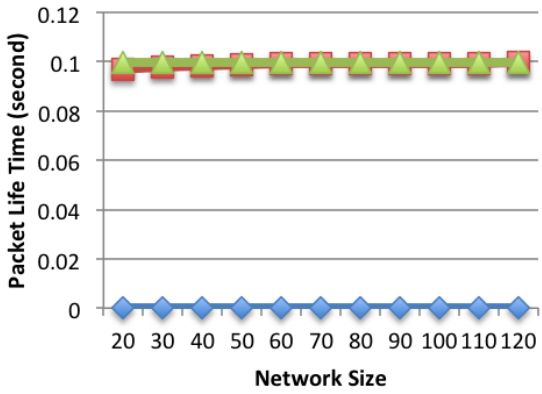
Figure 5.10: Comparison Of Packet Life Time

connection usage between LAN and WAN. The high speed in LAN benefits the network a great deal. Nevertheless, we have to consider tradeoffs from both aspects. More LAN connections denote fewer relayers though each listener will burden the relayer.

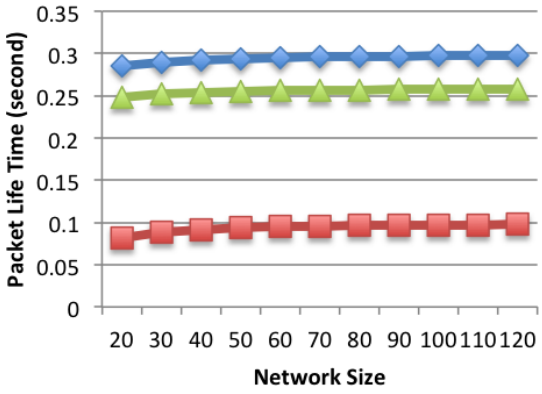
We compare our network configuration with the client-server network in average packet life time over all. As we can see from Figure 5.10, the lines are all fairly stable without any exceptional movement, which again would be ideal and only reveals a trend and not a detailed analysis. Given the configuration setup, the packet life time in the client-server model is lower than our configuration with 5 or more relayers, with the exception of our configuration with 1 relayer. This identifies a point in the system that could be considered for further more detailed evaluation informed by real world parameters.

Figure 5.11 compares the relationship between packet life time in LAN, packet life time in WAN and over all packet life time by assigning a different number of relayers to the network. Generally speaking, the slope of each line in the plots is quite flat, which indicates the significance of network size is not the key factor in this instance. The number of nodes only has a small effect on the packet life time in a network.

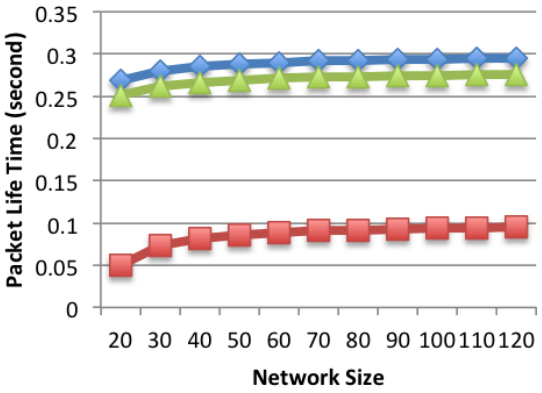
Figures 5.11b, 5.11c, 5.11d and 5.11e, show some correspondence with the number of relayers with more clarity. The packets spend the most of time in WAN



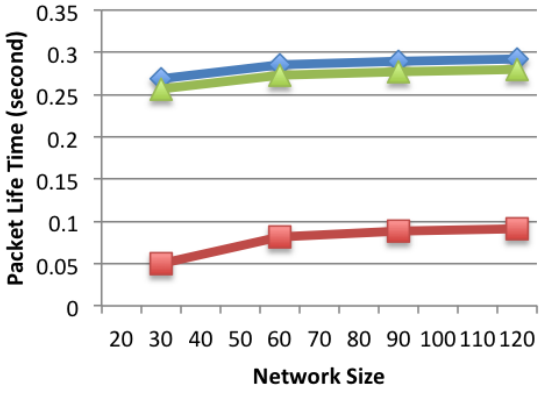
(a) with 1 Relayer



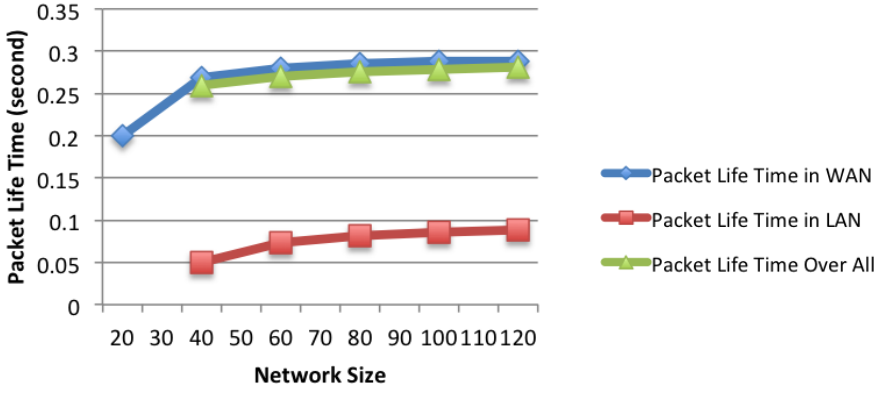
(b) with 5 Relayers



(c) with 10 Relayers



(d) with 15 Relayers



(e) with 20 Relayers

Figure 5.11: Packet Life Time With New Configuration

while the least time in LAN. The packet life time over all averages these two together. The packet life time in WAN dropped a bit when there are more relayers, the same as the packet life time in LAN. Packet life time in both WAN and LAN tends to the limitation in the network with the increase in network size. As for the packet life time over all, it can be found that the line is approaching the line representing the packet life time in WAN while the number of relayers increases. The gap between the packet life time in WAN and over all is getting smaller and smaller. Both of the lines are getting close to the boundary of the limit that was set in the network. As a consequence, more relayers create more WAN connections in the network, which enhances the overall network performance.

Additionally, Figure 5.11a presents the special cases and exceptions in this experiment. The packet life time in WAN is almost zero since all the clients communicate with each other through the LAN and the only channel using WAN is the one between the cloud server and the only relayer. The packet life time over all is remarkably close to the packet life time in LAN due to the same reason. They also tend to reach 0.1 as the limitation in the network.

5.3 Limitations

Our simulator is very preliminary and has limitations due to a variety of oversimplifications. However, it serves the purpose of addressing a proof-of-concept that could be further refined in the hands of a product manager with access to relevant application-specific parameters. The upfront costs of creating the simulation may not be trivial, but once it is in place, it could be incorporated into an agile methodology along side prototype implementations that inform the manager about appropriate setting for given parameters.

Specifically, the connection between each peer is represented as the simple channel in the simulator. Every data transmission is employed in a similar way as UDP, which means the packet is sent directly. Unlike TCP, it does not require a three-way handshake to establish a connection. Furthermore, it does not support resending the packet whenever the packet is lost during the transfer or has errors. This could be modified if future versions of the configuration. Additionally, network latency is barely taken into consideration in our configuration as it is beyond our focus scope

in this study. Moreover, we restrict the distribution methods in data transmission in unicasting that sends messages to a single network destination identified by a unique address [20] while the communication between peers in multicast is not used in the simulator.

In the network configuration evaluation, there are several factors not examined. For instance, we do not consider any cost during peer adaptation or recovery. The influence on other peers during these operations is disregarded in the analysis of data transmission efficiency. Furthermore, we presume the data transfer volume is a set amount for every peer in the simulator, even though individual conditions inevitably vary in practice. The additional complexity of consistency and consensus models is also deferred to future work.

5.4 Discussion

Table 5.8: Typical research questions versus product management questions in network simulation

	Network Simulation	**Question Status**
Research Questions	What spectrum of configurations are feasible? What metrics are critical? How are the informed by real-world results? What is the relationship between a theoretical result, one shown in the simulator, and the real-world?	Answered or could be answered with fairly straightforward use of the simulator.
Product Management Questions	What are the bottlenecks in the current system? What is the most cost-effective way to mitigate the bottleneck for the next version of the product?	Answered in a more informed way than possible without the simulator.

While we have conducted the experiment with only an extremely basic network simulation, we believe this approach shows the potential to address both of the ques-

tions from the research and product management perspectives. Such quantitative methodology provides solid data points and trends to start as a foundation for the research questions as well as gaining the insights of the tradeoffs in concrete terms for a product manager. We argue that simulation is extremely useful for the product manager if the simulation tool is easy enough to use. We have shown that a simple configuration is possible, with the code used in this example provided in the Appendix. As a product manager can not afford a huge learning curve on simulation tools, in particular in an agile development environment. However, if the barrier to entry is kept low, it could be beneficial and useful for product management to learn from research perspectives.

5.5 Summary

In this chapter, we report on a sequence of experiments to evaluate divergent prospects of our configuration. The main emphasis on the experiment is productivity and responsiveness. In order to assess this kind of user experience, we measure bandwidth consumption in the server by analyzing the number of packets sent out within the setup of our configuration. The results show how the network size and number of relayers have impact on the bandwidth consumption. Moreover, we find that our configuration spares more bandwidth for the server compared with the client-server model in the same network settings.

We investigate packet life time in different network environments to examine the responsive efficiency of our network configuration. It turns out that the network size is not the key factor here and the number of relayers has more influence on packet life time. Differentiating with the client-server model, it usually takes longer to transfer a packet in our configuration, which can be considered as a tradeoff. On top of that, we contrast five sets of data to understand that the over all packet life time is always getting closer to the packet life time in WAN when there are more relayers.

While we present evaluation with that would be a first step in network simulation within research perspectives, we also have found answers for product management questions. This reveals that research tooling and practices would be useful to introduce to product management if the tools are user-friendly and simple to use.

Chapter 6

Conclusions and Future Work

This thesis provides a comparison of the processes and tools used by product managers and researchers in the context of a case study on how to best scale the *Think Together* application. An exploratory study was conducted, and a new network configuration was developed to explore the crucial performance characteristics involved.

The evaluation illustrates that the simulation provides an effective means of exploring very specific aspects of the key performance indicators at a low cost. The proposed changes to the infrastructure hold promise, however, in the next iteration, they have to be informed by parameters set according to real world characteristics of the application. As the first step, however, the proposal is shown to meet the needs identified in the exploratory study. This information would allow the project manager to then make an investment in prototype implementation scenarios to inform the next round of simulation.

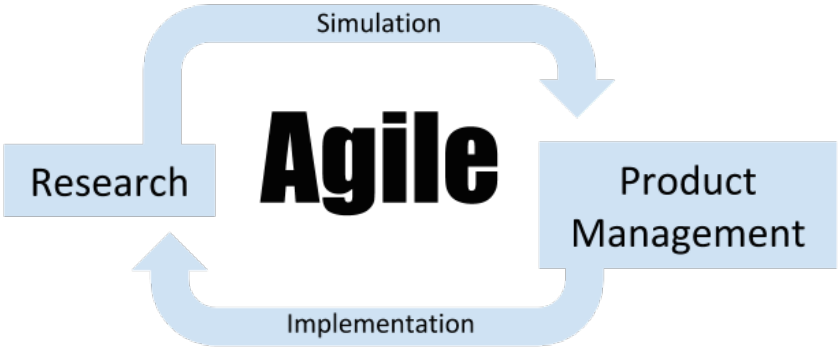


Figure 6.1: Life cycle showing how research and product management can inform each other in an agile methodology.

Specifically, the exploratory study shows its potential to inform the research of the simulation phase. The simulator is useful to product managers as long as the simulation tools like OMNeT++ are easy to use. Currently, it may be true that product managers can not afford to spend time on a tool with a steep learning curve, and OMNeT++ may fall into that category. As Figure 6.1 indicates, our work shows that ideally research and product management should be in the same agile development cycle. However, our study does not cover a complete life cycle. We can see that the results from the simulation inform the product manager of the trade-offs, but it has not been implemented in production. As a result, the parameters cannot be returned to the research to complete a life cycle without the product. However, a product manager can still take advantage of research methodology and provide parameters from an implementation to a simulation. With a user-friendly simulator, it is greatly beneficial for the product manager to identify and verify the structure design before implementation. Meanwhile, the simulation with parameters from real world enhances the accuracy of the research results as well as provides more meaningful ideas in research. As agile development is an incremental process, small implementations improve the simulation in the research while the simulation results help to verify the ideas for implementation.

6.1 Future Work

One of the most interesting avenues for future work would be to study the life cycle in Figure 6.1 through several iterations of the product. The configuration can be elaborated in numerous ways immediately, for example bringing in Bluetooth for peers who are within proximity. The use of Bluetooth in mobile applications is growing as product managers learn more about its potential in the industry. For instance, Bluetooth would be very practical and dominant when the devices are in a close distance or the other wireless connections are weak. This would be an example of a lightweight and incremental change that could be greatly informed by simulations using information gathered from prototype implementations of the application and its use cases. In terms of accurately setting parameters, establishing performance from a per-device perspective would be one way to gain insights into some of the challenges associated with the new infrastructure [17]. Besides, different network connections might lead to diametrically opposed results on data loss, which can give us a more detailed performance evaluation.

Appendix A

Source Code of omnetpp.ini

```
[General]
cmdenv-interactive = true
cmdenv-express-mode = true
sim-time-limit = 20s
tkenv-plugin-path = ../../../etc/plugins

[Config client_server_network]
network = client_server_network

[Config t3_simple_module_network]
network = t3_simple_module_network

[Config t3_simple_module_network_1R]
network = t3_simple_module_network
*.numOfRelayers = 1
*.numOfNodesUnderOneRelayer = ${N=19..119 step 10}

[Config t3_simple_module_network_5R]
network = t3_simple_module_network
*.numOfRelayers = 5
*.numOfNodesUnderOneRelayer = ${N=3..23 step 2}

[Config t3_simple_module_network_10R]
network = t3_simple_module_network
*.numOfRelayers = 10
*.numOfNodesUnderOneRelayer = ${N=1..11 step 1}

[Config t3_simple_module_network_15R]
```

```
network = t3_simple_module_network
*.numOfRelayers = 15
*.numOfNodesUnderOneRelayer = ${N=1..7 step 2}

[Config t3_simple_module_network_20R]
network = t3_simple_module_network
*.numOfRelayers = 20
*.numOfNodesUnderOneRelayer = ${N=0..5 step 1}
```

Appendix B

Source Code of t3_simple_module.ned

```
package t3_thinktogether;

simple T3_Server {
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];
}

simple T3_Relayer {
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];
}

simple T3_Node {
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];
}

// t3 simple module client server network
network client_server_network {
    parameters:
```

```
        int numOfNodes;
        @display("bgb=346,227");
    types:
        channel InternetChannel extends ned.DatarateChannel
        {
            datarate = 100Mbps;
            delay = 100ms;
        }
    submodules:
        serverHost: T3_Server {
            @display("i=block/square;p=170,50");
        }
        nodeHost[numOfNodes]: T3_Node {
            @display("i=block/users");
        }
    connections:
        for i=0..numOfNodes-1 {
            serverHost.gate++ <--> InternetChannel <-->
               nodeHost[i].gate++;
        }
}

network t3_simple_module_network {
    // Set numOfNodesUnderOneRelayer, Same amount of nodes under
       one relayer
    parameters:
        int numOfRelayers;
        int numOfNodesUnderOneRelayer;
        @display("bgb=400,342");
    types:
        channel InternetChannel extends ned.DatarateChannel
        {
            datarate = 100Mbps;
            delay = 100ms;
        }
        channel localChannel extends ned.DatarateChannel
        {
            datarate = 1000Mbps;
            delay = 50ms;
        }
    submodules:
        serverHost: T3_Server {
```

```
            @display("i=block/square;p=195,59");
        }
        relayerHost[numOfRelayers]: T3_Relayer {
            @display("i=block/users");
        }
        nodeHost[numOfRelayers*numOfNodesUnderOneRelayer]: T3_Node {
            @display("i=block/users");
        }
    connections:
        for i=0..numOfRelayers-1 {
            serverHost.gate++ <--> InternetChannel <-->
                relayerHost[i].gate++;
        }
        for i=0..numOfRelayers*numOfNodesUnderOneRelayer-1 {
            nodeHost[i].gate++ <--> localChannel <-->
                relayerHost[i/numOfNodesUnderOneRelayer].gate++;
        }
}
```

Appendix C

Source Code of t3_simple_module.msg

```
enum T3NodeType {
    T3_SERVER = 1;
    T3_RELAYER = 2;
    T3_NODE = 3;
}

message AnnotationMsg {
    int senderType @enum(T3NodeType);
    int senderIndex = 0;
    int sourceType @enum(T3NodeType);
    int sourceIndex = 0;
    bool isLocalMsg = true;
    bool isPacketError = false;
    simtime_t duration;
}
```

Appendix D

Source Code of T3_Node.cc

```
#include <string.h>
#include <omnetpp.h>
#include "t3_simple_module_m.h"

class T3_Node : public cSimpleModule {
    private:
        double lanPacketErrorRatio = 0.01;     // Packet has
            possibility to have error when sent

        long numReceived;
        long numReceivedFromServer;

        long numErrorMessageReceived;

        cOutVector durationVector;
        cOutVector lanMessageDurationVector;
        cOutVector wanMessageDurationVector;
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
        virtual void finish();

        virtual AnnotationMsg *generateAnnotationMessage();
};

Define_Module(T3_Node);

void T3_Node::initialize() {
```

```
    numReceived = 0;
    numReceivedFromServer = 0;
    numErrorMessageReceived = 0;

    durationVector.setName("duration");
    lanMessageDurationVector.setName("lanMessageDuration");
    wanMessageDurationVector.setName("wanMessageDuration");

    cMessage *timerMessage = new cMessage();
    scheduleAt(intrand(6), timerMessage);
}

void T3_Node::handleMessage(cMessage *msg) {
    if(msg->isSelfMessage()){
        scheduleAt(simTime() + 3, msg);

        AnnotationMsg *newmsg = generateAnnotationMessage();
        if(lanPacketErrorRatio != 0 && dblrand() <
           lanPacketErrorRatio) {
            newmsg->setIsPacketError(true);
        }
        send(newmsg, "gate$o", 0);  // Send to parent (relayer or
           server)
        bubble("Generate a new msg!");

    }else {
        AnnotationMsg *ttmsg = check_and_cast<AnnotationMsg *>(msg);

        if(ttmsg->getSenderType()==T3_RELAYER ||
           ttmsg->getSenderType()==T3_SERVER) {
            ++numReceived;
            if(ttmsg->getSourceType()==T3_SERVER) {
                ++numReceivedFromServer;
            }
            if(ttmsg->getIsPacketError()) {
                ++numErrorMessageReceived;
            }

            durationVector.record(simTime() - ttmsg->getDuration());
            if(ttmsg->getIsLocalMsg()) {
                lanMessageDurationVector.record(simTime() -
                   ttmsg->getDuration());
```

```
            }else {
                wanMessageDurationVector.record(simTime() -
                    ttmsg->getDuration());
            }
        }

        delete msg;
    }
}

void T3_Node::finish() {
    recordScalar("#received", numReceived);
    recordScalar("#receivedErrorpacket", numErrorMessageReceived);
}

// Helper methods
AnnotationMsg *T3_Node::generateAnnotationMessage() {
    char msgname[20];
    sprintf(msgname, "AnnotationMsg from Node %d", getIndex());

    // Create message object and set source and destination field.
    AnnotationMsg *msg = new AnnotationMsg(msgname);
    msg->setSenderType(T3_NODE);
    msg->setSenderIndex(getIndex());
    msg->setSourceType(T3_NODE);
    msg->setSourceIndex(getIndex());
    msg->setDuration(simTime());
    return msg;
}
```

Appendix E

Source Code of T3_Relayer.cc

```
#include <string.h>
#include <omnetpp.h>
#include "t3_simple_module_m.h"

class T3_Node : public cSimpleModule {
    private:
        double lanPacketErrorRatio = 0.01;     // Packet has
            possibility to have error when sent

        long numReceived;
        long numReceivedFromServer;

        long numErrorMessageReceived;

        cOutVector durationVector;
        cOutVector lanMessageDurationVector;
        cOutVector wanMessageDurationVector;
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
        virtual void finish();

        virtual AnnotationMsg *generateAnnotationMessage();
};

Define_Module(T3_Node);

void T3_Node::initialize() {
```

```
    numReceived = 0;
    numReceivedFromServer = 0;
    numErrorMessageReceived = 0;

    durationVector.setName("duration");
    lanMessageDurationVector.setName("lanMessageDuration");
    wanMessageDurationVector.setName("wanMessageDuration");

    cMessage *timerMessage = new cMessage();
    scheduleAt(intrand(6), timerMessage);
}

void T3_Node::handleMessage(cMessage *msg) {
    if(msg->isSelfMessage()){
        scheduleAt(simTime() + 3, msg);

        AnnotationMsg *newmsg = generateAnnotationMessage();
        if(lanPacketErrorRatio != 0 && dblrand() <
           lanPacketErrorRatio) {
            newmsg->setIsPacketError(true);
        }
        send(newmsg, "gate$o", 0);  // Send to parent (relayer or
           server)
        bubble("Generate a new msg!");

    }else {
        AnnotationMsg *ttmsg = check_and_cast<AnnotationMsg *>(msg);

        if(ttmsg->getSenderType()==T3_RELAYER ||
           ttmsg->getSenderType()==T3_SERVER) {
            ++numReceived;
            if(ttmsg->getSourceType()==T3_SERVER) {
                ++numReceivedFromServer;
            }
            if(ttmsg->getIsPacketError()) {
                ++numErrorMessageReceived;
            }

            durationVector.record(simTime() - ttmsg->getDuration());
            if(ttmsg->getIsLocalMsg()) {
                lanMessageDurationVector.record(simTime() -
                   ttmsg->getDuration());
```

```
            }else {
                wanMessageDurationVector.record(simTime() -
                    ttmsg->getDuration());
            }
        }

        delete msg;
    }
}

void T3_Node::finish() {
    recordScalar("#received", numReceived);
    recordScalar("#receivedErrorpacket", numErrorMessageReceived);
}

// Helper methods
AnnotationMsg *T3_Node::generateAnnotationMessage() {
    char msgname[20];
    sprintf(msgname, "AnnotationMsg from Node %d", getIndex());

    // Create message object and set source and destination field.
    AnnotationMsg *msg = new AnnotationMsg(msgname);
    msg->setSenderType(T3_NODE);
    msg->setSenderIndex(getIndex());
    msg->setSourceType(T3_NODE);
    msg->setSourceIndex(getIndex());
    msg->setDuration(simTime());
    return msg;
}
```

Appendix F

Source Code of T3_Server.cc

```
#include <string.h>
#include <omnetpp.h>
#include "t3_simple_module_m.h"

class T3_Node : public cSimpleModule {
    private:
        double lanPacketErrorRatio = 0.01;     // Packet has
            possibility to have error when sent

        long numReceived;
        long numReceivedFromServer;

        long numErrorMessageReceived;

        cOutVector durationVector;
        cOutVector lanMessageDurationVector;
        cOutVector wanMessageDurationVector;
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
        virtual void finish();

        virtual AnnotationMsg *generateAnnotationMessage();
};

Define_Module(T3_Node);

void T3_Node::initialize() {
```

```
    numReceived = 0;
    numReceivedFromServer = 0;
    numErrorMessageReceived = 0;

    durationVector.setName("duration");
    lanMessageDurationVector.setName("lanMessageDuration");
    wanMessageDurationVector.setName("wanMessageDuration");

    cMessage *timerMessage = new cMessage();
    scheduleAt(intrand(6), timerMessage);
}

void T3_Node::handleMessage(cMessage *msg) {
    if(msg->isSelfMessage()){
        scheduleAt(simTime() + 3, msg);

        AnnotationMsg *newmsg = generateAnnotationMessage();
        if(lanPacketErrorRatio != 0 && dblrand() <
           lanPacketErrorRatio) {
            newmsg->setIsPacketError(true);
        }
        send(newmsg, "gate$o", 0);  // Send to parent (relayer or
           server)
        bubble("Generate a new msg!");

    }else {
        AnnotationMsg *ttmsg = check_and_cast<AnnotationMsg *>(msg);

        if(ttmsg->getSenderType()==T3_RELAYER ||
           ttmsg->getSenderType()==T3_SERVER) {
            ++numReceived;
            if(ttmsg->getSourceType()==T3_SERVER) {
                ++numReceivedFromServer;
            }
            if(ttmsg->getIsPacketError()) {
                ++numErrorMessageReceived;
            }

            durationVector.record(simTime() - ttmsg->getDuration());
            if(ttmsg->getIsLocalMsg()) {
                lanMessageDurationVector.record(simTime() -
                   ttmsg->getDuration());
```

```
            }else {
                wanMessageDurationVector.record(simTime() -
                    ttmsg->getDuration());
            }
        }

        delete msg;
    }
}

void T3_Node::finish() {
    recordScalar("#received", numReceived);
    recordScalar("#receivedErrorpacket", numErrorMessageReceived);
}

// Helper methods
AnnotationMsg *T3_Node::generateAnnotationMessage() {
    char msgname[20];
    sprintf(msgname, "AnnotationMsg from Node %d", getIndex());

    // Create message object and set source and destination field.
    AnnotationMsg *msg = new AnnotationMsg(msgname);
    msg->setSenderType(T3_NODE);
    msg->setSenderIndex(getIndex());
    msg->setSourceType(T3_NODE);
    msg->setSourceIndex(getIndex());
    msg->setDuration(simTime());
    return msg;
}
```

Bibliography

[1] Omnet++. `https://omnetpp.org`.

[2] Syed Muhammad Abbas and Christopher Henricsson. A simulation framework for efficient search in p2p networks with 8-point hypercircles. 2008.

[3] Jerry Banks, John S Carson II, and Barry L Nelson. Discrete-event system simulation. international series in industrial and systems engineering, 1996.

[4] David Barkai. *Peer-to-Peer Computing: technologies for sharing and collaborating on the net*. Intel Press, 2001.

[5] Rimon Barr, Zygmunt J Haas, and Robbert van Renesse. Jist: An efficient approach to simulation using virtual machines. *Software: Practice and Experience*, 35(6):539–576, 2005.

[6] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[7] Tim Berners-Lee and Robert Cailliau. Worldwideweb: Proposal for a hypertext project. *Retrieved on February*, 26:2008, 1990.

[8] Tim Berners-Lee, Mark Fischetti, and Michael L Foreword By-Dertouzos. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. HarperInformation, 2000.

[9] Alex Berson. *Client-server architecture*. Number IEEE-802. McGraw-Hill, 1992.

[10] SIG Bluetooth. Specification of the bluetooth system, version 1.1. *http://www. bluetooth. com*, 2001.

[11] Lee Breslau, Deborah Estrin, Haobo Yu, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, et al. Advances in network simulation. *Computer*, (5):59–67, 2000.

[12] Josh Broch, David A Maltz, David B Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97. ACM, 1998.

[13] George P Browman, Mark N Levine, E Ann Mohide, RS Hayward, Kathleen I Pritchard, Amiram Gafni, and Andreas Laupacis. The practice guidelines development cycle: a conceptual tool for practice guidelines development and implementation. *Journal of Clinical Oncology*, 13(2):502–512, 1995.

[14] Xinjie Chang. Network simulations with opnet. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pages 307–314. ACM, 1999.

[15] Qi Chen, Felix Schmidt-Eisenlohr, Daniel Jiang, Marc Torrent-Moreno, Luca Delgrossi, and Hannes Hartenstein. Overhaul of ieee 802.11 modeling and simulation in ns-2. In *Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems*, pages 159–168. ACM, 2007.

[16] Martin Christopher. The agile supply chain: competing in volatile markets. *Industrial marketing management*, 29(1):37–44, 2000.

[17] Leandro Collares, Chris Matthews, Justin Cappos, Yvonne Coady, and Rick McGeer. Et (smart) phone home! In *Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11*, SPLASH '11 Workshops, pages 283–288, New York, NY, USA, 2011. ACM.

[18] Nick Collier. Repast: An extensible framework for agent simulation. *The University of Chicagos Social Science Research*, 36:2003, 2003.

[19] Apple Developer Connection. Networkingbonjour, b. *URL http://developer. apple. com/networking/bonjour*.

[20] John Day. *Patterns in network architecture: a return to fundamentals*. Pearson Education, 2007.

[21] Alexander Dupuy, Jed Schwartz, Yechiam Yemini, and David Bacon. Nest: A network simulation and prototyping testbed. *Communications of the ACM*, 33(10):63–74, 1990.

[22] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9):833–859, 2008.

[23] Robert C Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–1797, 2004.

[24] Kathleen M Eisenhardt. Building theories from case study research. *Academy of management review*, 14(4):532–550, 1989.

[25] Samer Faraj, Srinivas Kudaravalli, and Molly Wasko. Leading collaboration in online communities. *MIS Quarterly*, 39(2), 2015.

[26] George S Fishman. Principles of discrete event simulation.[book review]. 1978.

[27] Ali Hajimiri, Sotirios Limotyrakis, and Thomas H Lee. Jitter and phase noise in ring oscillators. *Solid-State Circuits, IEEE Journal of*, 34(6):790–804, 1999.

[28] Thomas R Henderson, Sumit Roy, Sally Floyd, and George F Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, page 13. ACM, 2006.

[29] Jim Highsmith. *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley, 2013.

[30] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *Computer*, 34(9):120–127, 2001.

[31] Pierre Humblet, Murat Azizoğlu, et al. On the bit error rate of lightwave systems with optical amplifiers. *Lightwave Technology, Journal of*, 9(11):1576–1582, 1991.

[32] Kamal Jain, Jitendra Padhye, Venkata N Padmanabhan, and Lili Qiu. Impact of interference on multi-hop wireless network performance. *Wireless networks*, 11(4):471–487, 2005.

[33] M Eric Johnson, Dan McGuire, and Nicholas D Willey. The evolution of the peer-to-peer file sharing industry and the security risks for users. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 383–383. IEEE, 2008.

[34] David H Judson. Web browser with dynamic display of information objects during linking, November 5 1996. US Patent 5,572,643.

[35] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651. ACM, 2003.

[36] W David Kelton and Averill M Law. *Simulation modeling and analysis*. McGraw Hill Boston, 2000.

[37] Srinivasan Keshav. *REAL: A network simulator*. University of California, 1988.

[38] Henrik Kniberg. *Scrum and XP from the Trenches: How we do Scrum*. Lulu. com, 2007.

[39] Feiyu Lin and Kurt Sandkuhl. Towards efficient search in p2p networks with 8-point hypercircles. 2006.

[40] Virginia Lo, Dayi Zhou, Yuhong Liu, Chris GauthierDickey, and Jun Li. Scalable supernode selection in peer-to-peer overlay networks. In *Hot Topics in Peer-to-Peer Systems, 2005. HOT-P2P 2005. Second International Workshop on*, pages 18–25. IEEE, 2005.

[41] C Mallanda, A Suri, V Kunchakarra, SS Iyengar, R Kannan, A Durresi, and S Sastry. Simulating wireless sensor networks with omnet++. *submitted to IEEE Computer*, 2005.

[42] Micah Martin and Robert C Martin. *Agile principles, patterns, and practices in C#*. Pearson Education, 2006.

[43] Norm Matloff. Introduction to discrete-event simulation and the simpy language. 2008.

[44] Sheila A McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE intelligent systems*, (2):46–53, 2001.

[45] Su-Hong Min, Joanne Holliday, and Dong-Sub Cho. Optimal super-peer selection for large-scale p2p system. In *Hybrid Information Technology, 2006. ICHIT'06. International Conference on*, volume 2, pages 588–593. IEEE, 2006.

[46] Steve R Palmer and Mac Felsing. *A practical guide to feature-driven development*. Pearson Education, 2001.

[47] Kihong Park and Walter Willinger. *Self-similar network traffic and performance evaluation*. Wiley Online Library, 2000.

[48] Sung Park, Andreas Savvides, and Mani B Srivastava. Sensorsim: A simulation framework for sensor networks. In *Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 104–111. ACM, 2000.

[49] Roman Pichler. *Agile product management with scrum: Creating products that customers love*. Addison-Wesley Professional, 2010.

[50] Klaus Pohl, Günter Böckle, and Frank J van der Linden. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[51] Bianca Schroeder and Mor Harchol-Balter. Web servers under overload: How scheduling can help. *ACM Transactions on Internet Technology (TOIT)*, 6(1):20–52, 2006.

[52] K Schwaber and Mike Beedle. gilè software development with scrum. 2002.

[53] Patrick Seeling, Martin Reisslein, and Beshan Kulapala. Network performance evaluation using frame size and quality traces of single-layer and two-layer video: A tutorial. *Communications Surveys & Tutorials, IEEE*, 6(3):58–78, 2004.

[54] James Shore et al. *The art of agile development*. " O'Reilly Media, Inc.", 2007.

[55] Preston G Smith and Donald G Reinertsen. *Developing products in half the time: new rules, new tools*, volume 298. John Wiley & Sons New York, 1998.

[56] Robin Snader and Nikita Borisov. Eigenspeed: secure peer-to-peer bandwidth evaluation.

[57] Harry M Sneed. Encapsulating legacy software for use in client/server systems. In *Reverse Engineering, 1996., Proceedings of the Third Working Conference on*, pages 104–119. IEEE, 1996.

[58] Penna Sparrow. Peer-to-peer (p2p) : Advantages and disadvantages, 2015. [Online; accessed 01-December-2015].

[59] Nancy K Squires, Kenneth C Squires, and Steven A Hillyard. Two varieties of long-latency positive waves evoked by unpredictable auditory stimuli in man. *Electroencephalography and clinical neurophysiology*, 38(4):387–401, 1975.

[60] Sean Stolberg. Enabling agile testing through continuous integration. In *Agile Conference, 2009. AGILE'09.*, pages 369–374. IEEE, 2009.

[61] David Talby, Arie Keren, Orit Hazzan, and Yael Dubinsky. Agile software testing in a large-scale project. *Software, IEEE*, 23(4):30–37, 2006.

[62] Stefan Thomke and Donald Reinertsen. Agile product development: Managing development flexibility in uncertain environments. *California management review*, 41(1):8, 1998.

[63] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[64] N Viswanadham and Y Narahari. *Performance modeling of automated manufacturing systems*. Prentice Hall Englewood Cliffs, NJ, 1992.

[65] Quang Hieu Vu, Mihai Lupu, and Beng Chin Ooi. Architecture of peer-to-peer systems. In *Peer-to-Peer Computing*, pages 11–37. Springer, 2010.

[66] Tianming Wei, Yongjun Xu, Yiyun Zhao, Nishant Khanna, Bing Gao, and Yvonne Coady. Exploring peer-to-peer infrastructure for computer supported collaborative work applications. 2015 IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, 2015.

[67] Elias Weingärtner, Hendrik Vom Lehn, and Klaus Wehrle. A performance comparison of recent network simulators. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–5. IEEE, 2009.

[68] Steve Whittaker and Candace Sidner. Email overload: exploring personal information management of email. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 276–283. ACM, 1996.

[69] Moe Z Win, Robert A Scholtz, et al. Ultra-wide bandwidth time-hopping spread-spectrum impulse radio for wireless multiple-access communications. *IEEE Transactions on communications*, 48(4):679–689, 2000.

[70] Xiaodong Xian, Weiren Shi, and He Huang. Comparison of omnet++ and other simulator for wsn simulation. In *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, pages 1439–1443. IEEE, 2008.

[71] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 49–60. IEEE, 2003.