SERVICE DISTRIBUTION MECHANISMS IN INFORMATION-CENTRIC NETWORKING

Bachelorarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Oliver Stapleton
2015

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Contents

Contents **i**

List of Figures **iii**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Task Formulation . . . 2
- 1.3 Outline . . . 2

2 Related Work **3**
- 2.1 Distributed Computing . . . 3
  - 2.1.1 Terminology . . . 3
  - 2.1.2 Steps . . . 5
  - 2.1.3 Goals of Distributed Computing . . . 7
  - 2.1.4 Challenges . . . 7
- 2.2 Information-Centric Networking (ICN) . . . 8
  - 2.2.1 Content-Centric Networking (CCN) . . . 11
  - 2.2.2 Service-Centric Networking (SCN) . . . 12
  - 2.2.3 CCNx . . . 12
  - 2.2.4 NextServe . . . 15

3 Design and Implementation **17**
- 3.1 Trivial Approach (TA) . . . 17
  - 3.1.1 Design . . . 17
  - 3.1.2 Implementation . . . 19
  - 3.1.3 Analysis . . . 20
- 3.2 Delegated Approach (DA) . . . 21
  - 3.2.1 Design . . . 21
  - 3.2.2 Implementation . . . 24
  - 3.2.3 Analysis . . . 25
- 3.3 Probability-based Approach (PBA) . . . 26
  - 3.3.1 Design . . . 26
  - 3.3.2 Implementation . . . 31
  - 3.3.3 Analysis . . . 33

4 Evaluation **35**
4.1 Model Testing 35
4.1.1 Testing Setup 36
4.1.2 Measurements & Discussion 37
4.2 Basic Scenario 45
4.2.1 Testing Setup 46
4.2.2 Measurements & Discussion 46
4.3 Real-world Scenario 47
4.3.1 Testing Setup 48
4.3.2 Measurements & Discussion 49

5 Conclusion **51**
5.1 Summary & Conclusions 51
5.2 Future Work 52

Bibliography **53**

List of Figures

2.1 Terminology of Distributed Computing 5
2.2 Steps of Distributed Computing 6
2.3 Example: ICN Routing I 9
2.4 Example: ICN Routing II 10
2.5 Example: ICN Routing III 10
2.6 Example: ICN Routing IV 11
2.7 Example: ICN Routing V 11

3.1 Flow chart: Requester in TA 18
3.2 Flow chart: Provider in TA 19
3.3 Topology: Providers connected directly to Requester 19
3.4 Topology: Providers not connected directly to Requester 20
3.5 Flow chart: Requester in DA 23
3.6 Flow chart: Provider in DA 23
3.7 Sequence diagram: Actions performed by Requester, Information Forwarders, and Providers 24
3.8 Example: Possible Topology for DA 25
3.9 Flow chart: Requester in PBA 27
3.10 Architecture: Layers of Provider in PBA 28
3.11 Flow chart: Provider in PBA 29
3.12 Example: Situation before iteration 1 30
3.13 Example: Situation after iteration 1 30
3.14 Example: Situation after iteration 2 31
3.15 Example: Situation after iteration 4 31

4.1 Graph: Mean duration with 2 Providers for various P-values 38
4.2 Graph: Mean duration with 4 Providers for various P-values 39
4.3 Graph: Mean duration with 6 Providers for various P-values 40
4.4 Graph: Mean duration with 8 Providers for various P-values 41
4.5 Graph: Mean duration with 10 Providers for various P-values 42
4.6 Graph: Mean duration depending on number of jobs for various P-values 43
4.7 Graph: Mean duration depending on number of Providers for various P-values . 44
4.8 Graph: Duration to solve task depending on P-value 45
4.9 Testing setup for the Basic Scenario 46

4.10 Graph: Duration of Basic Scenario for various P-values 47
4.11 Testing setup for Real-world Scenario 49
4.12 Graph: Duration of Real-world Scenario for various P-values 50

Chapter 1

Introduction

1.1 Motivation

Information-Centric Networking (ICN) [1] is a new networking paradigm that decouples information from location. When an information consumer expresses his interest for some information, it declares only the name (information identifier) of the information but not the location (host identifier). The network routing across all the nodes takes care of locating the information with the matching name. In ICN networks, information can be cached in many locations of the network. This can reduce traffic significantly in an ICN network because information is delivered to a requester from the closest location it is cached at rather than the location of the original source.

Information can be defined in different ways: In Content-Centric Networking (CCN) [2], information is regarded simply as (static) content. In Service-Centric Networking (SCN) [3], information is regarded as a (dynamic) service. A service could be to solve a specific task, e.g. to encrypt a given file, perform a complex calculation or to transcode a video file from one format to another.

Research projects of recent years have led to several implementations of the ICN paradigm. Project CCNx [2] offers a solid implementation of CCN, and can be extended with NextServe [4] for full SCN capabilities. What is missing in current implementations of ICN is the ability to perform Distributed Computing in an ICN network: The processing of a specific requested service currently always relies on a single ad-hoc server, whereas for some services it might instead be more suitable to split the task into multiple jobs that are each processed as services individually. For example, the task of transcoding a video file of n minutes length could be split into n jobs of transcoding a one minute video file each. This would lead to more than one server working on a task at the same time, resulting in possibly a faster turnaround time to process the whole task. Also, caching could be optimized: Imagine the case where in the past, the first minute of a video file has been transcoded as a service and therefore is cached at some location in the network already. Now, some consumer is interested in transcoding the first two minutes of the same video file. In the current state of SCN, the entire two minutes of the video file would need to be transcoded again, even though the first minute is actually already available on the network. This would not be the case if services were distributed before processing, since then the transcoded version of the first minutes could be retrieved from its cached location.

The goal of this thesis is to design, implement and evaluate such a distribution mechanism for services in an ICN network that solves the aforementioned problems of the current state of ICN implementations. As a code base, the current versions of CCNx and NextServe are used, the latter of which is extended with the code for a distribution mechanism.

1.2 Task Formulation

Work on this thesis consisted of the following tasks:

- Get familiar with Content-Centric Networking (CCN)
- Get familiar with Service-Centric Networking (SCN)
- Design a service distribution mechanism
- Implement the service distribution mechanism
- Evaluate the performance of the service distribution mechanism

1.3 Outline

In Chapter 2, the related work on the topics of ICN and Distributed Computing is discussed. Chapter 3 presents and analyses three approaches of designing and implementing a service distribution mechanism: The very basic and straight-forward Trivial Approach, followed by two improvements, the Delegated Approach and the Probability-based Approach. An evaluation of the Probability-based Approach is made in Chapter 4 and presented together with the results. Finally, Chapter 5 contains the conclusion of this thesis and takes a short look at further work that is necessary on the topic of distribution mechanisms in ICN networks.

Chapter 2

Related Work

This chapter introduces the topics of Distributed Computing and Information-Centric Networking (ICN) and highlights the important elements of both fields in regard to the topic of the thesis.

2.1 Distributed Computing

Distributed Computing is a form of computing where a Distributed System is used to perform a given task [5]. A Distributed System is a collection of independent computers [6]. Distributed Systems usually are classified into two classes: Clusters, where the network consists of fairly homogeneous machines that are coupled very closely, and Grids, where the machines are a lot more heterogeneous and loosely coupled. For the moment, we will focus on Distributed Computing on Grids (also known as Grid Computing), as we will later see that ICN is, at best, a Distributed System of extremely loose coupling. An important aspect of Distributed Computing that should be considered right from the start is the following: Distributing a task will always add a certain overhead to the workload, because all the involved parties need to communicate with each other and also because it is necessary to make some modifications to the task before it can be distributed. Let's assume we have a Distributed System with two equivalent machines: It is not realistic to expect that the two machines in combination can compute a specific task in exactly half the time that it would take each of the machines individually. On the other hand, it is realistic to assume that the two machines in combination will be able to compute the task quicker than a single machine. However, the sum of the actual processing power consumed by both machines when distributing the task will be greater than the processing power a single machine would consume if it processed the task locally.

2.1.1 Terminology

In the literature on the topic of Distributed Computing there is no global terminology standard for the core elements of Distributed Computing that are relevant for this thesis. Therefore, we will first define our terminology that will be used consistently in the forthcoming chapters.

Requester A Requester is a node in a network that wants to distribute parts of its workload to be computed on other nodes of the network, the so called Providers. A Requester can either distribute entire tasks or only parts of these tasks, the so called jobs.

Provider A Provider is a node in a network that provides some of its computing resources to compute jobs requested by other nodes (the Requesters) of the network. A node can be a Requester and a Provider at the same time.

Task A task represents a coherent amount of work that needs to be computed. It can sometimes be split into smaller, independent sub-tasks, named jobs. Example: To transcode a video file from one video format to another.

Task result A task result is the outcome when a task has been completely computed.

Job A job represents a sub-task (part of a task) that is independent of any other sub-task. In many cases it is also a task by itself. Whether a task is referred to as a task or as a job depends heavily on the context: In a scenario there is always an (overall) task that in many cases is split into multiple jobs. Example: The task of transcoding a complete video file could be split into the jobs of transcoding each minute of the video file individually.

Job result A job result is the outcome when a job has been computed completely.

Define Defining is the process where the Requester combines coherent parts of its workload to form a task.

Split Splitting is the process where the Requester divides a task into a number of jobs.

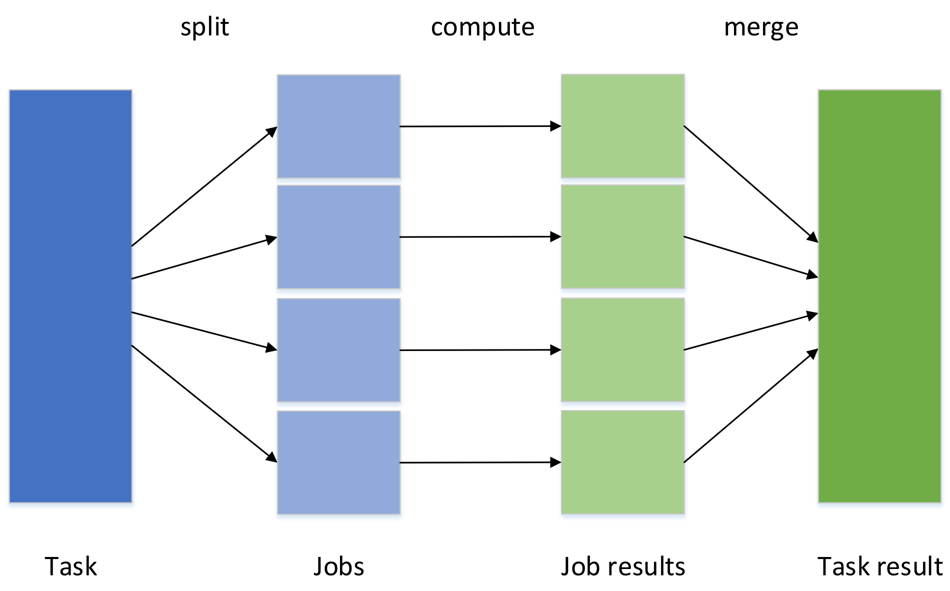
Distribute Distributing is the process where the Requester sends the jobs over the network to one or more Providers each.

Compute Computing is the process where a node processes a job (or a task) to receive the job result (or task result).

Merge Merging is the process where the Requester combines the job results (of all the jobs belonging to the same task) to receive the task result.

Figure 2.1 visualizes the terminology of Distributed Computing from a global perspective.

Figure 2.1: Terminology of Distributed Computing



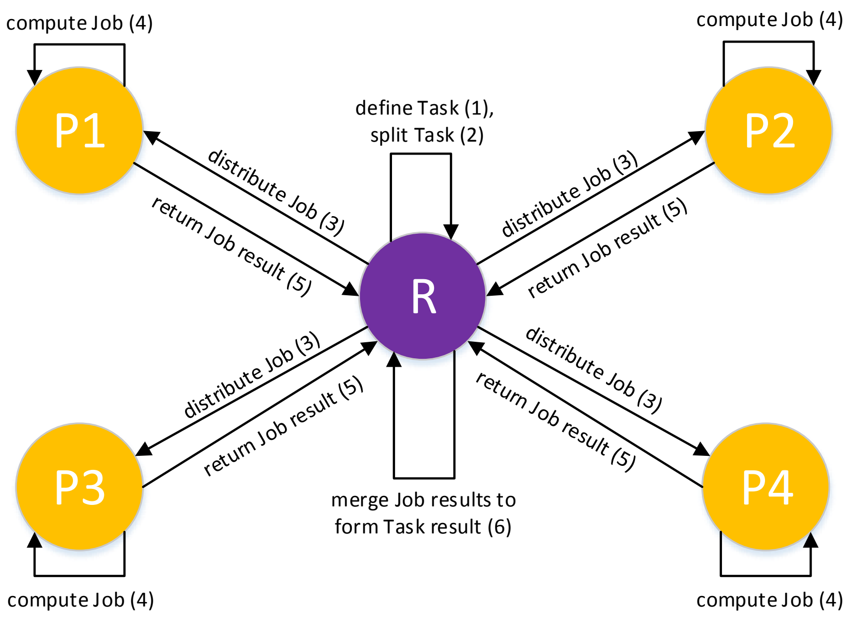
2.1.2 Steps

The work flow of Distributed Computing can be divided into six steps:

1. The Requester defines the task that needs to be computed.
2. The Requester splits the task into a number of jobs.
3. The Requester distributes the jobs to the Providers. Each job must be distributed to at least one Provider.
4. Each Provider computes the job(s) he receives.
5. Each Provider returns its job result(s) back to the Requester.
6. The Requester waits until all job results have arrived and merges all the job results to form the overall task result.

Figure 2.2 visualizes the steps of Distributed Computing in combination with the nodes on which the steps are executed.

Figure 2.2: Steps of Distributed Computing



Example

1. The Requester wants the video file tarzan.mpg to be transcoded from a high-resolution 1080p format to lower resolution DVD quality 576p format.

2. The Requester partitions the video file tarzan.mpg into four smaller files:

   ```
   tarzan_part1.mpg
   tarzan_part2.mpg
   tarzan_part3.mpg
   tarzan_part4.mpg
   ```

   The task is split into four jobs, each of which consists of transcoding one of the four partial files.

3. The Requester distributes each job to a specific Provider of a list of Providers that are available over the network.

4. Each Provider computes the jobs it receives by transcoding the partial 1080p video file (that is associated with the job) from 1080p to 576p (partial) video files.

5. Each Provider returns the partial 576p video file as the job result back to the Requester.

6. The Requester waits until all transcoded partial 576p video files (the job results) have arrived and then merges them to form the desired 576p tarzan.mpg video file (the task result).

2.1.3 Goals of Distributed Computing

From the point of view of a Requester, there are many reasons for distributing a task. Jobs that would locally need to be computed sequentially can be computed in parallel by multiple Providers instead. This results in a faster turnaround time per task for the Requester and potentially also a higher throughput (if the Requester has more than one task to compute), as long as the overhead caused by the distribution is smaller than the time gained through parallel computing of the jobs. The same criteria also apply to the power consumption of the Requester. If the overhead in power consumption caused by distribution is less than the power consumption saved by not locally computing the task, then it is also attractive for the Requester to distribute the task. This is especially important for mobile nodes on a network, since they might only have limited (battery) power available.

For a Provider, all these goals do not apply. Theoretically, in terms of improving the local turnaround time and reducing the power consumption, it makes not much sense for a Provider to provide computing power to other nodes on a network. In most cases, nodes in a network will be able to act both as a Provider and a Requester. Thus, a node might volunteer as a Provider knowing that it will at some stage also act as a Requester and profit from the network. It is imaginable that a Distributed System might actually be implemented in a way that a node must first provide some of its computing power to other nodes of the network before being able to distribute its own tasks.

In the context of this thesis, we will treat the reduction of the turnaround time per task as the main (and only) goal of Distributed Computing. The evaluation will measure the time duration on the Requester's side between launching a task and having the final task result available locally. A distribution mechanism will be considered an improvement (and consequently a success) if its measured turnaround time is smaller than the turnaround time of locally computing the task.

2.1.4 Challenges

Distributed Computing leads to the sharing of various resources between the machines in a Distributed System: Computation power, information about tasks and jobs, and also data. This leads to many challenges concerning security and fairness. Which nodes can be trusted with what kind of data? Can the results of a certain node be assumed to be correct? How can it be avoided that not always the same Requester occupies all Providers?

Another challenge is recovery from errors. There is never a complete guarantee that a Provider will actually compute a job it received. In a Distributed System, there is always the possibility that certain parts of a network suddenly disconnect from the rest of the network. For a Requester it is important that it does not just distribute the jobs and infinitely wait for the results, but instead have some sort of mechanism to detect if a Provider has taken too long to solve a job and react by re-distributing this job to another Provider.

2.2 Information-Centric Networking (ICN)

Information-Centric Networking [1] is a networking concept where the routing of traffic focuses on the actual name of a specific information available on the network rather than on the location of the information. The idea is that an Information Consumer only has to express the name of the information (i.e. some static content such as a video file, or a dynamic service such as performing an arithmetic operation and returning its result) he is interested in. This is in contrast to the current state of the Internet, where the information requester first needs to find out the location of the information he is interested in (i.e. by using a web search engine or by receiving the URL of the information by some other party) before sending a request with the name of the actual information to that exact location to initiate a data transfer. The motivation behind ICN is the fact that there are usually a lot more consumers of a specific piece of information than producers [8]. By each node actively caching information that passes through it, consumers can in many cases find the information they are looking for located much closer to them than the actual location of the producer. This helps to reduce a lot of redundant traffic on a network and, if successful, also decreases the time it takes to retrieve a given piece of information.

Information-Centric Networking is a fairly generic term that is the subject of research in many projects, such as Content-Centric Networking (CCN) [2], Data-Oriented Network Architecture (DONA) [9], Publish-Subscribe Internet Routing Paradigm (PSIRP) [10], and Network of Information (NetInf) [11]. While each of these research projects does have an own understanding and definition of ICN, there still are some characteristics that are common with all the projects [7]. Because the implementation part of this thesis is based on the CCNx implementation which is part of the CCN project, the general interpretation of ICN used in this thesis will also be similar to that of the CCN project.

Components

ICN networks have some core components that need a closer look.

- There are two different message types:

  Interest Messages Interest Messages represent the request of an Information Consumer for a specific piece of information. They contain the name of the requested information plus optionally some additional flags, e.g. the ability to set the lifetime of the Interest Message (how long until it expires if no information by this name can be found), or the answer origin kind (can Information Providers produce new information to satisfy the Interest Message or can the Interest Message only be satisfied by already existing information that is cached in some location).

  Data Messages Data Messages represent the corresponding piece of actual information to an Interest Message. They are returned to the Information Consumer that expressed an Interest Message that matches their name. Data Messages can be any kind of file.

- There are three different network components:

Information Consumers Information Consumers are nodes in the network that consume information by expressing Interest Messages and receiving Data Messages.

Information Providers Information Providers are nodes in the network that provide information to other nodes by receiving Interest Messages and issuing Data Messages.

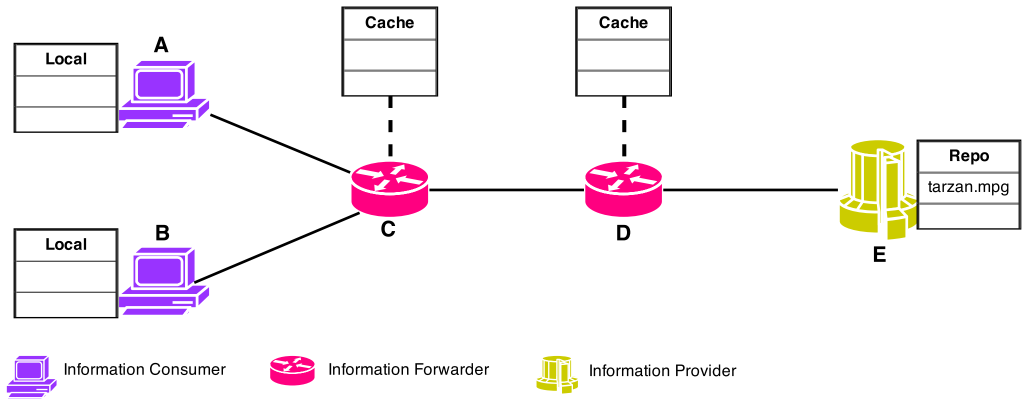
Information Forwarders Information Forwarders are nodes in the network that route Interest Messages and Data Messages between other Information Forwarders, Information Consumers and Information Providers. Information Forwarders usually are connected to a local repository where the passing Data Messages are cached.

Routing Example

The idea behind routing in an ICN network can best be explained with an example. A detailed explanation of a concrete implementation of ICN routing follows in Section 2.2.3.

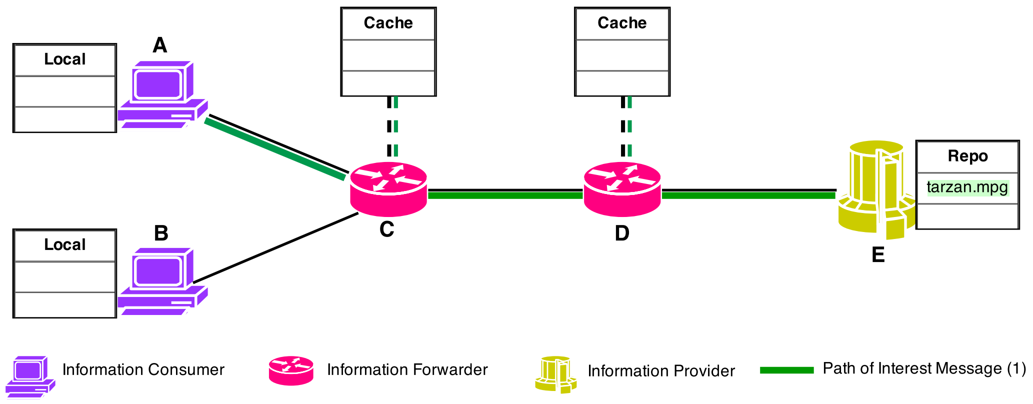
Two nodes, A and B, that both represent Information Consumers on an ICN network are successively interested in the same information, namely a video file called tarzan.mpg. The owner of tarzan.mpg is the node E, that represents an Information Provider. At first, all caches and local repositories are empty, and tarzan.mpg only exists on node E (Figure 2.3).

Figure 2.3: Example: ICN Routing I



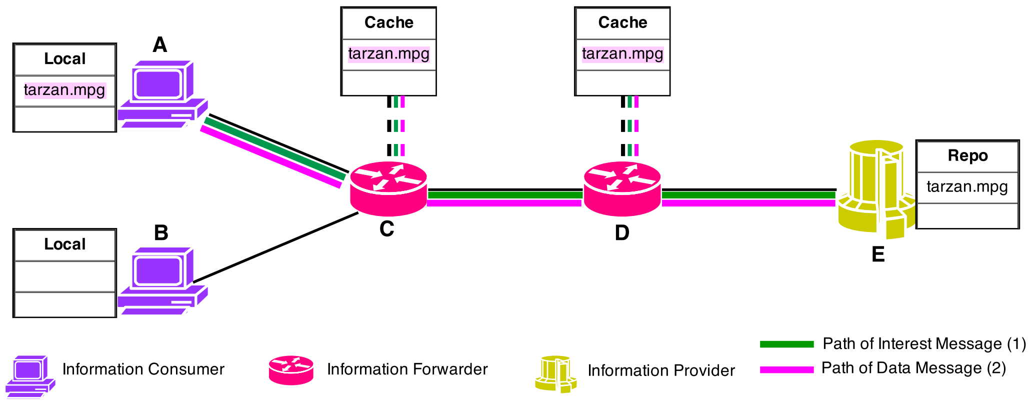
To communicate to the network that it is interested in the file tarzan.mpg, node A expresses an Interest Message (1) to the Information Forwarder C it is directly connected to. C receives the Interest Message (1) and first checks with its local repository to see if the file is stored there. Because this is not the case, C forwards the Interest Message (1) to Information Forwarder D with which it is connected. D also checks its local repository for a potential file named tarzan.mpg but the result is negative. D forwards the Interest Message (1) to the neighboring node E, which happens to be the Information Provider of tarzan.mpg (Figure 2.4).

Figure 2.4: Example: ICN Routing II



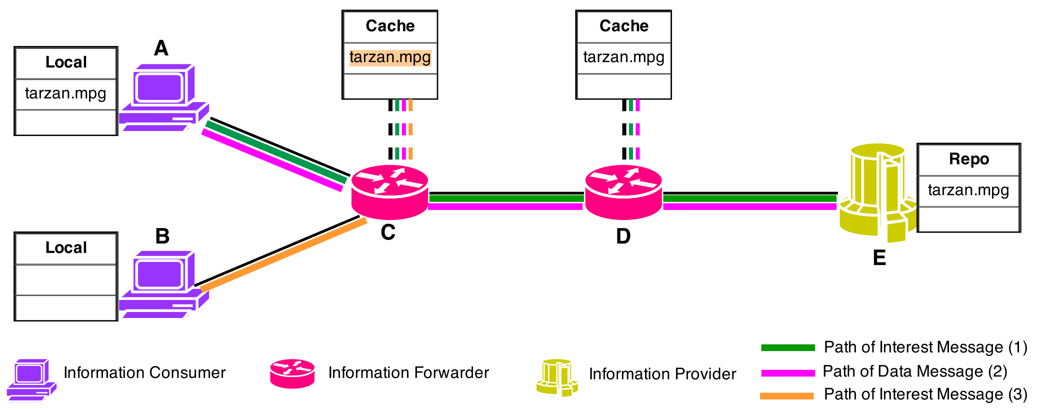
When node E receives the Interest Message (1) from node D, E responds to D with a Data Message (2) containing tarzan.mpg. D and every other Information Forwarder that forwards Interest Messages memorizes for every incoming Interest Message from which node it was received. The Data Message (2) just follows the opposite direction of the route of the Interest Message (1) until it ultimately ends up at the Information Consumer that expressed the Interest Message (1). Every Information Forwarder that forwards a Data Message (2) also caches it by saving the information contained in the Data Message (2) to the local repository (Figure 2.5).

Figure 2.5: Example: ICN Routing III



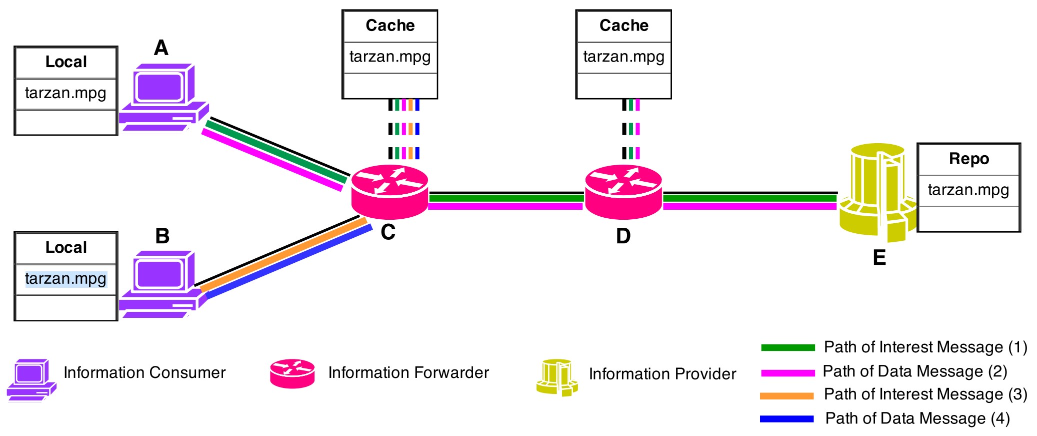
Now, when node B is also interested in tarzan.mpg, it will also express an Interest Message (3) that it forwards to the connected Information Forwarder C. Because C has already forwarded the file tarzan.mpg in an earlier instance (when node A requested it), a copy of the file will be cached in C's local repository. Therefore, the check for the file in the cache of C will return a positive result this time. (Figure 2.6).

Figure 2.6: Example: ICN Routing IV



There is no longer the need for C to forward the Interest Message (3) any further towards the original location of tarzan.mpg. Instead, C returns the file from its own cache as a Data Message (4) back to Information Consumer B (Figure 2.7).

Figure 2.7: Example: ICN Routing V



2.2.1 Content-Centric Networking (CCN)

While the term Content-Centric Networking can be used to generally describe a form of Information-Centric Networking where information is regarded strictly as (static) content, it is more importantly the name of a research project [2] that focuses on the implementation of ICN. The definition of CCN is based on these three tenets:

- Content is accessed by name, not machine address.
- Security is applied directly to the content, not the connection.

- Content may be cached opportunistically in the network for more efficient and scalable retrieval.

As part of the research on CCN, the CCNx Protocol was defined. Based on this, the actual software implementation of CCN, called CCNx, was developed. Details about CCNx can be found in Section 2.2.3.

2.2.2 Service-Centric Networking (SCN)

The term Service-Centric Networking (SCN) can be used to describe a form of Information-Centric Networking where information is regarded as a service. Because many initial definitions and implementations of ICN, such as the CCN project, make the assumption that information is just static content, it has been proposed [3; 12] that a better abstraction of information in regard to current Internet applications would be as a service. This is due to the fact that content can be easily regarded as a special kind of service (namely the service that returns the specific content requested) but not vice-versa. Additionally, this definition of information offers a lot more flexibility to Information-Centric Networking than to just return static content: A service can be any kind of function a computer can perform, e.g. solving an arithmetic operation, printing the current date, or encrypting a content file and returning it to the Requester.

Services in SCN are addressed the same way as content is addressed in CCN: by the name. The parameters required by a service can be encapsulated within its name. This way, the routing of content and services in an ICN network works identical. To a router it is not visible whether the name in an Interest Message corresponds to some content or some service. Moreover, the result of a service with specific arguments that has been processed is returned to the requesting Information Consumer as a Data Message. This is the same way static content is transferred over the network, which means the results of processed services can be regarded just as regular content and also cached among the Information Forwarder.

Because SCN works with the regular naming scheme of Interest Messages in ICN, current implementations of ICN can be extended to support also services as information, and not just static content. For example, for the CCNx implementation several projects are concerned with extending it with SCN capabilities [12], such as NextServe [4], CCNxServe [13], SoCCeR [14] and Serval [15].

For the programming part of this thesis, NextServe was used as the concrete implementation of SCN on top of CCNx (see Section 2.2.4).

2.2.3 CCNx

CCNx is a popular implementation of Information-Centric Networking that uses the definition that information is content. It is the core of the CCN research project [2]. The source code is written in the programming language C, but there is also an API available for the programming language Java. The following section mainly focuses on highlighting the aspects of CCNx that are relevant to achieving our distributed service mechanism. Extensive further documentation can be found on the CCNx website [16].

CCNx Message Types

The message types in CCNx are the same as those of ICN described earlier. In regard to our service distribution mechanism, there are some important implementation-specific details that need a closer look.

CCNx Interest Message The Interest Message is expressed by a Content Consumer and contains information about the content that the Content Consumer is interested in. The expressing Content Consumer passes the Interest Message to the connected Content Forwarder, from which it is routed to other Content Forwarders until there is a corresponding Data Message retrieved from either the Content Store or a connected Content Provider. The Interest Message contains eleven fields of parameters or flags that can be set, whereby only the name parameter is mandatory. Relevant to our implementation is the 'AnswerOriginKind' which defines how the origin of the corresponding Data Message should be. Options for the value include "Answer from cache only", "Answer may be generated" and "Answer may be marked stale".

CCNx Data Message Data Messages are used to transfer actual data over the network. An Data Message is always sent as a response to an Interest Message. Data Messages are also referred to as Content and Content Objects in the CCNx documentation.

CCNx Network Components

Just like the message types, also the network components in CCNx correspond to those generally considered in Information-Centric Networking, with some notable details that need looking at.

Content Consumer As the name says, Content Consumers consume content by sending Interest Messages and receiving the corresponding content as Data Messages. Content Consumers are always connected to a Content Forwarder, to which they express their Interest Messages.

Content Provider Content Providers are responsible for providing the network with actual content. They are connected to a Content Forwarder. A Content Provider can be an instance of an application that produces new content based on the incoming Interest Messages. NextServe connects to CCNx in this way. A Content Provider can also be a data repository or a file server with an appropriate interface.

Content Forwarder Content Forwarders act as routers in the CCNx network. They can be connected to other Content Forwarders as well as Content Providers and Content Consumers at the same time over a separate face for each Content Provider/Content Consumer/Content Forwarder. A face can be understood as a port into which another network component can be plugged in. A Content Forwarder forwards Interest Messages and Data Messages in a systematic and efficient way. To achieve this, it maintains a so called Forwarding Information Base (FIB), a Pending Interest Table (PIT) as well as a Content Store (CS).

Forwarding Information Base (FIB) This table contains entries for all outbound faces and the name prefixes that point to these outbound faces. It acts as a look-up table when the Content Forwarder needs to forward an Interest Message. The name in the Interest Message gets compared to the entries in the FIB and the Interest Message subsequently gets forwarded over the outbound face with the longest matching name prefix.

Pending Interest Table (PIT) This table contains entries for all pending Interest Messages that have been forwarded to outbound faces but have not yet received an answer in the form of a Data Message. For each Interest Message that was forwarded, the face of its arrival is stored. This information is used when a Data Message arrives that needs to be returned to the original requesting Content Consumer. When two or more equal Interest Messages arrive at the same Content Forwarder, only for the first one an entry in the PIT is created. This entry then just gets extended with the arrival face of the newer Interest Message with the same name.

Content Store (CS) The Content Store is a local repository that acts as the cache storage of the Content Forwarder. Frequently or recently delivered Data Messages (containing information) that have been forwarded by this Content Forwarder are cached here together with the name by which the information is identified.

Routing

Routing takes place on each Content Forwarder and consists of the two core operations of processing Interest Messages and processing Data Messages.

Processing Interest Messages

The processing of an incoming Interest Message by a Content Forwarder is performed in the following order (highest to lowest priority):

1. Check the Content Store if any cached content matches the name contained in the Interest Message. This might apply for multiple entries of the Content Store, in which case also the closer specifications of the Interest Message for making a selection from multiple entries apply. If some cached content in the Content Store indeed matches the Interest Message, forward this content to the arrival face of the Interest Message and, since it has been satisfied, discard the Interest Message. Otherwise, if no cached content matches the Interest Message, continue with 2.

2. Check the Pending Interest Table (PIT) for equivalent Interest Messages. If there is already an entry for an exact same Interest Message in the PIT, simply add the arrival face of the new incoming Interest Message to the existing entry and discard the new incoming Interest Message. Otherwise, continue with 3.

3. Check the Forwarding Information Base (FIB) for an entry with a matching prefix. If such an entry in the FIB exists, create a new entry of the Interest Message plus its arrival face in

the PIT and forward the Interest Message to the faces defined in the FIB entry. Otherwise, if no such entry in the FIB exists, continue with 4.

4. Because steps 1-3 have failed, there is no way this Content Forwarder can satisfy the Interest Message. The Interest Message can either be discarded immediately or kept for a short while, during which new FIB entries might be created that match the Interest Message.

Processing Data Messages

The processing of Data Messages is performed in the following order (highest to lowest priority):

1. Check the Content Store (CS) if any cached content matches the incoming Data Message. If this is the case, discard the incoming Data Message since the Interest Message that requested the Data Message must already have been satisfied in the meantime by the content in the CS. Otherwise, continue with 2.

2. Check the Pending Interest Table (PIT) for an entry that matches the Data Message. If this is the case, forward the Data Message to all the source faces defined in the PIT entry and then, because it has been satisfied and is no longer pending, remove the PIT entry. Otherwise, if no PIT entry matches the Data Message, continue with 3.

3. Because steps 1 and 2 have failed, the incoming Data Message is unsolicited and can be discarded without further action. The Content Forwarder can optionally still store the Data Message in the Content Store before discarding it.

2.2.4 NextServe

NextServe [4] is an extension to CCNx that provides CCNx with the functionality of Service-Centric Networking. NextServe integrates with CCNx as an application that represents both a Content Provider (for the processing of services) and a Content Consumer (for the parameter retrieval). It connects to a Content Forwarder in a CCNx network. CCNx and NextServe together offer the complete ICN experience by providing both content and service retrieval functionality.

Naming Scheme

Because NextServe is an extension of CCNx, and service Interest Messages are treated just as regular Interest Messages by CCNx routing, the naming scheme of services must be compatible with the naming scheme for regular content in CCNx. Names of services in NextServe have a special syntax similar to the method invocation style of object oriented programming languages. The first part of the name consists just of the name of the service, which is constructed of hierarchical naming. This ensures that CCNx correctly routes service Interest Messages to instances of NextServe so the services can be processed. The hierarchical part of the name is followed by the symbols "`/(`" which enable NextServe to locate the parameters that the service might need within the name of an Interest Message. Individual parameters are separated by the symbol "`,`". Finally, to mark the end of the parameter part of the name, the symbols "`/)`" are used.

Parameters

NextServe allows for three different types of parameters:

- Primitive types (strings, integers, doubles, etc.)
- Content
- Services

When a parameter is a primitive type, it is encapsulated within double quotations. Otherwise, NextServe interprets the parameter as the name of either some content or another service. NextServe will then look up the actual information stored under that name on the network by expressing an Interest Message. This happens recursively until there is a primitive type for each parameter. A total service composition functionality is ensured by this: It is possible to nest both a regular content Interest Message or another service Interest Message within the parameters of a service Interest Message.

Examples

A mathematical service to calculate the sum of two integers:

```
/math/sum/("5","2"/)                        => "7"
```

Another mathematical service to round a double value down to the next lower integer:

```
/math/floor/("1.54534"/)                    => "1"
```

A constant value like Pi can exist as content in a network:

```
/math/pi                                    => "3.14159265359"
```

With NextServe it is possible to perform service composition:

```
/math/sum/(/math/floor/(/math/pi/)/,"1"/)   => "4"
```

Implementing and Publishing Services

NextServe is written in the programming language Java and provides an interface for developers to implement and publish their own services on the network. The interface is called *Service Publisher* and has only the *publish* method that needs implementing. *publish* takes two parameters: The first one is a string that represents the name under which the service can be addressed by Interest Messages. The second parameter is an object which must have a method called *execute* that takes a list of the type ByteArray as a parameter. The ByteArray consists of all the primitive type parameters that were retrieved automatically by NextServe.

The actual business logic of the service is located in the *execute* method of the custom object and has no imposed limitations by NextServe. This means the business logic can consist of basically any possible functionality that can be implemented in Java.

Chapter 3

Design and Implementation

The ultimate goal of this thesis is to develop a mechanism to distribute tasks as services over a SCN network (in our case using CCNx and NextServe plus some custom Java code) in an efficient way by fulfilling the main criteria of performing Distributed Computing of a task: Faster turnaround time than locally processing the task. Our first approach, called the Trivial Approach (TA), is a very trivial, one-to-one ”translation” of our defined steps of Distributed Computing combined with the ideas of SCN. Because this does not satisfy our criteria of improving the turnaround time for solving a task, two further approaches are introduced: The Delegated Approach (DA) and the Probability-based Approach (PBA). The design of each approach is first outlined in a neutral way, independent to any implementation-related questions. After that, in the implementation section, the focus is to show how the design of each approach can be implemented using just CCNx, NextServe and some custom Java code.

3.1 Trivial Approach (TA)

The TA is an attempt to very closely combine the idea and steps of Distributed Computing with the principles of Information-Centric Networking (ICN). All the steps of Distributed Computing that involve communication between two or more nodes are handled in an ICN manner.

3.1.1 Design

A translation of the definition of Distributed Computing to the principles of ICN would consist of the following steps:

1. The Requester, which represents an Information Consumer attached to an Information Forwarder, defines the task that needs to be solved.

2. The Requester splits the task into several jobs. The jobs must be independent to each other: No job should rely on results of or use the same resources as another job.

3. The Requester forwards the jobs as Interest Messages via the connected Information Forwarder to all its neighboring Information Forwarders and Information Providers. The Interest Messages consist of the type of job along with the parameters for the job.

4. All Information Forwarders and Information Provider on the ICN network that receive an Interest Message will first check their own cache to see if there is any content already stored that satisfies the Interest Message. If this is the case, this content gets returned to the Requester as a Data Message and the Interest Message can be discarded. If there is no such content in the cache, an Information Provider either decides to compute the job by itself, discarding the Interest Message and returning the result as a Data Message to the Requester, or to ignore the Interest Message. This depends mainly on whether or not the Information Provider is capable of processing the Interest Message or not. Information Providers that are capable of processing the Interest Message are called Providers in our terminology. This means, there can exist Information Providers that are not regarded as Providers by our terminology: These Information Providers can be ignored, since they do not contribute any work to the distribution or processing of the Interest Message and simply represent a dead-end in the network. Information Forwarders that do not have a matching content in their cache will always forward the Interest Messages to the neighboring Information Forwarders and Information Providers.

5. After a Provider has processed an Interest Message, a Data Message containing the job result is returned to the Requester over the ICN network by breadcrumb routing.

6. The Requester waits until each expressed Interest Message has returned a corresponding Data Message, which contains the result of the job the Interest Message was representing. Once this is the case, the results of each job are merged to form the result of the initial task.

The actions in these steps can be divided into those that concern the Requester (flow chart in Figure 3.1) and those that concern the Providers (flow chart in Figure 3.2). As can be seen in the flow charts, only the Requester knows the current state of the distribution process. After the Providers publish their service, they simply operate in an infinite loop, processing all service Interest Messages they receive.

Figure 3.1: Flow chart: Requester in TA

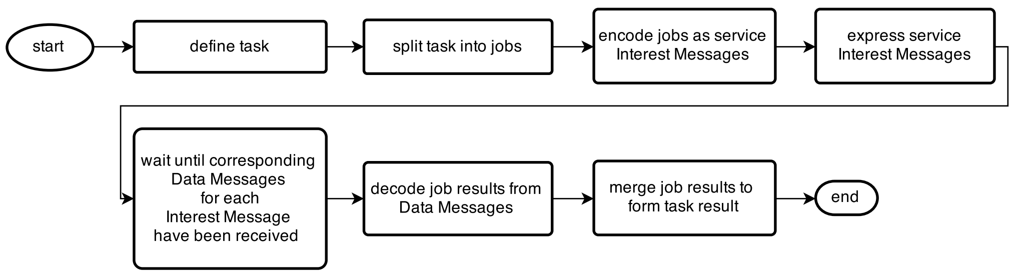
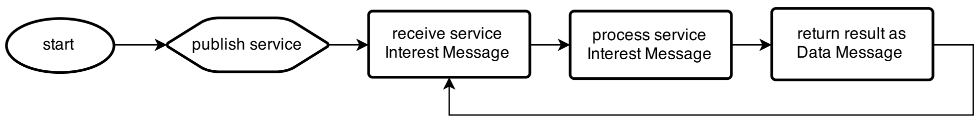

start
define task
split task into jobs
encode jobs as service Interest Messages
express service Interest Messages
wait until corresponding Data Messages for each Interest Message have been received
decode job results from Data Messages
merge job results to form task result
end

Figure 3.2: Flow chart: Provider in TA


start
publish service
receive service Interest Message
process service Interest Message
return result as Data Message

3.1.2 Implementation

Our implementation of the TA is written in the programming language Java as an extension to the existing code of CCNx and NextServe. The Requester is a custom Java application that acts as Content Consumer and is hosted on a machine together with a regular Content Forwarder, to which it is connected. The Providers are instances of NextServe acting as Content Providers that are connected to a regular Content Forwarder which is hosted on the same machine. The Content Forwarders of both the Requester and all the Providers are connected with each other over the CCNx network, either directly (Figure 3.3) or via one or more intermediate Content Forwarders (Figure 3.4).

Figure 3.3: Topology: Providers connected directly to Requester

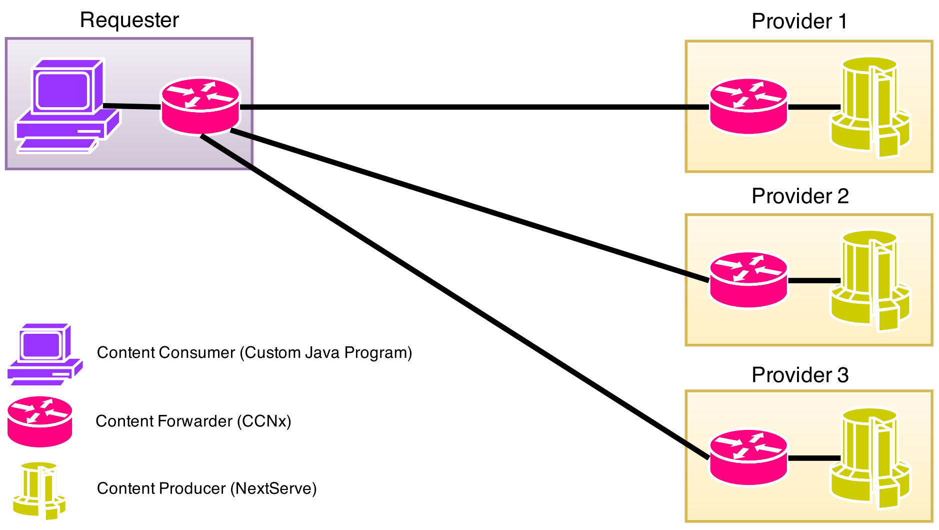
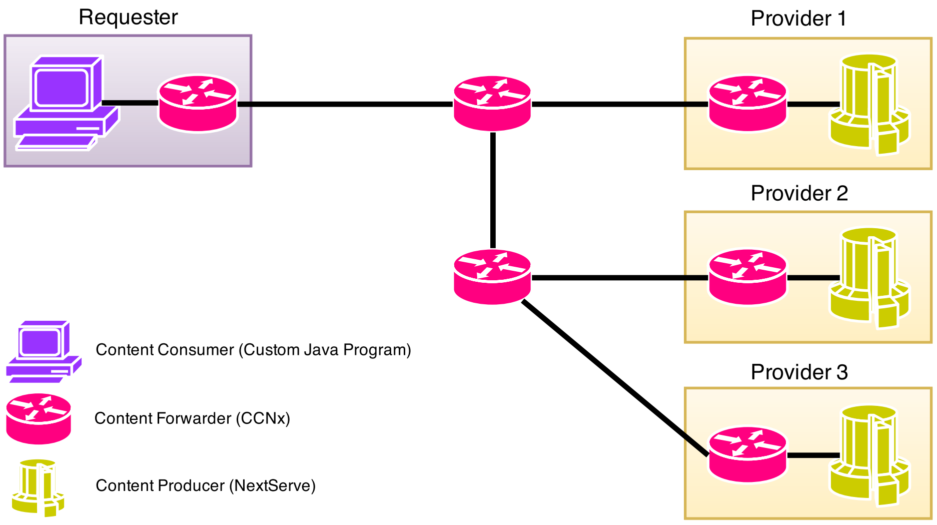

Requester
Provider 1
Provider 2
Provider 3
Content Consumer (Custom Java Program)
Content Forwarder (CCNx)
Content Producer (NextServe)

Figure 3.4: Topology: Providers not connected directly to Requester


Requester
Provider 1
Provider 2
Provider 3
Content Consumer (Custom Java Program)
Content Forwarder (CCNx)
Content Producer (NextServe)

Because a node in CCNx usually only has knowledge of its direct neighbors, with which it can only exchange Interest Messages and Data Messages but not additional information such as the amount of neighbors or other characteristics of the network, it is not possible for the Requester to know how many Providers there are on the network.

3.1.3 Analysis

In some specific cases, the TA can be very efficient in regards to reducing the turnaround time to solve a task. Imagine the case where many or all of the Interest Messages (that represent the jobs) can be satisfied by content already cached somewhere in the network, close to the Requester. The turnaround time will be much quicker than to locally compute the task, because there is only retrieving of data instead of actual computing to be done.

However, there are also opposite cases. Consider the network topology being similar to the one seen in Figure 3.4 and all caches on the network being empty. In this case, the Content Forwarder that is connected directly to the Content Consumer receives all the Interest Messages in the order they are expressed. Because no Interest Message is satisfied by any cached content in the Content Store of the Content Forwarder, in the current implementations of CCNx all Interest Messages will end up being broadcast to the neighboring Content Forwarders on the network. This means that all Interest Messages arrive at each connected Content Forwarder. On each of these Content Forwarders, the Interest Messages will either be passed on to the connected NextServe instance that is serving as a Content Provider (if this node is part of a Provider) or be forwarded once more to all connected Content Forwarders.

Every Provider in this scenario ends up processing each job that was expressed by the Requester. Even more so, the order in which the jobs are processed by each Provider will be identical. So before the Requester is able to merge all the results of all the jobs that the initial task was split into, it must first wait until at least one Provider has processed all of the jobs and returned the job results. Ultimately, it would have been better for the Requester to express the whole task as a single Interest Message or compute the task locally rather than splitting it up into several jobs and expressing each of these jobs separately.

Examining this problem a bit more closely reveals that this situation actually occurs for any network topology. Unless the strategy layer of CCNx routing is modified to support other strategies instead of just broadcasting (e.g. Round Robin), there is no way that the routing on a Content Forwarder would ever split or change the order of a set of incoming and outgoing Interest Messages with the same service name. This is because for routing, the Content Forwarder currently just queues the incoming Interest Messages and forwards them one-by-one while maintaining the order of the jobs.

In conclusion, the TA of distributing a task by splitting it into multiple jobs and expressing these as Interest Messages has proven to be less efficient than to just express the entire task as a single Interest Message, except for the (probably very rare) occasion when all or most of the Interest Messages can already be satisfied from cache.

3.2 Delegated Approach (DA)

The DA tries to overcome the problems experienced with the TA by altering the way the jobs get distributed to the Providers over the network. Basically, the idea is to not rely on the ICN routing to distribute Interest Messages but to establish a separate, non-ICN connection between the Requester and the Providers where the Requester can communicate with each Provider individually.

3.2.1 Design

The order of events in the DA is the following: At first, the Requester scans the network for potential Providers by expressing an Interest Message of the /scan service. The /scan service is designed to recursively spread itself further over the ICN network: It contains information about the kind of task the Requester wants to solve, along with the address over which the Requester can be reached over a non-ICN connection. Each node that processes the Interest Message is a potential Provider. If it provides the kind of service that is requested by the Requester (this service is stated as a parameter within the Interest Message of the /scan service), the potential Provider opens such a non-ICN connection to the Requester. Either way, whether it supports the stated kind of service or not, each potential Provider re-expresses an Interest Message of the /scan service to the network that again contains the kind of service that needs solving and the address of the initial Requester. The origin flag in these Interest Messages strictly must be set to 'non-cache', because it's crucial that the nodes actually process the /scan Interest Message instead of serving it with an answer from the cache. The content of the corresponding Data Messages that are returned is irrelevant. Due to the fact that each Provider that processes an

Interest Message of the /scan service also re-expresses it, we can be sure that the /scan Interest Message will arrive at each node of the ICN network.

After the Requester has sent out the initial /scan service Interest Message, it waits for a given period of time to be contacted by potential Providers at the address it specified within the Interest Message. To each Provider that reaches out to the Requester, a separate and independent connection is established. When the waiting period is over, the Requester will know exactly how many Providers are willing and capable of helping to solve the task. Based on this knowledge, the Requester can optimize the number of jobs it splits the task into. For example, it is advisable to have at least as many jobs as Providers so that each Provider can contribute in the first place. Because the Requester and Providers are connected in a peer-to-peer, non-ICN way, they can theoretically exchange all kinds of information, including also each Provider's computational resources. The scheduling (number of jobs, size of jobs, which job gets distributed to which Provider, etc.) can therefore be optimized extremely well.

Now the Requester splits the task into separate jobs and encodes them as Interest Messages, just as in the TA. However, these Interest Messages are then not expressed and distributed over ICN routing, but instead each Interest Message gets forwarded over the non-ICN connection to a specific Provider. Basically, the Provider could start processing its received Interest Message immediately, but this would mean that during the whole distribution process, there is never a check if any of the Interest Messages can be satisfied from the cache. Therefore, before a Provider starts processing an Interest Message, it first expresses the exact same Interest Message with the origin flag set to 'cache-only' on to the ICN network. If after a certain time the Provider does not receive a Data Message as a response to the Interest Message, it starts processing the Interest Message itself. Once the Provider is in possession of the Data Message that contains the result of the job represented by the Interest Message it returns this Data Message to the Requester via the non-ICN connection.

After the Requester has distributed all the jobs to the Providers as Interest Messages via the non-ICN connections, it waits until it receives for each of these Interest Messages a Data Message that contains the job result. Once this is the case, it closes all the connections to the Providers and merges the job results to form the overall task result.

Figures 3.5 and 3.6 show the sequence of actions performed by the Requester and the Providers, respectively.

Figure 3.5: Flow chart: Requester in DA

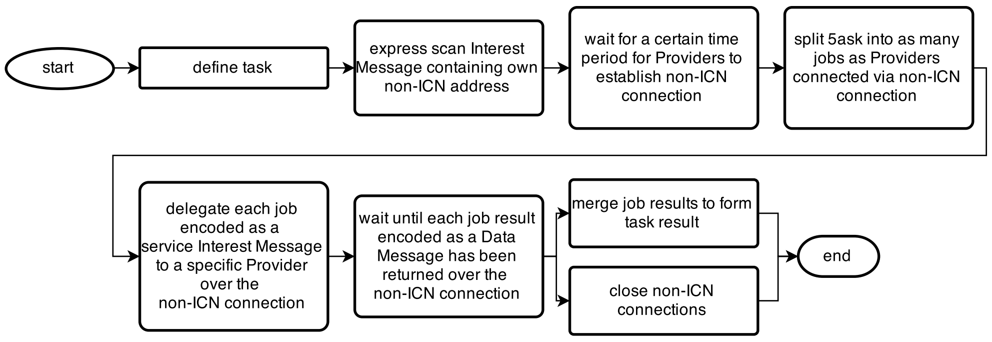
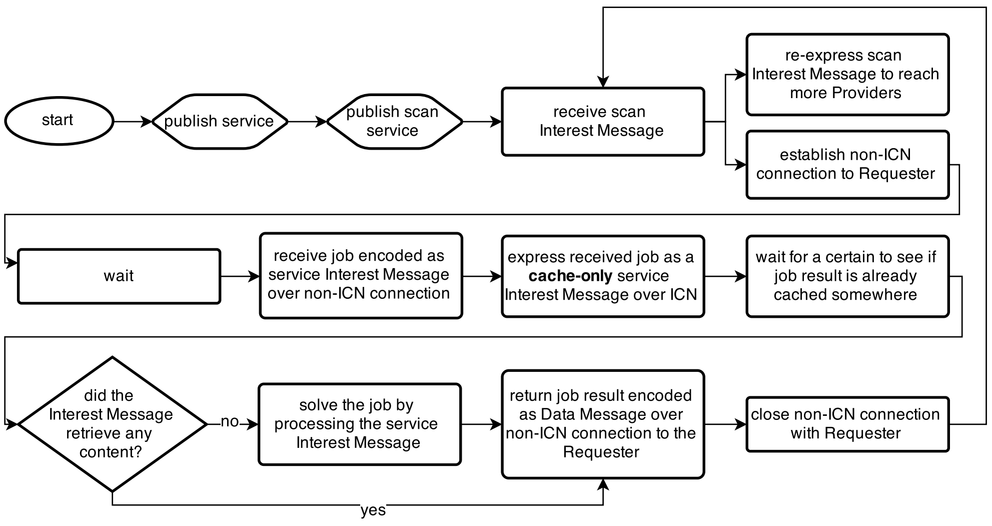

start
define task
express scan Interest Message containing own non-ICN address
wait for a certain time period for Providers to establish non-ICN connection
split 5ask into as many jobs as Providers connected via non-ICN connection
delegate each job encoded as a service Interest Message to a specific Provider over the non-ICN connection
wait until each job result encoded as a Data Message has been returned over the non-ICN connection
merge job results to form task result
close non-ICN connections
end

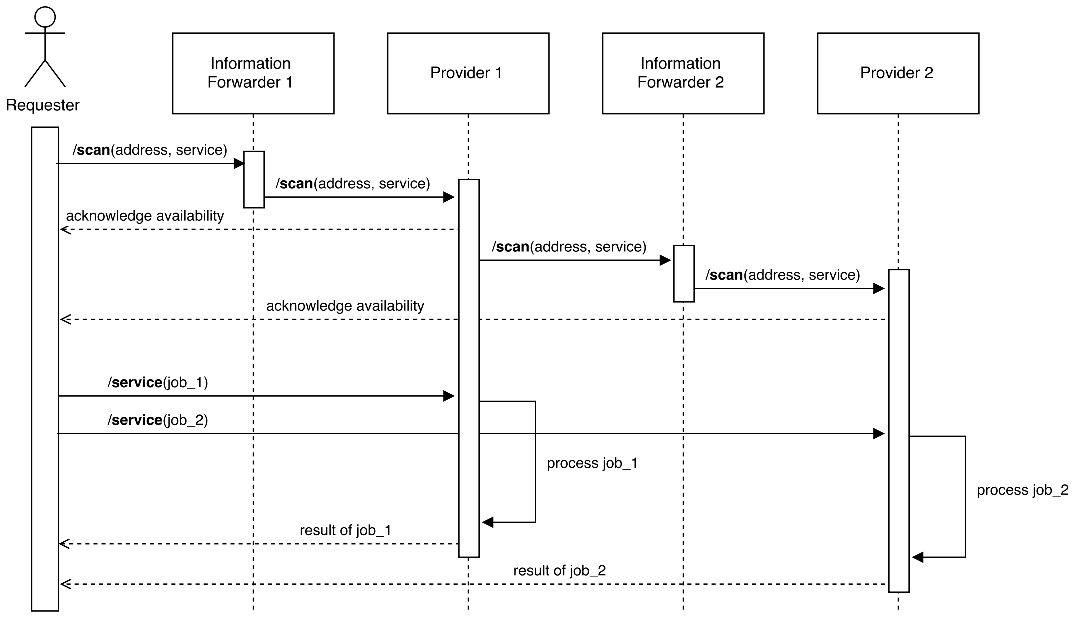
Figure 3.6: Flow chart: Provider in DA


start
publish service
publish scan service
receive scan Interest Message
re-express scan Interest Message to reach more Providers
establish non-ICN connection to Requester
wait
receive job encoded as service Interest Message over non-ICN connection
express received job as a **cache-only** service Interest Message over ICN
wait for a certain to see if job result is already cached somewhere
did the Interest Message retrieve any content?
no
yes
solve the job by processing the service Interest Message
return job result encoded as Data Message over non-ICN connection to the Requester
close non-ICN connection with Requester

Example

Imagine a network with one Requester, two Information Forwarders and two Providers. Information Forwarder 1 is connected to the Requester and Provider 1, Information Forwarder 2 is connected to Providers 1 and 2. Figure 3.7 shows the sequence of messages exchanged between these actors (for simplicity reasons without the cache-only Interest Messages that check the repositories of the network).

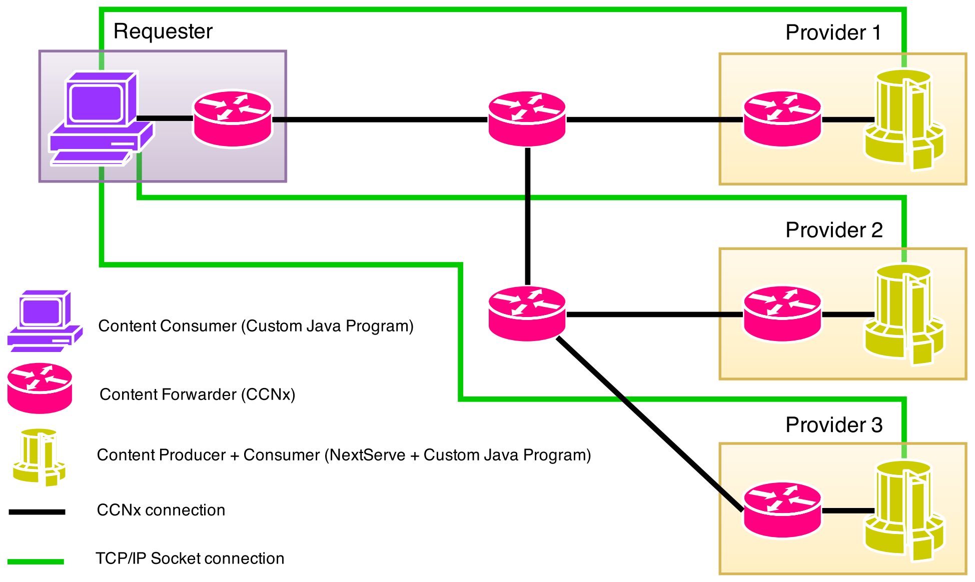
Figure 3.7: Sequence diagram: Actions performed by Requester, Information Forwarders, and Providers



3.2.2 Implementation

The DA can be implemented using CCNx, NextServe and some custom Java code that makes use of TCP/IP socket connections. The Requester is represented by a custom Java program that acts as a Content Consumer together with a connected Content Forwarder. The Providers consist of a NextServe instance with a custom extension that acts both as a Content Consumer (because it expresses Interest Messages to check the cache) and as a Content Provider (because it processes service Interest Messages). The NextServe instance has the /scan service published and is connected to a Content Forwarder. An example topology together with how the components are connected to each other can be seen in Figure 3.8.

Figure 3.8: Example: Possible Topology for DA



The custom Java code that gets executed by the Requester first retrieves the current IP-address of the Requester and sees which ports are available to be used for the socket connections. This information is then broadcast with a /scan service Interest Message over the CCNx network. It is important that the Interest Messages for the /scan service always have the origin flag set to 'no-cache', otherwise a Content Forwarder might return some cached information for the /scan service Interest Message instead of routing it to the Providers. As mentioned earlier, in NextServe the business logic of a service (that gets invoked when a Interest Message of the specific service is processed) is written in Java, meaning it is no problem to implement the opening of a socket connection as part of the processing of the /scan service. Also, it is possible to express Interest Messages from within the processing of a service and to wait for the corresponding Data Message (or time-out when it takes too long). This is exactly what happens when the Provider expresses its received job as a cache-only Interest Message to the CCNx network to find out if the processing of the service Interest Message is actually necessary.

3.2.3 Analysis

The problems with the DA are very clear: It involves nodes exchanging their addresses to establish non-ICN connections. Over these non-ICN connections jobs are distributed and job results are returned. Even though some parts of the communication (the /scan service Interest Messages) are still handled purely by ICN and the jobs are correctly represented by Interest Messages and job results by Data Messages, this is still too much of an infringement of the idea of ICN. As a service distribution mechanism over ICN, the DA is not sufficient. To show that the DA actually works, it was completely implemented as part of this thesis. However, we did not perform any evaluations with the DA because the results would not prove anything. The goal of the thesis is clearly to design, implement and evaluate a service distribution mechanism for ICN networks. On a side-note: The quick tests we performed did show that the DA does actually work very well and would absolutely be a suitable solution for a peer-to-peer network service distribution mechanism.

3.3 Probability-based Approach (PBA)

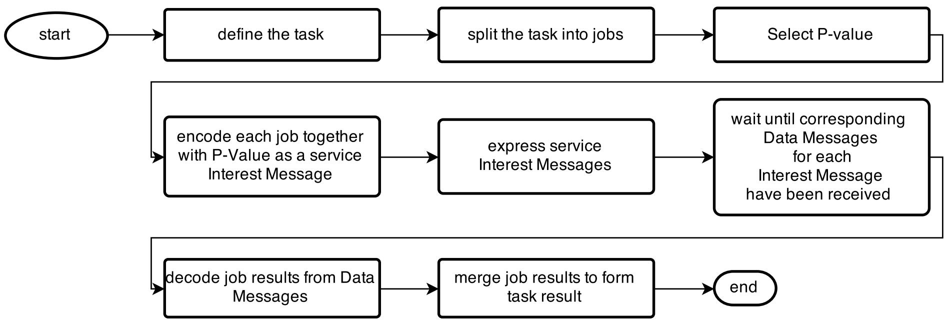
As seen in the TA, it is not possible to achieve an efficient distribution mechanism of services by simply encoding the jobs as Interest Messages and expressing them over the ICN network. In the DA, we found a way to delegate a job to a specific Provider (rather than to just distribute it to every Provider like in the DA), but this was established by additional communication between the Requester and the Providers using a non-ICN connection. This is to some extent not conform with our actual goal of creating a service distribution mechanism for ICN. The PBA is an ICN-only approach that reduces communication between the Requester and the Providers to a minimum. The only additional information (in comparison to the TA) the Requester distributes to the Providers is a probability value called ’P-value’ that makes each Provider randomize the order in which the incoming Interest Messages are processed.

3.3.1 Design

The P-value is the core element of the PBA. It is set by the Requester and added to the service Interest Message (that contains a job) before it is expressed. The P-value acts as a probability-based measure with which the Providers randomize the order of incoming service Interest Messages. When each Provider processes the Interest Messages that contain the jobs in a different order, in almost any case this reduces the time it takes until the Requester has received the result of each individual job via Data Message. Ultimately, this leads to a shorter turnaround time for the distribution of a given task.

Apart from setting the P-value, the work-flow of the Requester in the PBA is the same as in the TA.

Figure 3.9: Flow chart: Requester in PBA


start
define the task
split the task into jobs
Select P-value
encode each job together with P-Value as a service Interest Message
express service Interest Messages
wait until corresponding Data Messages for each Interest Message have been received
decode job results from Data Messages
merge job results to form task result
end

The behaviour of the Providers in the PBA is a lot more complicated than that of the Providers in the TA. Basically, the design of a Provider can be abstracted into four layers:

Service processing layer This layer is responsible for processing the actual jobs that are encoded as Interest Messages. After processing, it returns the job results as Data Messages directly back to the ICN network facing layer.

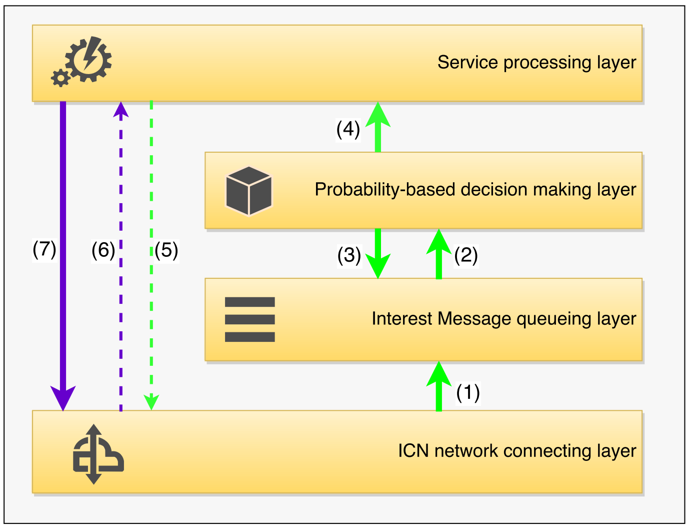
Decision making layer This layer takes Interest Messages from the queue (in the Interest Message queueing layer), reads their P-value and makes the decision to either forward the job contained within the Interest Message to the Service processing layer or to return it back to the queue. On average, only once in P times the decision will be to forward the job.

Interest Message queueing layer This layer contains a queue where all incoming Interest Messages are stored.

ICN network connecting layer This layer connects to an Information Forwarder of an ICN network. It publishes the services that the Service processing layer is capable of processing to the ICN network.

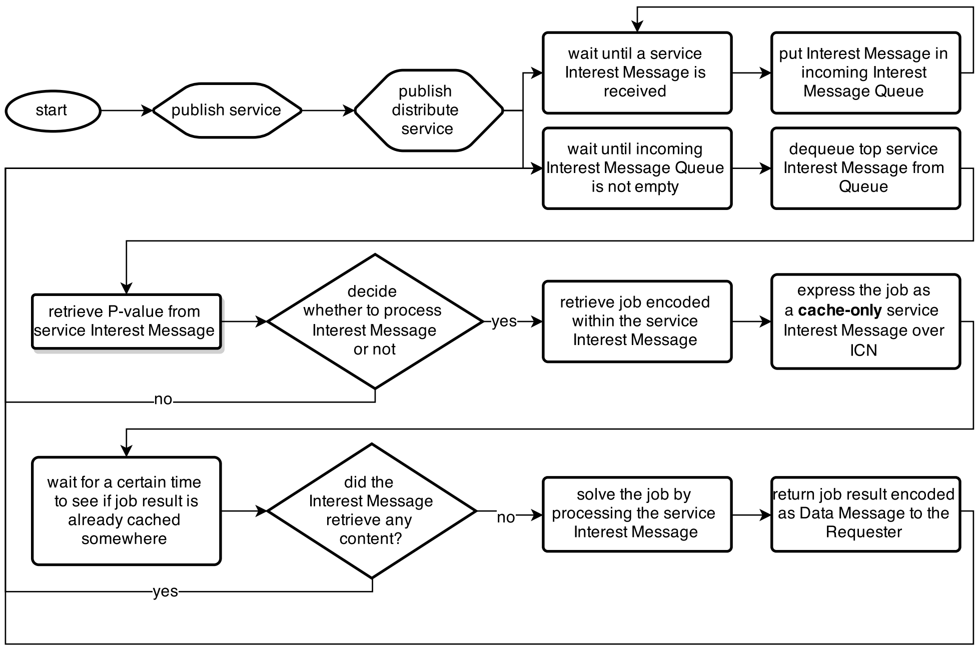
These layers interact with each other with the goal to randomize the order in which Interest Messages are processed. The routes that Interest Messages and Data Messages take between the layers can be seen in Figure 3.10.

Figure 3.10: Architecture: Layers of Provider in PBA


Service processing layer
(4)
Probability-based decision making layer
(7)
(6)
(5)
(3)
(2)
Interest Message queueing layer
(1)
ICN network connecting layer

The incoming Interest Messages arrive via the ICN network connected layer. They are then passed (1) on to the Interest Message queueing layer, where they are enqueued. The Decision making layer constantly dequeues (2) the top Interest Message from the queue and reads the P-value from the Interest Message. It then makes the decision whether or not the Interest Message is processed and passed (3) on to the Service processing layer. If it decides not to process, the Interest Message is returned (4) back to the Interest Message queueing layer where gets enqueued again. This decision is based on a probability that depends on the P-value: On average in 1 out of P times the decision is positive and the Interest Message is processed. In P-1 out of P times the decision is negative. When an Interest Message ends up in the Service processing layer, it is not directly processed right away. First, a cache-only Interest Message is expressed (5) to specifically search for cached information in the ICN network with the same name as current service Interest Message. This is due to the fact that the Interest Message may have spent a significant amount of time in the queue of incomingInterest Messages. It is possible, that in the mean time another Provider might have already processed the same service Interest Message. Should it be the case that another Provider has already processed exactly this Interest Message, then a corresponding Data Message would have been cached at some points in the network. These would react to the cache-only Interest Message and return the cached result back (as a Data Message) to the Provider that expressed the cache-only Interest Message. The Interest Message processing layer in this case would know that it is no longer necessary to process this service Interest Message (because it already received a Data Message containing its result). If no Data Message has arrived after a certain time, then the Interest Message processing layer finally can start to process the actual job. After the Interest Message has been processed, the corresponding Data Message is passed (7) on to the ICN network connecting layer from where it is returned to the Requester and cached in local repositories along the route.

Figure 3.11: Flow chart: Provider in PBA

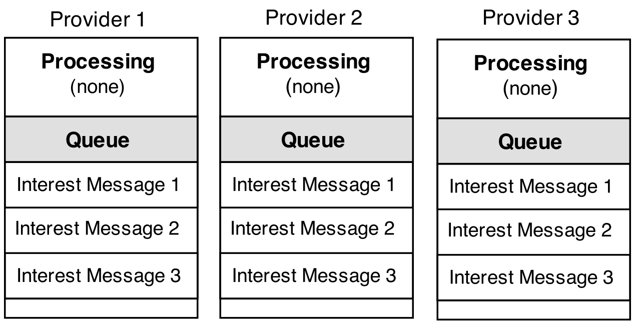

start
publish service
publish distribute service
wait until a service Interest Message is received
put Interest Message in incoming Interest Message Queue
wait until incoming Interest Message Queue is not empty
dequeue top service Interest Message from Queue
retrieve P-value from service Interest Message
decide whether to process Interest Message or not
yes
no
retrieve job encoded within the service Interest Message
express the job as a **cache-only** service Interest Message over ICN
wait for a certain time to see if job result is already cached somewhere
did the Interest Message retrieve any content?
no
yes
solve the job by processing the service Interest Message
return job result encoded as Data Message to the Requester

Randomizing processing order: Example

Imagine the case where the Requester splits a task into three jobs and sets P=2. The jobs together with the P-value form three Interest Messages. The network consists of one Requester and three Providers. For simplicity reasons, let's assume that it takes one iteration for a Provider to dequeue the next Interest Message and decide to either process the encapsulated job or enqueue the Interest Message again, and two iterations to process the job contained in an Interest Message.

Iteration 0: Immediately after the Interest Messages representing the jobs have been distributed over the ICN network by the Requester, the incoming Interest Message queue on all Providers looks identical.

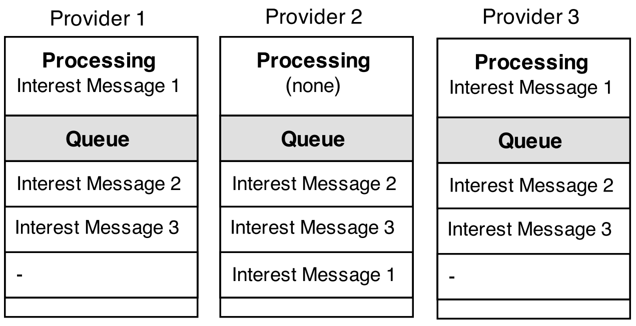
Figure 3.12: Example: Situation before iteration 1



Provider 1	Provider 2	Provider 3
Processing (none)	**Processing** (none)	**Processing** (none)
Queue	**Queue**	**Queue**
Interest Message 1	Interest Message 1	Interest Message 1
Interest Message 2	Interest Message 2	Interest Message 2
Interest Message 3	Interest Message 3	Interest Message 3

Iteration 1: The Providers each "throw a coin" to decide whether or not they process the job within the first Interest Message of the queue. Because in all Interest Messages the P-value is 2, the Providers will decide to process, on average, in one out of two cases. Let's say Providers P_1 and P_3 decide to process, while P_2 decides not to process and place the Interest Messages at the end of the queue instead. P_1 and P_3 now are busy processing the job, while P_2 iterates through its queue Interest Messages.

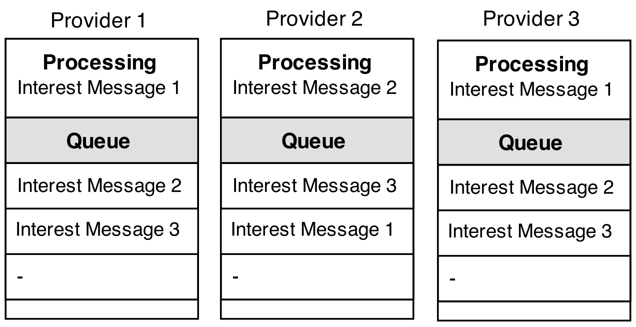
Figure 3.13: Example: Situation after iteration 1



Provider 1	Provider 2	Provider 3
Processing Interest Message 1	**Processing** (none)	**Processing** Interest Message 1
Queue	**Queue**	**Queue**
Interest Message 2	Interest Message 2	Interest Message 2
Interest Message 3	Interest Message 3	Interest Message 3
-	Interest Message 1	-

Iteration 2: Only Provider P_2 "throws a coin" to decide whether or not it processes the job within the top Interest Message of their queue. P_1 and P_3 are busy processing their first Interest Message (as defined earlier, the processing takes two iterations). Let's assume that P_2 decides to process the Interest Message.

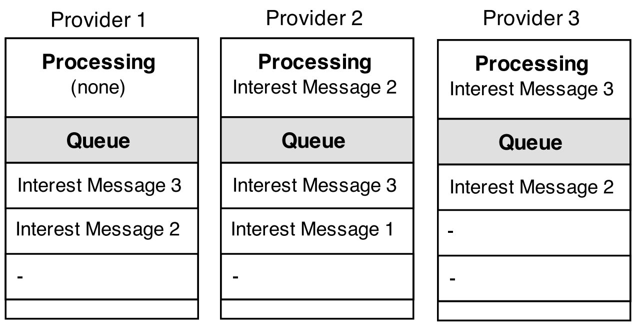
Figure 3.14: Example: Situation after iteration 2



Iteration 3: All Providers are busy processing an Interest Message. The situation remains the same as in Figure 3.14.

Iteration 4: Providers P_1 and P_3 have finished processing their first Interest Messages. They return the result of the Interest Messages back to the Requester (as Data Messages) and both take the next Interest Message from their queues to ”throw a coin”. Let’s assume that this time P_3 decides to process the Interest Message while P_1 decides not to.

Figure 3.15: Example: Situation after iteration 4



This mechanism continues on each Provider as long as there are Interest Messages in the queue. The Requester always receives the Data Message of a processed Interest Message immediately, meaning it only needs to wait until each job it expressed as an Interest Message has been processed by just one Provider to be able to merge the job results to form the overall task result. In the example above, this would be the case after iteration 5 when P_3 finished processing Interest Message 3.

3.3.2 Implementation

The PBA was already designed with certain limiting aspects of CCNx and NextServe in mind. However, there is one CCNx related complication that was not discussed in the design section:

Encapsulating the P-value in an Interest Message. The most straight-forward way would be to just add a parameter to the service for the P-value. However, this proves not to be practical because caching would be very limited. The problem is that two service Interest Messages that contain the same job but different P-values would not be considered identical. Take the following service as an example:

```
/math/factorial/("1000"/)
```

After extending this service with a parameter for the P-value, and setting P=3, it would look like this:

```
/math/factorial/("3","1000"/)
```

If this service were processed once, the result of factorial(1000) would be cached under the complete name:

```
/math/factorial/("3","1000"/)
```

When another Requester is interested in the same service but with a different P-value (e.g. P=5), CCNx routing would not recognize that the result is actually already cached. This is due to the fact that the new Interest Message would request a slightly different name:

```
/math/factorial/("5","1000"/)
```

Thankfully, there is a workaround the caching issue: NextServe supports nesting of Interest Messages, meaning that an ”inner” Interest Message can be placed within the parameters of an ”outer” Interest Message. This way, the earlier example of the /factorial service including the P-value can be split up into two Interest Messages, nested within each other:

```
/math/distribute/("10",/service/factorial/("1000"/)/)
```

Now, CCNx routing correctly routes the (outer) distribute service Interest Message to a NextServe instance that has the /distribute service published. When this instance receives an incoming /distribute service Interest Message, it can read the value of P, and decide if it chooses to process the service Interest Message located within the 'service' parameter of the /distribute service Interest Message. The result of the inner Interest Message, in our case the Interest Message

```
/math/factorial/("1000"/)
```

will then be issued and cached as a Data Message for both the inner and the outer Interest Messages.

In our implementation a Requester consists of a custom Java program (acting as a Content Consumer) together with a connected Content Forwarder, just like in the TA. Basically, the custom Java program for the Requester used in the implementation of the TA just needs to be extended to support the setting of the P-value and properly encapsulating the inner service Interest Message (with the actual job) inside the outer service Interest Message (with the P-value).

The Provider consists of an instance of NextServe with some extended code. Together they act as both a Content Provider and a Content Consumer. They are also connected to a Content Forwarder. The Content Forwarder would represent the ICN network connecting layer explained in the design section. The three other layers (the Interest Message queueing layer, the decision making layer, and the service processing layer) are located within an extended code base of the NextServe instance.

3.3.3 Analysis

The PBA seems to be an efficient way to distribute services on an ICN network. Because Interest Messages representing a job cannot be delegated to specific nodes in an ICN-only network, and there is no communication between nodes apart from Interest Messages and Data Messages, the use of probability seems like the next best measure to prevent all Providers from processing all jobs in the exact same order (which is the problem with the TA). The overhead to both computation power and time, caused by having to make the decision whether to process Interest Messages or not (and putting them back on top of the queue quite often), is a sacrifice that has to be made. Of course, the PBA could still be optimized in many ways. For instance, is there a clever way of choosing the right parameters for P-value and amount of jobs? If the Requester has absolutely no information about the ICN network and has never tried to distribute a task before, these parameters can really only be guessed. There could be no Providers available or there could be thousands. If the Requester has distributed a task before, it really only has the time it took for service Interest Messages to retrieve a corresponding Data Message and the order the Data Messages arrive as an indication of the characteristics of the ICN network. The evaluation part of this thesis will focus on testing the PBA with various for the P-value, the number of jobs and the number of Providers, to see whether they have actually any influence at all on the duration it takes to distribute a task or not.

Chapter 4

Evaluation

This chapter covers the evaluation process. All performed tests are explained and the corresponding measurements are presented and discussed. Due to the limitations that the Trivial Approach (TA) and the Delegated Approach (DA) presented (namely not being efficient and not relying exclusively on mechanisms of ICN, respectively) the evaluation process focuses on the Probability-based Approach (PBA).

The evaluation process was split into two phases: The first phase focused on evaluating the performance of the PBA as a service distributing mechanism in an ICN network independent from the the specific ICN implementation used. For this, we developed a Java application that can be used to measure the performance of the PBA on a model that behaves exactly like an ICN network. In the second phase we performed a series of tests on an actual CCNx network together with NextServe and the custom Java programs. These tests were divided into two scenarios: The Basic Scenario with very little data traffic between the nodes, and a Real-world Scenario with a lot of traffic between the nodes.

4.1 Model Testing

For model testing of the PBA, a Java application called PBA-Model was created that simulates the behaviour of an ICN network that uses the PBA as the service distribution mechanism. The PBA-Model allows for a four-dimensional input:

- `duration`

  The duration of the overall task processing time (if it were computed locally).

- `number of providers`

  The number of Providers that help process the jobs.

- `number of jobs`

  The number of jobs that the task is split into.

- `p-value`

  The P-value that remains constant for all jobs.

If the task is split into n jobs, we assume that the sum of the durations of the processing time of each job is equal to the overall task processing time. To achieve this, a special count task was defined.

Count task

The count task has a constant execution time, independent of the actual processing power of the machine that computes it. Furthermore, the task minimizes the amount of data exchanged between Requester and Provider. It consists of counting from 1 to a given number N in steps of one number per time interval and printing each number to a console. For example, the task could be ”count from 1 to 10, one number per second”. The console output would be

```
1 2 3 4 5 6 7 8 9 10
```

and the execution time would be exactly 10 seconds. If we were to split this task into two jobs that could be ran in parallel, the console outputs would be `1 2 3 4 5` and `6 7 8 9 10`. Assuming that the jobs are distributed to at least two separate machines, the execution would take exactly 5 seconds plus the splitting, distributing and merging overhead.

The PBA-Model works and behaves exactly like the PBA described in Section 3.3, except that there is no caching. If it can be proven that the PBA improves the turnaround time for a given task (compared to locally computing it) even without utilizing cached information of previously computed jobs, then it will most definitely improve the turnaround time when using caching. This is because the retrieval of cached information for a specific Interest Message will always be quicker than to compute it, so caching will even further improve the performance of the PBA.

The PBA-Model has some global configuration variables: The counting speed by default is 1 number per 1000 milliseconds. The speed at which the Providers dequeue the top Interest Message on their incoming Interest Message queue before they decide whether to process the contained job is also 1000 milliseconds. Finally, there is the speed variable. This by default is 1, which means the unit of the time measurement (in milliseconds) that is printed in the output for a certain set of parameters is also how it was measured. When the speed is increased, for example to `speed = 100`, then the output for the same parameters will be identical to those at `speed = 1`, but the computation will only take 1/100 as long. There are two modes that the PBA-Model can be executed in:

- Command-line mode: only one set of parameters
- Batch mode: varying, multiple sets of (iterating) parameters

4.1.1 Testing Setup

For all tests we performed with the PBA-Model, we used the batch mode and had the following global configuration for PBA-Model:

- `speed = 100`
  (the printed time measurements are multiplied by factor 100)

- `provider count interval = 1000 ms`

  (the Providers count 1 number per second)

- `provider queue refresh interval = 1000 ms`

  (the Providers iterate through the queue of incoming Interest Messages at one Interest Message per second)

4.1.2 Measurements & Discussion

From the parameters that need to be set each time the PBA-Model is executed, our only constant one was `duration = 100`. This means that the task would take 100 seconds to compute locally. All other parameters varied depending on what we were measuring. Because we constantly set `duration = 100`, we are left with a three-dimensional input of parameters for all measurements with the PBA-Model.

1st Series

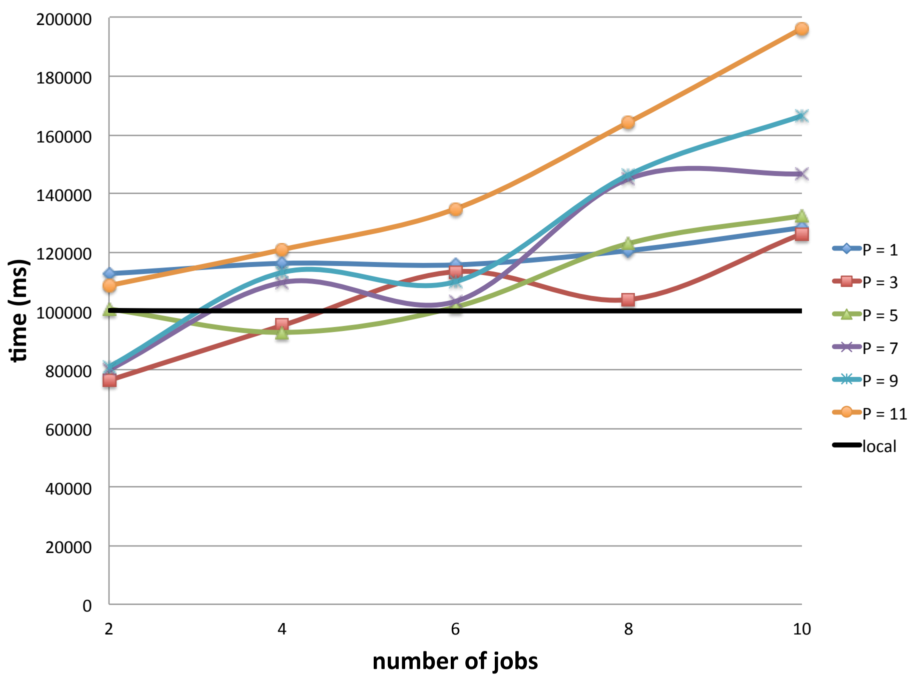
In the first series we defined a range of values for each of these dimensions and measured the duration to distribute the count task for every possible combination of values 10 times. These measurements were intended to show whether or not the PBA actually is an improvement over locally computing the task in the first place, and also which values make sense as an input for with parameter.

The values we considered as inputs were:

- `duration = 100`
- `number of providers = {2, 4, 6, 8, 10}`
- `number of jobs = {2, 4, 6, 8, 10}`
- `p-value = {1, 1.5, ..., 11.5, 12}`

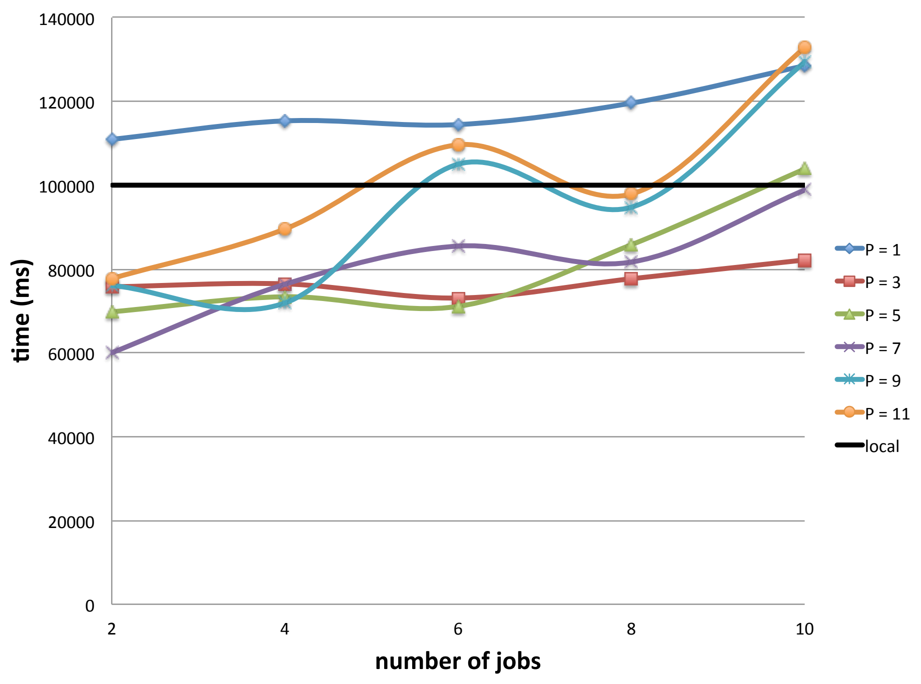
The measurements obtained this way can be aggregated for each considered number of Providers separately to show the mean time duration it takes per P-value to solve the count task with the PBA given the number of jobs the task is split into. For visualization reasons, the plots in Figures 4.1, 4.2, 4.3, 4.4 and 4.5 only contain the graphs with the P-values of the set $\{1, 3, 5, 7, 9, 11\}$.

Figure 4.1: Graph: Mean duration with 2 Providers for various P-values



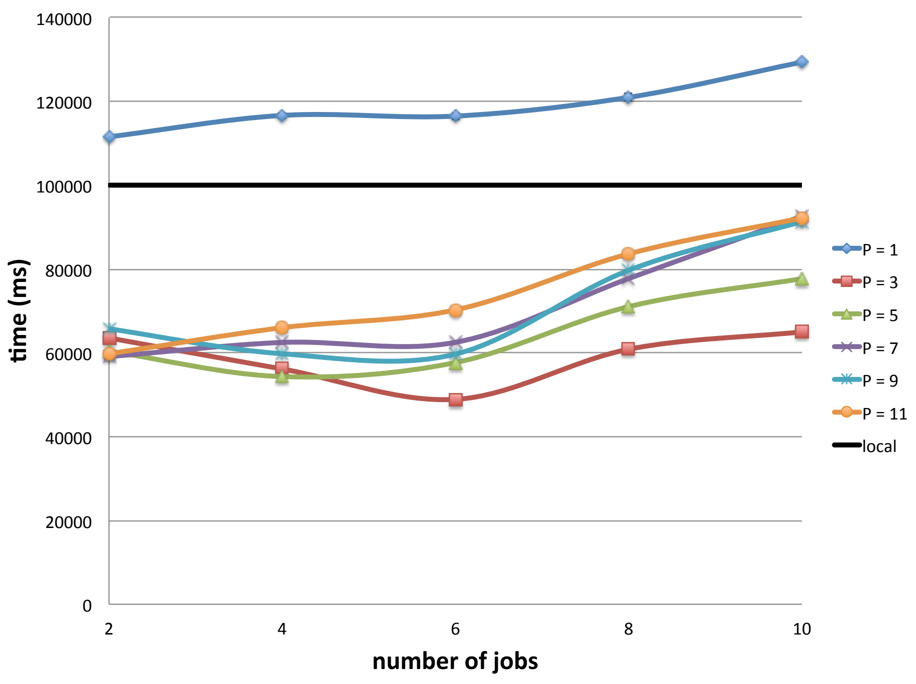
For the very small amount of only 2 Providers we can see in Figure 4.1 that for almost every combination of P-value and number of jobs, the mean duration to solve the task is longer than to just locally compute the task on a single machine.

Figure 4.2: Graph: Mean duration with 4 Providers for various P-values



Increasing the number of Providers from 2 to 4, as seen in Figure 4.2, already results in reducing the mean duration to solve the task for any combination of P-value and number of jobs, except for `P-value = 1`, which seems to be a special case. Interestingly, a lower P-value seems to result in a lower mean duration, as long as the P-value is greater than 1.

Figure 4.3: Graph: Mean duration with 6 Providers for various P-values



With a total of 6 Providers in Figure 4.3, the same observation as for Figure 4.2 can be made: The mean duration decreases (compared to the measurements obtained with 2 Providers less) for every combination of P-value and number of jobs, expect for `P-value = 1`.

Figure 4.4: Graph: Mean duration with 8 Providers for various P-values

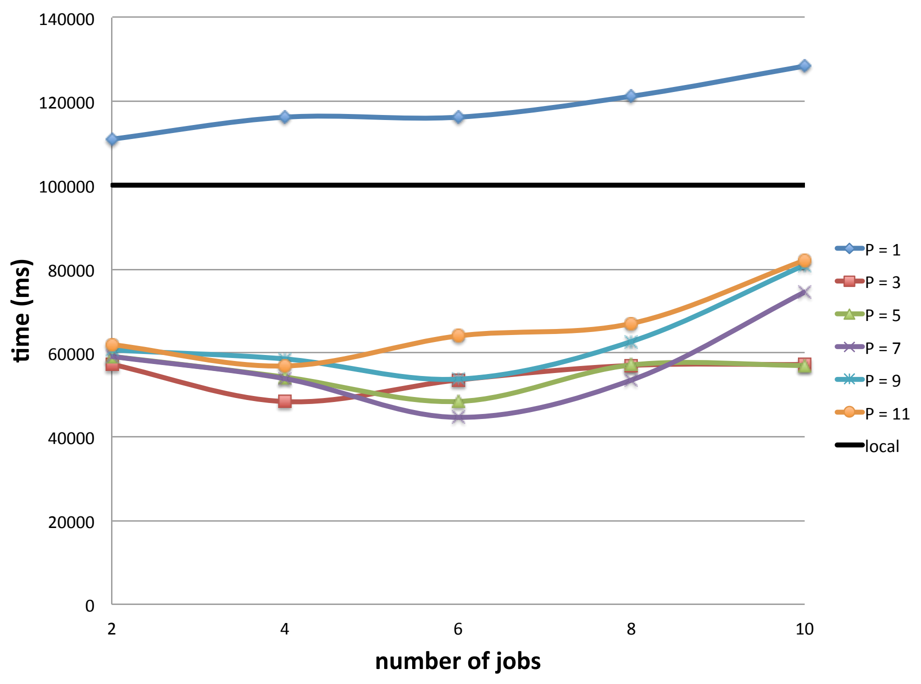
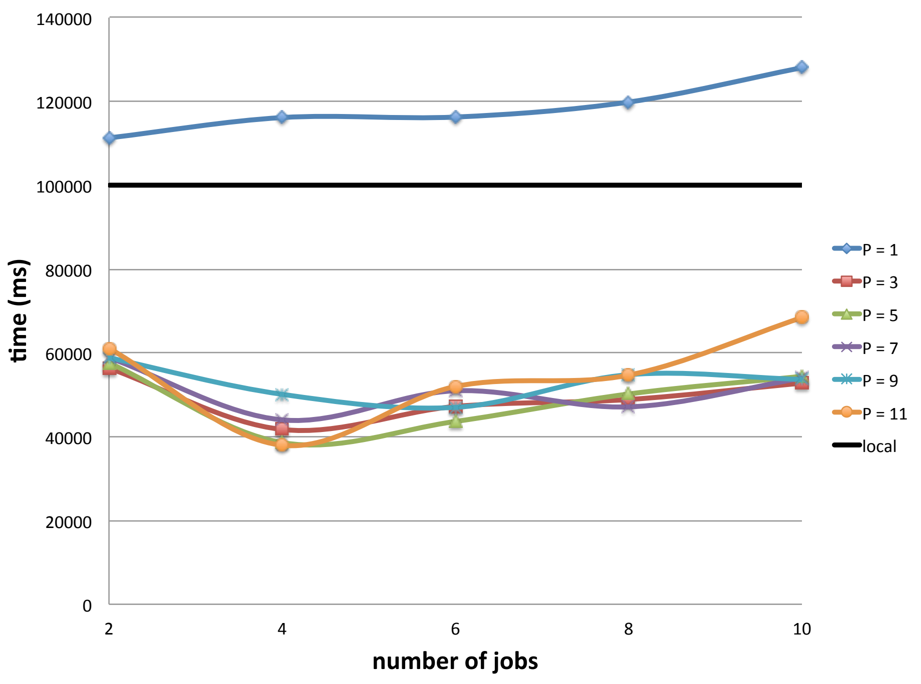


Figure 4.4 shows the graphs for combinations of P-value and number of jobs when distributing the task to 8 Providers. It appears that the shortest mean duration of solving the task is no longer necessarily achieved with a small P-value (such as `P-value = 3`) but rather for a P-value in the middle of our specified range of values (when splitting the task into 6 jobs or 8 jobs, `P-value = 7` actually has the shortest mean duration).

Figure 4.5: Graph: Mean duration with 10 Providers for various P-values



When using a total of 10 Providers to help processing the task (shown in Figure 4.5) it is visible that the choice of P-value no longer has much influence on the mean duration of processing the task: For any P-value of the set 3, 5, 7, 9, 11, the measurements are very close together for the same number of jobs that the task is split into.

As we can see from Figures 4.1 to 4.5, the mean duration to solve the task decreases for a growing number of Providers. This makes a lot of sense since having an additional Provider available to distribute and compute a task (while not changing the parameters of the P-value and the number of jobs) will not increase the duration of computing the task. In some cases, this will actually decrease the duration of solving the task because the Requester doesn't have to wait as long to have all the job results available to merge.

The critical number of Providers that help processing the task seems to be around 4. If there are less than 4 Providers available, for almost all combinations of P-value and number of jobs, distributing the task and having multiple Providers compute the task actually takes longer than to just compute the task locally on a single machine. As for the number of jobs, it is clearly visible that this should not exceed 6 jobs, as the duration increases for any number after that. However, the higher the number of Providers, the more equal is the mean duration of distributing the task for a varying number of jobs.

What also is visible in the graphs of Figures 4.1 to 4.5 is the fact that setting `P-value = 1` will always lead to a longer duration of solving the task than locally computing it. The reason is that with a P-value of 1, every Provider will always decide to process each of the incoming service Interest Messages. Essentially, the situation will be identical to that experienced with the

TA: Every Provider will process the service Interest Messages in the order they arrive (which is the same for each Provider) and the Requester will have to wait until the first Provider has finished processing every single service Interest Message (representing every single job). It would be faster to just send a service Interest Message consisting of the complete task and splitting the task into multiple jobs and distributing them separately.

In Figures 4.1 to 4.5 we can also observe that lower P-values seem to result in a shorter duration, as long as the P-value is greater than 1. This also seems understandable, since a higher P-value means that the Providers spend more time iterating through the queue of incoming Interest Messages before they decide to process a specific one.

Another way to aggregate the measurements is per number of jobs (Figure 4.6) and per number of Providers (Figure 4.7). For the aggregation of number of jobs, the measurements for all values of numbers of Providers are consolidated (per considered P-value), and vice-versa.

Figure 4.6: Graph: Mean duration depending on number of jobs for various P-values

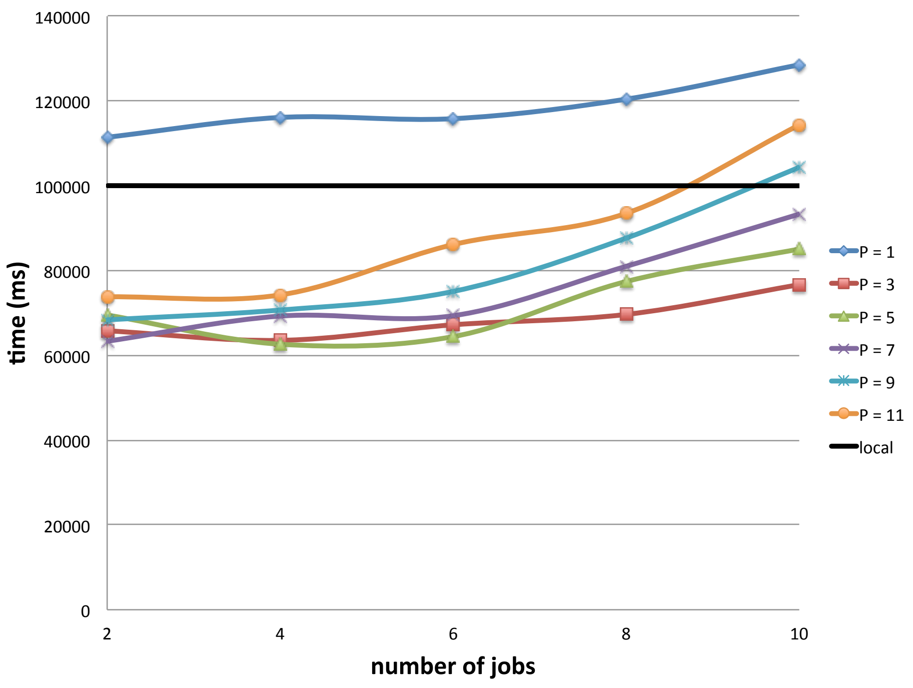


Figure 4.6, where the measurements are aggregated per number of jobs, confirms our earlier observations: The optimal P-value seems to be in the range between 3 and 5, and the optimal number of jobs around 4.

Figure 4.7: Graph: Mean duration depending on number of Providers for various P-values

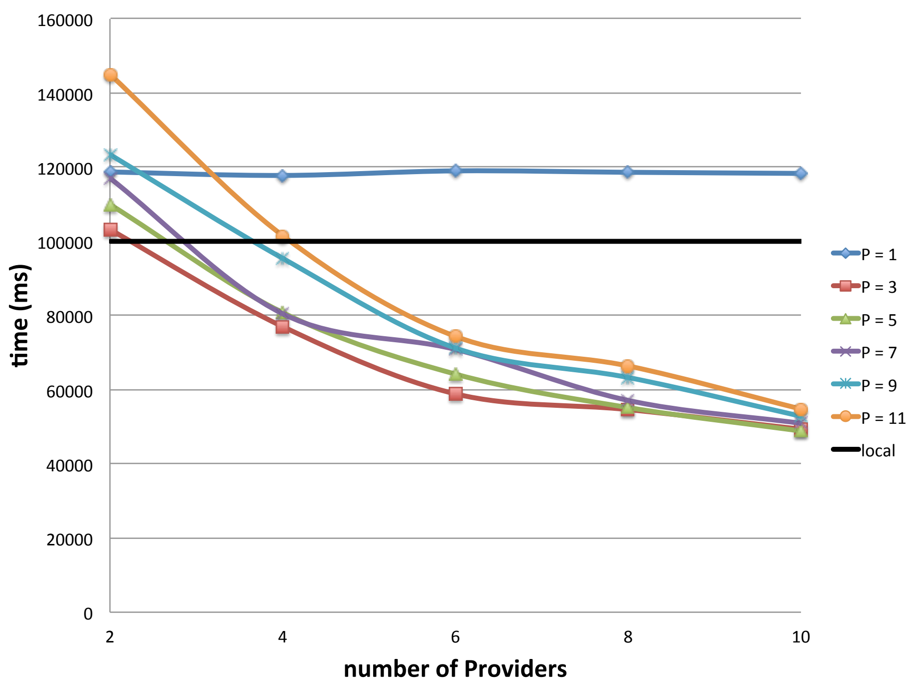


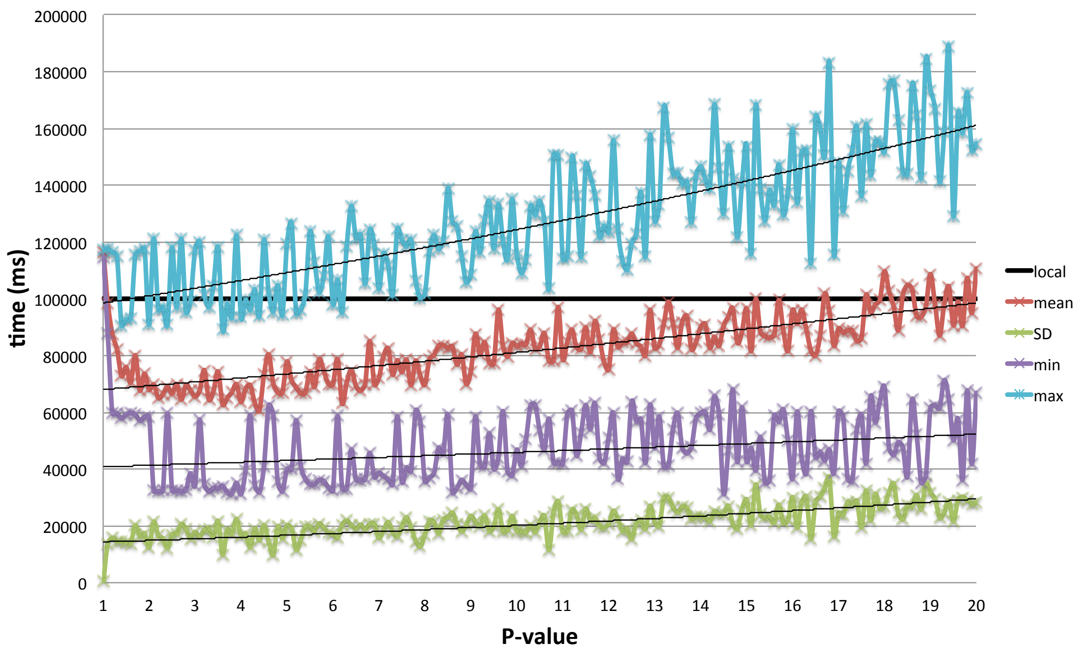
Figure 4.7 confirms the fairly logical conclusion that the mean duration (for any tested P-value except 1) decreases as the number of Providers grows. Interestingly, for a growing number of Providers the influence of the P-value is reduced: The larger the number of Providers, the closer together are the mean durations of all the tested P-values (except `P-value = 1`).

2nd Series

The second series of measurements with the PBA-Model focused on a detailed examination of the P-value's influence. In addition to the `duration` parameter, we also set `number of providers` and `number of jobs` to constant values. The only variable parameter for the 2nd series therefore was `p-value`. For each P-value we measured the turnaround time 20 times. From this we determined the mean, the standard deviation (SD), the minimum and maximum turnaround time for each P-value.

- `duration = 100`
- `number of providers = 4`
- `number of jobs = 4`
- `p-value = {1, 1.1, ..., 19.9, 20}`

Figure 4.8: Graph: Duration to solve task depending on P-value



In Figure 4.8 (where the mean duration, standard deviation (SD) of the duration, minimum value and maximum value of 100 measurements of the duration per tested P-value are plotted) it is immediately visible that after about `p-value = 4` the mean duration and all associated values slowly increase as the P-value grows. Therefore, at least for our chosen number of 4 Providers, the chosen P-value should not be greater than 4. For just about every tested P-value, the maximum duration that was measured was greater than 100 seconds. This means that there is never a guarantee that distributing the task will necessarily have a positive influence on the turnaround time. However, looking at the minimum duration that measured during the 100 iterations for each P-value, we can also see that it is still possible for the duration to be below 40 seconds, regardless of P-value.

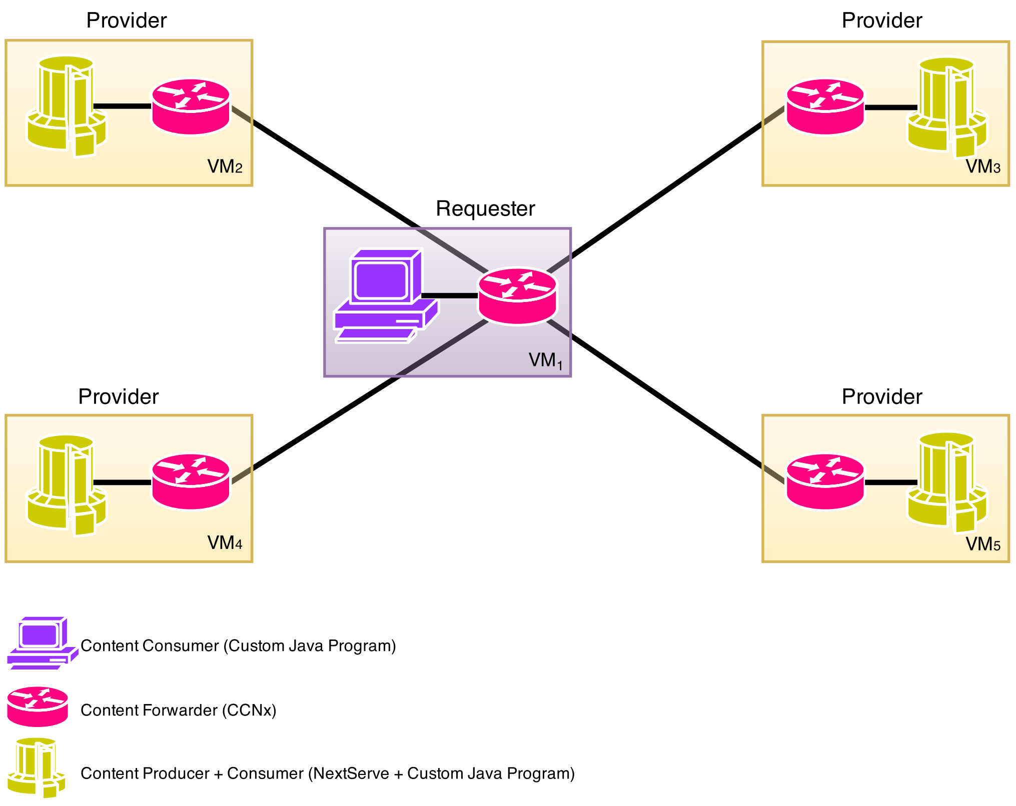
4.2 Basic Scenario

For the Basic Scenario the count task (as described in Section 4.1) was fully ported to a PBA implementation in an CCNx/NextServe environment. The /count service for NextServe that behaves just like the count task was implemented as part of this. The goal of the Basic Scenario is to prove that the results we obtained with PBA-Model also hold in a concrete implementation of the SCN paradigm. As described earlier, the count task tries to rule out all possible outside factors of the evaluation process that could interfere with the time measurements. Possible outside factors include limitations of the disk reading/writing speed and processing power on the physical machines, as well as limited speed and bandwidth between the nodes for data transfers.

4.2.1 Testing Setup

For the measurements, we used just one physical machine that hosted five Virtual Machines (VMs). VM_1 represented the Requester and ran a CCNx node acting as a Content Forwarder, together with a custom Java program that took care of splitting the the count task into count jobs, expressing the service Interest Messages and finally merging the jobs results. VM_2, VM_3, VM_4 and VM_5 were the Providers: They all ran CCNx as a Content Forwarder together with an instance of NextServe extended by some custom code that was capable of performing the count service required to solve the jobs. All five machines were assigned their own IP-address, with which they all were connected to each other over a CCNx network.

Figure 4.9: Testing setup for the Basic Scenario

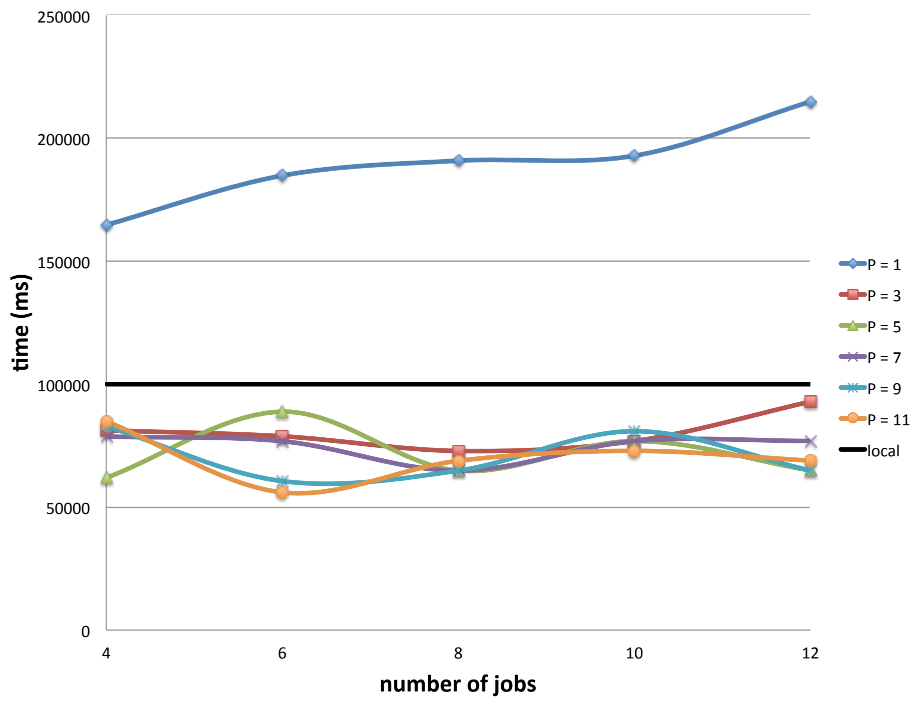

Provider
VM2
Provider
VM3
Requester
VM1
Provider
VM4
Provider
VM5
Content Consumer (Custom Java Program)
Content Forwarder (CCNx)
Content Producer + Consumer (NextServe + Custom Java Program)

4.2.2 Measurements & Discussion

Our measurements had `number of providers` constantly set to 4. The `duration` of the count task was set to 100 seconds. For `number of jobs` we tried all of the values of the set

{4, 6, 8, 10, 12}, and for `p-value` those of the set {1, 2, ..., 11, 12}. Again, for readability reasons the graph only display the measurements for the P-values of the set {1, 3, 5, 7, 9, 11}.

Figure 4.10: Graph: Duration of Basic Scenario for various P-values



From the visualized measurements in Figure 4.10 we can read that the PBA also performs well in a concrete implementation of SCN (in our case CCNx/NextServe). We can observe that again for `P-value = 1` the duration for distributing the task is longer than to locally solve the task, but for any other P-value greater than 1 this is no longer the case.

4.3 Real-world Scenario

The Real-world Scenario aims to be a lot more realistic than the Basic Scenario: The task is to transcode a video file from a specific file format to another.

We have the video file movie.avi (MPEG-4 video and AAC sound) of 10 minutes length with a 1080p resolution. This needs to be transcoded to a .mpg video file (PAL DVD format MPEG-2/H.262) with a smaller 576p resolution. As in the Basic Scenario, we compared and evaluated the duration of computing this task locally to distributing it using the PBA distribution mechanism in a CCNx/NextServe network. Again, we also considered different P-values and variable numbers of jobs that the task was split into. To reduce the complexity of the Real-world Scenario, we made some basic assumptions:

- There is only one transcode operation, meaning it is immediately clear that a given file needs to be transcoded from 1080p .avi (MPEG-4 video and AAC sound) to 576p .mpg

(PAL DVD format MPEG-2/H.262). So there are no parameters necessary to define the input and output file format of the /transcode service.

- All Providers already have the original to be transcoded movie.avi video file available locally. So as soon as a Provider decides to process a an Interest Message of the /transcode service, it does not first have to retrieve the original video file. If this weren't the case, our measurements would depend a lot more on how the original video file movie is available on the CCNx network initially, rather than on the effectiveness of our service distribution mechanism.
- The /transcode service requires the parameters `file`, `start` and `end`. `file` needs to be name of the movie file, `start` is the second mark of where to start transcoding and `end` is the second mark of where to end transcoding.

Example

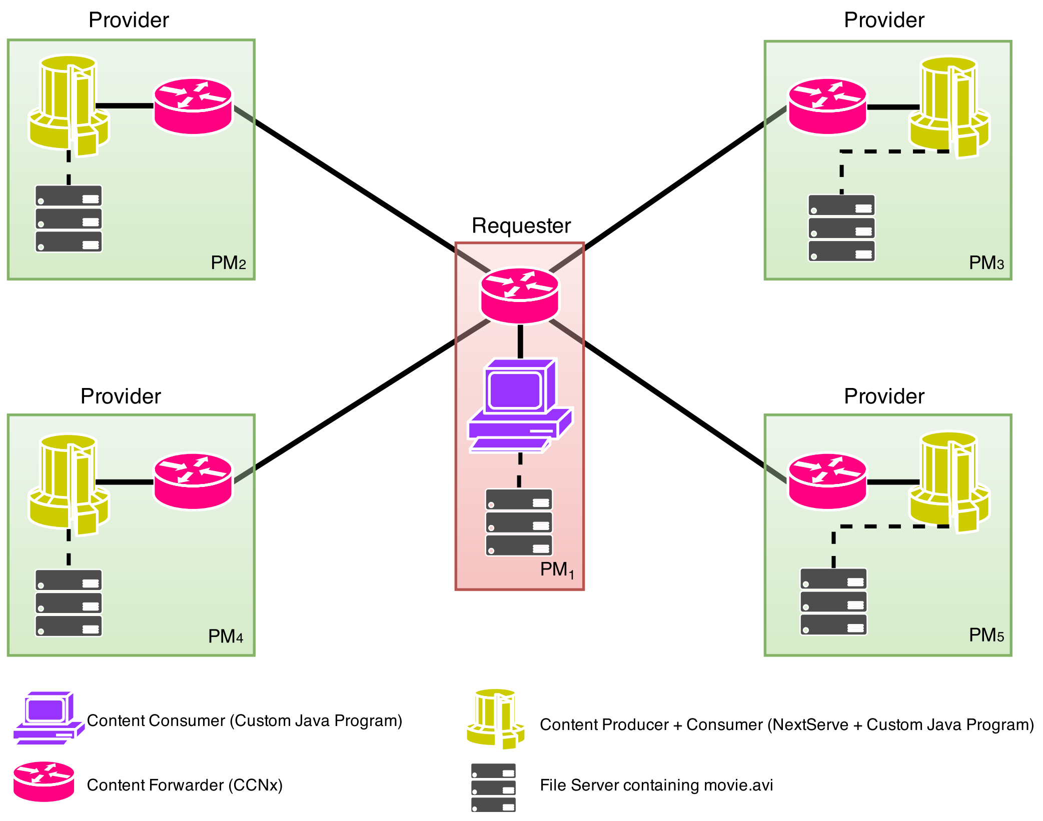
A service Interest Message to receive the transcoded part between minutes 1 and 3 of the file movie.avi would have the following name:

```
/video/transcode/(movie.avi,"60","180"/)
```

4.3.1 Testing Setup

For the measurements, we used five physical machines (PMs). PM_1 represented the Requester and ran a Content Forwarder together with the custom Java program. PM_2, PM_3, PM_4 and PM_5 were the Providers: They all ran a Content Forwarder together with an instance of NextServe. They also had published the services required by the Requester in this scenario, namely the /distribute and the /transcode services. All five physical machines got assigned their own IP-address, with which they were connected to each other over a CCNx network.

Figure 4.11: Testing setup for Real-world Scenario


Provider
PM2
Provider
PM3
Requester
PM1
Provider
PM4
Provider
PM5
Content Consumer (Custom Java Program)
Content Producer + Consumer (NextServe + Custom Java Program)
Content Forwarder (CCNx)
File Server containing movie.avi

4.3.2 Measurements & Discussion

For the parameter `number of jobs`, we considered the values of the set $\{2, 4, 8\}$. For the parameter `p-value`, we considered those of the set $\{1, 2, 3, 4, 6, 8, 12, 16, 32, 64\}$. For each combination of number of jobs and P-value, we ran the transcode task four times and measured the duration from the moment the task is launched until the final transcoded video file is fully merged and available on the Requester. We also measured how long it would take the Requester to locally perform the complete task of transcoding the entire 10 minute video file.

Figure 4.12: Graph: Duration of Real-world Scenario for various P-values

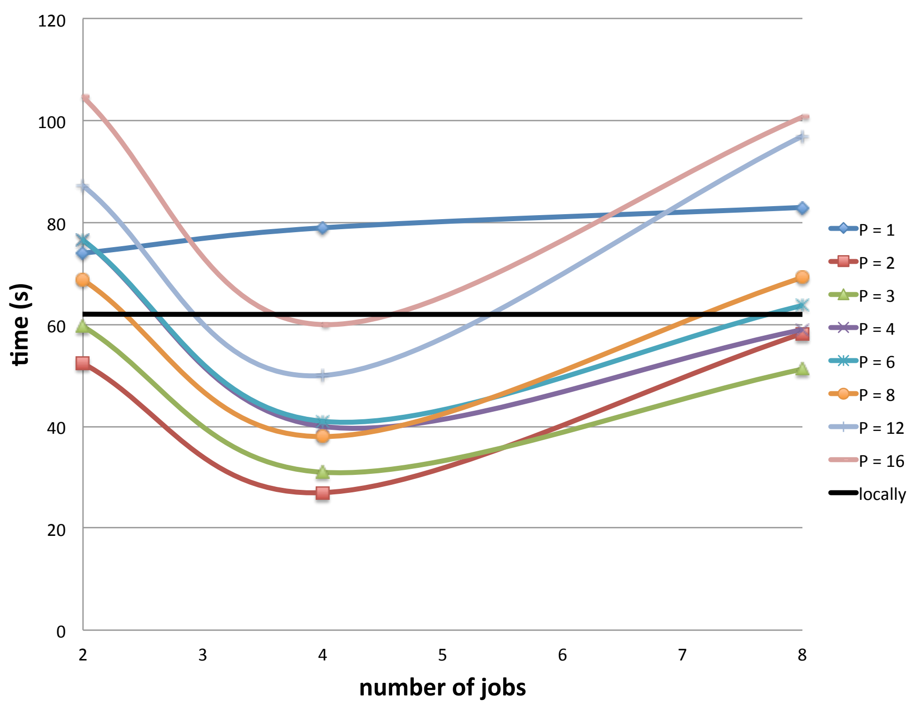


Figure 4.12 proves to us that the PBA is also successfully able to reduce turnaround time for a task we consider to be a Real-world task, albeit with less margin than we experienced on the Basic Scenario. It becomes clear that for the PBA to be efficient, parameters such as number of jobs and P-value need to be selected carefully. If the number of jobs is too small (e.g. 2) or too big (e.g. 8), for most P-values distributing the task is no longer more efficient than locally processing it. Once again, we can observe that for our rather small number of 4 Providers the best results are achieved with low P-values no greater than 6.

Chapter 5

Conclusion

5.1 Summary & Conclusions

In this thesis we designed and implemented three different approaches of a service distribution mechanism for Information-Centric Networking (ICN). The Trivial Approach (TA) appeared to not reach our goal of reducing the turnaround time for solving a task compared to locally processing it. The Delegated Approach (DA) seemed very promising with regard to performance, but it used some elements such as direct peer-to-peer connections that were not conform with the principles of ICN. Our third and final approach, the Probability-based Approach (PBA), finally appeared to both satisfy our criteria of being completely ICN-conform and actually distributing a task in an ICN network in a more efficient way than having it processed locally on a single machine. We developed a concrete implementation of a service distribution mechanism based on the PBA as an extension to CCNx and NextServe with which we could also successfully obtain measurements that confirmed our theoretical evaluation.

However, it must also be pointed out that it became evident that the ideas behind ICN and Distributed Computing are not fully compatible with each other: A very important aspect of Distributed Computing is the fact that a task can be split into jobs which are delegated to specific Providers, and not just broadcast to the entire network. To achieve delegation of a job to a specific Provider, the exact identity and therefore the location of the Provider would need to be known by the Requester. This is a contradiction to the main idea of ICN, by which the location of information is irrelevant and only its name is of importance. This leaves the conclusion that it is not completely possible to implement straight-forward Distributed Computing with the current form of ICN, simply because exact delegation of jobs is not permitted in ICN. Regardless of this, we were still successfully able to prove that it is certainly possible to find a consensus between the the two fields of ICN and Distributed Computing: The PBA is a service distribution mechanism conform to the ideas of ICN that works in many ways according to the functioning of Distributed Computing.

5.2 Future Work

There are multiple areas where future work could be considered on this topic. For a start, the scheduling mechanism of the PBA could be improved further. Rules need to be defined how to choose the best P-value with which to begin a distribution process. A mechanism for the Requester to dynamically adjust the P-value depending on the return rate of expressed Interest Messages could also be considered. Another research topic could be to adapt the idea of ICN so that the functionality of Distributed Computing can more closely be implemented.

Bibliography

[1] B. Ahlgren, M. D'Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone, "Design considerations for a network of information," in *ReArch*, 2008.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, (New York, NY, USA), pp. 1–12, ACM, 2009.

[3] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner, and M. Varvello, "Service-Centric Networking," pp. 1–6, June 2011.

[4] D. Mansour, T. Braun, and C. Anastasiades, "Nextserve framework: Supporting services over content-centric networking," in *The 12th International Conference on Wired and Wireless Internet Communications,* Springer, 2014.

[5] D. P. Vidyarthi, B. K. Sarker, A. K. Tripathi, and L. T. Yang, "Scheduling in Distributed Computing Systems: Analysis, Design and Models," Springer Publishing Company, Incorporated, 2008.

[6] A. S. Tanenbaum, and M. van Steen, "Distributed Systems: Principles and Paradigms (2nd Edition)," Prentice-Hall, Inc., Upper Saddle River, NJ, 2006

[7] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A Survey of Information-Centric Networking," in *Communications Magazine, IEEE,* vol. 50, pp. 26–36, July 2012.

[8] T. Zahariadis, D. Papadimitriou, H. Tschofenig, S. Haller, P. Daras, G. Stamoulis, and M. Hauswirth, "Towards a future internet architecture," in *The Future Internet*, vol. 6656 of Lecture Notes in Computer Science, pp. 7–18, Springer Berlin Heidelberg, 2011.

[9] T. Koponen et al., "A Data-Oriented (and Beyond) Network Architecture," Proc. SIGCOMM '07, Kyoto, Japan, Aug. 27–31, 2007.

[10] M. Ain et al., "D2.3 – Architecture Definition, Component Descriptions, and Requirements," Deliverable, PSIRP 7th FP EU-funded project, Feb. 2009.

[11] B. Ahlgren et al., “Second NetInf Architecture Description,” 4WARD EU FP7 Project, Deliverable D-6.2 v2.0, Apr. 2010,.

[12] D. Mansour, and T. Braun, “Survey in Service-Centric Networking,” 2014.

[13] S. Srinivasan, A. Singh, D. Batni, J. Lee, H. Schulzrinne, V. Hilt, and G. Kunzmann, “Ccnxserv: Dynamic service scalability in information-centric networks,” in *2012 IEEE International Conference on Communications (ICC)*, pp. 2617–2622, 2012.

[14] S. Shanbhag, N. Schwan, I. Rimac, and M. Varvello, “Soccer: services over content-centric routing,” in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ICN ’11, (New York, NY, USA), pp. 62–67, ACM, 2011.

[15] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Y. Ko, J. Rexford, and M. J. Freedman, “Serval: an end-host stack for service-centric networking,” in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI’12, (Berkeley, CA, USA), pp. 7–7, USENIX Association, 2012.

[16] “Ccnx project official website.” http://www.ccnx.org/. Last Checked, May 28th, 2015.